Verifiability and accountability in the Cloud

Monir Azraoui

2016-ENST-0032

EDITE - ED 130

Doctorat ParisTech

THÈSE

pour obtenir le grade de docteur délivré par

TELECOM ParisTech

Spécialité « Informatique et Réseaux »

présentée et soutenue publiquement par

Monir AZRAOUI

le 7 juin 2016

Vérifiabilité et Imputabilité dans le Cloud

Directeur de thèse : **Refik MOLVA**
Co-encadrement de la thèse : **Melek Önen**

Jury

Dr. Sébastien GAMBS, Université du Québec, Canada	**Rapporteur**
Dr. Keith MARTIN, Royal Holloway, University of London, Royaume-Uni	**Rapporteur**
M. Hervé CHABANNE, Morpho, France	**Examinateur**
Dr. Josep DOMINGO-FERRER, Université Rovira i Virgili, Catalogne	**Examinateur**

TELECOM ParisTech
école de l'Institut Télécom - membre de ParisTech

THÈSE

Thesis Dissertation

for the degree of Doctor of Science from

Telecom ParisTech
Computer Science and Networks

Monir Azraoui

Verifiability and Accountability in the Cloud

Jury composed of

Sébastien Gambs	Reviewer
Keith Martin	Reviewer
Hervé Chabanne	Examiner
Josep Domingo-Ferrer	Examiner
Refik Molva	Thesis adviser
Melek Önen	Thesis adviser

À mes proches, eux qui m'ont témoigné
un soutien inconditionnel lors de mon parcours.

Acknowledgments

Looking back to these years as a PhD student, any achievement I made has been possible thanks to the support and encouragement of numerous people. As this thesis is coming to completion, I would like to thank all these individuals and organizations that contributed to make this unforgettable experience possible.

I am sincerely indebted to three persons with whom I closely worked on the topics presented in this thesis. First, my deep gratitude goes to my supervisor **Professor Refik Molva** who sowed the seeds of success for our research. It has been a privilege to learn from his outstanding experience and knowledge and to receive his guiding support and invaluable advice. I also thank him for his steady enthusiasm and passion that he conveyed to me. His state of mind will always inspire me. As he often told me, I had much luck during my PhD studies. I would add that I was very lucky to work with such a great professor and to receive from him advice that will always be helpful in my career.

I also would like to express my sincere gratitude to my second supervisor **Doctor Melek Önen** beside whom I learned a lot. Her great experience, guidance and support helped me mature my research skills. I deeply thank her for her patience and tact when I sometimes showed stubbornness. I also acknowledge her effort to draw the best from me by giving me responsibilities in such a way that today I feel I have grown a lot. I finally thank her for reviewing my manuscript, for her valuable suggestions and corrections; it was a long process but we did it!

I am also deeply indebted to **Doctor Kaoutar Elkhiyaoui**, who highly contributed to the success of this research work. I thank her for sharing her eminent knowledge and open-mindedness. She is probably one of the cleverest persons I have ever met and it was a great pleasure to discuss with her, not only about our research work but also about history, literature or the news.

I gratefully acknowledge the thesis committee, **Mister Hervé Chabanne**, **Professor Josep Domingo-Ferrer**, **Professor Sébastien Gambs** and **Professor Keith Martin**, who kindly agreed to evaluate this work. I sincerely thank them for their availabilities and their encouraging feedback. I hope that we will have other interactions in the future.

This work would not have been possible without the funding of the **A4Cloud project**. Special thanks go to all the A4Cloud partners I have collaborated with. It has been very instructive to pursue this PhD thesis in the context of this project: I especially enjoyed working with very skilled people and observing how research and industry interact with each other.

I take this opportunity to thank the **Eurecom staff** for making this journey so special. My gratitude goes to **Gwenaëlle Le Stir** and **Audrey Ratier** for the administrative procedures related to my thesis and to **Christine Mangiapan**, **Audrey Ratier** and **Delphine Tales** for their caring about the travel procedures when I had the opportunity to travel for conferences and project meetings.

Pursuing PhD studies is truly a challenging endeavour whose success also depends on the encouragement of family and friends. I am therefore grateful to all the people who gave me their emotional support, sometimes by only asking the (somewhat disturbing) question *How is your PhD going?* First, I would like to thank all the friends I made at Eurecom. The school gives us the opportunity to thrive in a multicultural environment and I am delighted to have met so many Greek, Turkish, Italian and Spanish people. I thank my office mates Iraklis, Cédric and Dimitrios: An anodyne *Bonjour, ça va ?* is often good for morale. Special thoughts also go to Salvatore, Nikos, Panos, Katerina and Konstantinos whose friendships go beyond the walls of Eurecom and other thoughts go to the extended Greek and Italian communities in Nice and Sophia Antipolis with whom I shared lots of memories.

I always believed in the Latin phrase *mens sana in corpore sano*, a healthy mind in a healthy body. Therefore, alongside my PhD studies I practiced volleyball, squash and swimming within the Groupe Azur sport association. I thank all the athletes I met there, especially Benoît and Pierre-Yves, for their enthusiasm and their support. We had a lot of fun and those nice sporting moments helped me temporarily get away from my studies.

Aside my PhD work, I have followed theater classes for more than two years. I would like to thank all my fellow amateur actors, from Le Théâtre du Bocal, La Friche, Le Théâtre Athéna, as well as the crew of *Folle Amanda*, for these delightful moments that enabled me to open my mind wider. I specially thank Emmanuelle, our teacher and director, who became a real friend. I will always apply her instructions to improve my acting skills whenever I will have to give a talk in front of an audience. Her advice is a precious treasure.

I express my sincere gratitude to the following people living in Nice for their emotional support during this journey: I thank Yoann, for being so special and understanding; I thank the two “word in S”-est medical doctors that everyone would dream to have, Vanessa and Clément, for their friendship, the funny moments we had together and, of course, for their medical help!; I thank the greatest neighbours one can ever imagine to have, Lila and David, whose help and care were unhoped in a city like Nice; I thank Gregory for his kindness in spite of his life difficulties and for making me discover special places of Nice; I thank Juanjo for his humour, his Spanish dinners (and accent) and his drives between Sophia and Nice; I thank my great friend Guillaume for his support *concomitantly* to ours numerous laughters, especially about the ESC and Céline..; I thank Catherine, a long-time friend, for sharing thoughts about Mazamet. Warmhearted thoughts go to Cyrielle, for her love and care. She is maybe the most incredible person I have ever met and became rapidly important to my eyes. I thank her for our talks and the special moments we shared together. I thank her for being supportive even when my PhD thesis took up all my time and energy.

I am thankful to the friends I met in Paris, before the beginning of my thesis, and who, in spite of my gloominess that time, stood by me: I thank Gabriel for his regular messages caring about my PhD work and for his warm welcome in Sevilla; I thank Christophe, whose passion for philosophy inspired my work and state of mind; I thank my great friend, Michael, who came many times in Nice to create new memories to share; and finally I thank Pauline, for her love and her sweetness, our crazy moments and our *holorimes*.

I am lucky to have friends all around the world and I want to acknowledge the ones I met in Munich: Arnaud, Peryoun, Wolf and Guillaume. I thank them for their warm welcome when I went back in Munich, for their friendships and their nice messages asking about the evolution of my thesis. I also thank Arnaud for his visit in Nice and his company in Wrocław (*Polska biało czerwoni!*).

Another group of friends deserves my deep gratitude: la 846 de Toulouse. We celebrate this year the 10th anniversary of our friendship, and in spite this long period, they are still there. We were encouraging each other for our respective PhD studies and I am always glad to receive supportive messages from them. These people are Philippe, who defended his PhD thesis exactly at the same date and time as mine; Dr. Christophe, whose friendship is golden to me; Oriane, who is also about to be a doctor; Dr. Alexandre, one of the coolest and cleverest man I have ever met; President Fathi, whose eloquence is admirable. More thoughts go to the other members of the 846, with whom I had, unfortunately, less contact: Divin, Bastien, Matthieu and Patxi.

I cannot mention Toulouse without thinking of my dear friends, met there, ten years ago, but spread all around the world. Despite the distance between us, they always dropped a message to say hello and ask about my PhD work. Those are true friends that I know I can rely on, eyes shut. I thank Dr. Camille for her supportive messages and her cartoons that accurately describe the life of a PhD student; I am beholden to Boris, who opened his door each time I went to Paris during my PhD studies; I am thankful to Laure, whose messages from Congo-Kinshasa are always a great pleasure; and finally, I am grateful to Claire, for her cheering kindness that I will always remember.

I take the opportunity to express my profound gratitude to Nadine, who knows me since my early childhood, who has always been trying to be aware of my university and professional career and who has given me a lot of inspiring and guiding advice.

Last but not least, my most loving thoughts go to my family. Je voudrais adresser mes pensées les plus tendres et mes remerciements les plus intimes à ma famille, dont les membres sont dispersés entre l'Algérie et la France. Un chaleureux merci à ma famille à Alger qui m'a accueilli à bras ouverts pendant ma thèse, malgré 15 ans de séparation, comme s'il ne s'était passé que 2 jours. Je pense aussi à mes frères, Mouloud, Ahmed et Amine, qui, je sais, sont tacitement fiers de moi. Ils sont mon inspiration et mon moteur. Si j'ai poursuivi de longues études, c'est pour que plus tard, nous puissions chacun vivre dans de meilleurs conditions que l'ont été celles de notre enfance. Une douce pensée pour les nouveaux arrivés dans la famille: mes demi-frères Abou-Sofiane et Abdellah. J'ai tellement hâte de les voir grandir. Un grand merci à ma belle-mère Fatiha, qui m'accueille comme si j'étais son propre fils; merci pour ses encouragements et sa bienveillance. Je remercie aussi mon père Mohamed, pour son aide, pour son soutien et pour son amour. Sa présence a été et est très importante et son éducation m'a fait devenir ce que je suis aujourd'hui. Enfin, un profond merci à ma mère Fatma-Zohra, pour son amour, son soutien infaillible et inconditionnel ainsi que les soins qu'elle m'a réservé. Elle a toujours été là pour moi, même dans l'adversité, elle a fait preuve d'un grand courage pour élever quatre garçons et je sais que je lui en suis redevable tous les jours. Je les remercie tous pour avoir été derrière moi, pendant les moments heureux, comme les plus délicats, pendant les hauts, comme les bas. Cette aventure, je ne l'ai pas faite seul. Mes frères, ma belle-mère, mon père et ma mère, merci d'avoir cru en moi.

Abstract

Tremendous amounts of data are collected every day from sensors, social networks, mobile devices, etc., requiring large resources to store and process them. Cloud computing is perceived as the "holy grail" to cope with the handling of this "big data". As a matter of fact, the cloud offers virtually unlimited on-demand computational and storage resources. Hence, outsourcing the storage and processing of this data to the cloud cancels the need to invest in the maintenance of expensive storage and computing hardware or software. However, many organizations are still reluctant to resort to cloud computing technologies. Indeed, the inherent transfer of control over storage and computation from data owners to untrusted cloud servers raises various security and privacy challenges. In this regard, cloud users deem *verifiability* as an important security requirement: they should be empowered with the capability to check that the cloud processes their data and computations as expected. In particular, they should be able to verify that the data is correctly stored and that the computations are correctly performed. Besides, a further approach to encourage organizations to take advantage of the cloud is to consider *accountability* as a key prerequisite which enables cloud users to check that the cloud is compliant with policies and which allows these users to be aware on how their assets are handled. On the other hand, ensuring verifiability and accountability of cloud services should not sacrifice the benefits of migrating to the cloud. The literature describes cryptographic-based methods to verifiably outsource storage and computation. Recently, researchers introduced the concepts of Proofs of Storage and Verifiable Computation. However, many of the existing methods involve heavy cryptographic techniques, which render the solutions unpractical or inefficient. Besides, very few technical solutions were proposed to achieve accountability in the cloud.

This dissertation proposes new cryptographic protocols that empower cloud users to verify the correct storage of outsourced data and the correct execution of outsourced computation. Our solutions achieve their goals more efficiently than prior art while pursuing simpler approaches. We also introduce an accountability framework that expresses into machine-readable policies accountability obligations with respect to a correct provision of cloud services. In particular, we design a policy language for expressing accountability rules.

We first describe a cryptographic protocol that generates proofs of retrievability, which enable data owners to verify that the cloud respects its promise to correctly store their data. We then detail three cryptographic schemes for verifiable computation by focusing on three operations frequently used in data processing routines, namely polynomial evaluation, matrix multiplication and conjunctive keyword search. Each of these four protocols are publicly verifiable, provably secure under well-studied assumptions and we demonstrate their efficiency and practicality by means of prototypes. We finally present A-PPL, an accountability policy language that allows the expression of accountability obligations into machine-readable format in order to automate their enforcement. In particular, we show that A-PPL policies can be used to rule the execution of our protocol for verifiable storage.

We expect our contributions to highlight the fact that increasing trust in cloud computing technologies requires an interdisciplinarity of approaches (cryptographic protocols, policy language) in order to make the existence of verifiable and accountable cloud solutions possible.

List of Acronyms

ABE	Attribute-Based Encryption
AES	Advanced Encryption Standard
A-PPL	Accountability-PPL
A-PPLE	A-PPL Engine
aPRF	Algebraic Pseudo-Random Functions
BLS	Boneh-Lynn-Shacham
BOINC	Berkeley Open Infrastructure for Network Computing
CBF	Counting Bloom Filter
CDH	Computation Diffie-Hellman
co-CDH	co-Computation Diffie-Hellman
CE	Consumer Electronics
CS	Computationally Sound
CSA	Cloud Security Alliance
DDH	Decisional Diffie-Hellman
DL	Discrete Log
D-**SDH**	D-Strong Diffie-Hellman
D-**SBDH**	D-Strong Bilinear Diffie-Hellman
ECC	Error Correcting Code
ECRH	Extractable Collision-Resistant Hash Functions
ESA	European Space Agency
EU	European Union
FAQ	Frequently Asked Questions
FHE	Fully-Homomorphic Encryption
FFT	Fast Fourier Transform
GCD	Great Common Divisor
GUI	Graphical User Interface

i$\mathcal{O}$	Indistinguishability Obfuscation
IT	Information Technology
IP	Interactive Proofs
MAC	Message Authentication Code
m-DDH	multiple Decisional Diffie-Hellman
NASA	National Aeronautics and Space Administration
NIST	National Institute of Standards and Technology
O-RAM	Oblivious RAM
PAP	Policy Administration Point
PBC	Pairing-Based Cryptography
PCP	Probabilistically Checkable Proofs
PDP	Provable Data Possession
PDP	Policy Decision Point
PEP	Policy Enforcement Point
PIR	Private Information Retrieval
PRNG	Pseudo-Random Number Generator
POL	Proofs of Location
POR	Proofs of Retrievability
POS	Proofs of Storage
PPL	PrimeLife Policy Language
PPT	Probabilistic Polynomial-Time
PPWS	Privacy-Preserving Word Search
PRF	Pseudo-Random Function
PRP	Pseudo-Random Permutation
PVC	Publicly Verifiable Computation
PVCKS	Publicly Verifiable Conjunctive Keyword Search
QAP	Quadratic Arithmetic Program
QoS	Quality of Service
QSP	Quadratic Span Program
RIC	Remote Integrity Check
ROM	Random Oracle Model
RSA	Rivest-Shamir-Adleman

SCC	Signatures of Correct Computation
SETI	Search for ExtraTerrestrial Intelligence
SHA	Secure Hash Algorithm
SLA	Service Level Agreement
SNARK	Succinct Non-Interactive ARgument of Knowledge
SSE	Semantically-Secure Encryption
ToU	Terms of Use
TPM	Trusted Platform Module
VABKS	Verifiable Attribute-Based Keyword Search
VC	Verifiable Computation
VCKS	Verifiable Conjunctive Keyword Search
VKS	Verifiable Keyword Search
XACML	eXtensible Access Control Markup Language

List of Figures

List of Tables

Contents

Introduction

Cloud computing has recently emerged as a successful alternative computing paradigm, involving a large number of machines connected to a network delivering computing power, storage and Information Technology (IT) services over the Internet. The increasing use of cloud computing introduced **new security risks**, exposing cloud users to see the data they store and the applications they host in the cloud compromised. To alleviate this threat and increase trust into the cloud, the cloud service providers should implement mechanisms to ensure that the data and applications are correctly managed in their infrastructures. The discovery of security breaches in the cloud may result in heavy economical losses and poor reputation, especially if the provided services target the general audience. Consequently, security concerns act as strong impediments to the wide adoption of cloud technologies.

1 Topic of Research

1.1 Observations

Before characterizing our topic of research, we present some actual facts to illustrate our motivation in conducting research on some peculiar aspects of cloud security: **verifiability** and **accountability**. These observations will reflect the existence of these two concerns.

1.1.1 Lack of Trust in Cloud Computing: the Sidekick Data Loss Incident

The context. In the first decade of the new century, at a time when smartphones were not yet as popular as they are nowadays, three former employees of Apple[1] founded Danger, a Consumer Electronics (CE) company that designed and sold what is known as the first successful smartphone in the United States: the Sidekick smartphone. Sidekick phones were launched under T-Mobile carrier and their competitive advantage, compared to other phones available in the market at that time, relied on the pioneering provision of mobile Internet services and especially of an application marketplace (later popularized by the Apple's App Store and Google's Android Market). Sidekick caught the attention of certain numbers of celebrities such as Paris Hilton[2] making the phones rapidly popular among the general audience, in particular for teenagers and young adults[3]. Its popularity can also be assessed through the important number of thefts[4] and hackings[5] the devices were subject to. Sidekick phones also pioneered by their use of cloud computing services: all personal data, such as contacts, text messages, calendar entries and pictures, were not locally stored (not even

[1]An interesting historical note: One of the co-founder of Danger, Andy Rubin, left Danger in 2003 and created Android, later acquired by Google. Ben Elgin, "Google Buys Android for Its Mobile Arsenal", Bloomberg Business, August 16, 2005, http://tiny.cc/j6fr8x [Accessed: February 1, 2016].

[2]Tricia Duryee, "T-Mobile Uses Celebrity Buzz to Market its Sidekick", The Seattle Times, December 12, 2005, http://tiny.cc/17fr8x [Accessed: February 1, 2016].

[3]Melissa Trujillo, "Sidekicks Popular Among Teenagers, Thieves", NBC News, http://tiny.cc/dhgr8x [Accessed: February 1, 2016].

[4]Andrew Nusca, "T-Mobile Sidekick Most Stolen Phone in America", ZDNet, March 2, 2009, http://tiny.cc/ujgr8x [Accessed: February 1, 2016].

[5]Steven Musil, "Paris Hilton's cell phone hacked?", CNET, March 18, 2005, http://tiny.cc/eqgr8x [Accessed: February 1, 2016].

backed-up) in the devices but stored in cloud servers operated by Danger. The company has finally been acquired in 2008 by Microsoft[6].

The facts. At the beginning of October 2009, many users started to face data service outage: they were not able to access their pictures, contacts, calendar entries, or e-mails. This outage was caused by a server crash at Danger. The earliest complaints about this event were published on Twitter from October 2, 2009. Among dissatisfied users, Perez Hilton, a famous on-line American gossip blogger, launched the hashtag "#TMobileSucks" which became a number one trending topic on Twitter[7]. Microsoft (who bought Danger) and T-Mobile kept silent for a couple of days concerning the amount of damage this failure caused. But on October 10, 2009, Microsoft and T-Mobile said that the data may be permanently lost[8]. Fortunately, on October 12, 2009, T-Mobile and Microsoft finally declared that they would be able to recover Sidekick users' data[9]. Finally on October 15, 2009, Microsoft published a small report[10] announcing that it has recovered most of the Sidekick users' data. Besides, the report revealed that "the outage was caused by a system failure that created data loss in the core database and the back-up". Other sources[11] relate a hardware problem on servers run by Danger.

The impacts. The Sidekick data loss incident was known to be the "biggest disaster of cloud computing"[12]: it impacted an estimated number of 800,000 users. For a long period of nearly two weeks (from October 2, to October 15, 2009), they were unable to access their own data, ranging from calendar entries to pictures, including game high scores and notes.

A straightforward commercial repercussion arose in Sidekick phones sales. The American carrier T-Mobile temporarily interrupted sales of the smartphone[13], from October 12, 2009.

Another consequence results in several lawsuits being filled against both Microsoft and T-Mobile. These lawsuits alleged the negligence of the two international companies of guaranteeing an appropriate level of back-up. "T-Mobile and its service providers ought to have been more careful with the use of backup technology and policies to prevent such data loss.", one complaint says[14]. Another one[15] sums up the situation as following: "T-Mobile and Microsoft promised to safeguard the most important data their customers possess and then apparently failed to follow even the most basic data protection principles. What they did is

[6]Robert J. Bach, "Microsoft Agrees To Acquire Danger Inc.", Microsoft News Center, February 11, 2008, http://tiny.cc/hsgr8x [Accessed: February 1, 2016]. Rumors affirmed this acquisition was part of Microsoft's strategy to compete with the new successful Apple's smartphone: the iPhone.

[7]Cristina Lepore, "The T-Mobile Sidekick Data Outage: A Lesson in Social Media Crisis Management", 451 Heat, October 11, 2009, http://tiny.cc/4vgr8x [Accessed: February 1, 2016].

[8]Excerpt of T-Mobile Press Release on Sidekick Data Loss: "Regrettably, based on Microsoft/Danger's latest recovery assessment of their systems, we must now inform you that personal information stored on your device - such as contacts, calendar entries, to-do lists or photos - that is no longer on your Sidekick almost certainly has been lost as a result of a server failure at Microsoft/Danger". Pete Cashmore, "T-Mobile: All Your Sidekick Data Has Been Lost Forever", Mashable, October 10, 2009, http://tiny.cc/4ygr8x [Accessed: February 1, 2016].

[9]Saul Hansell, "T-Mobile Says Sidekick Owners May Recover Lost Data" The New York Times, October 12, 2009, http://tiny.cc/l1gr8x [Accessed: February 1, 2016].

[10]Roz Ho, "Microsoft Confirms Data Recovery for Sidekick Users", Microsoft News Center, October 15, 2009, http://tiny.cc/93gr8x [Accessed: February 1, 2016].

[11]Ina Fried, "Sidekick Users Share Their Horror Stories", CNET, October 12, 2009, http://tiny.cc/lehr8x [Accessed: February 1, 2016].

[12]Rory Cellan-Jones, "The Sidekick Cloud Disaster", BBC News, October 13, 2009, http://tiny.cc/xhhr8x [Accessed: February 1, 2016].

[13]Maggie Shiels, "Phone sales hit by Sidekick loss", BBC News, October 13, 2009, http://tiny.cc/xjhr8x [Accessed: February 1, 2016].

[14]Daniel Eran Dilger, "Sun, Oracle save Microsoft's Pink After Danger Data Disaster", AppleInsider, October 21, 2009, http://tiny.cc/nlhr8x [Accessed: February 1, 2016].

[15]Ina Fried, "Lawsuits Filed Over Sidekick Outages", CNET, October 14, 2009, http://tiny.cc/8nhr8x [Accessed: February 1, 2016].

unthinkable in this day and age."

Resulting from the data loss and the consecutive lawsuits, the reputations of T-Mobile and Microsoft have been harmed. In the case of T-Mobile, which was facing at that time, a substantial churn, this disaster did not help the carrier increase the number of its subscribers[16]. As far as Microsoft is concerned, 2009 was a key year for the company: the Sidekick disaster occurred just one month before Microsoft was believed to promote its cloud computing services, Azure[17].

More than the damage caused on Microsoft's reputation as a cloud service provider, the public perception of cloud computing in general suffered in this anecdote. As a matter of fact, loss of confidence in cloud computing came to light after the Sidekick data outage. This episode clearly supported the idea that cloud services cannot be trusted. Customers of other cloud services were likely to question the providers about their storage, back-up and recovery practices.

Analysis. The Sidekick smartphone users had no technical means to back-up their personal data. The devices were designed such that the data was not stored locally but directly in the cloud. This means that the users were putting their entire trust on Danger (Microsoft) competence in storing their data, making the supposition that the data would be correctly stored in the cloud, intact and available at any time. Even worse, users (legitimately) assumed that the cloud servers were redundantly storing their data to prevent such a data loss disaster. Hence, the Sidekick users were not backing-up their data locally. In spite of the awareness of the data loss threat and the evolution of cloud technologies[18], the concern remains the same: personal or sensitive data cannot be assumed to be trustfully stored in the cloud.

Along with the lack of trust concern, the Sidekick anecdote reveals another cloud computing issue: delegating all control over stored data to the cloud is problematic especially in the case of service outage. As a matter of fact, these two concerns also apply if the service failure was due to malicious acts.

The perspective of our work in this thesis is to bring some control on the way an untrusted cloud stores outsourced data. In particular, we believe that data owners may be willing to **check** that their data is, at any point of time, correctly stored and available and, under certain adversarial circumstances, **retrievable**. For such a verification, the data owners should be provided with **proofs** that their data is intact and recoverable.

1.1.2 Verifying Untrusted Computers: the SETI@home Project

The context. Since the dawn of time, humanity has contemplated the stars and wondered "Are we alone in the Universe?". Owing to the difficulty of space traveling and planet visiting, the chance to come face to face with an alien is extremely low. Nevertheless, the advent of radio communications gives hope that this question will not remain unanswered. Indeed, humanity has been broadcasting radio signals into space to find other civilizations. Humans also believe that other beings might be doing the same from other planets, from other galaxies. The Search for ExtraTerrestrial Intelligence (SETI) project is an American program started in the sixties whose goal is to detect signals from outer space that would witness the existence of forms of extraterrestrial intelligence. The project analyzes the electromagnetic radiations from space and tries to identify signals that are not the random noise due to gravitational forces. Researchers are collecting a huge amount of data from radio

[16]Dan Butcher, "T-Mobile May Recover Some Data but Hit To Reputation Stings", Mobile Marketer, October 14, 2009, http://tiny.cc/sshr8x [Accessed: February 1, 2016].

[17]Daniel Eran Dilger, "Microsoft's Danger Sidekick Data Loss Casts Dark On Cloud Computing", October 11, 2009, http://tiny.cc/cvhr8x [Accessed: February 1, 2016].

[18]Unlike Sidekick, Apple's iPhone syncs the data in users' computers using the software called iTunes. Then iTunes syncs the data with a cloud-based storage service. Therefore, if the cloud service was not available, the users are still able to access their data locally using iTunes.

telescopes around the world. However, processing this data is an expensive and resource intensive task which the researchers could not afford themselves. Hence, in 1999, researchers at the University of California, Berkeley, devised the SETI@home project: instead of analyzing the data themselves, the analyses are distributed among a large number of volunteers using the Internet. The SETI@home project is the first successful and popular implementation of distributed computing. The volunteers download the Berkeley Open Infrastructure for Network Computing (BOINC) client application which runs when their computers are not used. Thus, their computational resources are made available to the SETI project. All BOINC clients interface with a central BOINC server which divides the signal data and its respective computation (for analysis) into small units called *work units*. The server sends these work units to computers distributed over the Internet, which analyze their respective data and send back the results to the server. The computations performed by the BOINC clients mainly consist in statistical signal processing which involves operations such as the discrete Fourier transform. Participants to the SETI@home project are volunteers; Berkeley does not offer any material or financial reward to those who contribute most to the radio signal processing.

The facts. Since the beginning of the SETI@home project in 1999, BOINC publishes rankings showing which users and which machines processed the biggest amount of work units[19]. These rankings act as incentives for joining the project, downloading and running the BOINC client software. Indeed, as no material or financial compensation are offered, reputation and success provided by these rankings constitute the only reward which motivates volunteers to process signal data. However, the downside of rankings is that it creates competition between users, and thus some of them might cheat to get a better rank. Molnar [128] reported an episode in which "cheating" with the BOINC software compromised computation integrity, yielding incorrect results. A German volunteer, who called himself "Olli" created and published an unauthorized patch to improve and speed up the computations performed by the original BOINC client application. SETI@home researchers revealed that this patch actually return wrong results, substantially different from what the original BOINC software would have output. Besides, there was no way for the BOINC server to tell whether results are returned by a patched client or by a normal one. Camp and Johnson [50] also report users tampering with result data to generate false positives.

The impacts. This "Olli' anecdote raised awareness about the computation integrity concern in distributed computing, as shown in the SETI@home's Frequently Asked Questions (FAQ)[20]. This concern was referred as the "SETI@home problem" by Molnar [128] who summarized it by the following question: "how do you ensure that the client machines are doing the right thing?". Indeed, the results of a delegated computation to a volunteer computer cannot be trusted because (i) unintentional hardware problem at client hosts can cause errors in the computations; (ii) volunteers might intentionally return bogus results by sabotage of the application or simply by not performing the requested computation. The SETI@home project was then required to review their security practices in order to filter out incorrect results. One envisioned approach to prevent such a misbehavior was to remove the incentive that motivates users to run patched BOINC client software (such as the one released by "Olli"). Namely, SETI@home could have stopped ranking users to shrink competition and thus thwart cheating behaviors. However, this solution would also have removed the motivation to participate in the project. Hence, the BOINC platform copes with this computation integrity concern via replication. As the project involves a large number of volunteers[21], the same work unit (same computation on the same data) is randomly assigned to several

[19]BOINC rankings: http://boincstats.com/en/stats/0/user/list/ [Accessed: February 1, 2016].

[20]SETI@home FAQ: http://www.faqs.org/faqs/seti/at-home/questions/ [Accessed: February 1, 2016].

[21]The "BOINCstats" web page reports 1,598,622 volunteers: http://boincstats.com/en/stats/0/user/list/ [Accessed: February 1, 2016].

clients. Assuming that clients running unauthorized patched BOINC applications form a minority and that clients cannot collude[22], only the results that appear in a majority will be considered as valid. Molnar [128] also extrapolates another possible approach from some research conducted around the years 2000s about **proofs of correct computation**, that consider an adversary that would only disrupt the computation and do not mind about the reputation resulting from the BOINC rankings[23]. Applied to the SETI@home project, the volunteers would be required to generate a proof of correctness for each result they output and the BOINC server would verify the validity of this proof.

Analysis. Related to the SETI@home problem, Gennaro *et al.* [90] wrote: "A related fear plagues cloud computing". Indeed, when a user outsources some resource intensive computation to the cloud, the concern of computation integrity and result correctness arise. This user cannot trust the cloud to correctly perform the outsourced computation, as the BOINC server cannot trust the BOINC clients. While the incentives for BOINC clients to misbehave were related to the inherent competition in the SETI@home project, a malicious cloud may return bogus computation results to save computational resources. As a consequence of the "Olli" patch problem, the BOINC platform devised a measure to control the computations by the untrusted BOINC clients and detect fraudulent results. Similarly, cloud users should be given control on the way the cloud operates the outsourced computation. Namely, they should be convinced that the returned results are correct by **verifying the computation**. Namely, as extrapolated by Molnar [128] for the SETI@home project, the cloud server should generate **proofs of correctness**. Besides, the verification of these proofs must demand less computational resources than the computation itself. Otherwise, delegating the computation offers no gain to the user. The vision of our work consists in giving to cloud users such a control over the computation they outsource to a cloud server.

1.1.3 Security, Privacy and Accountability: the iCloud Data Breach

The context. The breakthrough of cloud computing technologies opened the way to a plethora of new services. Among others, iCloud, provided by Apple since 2011, hit the headlines during summer 2014, because of a media-hyped data breach. iCloud[24] is the leading cloud computing service offered by Apple Inc. It provides cloud storage and serves as an automatic back-up system for iOS[25] devices. Users can upload their documents, contact lists, e-mails, photos and other multimedia items to their iCloud account. They can easily access their data from any device and share them to others.

The facts. On August 31, 2014, a substantial collection of private pictures, portraying at least 100 female celebrities, was hacked and leaked on the Internet. Some of these stolen pictures contain nudity. They were initially published on the anonymous image-based forum, 4chan[26], and later advertised on popular social networks such as Reddit[27] and Tumblr[28]. All these images (even images that were previously deleted by the subjects) were stolen via a breach in iCloud. Many of the affected celebrities confirmed the genuineness of the pictures. Other data such as calendar entries, address books or text messages were also illegally ob-

[22]Molnar [128] assumes BOINC clients cannot collude but since they are connected to the Internet, this assumption seems too strong.

[23]As opposed to patched clients that only want to speed up their computations to increase the number of processed work units and thus to top the BOINC rankings.

[24]iCloud: https://www.icloud.com/ [Accessed: February 1, 2016].

[25]iOS is the Operating System of Apple devices.

[26]4chan: http://www.4chan.org/ [Accessed: February 1, 2016].

[27]Reddit: https://www.reddit.com/ [Accessed: February 1, 2016].

[28]Tumblr: https://www.tumblr.com/ [Accessed: February 1, 2016].

tained. Apple confirmed[29] within two days that the data leak resulted from targeted attacks on celebrities' iCloud accounts. As a matter of fact, the hackers obtained the pictures by the brute force guessing[30] of usernames, passwords and security questions.

The impacts. The illicit distribution of the hacked pictures was perceived as a major **privacy** breach for the affected celebrities. Some of the subjects were underage when the nudity pictures were taken, leading to prosecution for child pornography distribution for all web users that republished them.

Beyond the legal aspects of the leak, finding **responsibilities** and **evidence** of the data breach was one of the main priorities. The concern was to identify which party is responsible for the attack: did the iCloud service present vulnerability or were the accounts hijacked by brute-force attack? Investigations by Apple revealed that the problem stemmed from these two aspects. It appears that hackers were able to conduct an unbounded number of brute force iCloud credential guesses because iCloud was not locking the number of signing-in attempts. Besides, iCloud legitimate users did not receive **notifications** that their accounts were tried to be accessed. In a sense, Apple rejected responsibility for the hack and blamed iCloud users for choosing weak or multiple-time passwords as credentials.

As a result, while denying **accountability** for this incident[31], Apple took some remediation measures[32] to prevent similar attacks in the future: it added security alerts for the users when someone is trying to access and change their accounts; it increased users' awareness about the danger of weak passwords and it introduced a two-factor authentication to access the iCloud account.

This incident undermined iCloud's and Apple's reputation and induced bad publicity for Apple just before the release of their new operating system iOS8[32]. As a result, Apple published an accountability report[33] on their security and privacy practices. Beyond iCloud, cloud computing services in general suffered from the iCloud data breach. It aroused concerns on how these services guarantee privacy and security and the general audience's trust in the cloud.

Analysis. The iCloud data breach is an example of how important the concept of **accountability** is and how critical trust in cloud computing becomes. In its accountability report[33], Apple states that "strict **policies** govern how all data is handled". Nevertheless, we identify at least three issues that should have been stated in such policies in order to ensure a better level of privacy and accountability: (i) Deleted pictures were still recoverable. This may consist in a policy violation, or even a legal violation since many data protection regulations impose restrictive rules on data erasure[34]; (ii) No notifications or alerts were sent to legitimate users when an unauthorized user accessed the iCloud accounts or when data was retrieved from unknown devices; (iii) Collecting evidence[35] about the data breach was a tough issue in order to determine the responsibilities of the actors involved in the iCloud

[29]Apple Media Advisory, "Update to Celebrity Photo Investigation", Apple Press Info, September 2, 2014, http://tiny.cc/o8ir8x [Accessed: February 1, 2016].

[30]A script used to guess iCloud login credentials was on Github. "Brute Force Attack Burns Celebrities", LMG Security Blog, September 11, 2014, http://tiny.cc/udjr8x [Accessed: February 1, 2016].

[31]Nicole Arce, "iCloud? Find My iPhone? What Should Be Blamed For Nude Celebrity Photo Leaks? Neither, says Apple". Tech Times, September 4, 2014, http://tiny.cc/trjr8x [Accessed: February 1, 2016].

[32] Daisuke Wakabayashi, "Tim Cook Says Apple to Add Security Alerts for iCloud Users", The Wall Street Journal, September 5, 2014, http://tiny.cc/fvjr8x [Accessed: February 1, 2016].

[33] Tim Cook, "Apple's Commitment To Your Privacy", Apple, http://www.apple.com/privacy/ [Accessed: February 1, 2016].

[34]Regulation of the European Parliament and of the Council on the protection of individuals with regard to the processing of personal data and on the free movement of such data: http://tiny.cc/12jr8x [Accessed: February 1, 2016].

[35]Warwick Ashford, "Apple and FBI Launch iCloud Hack Investigation", Computer Weekly, September 2, 2014, http://tiny.cc/25jr8x [Accessed: February 1, 2016].

incident (Apple cloud service, users, malicious parties). Hence, we can appreciate the important role played by **appropriate policies** in order to mitigate risks and increase trust in cloud-based services, in which users give up the control over their data. Establishing policies and ensuring their enforcement enable those users to keep data under certain control.

1.2 Framing the Topic

In the observations we made in the previous section, we mentioned some key notions that we define in the present section in more detail. The expression of these definitions contribute to the delineation of the topics of this dissertation.

1.2.1 Cloud Computing: an emerging computing paradigm

The term "cloud computing" (or just "cloud") is perhaps one of the most popular buzzword in IT in this early stage of the 21st century. No other technologies has raised the same amount of enthusiasm and passion as cloud computing. Envisioned since the 1950s, the idea of providing shared access to a single resource (in the model of client/server) has constantly evolved. The advent of virtualization and virtual machines, the development of web services and the growing number of individuals and companies able to connect to the Internet made the ideal conjunction for cloud computing to exist. In 2006, Amazon unveiled Amazon Web Services[36], maybe the first cloud-based service offering storage and computation known to the general audience. 2009 marked a shift in cloud computing perception: two other multinational companies, namely Google and Microsoft, entered the field. Google launched its App Engine, a cloud-based platform for developing and hosting web applications in Google's servers[37], while Microsoft announced Azure, a cloud-based platform for developing applications that are hosted inside Microsoft's data centers[38]. Naturally, other big players in the IT market took part in cloud computing evolution (Oracle, HP, Apple, etc.) It was not until September 2011 that the National Institute of Standards and Technology (NIST) released a definition of cloud computing embraced by many research papers and IT articles:

Definition 1 (Cloud Computing). *"Cloud computing is a model for enabling ubiquitous, convenient, on-demand network access to a shared pool of configurable computing resources (e.g., networks, servers, storage, applications, and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction". [124]*

In other terms, the term "cloud computing" represents a computing paradigm with the following four characteristics:

On-demand service: Cloud clients can obtain cloud services whenever they want, and as long as they want, in an automated way, without requiring any human assistance.

Accessibility: Cloud services are delivered using Internet and through standardized mechanisms. The clients are enabled to access such services through a variety of devices (such as mobile phones, tablets or laptops) and from anywhere on the globe.

Pooled resources and multi-tenancy: Unlike the traditional IT model that provides services for one client, cloud-based services share pooled resources that are delivered to

[36]About Amazon Web Services: http://aws.amazon.com/about-aws/.

[37]Brady Forrest, "App Engine: Host Your Apps with Google", O'Reilly Radar, April 7, 2008, http://tiny.cc/4wkr8x [Accessed: February 1, 2016].

[38]Ray Ozzie, "Microsoft Unveils Windows Azure at Professional Developers Conference", Microsoft News Center, October 27, 2008, http://tiny.cc/q4kr8x [Accessed: February 1, 2016].

multiple clients in a multi-tenant model. Namely, a single instance of resources serves multiple clients. Note that resource mutualization implies location independence: the resources are distributed among different locations in order to maintain cloud performances (availability of resources, high Quality of Service (QoS), etc.).

Elasticity: Cloud computing virtually supplies computing, storage and network resources that the clients can easily access and use. This illusion is due to the capacity of the cloud to dynamically scale up or down the provision of a service in function of the demand. Because of resource pooling and multi-tenancy, the elasticity of cloud computing implies that a resource not used by one client can be assigned to another one who needs more resources.

Adopting cloud computing presents many benefits for both the service consumer and the service provider. Indeed, the cloud client, be it a company or an individual, can leverage cloud computing technologies for:

Reduced IT costs: Instead of spending a possibly large amount of money on IT infrastructures for storage and computation, an organization can rent resources from cloud providers. Reduced costs also concern IT maintenance or support, including salaries or energy expenditures.

Ease of access and usage: Cloud-based services empower their users to access the resources from any geographical location and simultaneously by different users, with various devices. Besides, the resources can be used in an automated and intuitive way, without the users being required to be trained to use cloud services.

High QoS: Cloud services supply their users with highly responsive applications with low latency on demand. This is so because cloud computing optimizes the use of pooled resources via the elasticity property.

Flexibility: This benefit also results from cloud computing's elasticity characteristic. For example, a company can face several seasonal changes in demand of resources (such as adding users into a cloud-based system for a short period of time). Cloud computing can handle such variations easily by scaling up and then down again the provision of the demanded resources. Besides, cloud providers often adopt the pay-as-you-go model: clients are charged only for the resources they consume. Hence, no resource is left idle.

From the cloud provider's perspective, benefits include:

Optimized utilization of resource: While in traditional computing systems, IT assets, such as hardware, are not used at their maximum capacity, cloud computing empowers cloud providers to effectively allocate their idle resources to a broad range of clients.

Increased profit: With an increased number of cloud users, cloud providers enter a new growing market that generates high revenue. Besides, the optimized use of their resources help make new high incomes.

Alongside cloud computing, the 21st century is facing the advent of "**big data**". This other buzzword not only describes data that is "big" in size (namely, tera, peta or exabytes of data) but also considers data that is constantly produced (such as streaming data or time-series data) and that presents a heterogeneity of formats and structures. Hence, the big data paradigm deals not only with storing these huge amount of data but also with processing, analyzing, visualizing and extracting relevant information and knowledge from different kinds of data. Organizations generating or collecting big data are confronted to the following concern: how to efficiently process this data? A possible solution to this concern arises from cloud computing that provides an optimal infrastructure to deal with big data.

The elasticity property of the cloud makes it ideal to handle big data processing. Together cloud computing and big data empower organizations to operate cost effective and scalable data analytics, from which they may gain valuable business information.

1.2.2 Cloud Computing: security, privacy and compliance challenges

In spite of the benefits we listed previously, cloud computing is prone to various security and privacy concerns (as shown in Section 1.1), which present barriers to the widespread adoption of cloud computing. Besides, for cloud service providers to be competitive in the cloud computing market place, these concerns must be addressed. Most of the impediments to the success of cloud computing stem from a combination of two interconnected issues, namely loss of control and lack of trust:

- Cloud users relinquish the control over their data to the cloud service providers. Indeed, by outsourcing their assets (data and computation) to the cloud, users do not physically possess them anymore and rely on the cloud providers to implement adequate control over a wide range of aspects, including: storage, access, privacy, confidentiality, integrity, availability and usage. This loss of control is typically critical for organizations which must adhere to regulatory obligations such as the European Data Protection directive [80], the Health Insurance Portability and Accountability Act (HIPAA) [1] or the EuroSOX[39] directive on transparency and accountability of financial control [79]. Hence, the loss of control over data by companies induces compliance challenges with respect to these regulations.

- Deterrence in cloud computing adoption can also be attributed to a lack of trust in cloud service providers. This lack of trust applies according to two aspects. First, unintentional failure in the cloud services may cause data loss or service disruption which can be critical for organizations that rely on these services. Secondly, cloud providers themselves are considered as untrusted and malicious. They can adopt two adversarial behaviors. A *curious* cloud compromises data and computation privacy and confidentiality, by, for instance, performing data mining operations to acquire valuable knowledge and information from the data they manage. This is typically the case with the huge amount of sensitive and personal data generated and collected by social networks such as Twitter, LinkedIn or Facebook. A *malicious* cloud intentionally misuses its resources in such a way that it compromises confidentiality, integrity or availability of the data and computations they handle. Furthermore, malicious third parties are another threat faced by cloud services. Alongside lack of trust, transparency on how, why and where the data and computations are processed is a fundamental concern in cloud-based services. Providers may be tempted to hide any system failure or data breach that might comprise their reputation.

To summarize, transferring control to an untrusted cloud poses several security and privacy risks, which are correlated with transparency, responsibility and compliance concerns. The threats associated with these risks were reported and analyzed by the Cloud Security Alliance (CSA) in their "Notorious Nine" report [174] that lists the nine cloud computing top threats. Here are some real-word examples of some of these threats:

Data breach: This threat is reported as the Top 1 in cloud computing. It refers to the theft and leakage of sensitive data falling into the hands of unauthorized parties. Over the past decade, not only have more and more organizations suffered from data breach incidents, but the amount of stolen data also increased. In addition, this stolen data is

[39]EuroSOX is the European version of the American Sarbanes-Oxley (SOX) Act stating that a company is responsible for any accounting or financial misbehavior even if financial data are processed by third-parties such as cloud providers.

more and more sensitive[40]. The term *sensitive* encompasses personal, business, medical and governmental/military data. In the following, we list some examples of data breach incidents. They do not all involve cloud technologies, but similar incidents are believed to occur in the cloud as well.

Personal data: The iCloud data breach we described in Section 1.1.3 is an example of personal data breach. Hackers stole and published celebrities' personal pictures.

Business data: In 2013, a major discount retailer, Target Corporation, unveiled that a serious data breach affected more than 70 million customers, during the Christmas holiday season. The stolen data included credit card information[41].

Medical data: In 2015, an important data breach at Premera Blue Cross, an American healthcare insurer[42], was discovered. This data breach affected more than 11 million people. Data includes date of birth, social security number and clinical information.

Governmental data: In 2015, the American government's personnel management agency (OPM) announced that hackers stole 21.5 million people's personal data from their IT system[43]. The stolen data includes information collected from security clearance applications, such as birth dates, addresses, current and former federal employees, information about relatives, eye colors, financial history, past substance abuse, social security numbers, etc.

Data loss: Cited as the Top 2 threat by the CSA [174], data loss refers to the unrecoverable destruction of data as was the case in the Amazon's cloud crash disaster in April 2011[44]. Data loss also corresponds to the temporary unavailability of the data as faced by the Sidekick users in the example we mentioned in Section 1.1.1. These two previous examples show data losses resulting from unintentional incidents. In addition to these, malicious behaviors can also compromise data storage, data integrity and data availability by, for instance, deleting or tampering with the data, unlinking the data from a larger context [174] or destroying encryption keys if the data is encrypted.

The above examples of cloud computing threats in real-life scenarios justify the recent research efforts in the topic of security and privacy in the cloud. Besides, with the increasing demand of cloud services and the proliferation of mobile devices, the cloud security market has been enjoying an important and rapid growth in the past few years. A report from Transparency Market Research estimated that the cloud security market reached $4.5 billion in 2014 and that it could be worth $11 billion by 2022[45].

However, the peculiar characteristics of cloud computing negate the straightforward utilization of *traditional* IT security mechanisms such as encryption to ensure confidentiality of data, cryptographic-based techniques to guarantee data or computation integrity, isolation to secure computations, etc. This is due to several challenges specific to the cloud:

1. The proliferation of data, the so-called "big data" movement, and the increasing number of mobile devices incite individuals and organizations to use cloud services to outsource and access at anytime and from anywhere their data and applications.

[40] A nice visualization of recent data breach incidents (since 2004) can be found at: http://tiny.cc/pwmr8x [Accessed: February 1, 2016].

[41] John Biggs, "Target Confirms Point-Of-Sale Data Breach, Announces It Exposed 40 Million Credit Card Numbers", TechCrunch, December 19, 2013, http://tiny.cc/7xmr8x [Accessed: February 1, 2016].

[42] Premera Blue Cross, "About the Cyberattack", http://tiny.cc/szmr8x [Accessed: February 1, 2016].

[43] Patricia Zengerle and Megan Cassella, "Millions More Americans Hit By government Personnel Data Hack", Reuters, July 9, 2015, http://tiny.cc/01mr8x [Accessed: February 1, 2016].

[44] Henry Blodget, "Amazon's Cloud Crash Disaster Permanently Destroyed Many Customers' Data", Business Insider, April 28, 2011, http://tiny.cc/d4mr8x [Accessed: February 1, 2016].

[45] James Martin, "Recent Research Into The Cloud Security Market That Is Expected To Reach US$ 11.84 Billion and CAGR of 12.8% by 2022 Just Published", WhaTech, August 18, 2015, http://tiny.cc/1dnr8x [Accessed: February 1, 2016].

2. This implies that the cloud users lose control over their data and lend it to the cloud. Besides, since the users do not have the physical possession of their data anymore, making it safe via traditional IT security mechanisms is impractical. Therefore they have to rely on cloud providers to implement the appropriate security measures.

3. Regulatory and contractual compliance makes challenging the application of security measures in the cloud, especially those laws and contracts that rule storage and use of data. They entail that verifications of the way the cloud stores and uses the outsourced assets must be performed to show that the cloud is compliant. In a wider perspective, compliance implies accountability with respect to a cloud's behavior.

In addition, solutions addressing cloud security concerns and challenges must be designed in a such a way that outsourcing to the cloud remains an attractive solution for cloud users. Security techniques must be efficient and must, at the same time, ensure ease of use and access of outsourced data.

1.2.3 Focus of the present work

This thesis addresses the problem of **verifiability** in the cloud. The goal is to devise mechanisms used by cloud users to *control* and *verify* that the cloud *correctly* delivers the promised services of storage and computation. It is legitimate to believe that users who consume and pay for these services expect that their data are correctly stored or that their computations are correctly performed. In other words, the terms of the "contract" (be it an explicit agreement such as Service Level Agreements (SLAs), terms of use, policies, or a tacit agreement) that binds the cloud and its users should be verifiable [42] and failures to meet this contract or violations to the terms should be detectable. Intuitively, we say that a property is verifiable if given an instance of this property, an agent can test whether this instance actually satisfies the property. The verifiability concept supports transparency: verifying cloud's behavior enables to know about the controls the cloud put in place to handle outsourced data and outsourced computation.

This thesis particularly focuses on two aspects of verifiability in the context of cloud computing, namely storage and computation:

Storage: We mentioned in Section 1.2.2 the data loss threat: a cloud user outsources to the cloud her possible large amount of data and expects the cloud to correctly store the data. Therefore, the user wants to verify the integrity of the data. Namely, she must be convinced that the data is not deleted, nor tampered with. In order to verify such a property, the cloud is required to generate some **proofs of storage**, that enable the user to check that the data is correctly stored.

Computation: Data is not the only asset that can be outsourced to the cloud. Many scenarios entail computation outsourcing: several kinds of operations may require a substantial amount of computational resources that may not be affordable for individuals or organizations. Hence, these users decide to migrate these operations to the cloud. In this case, they should be convinced that the cloud will always return correct results. In order to verify such a statement, the cloud is summoned to generate some **proofs of correct computation**, that empower the users to check that the returned results actually correspond to a correct execution of the outsourced operation. In this work, we focus on three types of computation: large degree polynomial evaluation, large matrix multiplication and keyword search in a large database

Besides verifiability, we study the broader concept of **accountability** for cloud computing, in relation with the transparency, responsibility and compliance challenges we identify in Section 1.2.2. We are particularly interested in policies for accountability. Such policies may convey the obligations that the cloud should comply with during the entire relationship between the cloud and its users. There currently exists no accountability framework that enables these users to understand how cloud providers honor accountability obligations, namely the rules making clear what the cloud is expected to do with the outsourced data. We believe that accountability policies provide means to express those obligations in terms of how the cloud should handle users' assets. Therefore, we look at the possibility to design a machine-readable policy language to convey accountability concepts.

2 Problem Statement

This thesis attempts to address a list of concerns that we formalize below.

2.1 Questions raised by this thesis

Associated with the security challenges identified in Section 1.2.3, a user who wants to adopt cloud technologies could raise the following questions:

- Is the service correctly provided by the cloud?
- Is it doing what it is supposed to do?
- How can I verify the execution of a service without wasting the cloud's benefits?
- Does the service comply with appropriate policies related to accountability and security?

Expressed in a more formal way, we characterize three problems that the present thesis considers as key issues in cloud computing security.

Problem 1: Verifiable Storage. This problem concerns the correct storage of cloud users' data. As users do not possess their data anymore, they relinquish the control over their storage to an untrusted cloud. *How can we give back cloud users some control over the data they outsource to the cloud?* In particular, the users must be convinced that the cloud complies with the promise to provide a storage service that does not compromise the integrity and the availability of users' outsourced data. In this perspective, the users must be empowered with the capability to verify the storage service offered by the cloud. Ideally, the cloud should produce a proof of storage stating that it actually stores an intact version of users' data. Besides, the verification of the proof by the users must not require computationally demanding operations since the users might not have sufficient resources to perform heavy verifications. Therefore, another question must be addressed: *Can we design a system where the cloud generates proofs of storage that enable to verify in an efficient way that users' data are correctly stored by the cloud?*

Problem 2: Verifiable Computation. Along with data outsourcing, cloud computing allows to delegate the computation of expensive operations to powerful servers that perform the computation on behalf of cloud users. In this thesis, we focus on primitives that are very common in today's world: polynomial evaluation, matrix multiplication and conjunctive keyword search. Similarly to the case of data storage, in the computation outsourcing scenario, users do not have control over the delegated computation and lend this control to untrusted servers. *How can we give back cloud users some control over the computation they outsource to the cloud?* More specifically, we regard computation integrity as an important issue. When the cloud returns the results of one of the above operations, the users must be convinced that these results are the ones they expect. Namely, they must be convinced that the returned results are equal to the one they would have obtained if they had performed the computation themselves. In other terms, the users must be able to verify that the cloud returned the correct outcome of the delegated computation. In this perspective, the cloud should produce a proof of computation ensuring the users that the returned results correspond to a correct evaluation of the outsourced operation. Furthermore, the verification of this proof by the users must be substantially less computationally demanding than the operation itself; otherwise outsourcing the computation would not be profitable for the users. Hence, this thesis considers the following problem: *Can we design a system where the cloud generates proofs of computation that enable to verify in an efficient way that users' computation is correctly performed by the cloud?*

Problem 3: Accountability. We identify in Section 1.2.3 that accountability is a peculiar challenge in cloud computing. Cloud providers must be compliant to regulation and contracts and be held accountable for the way they manage and operate cloud users' assets. Stated differently, a collection of accountability obligations binds cloud users and providers together so that the cloud operates in a transparent way. Machine-readable policies are a way to express these accountability obligations such that their enforcement can be handled easily in an automated way. This thesis intends to answer the following question: *How and to what extent can we convey accountability obligations via expressive and declarative policies in such a way that policies are easy to use, manage, enforce and validate and such that cloud provider can be held accountable for these obligations, thus increasing trust between users and providers?*

2.2 Contributions

This thesis answers the above problems and we propose the following contributions.

Proofs of Storage. Under an untrusted cloud, we design a protocol that generates some cryptographic proofs showing that the data outsourced at this cloud is correctly stored. The possible (and naïve) protocol whereby the data owner stores at the cloud the data along with a digital signature and, in order to verify the correct storage, downloads the data and checks the signature, would not scale in the context of cloud computing and big data, since it would incur large resource consumption. Hence the protocol for proofs of storage should be more efficient than this simple solution. Namely, the generation and the verification of these proofs of storage do not induce expensive costs in terms of computation, storage and bandwidth for the data owner (and/or any party that verifies the storage correctness). In addition, the protocol must be secure under a malicious cloud that would forge false proofs while not storing the data as expected. We propose our protocol for proofs of storage, $\mathcal{S}$tealthGuard, based on the idea of inserting in the outsourced data special blocks, called watchdogs.

Proofs of Computation. Under an untrusted cloud, we can delegate and verify the results of computationally demanding operations. The remote computer is required to send the result of such a computation along with a cryptographic proof that the operation was carried out as expected. We devise three protocols in which verifying the proof of correct computation is efficient, requiring less computational resources than computing the outsourced operation locally from scratch. These three protocols address the problem of verifiable computation for three types of operations: large degree polynomial evaluation, large matrix multiplication and conjunctive keyword search. These three solutions are based on simple mathematical techniques and well-established cryptographic primitives, rendering our protocol efficient compared to some prior work. One of the peculiar characteristics of our solutions is the fact that they allow two properties: public delegatability (anyone, not only the user who outsourced the computation, can request the server to perform the computation) and public verifiability (anyone, not only the user who requested the computation, can verify the results returned by the server). Besides, these protocols are secure against a malicious server that would return bogus results without performing the computation.

Accountability Policy Language. We design A-PPL, a policy language that enables the expression of accountability obligations which rule the conditions under which an accountable cloud must operate outsourced data. This policy language is machine readable to facilitate the automation of the enforcement of these obligations. We also devise the A-PPL engine, the system that enables the enforcement of the accountability policies written in our new language.

3 Organization

The present dissertation is organized into three parts:

Proofs of Retrievability: Part I investigates on storage verifiability. Chapter 1 provides the reader with the definition of a cryptographic proof of storage protocol and a new security model against malicious servers. We also review prior work in terms of proofs of storage solutions. Chapter 2 describes our protocol $\mathcal{S}$tealthGuard. We also prove the security properties of our proposal and implement a prototype showing the efficiency of $\mathcal{S}$tealthGuard.

Proofs of Correct Computation: Part II addresses Problem 2 on proofs of correct computation. In particular, Chapter 3 introduces the concept of verifiable computation and gives an exhaustive analysis of the literature in this domain. We then propose three efficient protocols for three different operations: polynomial evaluation (Chapter 4), matrix multiplication (Chapter 5) and conjunctive keyword search (Chapter 6). We study the security properties of our solutions, and demonstrate them. We finally build prototypes and analyze their efficiency.

An Accountability Policy Language: Part III introduces the concept of accountability in the context of cloud computing. We define the obligations related to an accountable cloud and derive from them some requirements that a machine-readable policy language for accountability should satisfy. We review existing policy languages and determine to which extent they meet the identified requirements. We then present A-PPL, a new policy language for accountability, together with the A-PPL engine. We finally illustrate with a scenario the expression and the enforcement of accountability policies.

Part I

Proofs of Storage

Chapter 1

Characterization of Proofs of Storage

1.1 Introduction to Proofs of Storage

In recent years, cloud computing became popular and received considerable attention since this paradigm allows users (industries, organizations, individuals) to outsource possibly large amounts of data to a remote cloud server who is then responsible for storing these data. While the users enjoy the advantage of offloading the storage burden to the cloud, some security issues inhibit some organizations or end users to shift from traditional storage systems where the data is locally stored (implying heavy costs in server maintenance) to cloud computing technology. Indeed, by storing data to the cloud, users lose the physical control over their data and relinquish their management to *untrusted* servers. In particular, as we showed in Section 1.2.2, outsourced data in the cloud may be the target of several threats identified by the CSA in their report on the “Notorious Nine” Top Threats of Cloud Computing [174]. The scope of this part of the thesis focuses on data losses, reported as the second biggest threat in cloud computing by the CSA [174]. The term *data loss* encompasses not only unauthorized deletion of data but also unrecoverable tampering of data. Data losses may result from (i) malicious attackers that intentionally erase or manipulate the data; and (ii) accidental deletion or changes due to system crashes, bogus software update or any unintentional system-based data corruption. Stated differently, the data loss threat may compromise the integrity and availability of outsourced data.

To be given back a certain form of control, the data owners need to be convinced that the cloud server is compliant with their storage expectations. In particular, data owners must be ensured that their data is intact and available all along the storage period. One of the challenges we mentioned in Section 1.2.2 concern big data. Since data owners cannot afford for the storage of large amounts of data, they use cloud storage services to store them. In such a scenario, data owners do not physically possess their assets anymore. Therefore, the traditional technique of integrity checking relying on a digital signature cannot be considered: Indeed, to check that large data is correctly stored, the owner downloads it, computes its signature, and compares it with the stored signature. Unfortunately, this solution does not scale in the context of cloud computing, since downloading big data incurs high communication costs that waste the advantage of outsourcing storage to the cloud.

This concern and the related challenges are addressed by a body of research in Proofs of Storage (POS), in which a user outsources the storage of large data to the cloud and further audits the cloud to check whether it stores the data as expected. In such an audit, the user challenges the cloud to return some *cryptographic proofs* asserting that the data is available and intact. Section 1.2 gives the definition of Proofs of Storage and requirements for such proofs are listed in Section 1.3. We introduce the security model specific to POS schemes in Section 1.4. We finally review prior work in Section 1.5.

1.2 Definition of a Proof of Storage Protocol

This section gives a formal definition of a POS protocol, inspired by the definition proposed respectively by Ateniese *et al.* [14], Juels and Kaliski [107] and Shacham and Waters [165].

1.2.1 Entities Involved in a POS Protocol

A POS scheme comprises the following entities:

Data owner $\mathcal{O}$: Data owner $\mathcal{O}$ wants to outsource the storage of her set of files, denoted $\mathcal{F}$, to a cloud server $\mathcal{S}$ and would like to obtain from $\mathcal{S}$ the assurance over the integrity of her files.

Cloud Server $\mathcal{S}$: Often mentioned as the *prover*, and considered as potentially malicious, cloud server $\mathcal{S}$ is presumed to store each file $F \in \mathcal{F}$ in its entirety. In practice, cloud server $\mathcal{S}$ stores an *enlarged*, *verifiable* version $\hat{F}$ of file F such that $\mathcal{S}$ can produce proofs showing that data owner $\mathcal{O}$ can retrieve her original file F.

Verifier $\mathcal{V}$: On behalf of data owner $\mathcal{O}$, verifier $\mathcal{V}$ enters a challenge-response protocol with cloud server $\mathcal{S}$ (*i.e.* the prover) to check whether $\mathcal{S}$ is storing $\mathcal{O}$'s file $F \in \mathcal{F}$ in its entirety. The role of verifier can be played either by data owner $\mathcal{O}$ herself or by any authorized entity, acting as an *auditor*.

1.2.2 System Model

We present here the definition of a POS system. The notion of POS was first formalized by Ateniese *et al.* [14] and Juels and Kaliski [107] then updated by Shacham and Waters [165]. The definition we propose here follows the challenge-response approach proposed in [107] while defining a *stateless* protocol as suggested by Shacham and Waters [165]. The term stateless means that the verifier does not need to maintain auxiliary information (a state) between several instances of verifications. We also define three POS phases (Setup, Challenge and Verification). Without loss of generality, we assume that each file $F \in \mathcal{F}$ is composed of n splits $\{S_1, S_2, ..., S_n\}$ of equal size of L bits. If necessary, F will be padded to a multiple of L. We also make the assumption that each split S_i comprises m blocks $\{b_{i1}, b_{i2}, ..., b_{im}\}$ of l bits, *i.e.* $L = m \cdot l$.

Definition 2 (POS Scheme). *A POS scheme consists of five polynomial-time algorithms* $(\mathsf{KeyGen}, \mathsf{Encode}, \mathsf{Challenge}, \mathsf{ProofGen}, \mathsf{Verify})$ *distributed across three POS phases.*

▶ **Setup.** *This phase only involves data owner $\mathcal{O}$. She runs* KeyGen *to produce the keying material required in the POS scheme and invokes* Encode *to prepare a verifiable version $\hat{F}$ of a particular file $F \in \mathcal{F}$ of hers:*

▷$\mathsf{KeyGen}(1^\kappa) \to K$: *This probabilistic key generation algorithm takes as input a security parameter 1^κ and outputs a secret key $K \in \{0,1\}^*$ for data owner $\mathcal{O}$.*

▷$\mathsf{Encode}(K, F) \to (\mathsf{fid}, \hat{F})$: *This algorithm takes key K and file $F = \{S_1, S_2, ..., S_n\}$ as inputs and returns the verifiable file $\hat{F} = \{\hat{S}_1, \hat{S}_2, ..., \hat{S}_n\}$ and F's unique identifier* fid.

It is worth mentioning that algorithm Encode *is invertible: There exists an algorithm* Decode *that lets data owner $\mathcal{O}$ recover her original file F from $\hat{F}$.*

At the end of the Setup phase, cloud server $\mathcal{S}$ is supposed to store file $\hat{F}$ together with F's identifier fid, *whereas data owner $\mathcal{O}$ removes F from her local storage, and only keeps the key output by* KeyGen. *Note that key K is independent of file F.*

► **Challenge.** *The Challenge phase consists in one or several challenge-response protocols involving verifier $\mathcal{V}$ and cloud server $\mathcal{S}$ (*i.e.* the prover). In essence, verifier $\mathcal{V}$ runs algorithm* Challenge *that generates POS requests to the prover so as to check the integrity of data owner $\mathcal{O}$'s file F. In turn, the prover invokes algorithm* ProofGen *that responds to verifier $\mathcal{V}$'s challenge by generating the requested proofs.*

▷$\mathsf{Challenge}(K, \mathsf{fid}) \to \mathsf{chal}$**:** *This probabilistic and randomized algorithm generates a challenge* chal *for an execution of the POS protocol for file F whose identifier corresponds to* fid*. It takes as inputs secret key K and file identifier* fid*, and returns challenge* chal*. For different executions of the POS protocol, algorithm* Challenge *always outputs different values for* chal*.*

▷$\mathsf{ProofGen}(\mathsf{fid}, \mathsf{chal}) \to \mathcal{P}$**:** *This algorithm is used by cloud server $\mathcal{S}$ to generate the proof of storage $\mathcal{P}$ for the target file $\hat{F}$ whose identifier is* fid*.*

The generated proof $\mathcal{P}$ is then sent to verifier $\mathcal{V}$ for Verification.

► **Verification.** *After receiving the proofs of storage for file F with identifier* fid *from cloud server $\mathcal{S}$, verifier $\mathcal{V}$ executes algorithm* Verify *to check their validity.*

▷$\mathsf{Verify}(K, \mathsf{fid}, \mathsf{chal}, \mathcal{P}) \to b \in \{0, 1\}$**:** *This deterministic algorithm decides whether $\mathcal{P}$ is a valid response to challenge* chal*. It takes as inputs key K, file identifier* fid*, challenge* chal *and proof $\mathcal{P}$. It outputs bit $b = 1$ if proof $\mathcal{P}$ is valid, $b = 0$ otherwise.*

1.3 Requirements for a POS protocol

Protocols for POS enable users of a remote storage service to verify that this service continuously and correctly stores their data, with the concern that such a verification does not waste the possibly limited bandwidth and computation resources of the users. In this section, we identify the design requirements and features that a POS must/should satisfy.

Security. The POS protocol must be sound, even against a malicious adversary, who would falsely claim that it stores the data correctly. The protocol must not allow such an adversary to produce correct proofs that will be accepted by the verifier (this is called the *unforgeability* of proofs of storage). We develop more the characterization of this requirement in Section 1.4.

Unbounded number of verifications. Data owners should be able to check that their data are correctly stored as many times as they want. Some POS schemes only provide limited number of verifications, compelling data owners to download their data back, in order to perform further verifications. Others guarantee an unbounded number of verifications.

Efficiency. Performances of a POS scheme can be assessed by means of four types of metrics: (i) the **communication cost** between the verifier and the server during the Challenge phase must not be large; (ii) the **computational cost** of algorithm Verify must be light for the verifier; (iii) the **computational cost** of algorithm ProofGen must be optimized for the server; (iv) the **amount of storage** induced by the POS protocol must be kept at minimum[46]. Besides, considering the two requirements of unbounded number of queries and efficiency, we authorize a POS protocol to adopt the *amortized*

[46]The data owner is not required to locally back-up a file to be checked.

model approach: an expensive Setup phase for the data owner is amortized over multiple instances of the Challenge and Verification procedures.

Along with the three above requirements, additional features can be adopted by a POS protocol.

POS with extractability. A certain number of solutions that verify storage are able to detect any data loss or modification, but they do not guarantee that the data can be recovered. As we will explain in Section 1.5.2.1, such schemes fall into the category of Provable Data Possession (PDP) or Remote Integrity Check (RIC). In other terms, they only assess whether the storage server possesses the data and stores it intact without allowing data owners to retrieve the data in its entirety. On the contrary, Proofs of Retrievability (POR) schemes, thoroughly described in Section 1.5.2.2, ensure that the verifier can retrieve her data in its entirety, at any point of time. In particular, we refer to *data retrievability* as a special security feature for data storage. Retrievability can be seen as a combination of *integrity* and *availability*.

POS with dynamic data. POS solutions allow to perform verifications on data that is prone to updates. The challenge here is to check that the outsourced data is correctly stored even in the case of modification by the data owner. Besides, this feature allows the data owner to update the data without the need to download it. Update operations include: deletion, modification or insertion of data blocks.

POS with public verification. We distinguish two modes for verifying storage: private verification and public verification. The feature of public verification is of special interest in the case where the verification procedure can be performed by anyone, not just the data owner. Precisely, anyone should be able to launch verification queries and verify the proofs of storage returned by the storage server.

POS with privacy. This security feature intends to protect the privacy of data owner's data and/or identity against third-party verifiers. The latter are authorized to assess whether the remote server stores the data correctly but not to infer any other information concerning the data.

1.4 Security Model of a POS Scheme

This section gives an outline of the two security requirements related to any POS scheme, namely *completeness* and *soundness* properties.

In a nutshell, the *completeness* property captures the fact that the POS scheme does not yield any false negatives. In other words, verifier $\mathcal{V}$ always accepts cloud server $\mathcal{S}$'s proof, whenever $\mathcal{S}$ stores the outsourced files. On the other hand, *soundness* requires that any prover, who convinces verifier $\mathcal{V}$ that she is storing some file F, is actually storing a verifiable version of file F.

1.4.1 Completeness

If cloud server $\mathcal{S}$ and verifier $\mathcal{V}$ are both honest, then on input of a challenge chal and some file identifier fid sent by verifier $\mathcal{V}$, using algorithm $\mathsf{Challenge}$, algorithm $\mathsf{ProofGen}$ generates a proof of storage that will be accepted by verifier $\mathcal{V}$ with probability 1.

1.4.2 Soundness

A POS scheme (KeyGen, Encode, $\mathsf{Challenge}$, $\mathsf{ProofGen}$, Verify) is **sound** if any malicious server cannot forge valid proofs of storage for a file F without storing a verifiable version of F in its entirety. In other words, a verifier can always detect (except with negligible probability) that a server deviates from a honest behavior. More details will be given in Section 2.1.2.

1.5 State of the Art on Proofs of Storage

Significant efforts to address the problem of POS were stimulated with the advent of cloud computing and big data in recent years. In particular, a large collection of research work proposes cryptographic solutions for a *verifier* (be it the data owner or third parties) to efficiently check that an untrusted remote server, the *prover*, correctly stores the outsourced data as expected, namely that the data is available and not tampered with.

We review in this section the existing POS solutions providing mechanisms to enable such verifications. These solutions include the pioneering work by Juels and Kaliski [107] on Proofs of Retrievability (POR) and Ateniese *et al.* [14] on Provable Data Possession (PDP).

This analysis of existing POS solutions will be based on the requirements (see Section 1.3) for the design of such solutions (security, unbounded number of verifications, efficiency), as well as additional features that some work adopt (extractability, handling of dynamic data, public verifiability and privacy-preserving mechanisms). These criteria allow us to establish a classification of the existing work.

Having identified the design requirements and features in Section 1.3, we propose to use the efficiency requirement as the first criterion for classification: we classify the existing work into two categories: *deterministic* and *probabilistic* solutions. In a nutshell, the first category (Section 1.5.1) ensures that the verifier can check the correct storage and be convinced with a probability of 1. This kind of solution often incurs heavy computations for the data owner, the prover and the verifier. The second category (Section 1.5.2) follows a sampling approach that involves some randomness in the verification protocol. This randomness makes uncertain the data integrity guarantee. Nevertheless, this probabilistic procedure makes more efficient solutions compared to deterministic solutions.

1.5.1 Deterministic Solutions

Early work [71, 84, 160] allow for Remote Integrity Check (RIC) that can convince a verifier with an integrity guarantee of 100 %. Deswarte and Quisquater [71], and independently Filho and Barreto [84], devise solutions that use RSA-based functions applied to the entire data for each verification challenge. The data owner needs first to pre-compute and to store a checksum on the data. The verifier challenges the prover based on a random number r and the server computes the proof of storage as $r^F \bmod N$ where F is the outsourced file and N a RSA modulus. The verifier then checks the validity of the proof against the pre-computed checksum. However, these RSA-based schemes suffer from a prohibitive cost for the server, especially in the case of large files, since the server has to exponentiate over the entire file F to compute its proof of integrity. Sebé *et al.* [160] revisit the approach followed by Filho and Barreto [84] and leverage the homomorphism property of the checksum: instead of generating the checksum over entire file F, the authors suggest to divide F into splits of same size $F = \{S_1, .., S_n\}$ and to compute the checksum over each split. The integrity verification requires the prover to compute a pseudo-random linear combination of all the splits in file F and to generate a proof based on this combination. Upon reception of the proof, the verifier checks its validity based on the checksum and the same combination of splits. In this setting, at the server side, the exponentiation time of the work in [84] is reduced to n sub-exponentiations. However, the data owner has to keep $\mathcal{O}(n)$ check values. Schwarz and Miller [159] develop the concept of algebraic signatures that, in combination with Error Correcting Code (ECC)[47], constitute the proofs of integrity. Algebraic signatures are generated such that the signature on the bits of redundancy (produced by the application of the ECC) is equal to the redundancy of the signatures on splits of data. The integrity check is based on the comparison of the signature on the ECC splits with the signature of the unencoded data (the check also relies on some additional technique to guarantee the freshness of the response). This retains the server from returning the entire file and the verifier to store

[47]We give a description of ECC in Section 2.2.3.2.

a local copy of that file. However, the communication complexity is linear in the size of the data, since the server has to return the signatures for all the splits. Besides, the authors of [159] did not provide any security proof of their scheme.

1.5.2 Probabilistic Solutions

To remove the burden of heavy communication and computation complexities, not always applicable in a cloud-based scenario, since this burden would cancel out the advantage of storing the data at a remote server, probabilistic solutions provide an alternative to the deterministic guarantee offered by the solutions described in Section 1.5.1. These probabilistic solutions rely on the technique of *random sampling*. In this method, each element of a set has an equal and independent chance of being selected. The rationale behind the use of random sampling is to optimize the communication and computation complexities of POS solutions: Instead of checking the integrity of the entire data (as performed in existing work presented in Section 1.5.1), probabilistic solutions for verifiable storage allow the verification of integrity of a randomly selected subset of data splits. Operating in such a way, a verifier can get the assurance that the data as a whole is correctly stored by the remote server. The optimization induced by the random sampling approach does not come without a price: sampling only ensures a probabilistic guarantee that the data is correctly stored in its entirety.

Two pioneer work on POS laid the basis for new solutions relying on random sampling: Provable Data Possession (PDP) introduced by Ateniese *et al.* [14] and Proofs of Retrievability (POR) defined by Juels and Kaliski [107]. These two seminal works on PDP and POR led to an intensive body of research on POS. PDP and POR protocols differ from the fact that only POR solutions satisfy the *extractability* requirement we established in Section 1.3. In other terms, PDP mechanisms only detect inconsistencies in the data but do not allow to recover from them.

1.5.2.1 Provable Data Possession

The *dynamicity* requirement enables us to provide a further categorization of methods that are based on PDP: we first review static solutions, then examine dynamic ones. We also analyze PDP schemes that provide other features, such as privacy-preserving data possession verification.

Static PDP schemes. PDP introduced by Ateniese *et al.* [14] enables a data owner to verify the integrity of outsourced data in an efficient way. The data owner is not required to keep a local copy of its outsourced data. The integrity check is performed according to a challenge-response protocol between the data owner and the remote server. In particular, its communication complexity is lighter that the one induced by the schemes presented in Section 1.5.1 and is independent of the data size.

RSA tags. Ateniese *et al.* [14] devise *homomorphic* verifiable tags as check-values for each data split. These items serve as an essential building blocks for many other PDP schemes but also POR techniques, as we will see in this review of the state of the art. In [14], the homomorphic tags are based on RSA and are of the form $\sigma_i = (w_i \cdot g^{S_i})^d \bmod N$, where w_i is a randomized function of the position i of split S_i in file $F = \{S_1, ..., S_n\}$, namely $w_i = \mathsf{PRF}(r, i)$, with r being a random number and PRF a pseudo-random function, N is a RSA modulus, d is the RSA private exponent such that $ed = 1 \bmod \varphi(N)$, e is the the RSA public exponent and g is a generator of the group of quadratic residues modulo N. The data owner keeps d secret, publishes N, e, r and g, and stores F and $\Sigma = \{\sigma_i\}_{1 \le i \le n}$ at the remote server (without keeping a local copy). To verify data possession, the verifier asks the server for tags of c pseudo-randomly chosen splits. The server generates a proof of storage based on the selected splits and their respective tags: The server generates a linear combination of the splits and thanks to the homomorphism on the tags, combines tags of the selected splits into a single tag. Since the homomorphic tags are encrypted using the secret RSA exponent d, this scheme provides *public verifiability* by means of public exponent e. The generation of the tags by the data owner is quite expensive, requiring $\mathcal{O}(n)$ exponentiations. However, the server exponentiations are reduced to a number c, corresponding to the number of challenged splits. In [16], the authors formalize the definition and use of homomorphic verifiable tags in publicly verifiable PDP protocols. In particular, they show how a homomorphic identification protocol can be derived to construct a PDP scheme (a homomorphic identification protocol originally authenticates an entity to a remote server without leaking any information). Curtmola *et al.* [67] envisioned the scenario where a file is replicated in t different (untrusted) servers to increase availability of the data. The authors devise a multi-replica PDP protocol (MR-PDP) that enables to verify the correct storage of the t replicas stored at the t servers. Instead of executing the single-replica PDP [14] for the t replicas, which may be computationally demanding since the data owner would have to compute homomorphic tags for each of the copies, the MR-PDP protocol generates a single set of tags (the tags are the one proposed in [14]) for the t copies of the same file. Besides, to prevent servers from colluding in order to correctly respond to a PDP challenge (when some of the servers do not store the data as expected), the protocol generates t unique and differentiable replicas by encrypting the original file and then applying a random mask of the encrypted file to produce the replicas. Even with this improvement, this solution still presents some weaknesses because the MR-PDP does not ensure that the proof generated by a challenged server really corresponds to the replica it was required to store.

BLS tags. Hanser and Slamanig [101] develop a PDP construction that simultaneously allow for private verification (that is only the data owner holding a secret key can perform the verification) and public verification. This scheme relies on elliptic curves and the computation of the tags is based on BLS signatures [41]. In a nutshell, a tag is computed as $\sigma_i = (H(i)g^{S_i})^\alpha$ and is publicly verified by checking that $e(\sigma_i, g) = e(H(i)g^{S_i}, g^\alpha)$, where e is a bilinear pairing[48], H a cryptographic hash function, g the generator of a cyclic group and α is data owner's secret key. The security of this scheme does not rely on standard assumptions but on the random oracle model.

Algebraic signatures. Chen [61] elaborates on algebraic signatures introduced in [159] to construct a PDP scheme that generates algebraic tags for data splits and imposes less computational overhead than homomorphic tags. The scheme pre-computes t verifications challenges. Each of these challenges randomly samples c data splits, that are combined into a single split corresponding to the sum of the c splits. This resulting split is then authenticated via its algebraic signature. To verify data possession, the verifier sends one of the pre-computed challenges and the server responds with the sum of the splits targeted by this challenge, and the corresponding algebraic signature. The obvious concerns with this method rely on the fact that the challenges are computed beforehand, limiting the number of verifications, and that this method induces a storage cost at the verifier.

Polynomial-based PDP. Krzywiecki and Kutyłowski [115] suggest tags that do not rely on RSA but on polynomial-based techniques. In [115], each data split S_i is divided in m blocks b_{ij} and is assigned to a pseudo-random polynomial P_i (the coefficients of P_i are pseudo-randomly generated) of degree m. The tag σ_i for split S_i corresponds to the set $((b_{i1}, P_i(b_{i1})), ..., (b_{im}, P_i(b_{im})))$. The data owner stores the data splits and the tags at the server. To test whether the server stores correctly split S_i, the verifier selects a random point x_{chal} (different from the blocks) and a (secret) nonce r and computes $y = g^{rP_i(x_{\mathsf{chal}})}$, with g being the generator of an appropriate group $\mathbb{G}$. The verifier sends $g^r, g^{rP_i(0)}, x_{\mathsf{chal}}$ to the server. The server computes the proof of data possession by computing $\Pi = g^{rP_i(x_{\mathsf{chal}})}$. To do so, the server employs the Lagrangian interpolation technique in the exponent, with the use of σ_i that contains the blocks and their image by P_i, and the challenge sent by the verifier. The latter then checks that Π corresponds to the expected value y. Beyond this interesting

[48]Bilinear pairings are introduced in Section 4.3.1.

effort to use polynomial interpolation, this scheme is not very efficient. Indeed, the protocol we just described applies to a single block only and involves expensive exponentiation and interpolation. Thus the verification of other blocks for a full PDP induces high cost. Some solutions to a problem orthogonal to verifiable storage can be applied as a technique for PDP. Indeed, verifiable polynomial evaluation[49] schemes [30, 90] enable to check that a remote server correctly evaluates an outsourced polynomial on some targeted inputs. The server must send a proof that the evaluation is correct. Benabbas *et al.* [30] and Gennaro *et al.* [90] suggest to use their respective protocol for verifiable polynomial evaluation as a building block for provable data possession. The idea is to encode the outsourced data as a polynomial whose coefficients correspond to the data splits. The proof consists in evaluating this polynomial over a random point sent by the verifier as a challenge. The server returns the outcome of this evaluation together with a proof of correct computation. The verifier checks the proof and recognizes that the server correctly stores the data if the proof is valid.

Dynamic PDP schemes. As we mentioned in Section 1.3, data can be subject to updates after its storage at a remote server. Enabling dynamic PDP is a useful feature: a verifier can run storage verification procedures while enabling the data owner to update the data (and verify the update) without the need to download the data.

Ateniese *et al.* [15] revisit their original PDP scheme [14] to address the problem of dynamic updates in the data. While in [14], public-key cryptography and homomorphic tags are used, the authors of [15] only employ symmetric-key cryptography. In this new scheme, the data owner pre-computes a collection of t verification tokens. Each of these tokens τ_j consists in c possible challenged splits of indices $i_1, ..., i_c$ and their corresponding answer v_j that is generated as $v_j = H(f(j), i_1, S_{i_1}) \oplus ... \oplus H(f(j), i_c, S_{i_c})$, where f is a pseudo-random function, H a cryptographic hash function and $\oplus$ the XOR operation. These tokens can either be locally stored or outsourced in an encrypted form to the server. Besides, the data owner should maintain a data structure to keep track of the structure of split indices in the data. To check that the server correctly stores the data, a verifier challenges the server with split indices $i_1, ..., i_c$ corresponding to token τ_j. The server then computes a proof of storage $z = H(f(j), i_1, S_{i_1}) \oplus ... \oplus H(f(j), i_c, S_{i_c})$. This proof is valid if z matches the corresponding verification token v_j. To update a data split stored at the server, the data owner must retrieve all the tokens that involve the updated split and update in each of the token answer the value of v_j $(1 \leq j \leq t)$ according to the new split. This technique is unpractical, since all the tokens have to be modified. Besides, the insertion of new data splits requires to update all the indices of consequent splits and thus update all the pre-computed tokens, which can be computationally intensive. Furthermore, this scheme suffers form the fact that the number of storage verifications is limited by the number t of precomputed tokens, which contravenes the requirement of unlimited number of verifications.

Skip list. An effective technique to support dynamic data updates relies on authenticated data structures, such as Merkle trees [125] or skip lists [153]. From this perspective, Erway *et al.* [77] (also Esiner *et al.* [78] that extend this work) propose a PDP protocol that is fully dynamic (all the update operations are supported in this scheme). The authors revisit the original skip list data structure to enable efficient insertions and deletions of splits, while being able to verify updates. Each split in the data is authenticated with a homomorphic tag $\sigma_i = g^{S_i} \bmod N$ which is stored in the skip list (N is a RSA modulus). The verification of storage is similar to the work proposed by Ateniese *et al.* [14]: the verifier challenges some randomly-selected blocks and the server returns a combination of the tags and the blocks, thanks to the homomorphism of the tags. The skip list is then used to authenticate that the combination is correctly computed for the requested blocks. Indeed, unlike in [14], the tag does not include the information on the index i. In addition, the skip list is used to perform the efficient updates. In this setting, the skip list induces a non-negligible storage overhead for the

[49]This problem will be considered in Part II.

server. Besides, deleting or inserting a data block can affect the computational complexities at the server.

Update tree. Zhang and Blanton [194] propose another PDP protocol that allows efficient updates, while maintaining a revision control history of past updates. In the scheme of Zhang and Blanton [194], each update is not directly verified. A verifier can check that the updates are well-performed only when she retrieves the outsourced data. This protocol relies on a new authenticated data structure that the authors name *balanced update trees*. Unlike in previous PDP proposals [77, 78], the authenticated data structure does not store information about the tags but about the updates themselves. The update operations are arranged into a balanced tree such that the verification of an update takes $\mathcal{O}(v)$ time, where v is the total number of the data versions (which is assumed to be identical to the number of updates performed on the data). While performing the update operations and their verification is efficient, this scheme requires the data owner to keep track of the update history. Thus, she needs to locally store the update tree (which is of size $\mathcal{O}(v)$). Furthermore, the tree must be updated, which may not be straightforward and lightweight, since keeping it balanced can require further computations. Besides, since the tags used to authenticate the data splits are just Message Authentication Codes (MACs), they cannot be aggregated into a single value as the homomorphic tags in [14]. Therefore, at each verification challenge, the communication complexity is linear with the number of challenged splits. Finally, the MACs have to be recomputed at each update, inducing additional costs for the data owner.

Merkle tree. Wang *et al.* [186] devise a publicly verifiable dynamic PDP scheme that employs homomorphic tags [14], Merkle hash trees[50] [125] and bilinear pairings[51] as building blocks. The authors observe that the tags of the scheme proposed by Ateniese *et al.* [14] cannot be devoted to a dynamic PDP protocol since they include the indices (the positions) of the splits in the data. Hence, when the data is updated, especially in the case of a new split insertion, the indices of subsequent splits are modified (incremented in the case of insertion), compelling the data owner to change (possibly many) other tags, which can be computationally demanding. Therefore, Wang *et al.* [186] pull away the index information for the computation of the tags. The data owner then computes the tags of the form $\sigma_i = (H(S_i)g^{S_i})^\alpha$, where H is a cryptographic hash function, α is a random number kept secret and g is the generator of a cyclic group (as a matter of fact, these tags are BLS signatures [41]). During a verification challenge, as in [14], the prover computes and returns a pseudo-random linear combination of splits and aggregation of their respective tags. The Merkle tree serves for authenticating $H(S_i)$. Indeed, to perform the verification, a verifier would need to recompute this value. For a matter of efficiency, instead of requesting the server to return each split S_i separately (along with the combination of challenged splits), the verifier only retrieves $H(S_i)$ and authenticates it using the tree, such that she can be sure that it really corresponds to the hash of the block at position i. Besides, the tree allows efficient split deletion and insertion (in $\mathcal{O}(\log n)$ time). Nevertheless, it is not clear whether a malicious server can be successful in a replay attack. Indeed, the server may use a previous version of the data to pass the verification.

PDP schemes with other features. In the static publicly verifiable PDP scheme devised by Wang *et al.* [185], third-party verifiers are considered as honest but curious, in the sense that they correctly follow the PDP procedure, but may learn unauthorized information about the data content. In particular, in the original PDP scheme [14], a verifier who has issued several challenges and received the same amount of proofs from the remote server, can recalculate the challenged splits via their combination $S_{\mathsf{chal}} = \sum_{i=1}^{c} \alpha_i S_i$. If the verifier receives enough S_{chal}, it can recover the S_i's using a simple linear equation system. To overcome this pitfall, Wang *et al.* [185] propose a privacy-preserving publicly verifiable PDP that requires

[50]Section 6.4.3 characterizes Merkle hash trees.

[51]Section 4.3.1 gives more details on bilinear pairings.

the server to apply a random mask over the split aggregation S_{chal}. Hence, such a technique prevents the verifier to infer any information about the data content from the proof of data possession. While this scheme protects data privacy against third-party verifiers, it does not protect against remote servers, to which the data blocks are sent in clear.

Shen and Tzeng [166] refine the concept of public verifiability. They propose a PDP protocol in which the data owner can delegate the verification ability to a third-party while preventing this party to re-delegate this capability to unauthorized users. In addition, the data owner can revoke the verification capability to an authorized verifier. These two features do not compel the data owner to compute new tags for the designated/revoked verifiers. Instead, the protocol "transforms" the tags computed from the data owner's secret key into tags that can be verified by the verifier's secret key, using a pre-computed token (note that the scheme involves similar tags to the ones in [186] based on BLS signatures [41]).

Wang *et al.* [183] propose Oruta, a PDP construction that considers the scenario of a (static) group of users sharing data in the cloud while preserving the identity of these users during integrity verifications performed by a third-party verifier (a similar scenario applies to their subsequent work [182, 184]). The splits are authenticated via homomorphic tags which are also shared among group users and may be subject to changes according to updates performed in the data. These tags are similar to the ones in [186] and are based on ring signatures [155, 40] that enable to verify that a signature is computed from the secret key of a member of a group of users. Oruta suffers from the fact that each group member generates tags for the shared data, which substantially increases the storage overhead at the server. Besides, Oruta applies to a group that is static, that is, it does not support group dynamicity, a new user joining the group, or an exisiting user leaving the group (or being revoked). In this perspective, in their consecutive works, the authors propose Knox [182] that accommodates to the issue of a new user joining the group and Panda [184] that allows to perform group member revocations, using homomorphic tags based on proxy re-signatures [38], that efficiently convert tags generated by a revoked group member to tags of remaining group members.

1.5.2.2 Proofs of Retrievability

The notion of POR was defined by Juels and Kaliski [107]. POR is stronger than PDP in the sense that POR offers the additional property that the data owner can be ensured that she can recover her data at any point of time. Indeed, the security model provided by POR schemes makes the definition of an extractor algorithm explicit. This algorithm can recover the outsourced data from a collection of POR challenges. Thus the security guarantee of POR is stronger than the PDP's. This extractability property is supported by means of Error Correcting Codes. In addition, the application of ECC mitigates an arbitrary number of data corruption. Sampling random splits detects substantial data corruption on the sampled splits and thus ensures that the remote server indeed committed to storing the outsourced data. However, small data corruption (tantamount to a couple of bits) may go undetected by the sampling technique, since it only targets large data corruption. In this case, the server can produce valid proofs of integrity for the sampled splits although the data is actually corrupted. To make the POR scheme robust to such corruptions, ECC is integrated in the pre-processing of the data before its outsourcing. Hence, corruptions on small parts of data provoke no damage since they can be recovered from the redundancy bits generated by the application of the ECC, while the corruptions affecting large amounts of data can be detected by random sampling. In the following, all the presented POR constructions involve the application of an ECC.

Static POR schemes. *Sentinels.* The original POR proposed by Juels and Kaliski [107] encodes each data split using an ECC algorithm, such as Reed-Solomon codes [154]. After encrypting the data with a semantically secure encryption, a bunch of pseudo-random valued blocks called *sentinels* are inserted in random positions in the data. These sentinels are in fact indistinguishable from other data blocks thanks to the semantically secure encryption of the data. Therefore, an adversary cannot tell whether a block is a real data block or a sentinel. To check the retrievability of the data, the verifier specifies the positions of a collection of randomly-selected sentinels and queries them from the remote server. The latter retrieves the queried sentinels and sends them back to the verifier who checks that they are intact. The rationale behind this protocol is that modifying part of the data will also, with a certain probability, impact the sentinels. Therefore, if the returned sentinels are not correct, this means that the data has been modified. This solution incurs very lightweight computational complexity, at the price of a limited number of verifications that can be performed by the verifier. Indeed, challenging the server discloses the position and value of the targeted sentinels. Thus the challenged sentinels cannot be used for further verification.

BLS signatures. Shacham and Waters [165] introduce the concept of Compact POR to improve the efficiency and the security guarantee provided by the POR construction of Juels and Kaliski [107], while enabling an unbounded number of verifications. The Compact POR protocol relies on homomorphic tags to minimize the bandwidth consumption during a POR challenge: indeed, the server can aggregate the tags into a single tag value, thanks to the homomorphism of the tags. Shacham and Waters [165] propose two distinct protocols: the first produces block tags that are based on MAC [27] for private verifiability and the second generates BLS signatures [41] as tags for public verifiability. The notion of POR is further formalized and generalized in [43]. In particular, Bowers *et al.* [43] observe that the ECC used in the protocol of Shacham and Waters [165] is inefficient and suggest different codes to improve the efficiency. Dodis *et al.* [75] improved the privately verifiable version in [165] to reduce the challenge communication complexity. Another direction of research considers the multi-server setting [44], where the outsourced data is replicated among several independent servers. If a verifier detects (using POR as a building block) that the data is not correctly stored in one of the servers, she can resort to the other servers to recover the parts of the data that are not intact. The POR protocol is conceived such that a block at a particular position is verified comparing the different blocks stored by the various servers at this particular position.

Polynomial-based POR. In [189], the authors revisit the POR scheme with private verifiability of Shacham and Waters [165] and construct a static POR protocol based on polynomial commitments defined in [108]. This scheme is very similar to the polynomial-based PDP proposed by Benabbas *et al.* [30] and Gennaro *et al.* [90] but with the additional application of an ECC to meet the extractability requirement. This polynomial commitment scheme enables to commit to a polynomial P and to generate a short proof (a witness) of the correct evaluation $P(r)$ on some input r. In [189], the block tags are computed using tags from [165] as $\sigma_i = H(i) + \sum_{j=1}^{m} b_{ij}\alpha^j = H(i) + P_i(\alpha)$ where each split S_i is regarded as a vector of blocks $\{b_{i1}, b_{i2}, ..., b_{im}\}$ that forms the coefficients of polynomial P_i. Then as in [165], to check the retrievability of the data, the verifier sends random positions to the server, who in turn computes a linear combination of the targeted splits and a linear aggregation of the corresponding tags. In addition, the server evaluates $P_{\mathsf{chal}}(r)$ where P_{chal} is the polynomial mapped to the challenged splits, and generates the corresponding witness of correct evaluation. Then the verifier checks that the proof is valid. This protocol improves the scheme by Shacham and Waters [165] in terms of communication complexity, but unlike [165] it does not ensure public verifiability.

RSA-based tags. Ateniese *et al.* [17] enhanced the original PDP construction of [14] with ECC to make it robust against small data corruptions. The authors observe that with the application of an ECC, three encoding strategies can be applied: (i) the data owner simply applies a Reed-Solomon code; (ii) to prevent an adversary to learn the dependencies between the splits and their corresponding redundancy blocks, the data owner applies the

Reed-Solomon code, permutes all the blocks in the data (including the redundancy blocks) and then encrypts the entire data; (iii) to prevent from executing an expensive preprocessing operation (due to the permutation step), the data owner only permutes the redundancy blocks. The choice between these three methods is made according to the desired security level of the data owner.

Dynamic POR schemes. As we will see in this paragraph, applying an ECC in the case of dynamic data is not straightforward. Indeed, updating a split also impacts the corresponding redundancy blocks, thus reveals to an adversary which redundancy blocks correspond to which splits. Having this extra knowledge enables the adversary to selectively delete some splits together with their redundancy blocks. If a POR challenge does not target the deleted blocks, the adversary is still able to generate valid PORs, that will be accepted by the verifier. The latter can then believe that the data is retrievable, even though some splits are deleted and cannot be recovered.

Batching updates via a proxy. Stefanov *et al.* [170] were the first to tackle the problem of dynamic PORs. They introduce an encoding based on XOR operations that is efficient and addresses the problem mentioned above. However, their setting involves an additional (trusted) proxy that caches update operations and performs them on behalf of the data owner. As the updates are not executed once requested, but cached by the proxy, the dependencies between redundant blocks and their corresponding data blocks are concealed.

Batching updates without a proxy. Chen and Curtmola [60] enhance the robust scheme of [17] to handle dynamic updates. In particular, they observe that updating even a small number of blocks may require to retrieve the entire file, since several blocks may be involved during the generation of the redundancy blocks. They present two protocols. The idea behind these two protocols is to leverage a variant of Reed-Solomon codes that is based on Cauchy matrices[52] [149]. The first protocol is efficient in encoding but suffers from high communication costs during updates (especially in the case of insertion operation, when the data owner needs to retrieve the entire data), while the second proposal reduces the communication costs but the encoding (and decoding) operation is more computationally demanding. Shi *et al.* [167] deploy a hierarchical log structure that *logs* the update operations to be performed. In other words, the protocol in [167] batches the update operations to handle them in a single time such that multiple data blocks are modified, concealing the dependencies between data and redundancy blocks.

O-RAM. Cash *et al.* [53] devise a (publicly verifiable) POR protocol with the technique of Oblivious RAM (O-RAM) [94]. In a nutshell, O-RAM consists in a data structure that allows the data owner to update the outsourced data in a privacy-preserving way. In particular, the *write* accesses are hidden, which conceals the dependencies between data and the redundancy blocks. This data structure is hierarchical: it consists of several layers of hash tables such that the top levels store the most recently accessed data and the bottom levels store the least recent ones. During an update operation, the modified split is inserted in the top table. During a read operation, that may occur before a write operation, the split is first searched in the top table. If it is found, the O-RAM structure simulates a search on the lower tables by looking for randomly-selected splits. Otherwise, the next layer is searched and so on and so forth until the last layer. This trick enables to conceal which data is about to be updated.

i$\mathcal{O}$ program. Guan *et al.* [100] innovate by proposing the first dynamic POR protocol with public verifiability that works in the symmetric-key setting. This protocol considers the static POR scheme with private verifiability proposed by Shacham and Waters [165] and brings the public verifiability property by means of an Indistinguishability Obfuscation (i$\mathcal{O}$) program. The indistinguishable obfuscation of a program [23, 88] intends to conceal the internal program operations while preserving its functionality. An i$\mathcal{O}$ program is thus able to embed secret information used by the program operations. In Guan *et al.* [100], the

[52]Cauchy matrices are matrices of elements $a_{ij} = \frac{1}{x_i + y_j}$, where x_i and y_j are elements of a Galois field.

authors suggest to create an i$\mathcal{O}$ program, available for any third-party verifier, that encodes the challenge and verification algorithms of the privately verifiable POR protocol in [165]. The secret key, used in that protocol as input to these two algorithms, is embedded in the i$\mathcal{O}$ program. Therefore, third-party verifiers do not need a secret key nor a public key to launch verification requests. By only running the i$\mathcal{O}$ program, they can verify the proofs of retrievability. Besides, the protocol of Guan *et al.* [100] supports dynamic updates thanks to the use of a Merkle tree. Resorting to an i$\mathcal{O}$ program may incur very expensive computational overhead since i$\mathcal{O}$ has not practical relevance yet. Besides, the work in [100] does not really handle the concern of concealing data and redundancy blocks under updates.

POR schemes with other features. Zheng and Xu [197] introduce the property of *fairness* in a dynamic POR protocol. Fairness implies the fact that the data owner is also considered as being malicious: for instance, she can accuse (to a judge in a court) a legitimate remote server of not storing her data correctly to gain financial compensation and damage server's reputation. The authors in [197] define a new authenticated data structure called 2-3 range-based tree that enables membership queries and dynamic operations in logarithmic complexity. To provide fairness, the protocol requires the server to verify a signature computed by the data owner at the time it receives the data, and whenever it receives an update request.

Similarly, Armknecht *et al.* [9] propose a publicly verifiable POR system in which the data owner and third-party verifiers are also considered as being potentially malicious. The authors formalize the concept of *outsourced* POR and build a concrete instantiation upon the scheme with private verification in [165]. Their construction requires the third-party verifier to create a log file which contains the collection of challenges and responses she performed. The data owner is able to check that the log entries are correct (which is claimed to be less expensive than running an entire POR protocol). Besides, the authors of [9] observe that the randomness involved in the sampling-based POR challenge can emanate neither from the data owner nor from the verifier only, since these are both untrusted. Therefore, they devise a technique based on the Bitcoin [130] mining process to produce (time-dependent) randomness, such that the data owner and the verifier are synchronized on a challenge without any interaction between each other. Since the randomness introduced by the Bitcoin process is somewhat verifiable, in case of conflict between the owner and the verifier, the latter can prove she correctly followed the POR protocol based on the provided randomness.

1.5.3 Conclusion of the State of the Art

Table 1.1 summarizes the analysis of the state of the art of Proofs of Storage. From the review of existing work, we are able to draw some conclusions that guide our own research perspective in the field of verifiable storage:

1. POR protocols are more efficient than deterministic remote integrity checking at the price of a storage assurance that is probabilistic.

2. POR protocols offer a better security guarantee than PDP schemes, with regard to storage integrity and retrievability.

3. The POR construction of Juels and Kaliski [107] incurs light bandwidth and computation overhead but suffers from the fact that it does not satisfy the requirement on unbounded number of verification.

From the last observation, we propose our own POR protocol, $\mathcal{S}$tealthGuard, that is inspired by the breakthrough of Juels and Kaliski [107] but which overcomes the problem of a limited number of verification.

		Deterministic	Probabilistic	PDP	POR	Unbounded	Static	Dynamic	Public Verifiability	Privacy-Preserving
Deterministic	Deswarte and Quisquater [71]	✓				✓	✓			
	Filho and Barreto [84]	✓				✓	✓			
	Sebé *et al.* [160]	✓				✓	✓			
	Schwarz and Miller [159]	✓				✓	✓			✓
Static PDP	Ateniese *et al.* [14]		✓	✓		✓	✓		✓	
	Curtmola *et al.* [67]		✓	✓		✓	✓			✓
	Wang *et al.* [185]		✓	✓		✓	✓		✓	✓
	Krzywiecki and Kutyłowski [115]		✓	✓		✓	✓			
	Chen [61]		✓	✓			✓			
	Hanser and Slamanig [101]		✓	✓		✓	✓		✓	✓
	Shen and Tzeng [166]		✓	✓		✓	✓		✓	
	Wang *et al.* [183]		✓	✓		✓	✓		✓	✓
Dynamic PDP	Ateniese *et al.* [15]		✓	✓			✓	✓		
	Erway *et al.* [77]		✓	✓		✓	✓	✓		
	Esiner *et al.* [78]		✓	✓		✓	✓	✓		
	Zhang and Blanton [194]		✓	✓		✓	✓	✓	✓	
	Wang *et al.* [186]		✓	✓		✓	✓	✓	✓	
Static POR	Juels and Kaliski [107]		✓		✓		✓			
	Shacham and Waters [165] (private)		✓		✓	✓	✓			
	Shacham and Waters [165] (public)		✓		✓	✓	✓		✓	
	Bowers *et al.* [43]		✓		✓	✓	✓			
	Dodis *et al.* [75]		✓		✓	✓	✓			
	Bowers *et al.* [44]		✓		✓	✓	✓		✓	
	Xu and Chang [189]		✓		✓	✓	✓			
	Ateniese *et al.* [17]		✓		✓	✓	✓		✓	
	Armknecht *et al.* [9]		✓		✓	✓	✓		✓	
Dynamic POR	Stefanov *et al.* [170]		✓		✓	✓	✓	✓		
	Cash *et al.* [53]		✓		✓	✓	✓	✓	✓	
	Shi *et al.* [167]		✓		✓	✓	✓	✓	✓	
	Chen and Curtmola [60]		✓		✓	✓	✓	✓	✓	
	Guan *et al.* [100]		✓		✓	✓	✓	✓	✓	
	Zheng and Xu [197]		✓		✓	✓	✓	✓		

Table 1.1: Existing work for Proofs of Storage

Chapter 2

StealthGuard: Proofs of Retrievability with Hidden Watchdogs

This chapter presents $\mathcal{S}$tealthGuard, a new Proofs of Retrievability (POR) scheme which combines the use of a Privacy-Preserving Word Search (PPWS) scheme [37] suited for large data stores with the insertion in the outsourced data of randomly generated short bit sequences, named **watchdogs**. We see *data retrievability* as a fundamental security requirement for data storage. Retrievability can be interpreted as a combination of *integrity* and *availability*.

Proofs of retrievability are a particular type of Proofs of Storage, with the extra property of *extractability*. Indeed, not only a data owner can check the integrity and the availability of her outsourced data, but she can also check that her data is retrievable. In a nutshell, this property implies that there exists an extractor that, via an interaction with the storage server (which can be malicious), can recover the outsourced data in its entirety, with some overwhelming probability. Being an instance of a POS scheme, a POR protocol inherits from the characteristics of POS and adopts the definition of a POS protocol we presented in Section 1.2. In particular, a POR solution must meet the requirements of security, unbounded number of verification and efficiency

After defining the security of a POR protocol in Section 2.1.2, we describe $\mathcal{S}$tealthGuard in Section 2.2. We analyze the security of $\mathcal{S}$tealthGuard in Section 2.3 and evaluate its performances in Section 2.4

2.1 Security Model of POR

A POR protocol must be complete and sound. *Completeness* means that the POR scheme does not yield any false negatives: a verifier $\mathcal{V}$ always accepts a proof of a honest cloud server $\mathcal{S}$. *Soundness* characterizes the fact that it is impossible for a malicious server to make the verifier accept forged (and false) proofs of retrievability.

2.1.1 Completeness

If cloud server $\mathcal{S}$ and verifier $\mathcal{V}$ are both honest, then on input of a challenge chal and some file identifier fid sent by verifier $\mathcal{V}$, using algorithm $\mathsf{Challenge}$, algorithm $\mathsf{ProofGen}$ generates a **proof of retrievability** that will be accepted by verifier $\mathcal{V}$ with probability 1.

Definition 3 (Completeness). *A POR scheme* (KeyGen, Encode, Challenge, ProofGen, Verify) *is **complete** if for any honest pair of cloud server $\mathcal{S}$ and verifier $\mathcal{V}$, and for any key $K \leftarrow$* KeyGen(1^κ)*, any file $F \in \mathcal{F}$ whose file identifier is* fid *and for any challenge* chal $\leftarrow$ Challenge(K, fid)*:*

$$\Pr[\mathsf{Verify}(\mathsf{K}, \mathsf{fid}, \mathsf{chal}, \mathcal{P}) \rightarrow 1 \mid \mathcal{P} \leftarrow \mathsf{ProofGen}(\mathsf{fid}, \mathsf{chal})] = 1$$

2.1.2 Soundness

In this section, we elaborate the soundness definition which is peculiar to any POR scheme.

2.1.2.1 Intuition behind the definition

A POR protocol is deemed sound, if for any malicious prover, *i.e.* malicious cloud server $\mathcal{S}$, the only way to convince a verifier $\mathcal{V}$ that a file F is retrievable is by actually storing a retrievable version of that file. As a consequence, if server $\mathcal{S}$ correctly executes a *polynomial* number of proof generations for file F, using algorithm ProofGen, thus yielding valid proofs, then it implies that server $\mathcal{S}$ should keep that file intact in such a way that data owner $\mathcal{O}$ can later retrieve her file F. According to Juels and Kaliski [107], such a definition gives way to the existence of a special algorithm $\mathcal{E}$, called the file extractor algorithm, which via an interaction with cloud server $\mathcal{S}$, is able to extract file F with an overwhelming probability, using the sound POR protocol. Shacham and Waters [165] revisit the definition of soundness by capturing the soundness of POR schemes that empower the verifier with unlimited number of *possible* POR challenges. However this definition does not apply to POR schemes where only limited number of *possible* challenges are available to the verifier such as in [107, 170]. Indeed, the definition in [165] asserts that whenever cloud server $\mathcal{S}$ generates valid PORs for some file F with a non-negligible probability and whenever the file is recoverable, then the corresponding POR scheme is sound. In the case of other POR schemes, such as the one proposed in [107, 170], where the number of possible POR challenges is limited, this definition cannot be employed to assess the soundness property. For example, if we take the POR scheme introduced by [107], and if we consider a scenario where cloud server $\mathcal{S}$ randomly corrupts half of the outsourced file, then the server will be able to correctly answer half of the POR challenges that the verifier issues, yet the file is irretrievable. This means that the POR mechanism adopted by Juels and Kaliski [107] is not secure under the definition of [165], still it is arguably "sound". In practice, to check whether a file is retrievable, the verifier generates a polynomial number of POR queries to which the server has to respond correctly. Otherwise, the verifier detects a corruption attack, be it malicious or unintentional. Here, the term "POR query" captures the ability of a verifier to issue requests for proofs of retrievability based on the challenges available to the verifier. In the following section, we aim at revisiting the soundness definition given in [165] with this notion of *number of POR queries that the verifier should generate either to be sure that a file is retrievable or to detect a corruption attack on the file.*

2.1.2.2 Definition details

The following lines formalize our definition of the protocol soundness. This definition sets a *soundness game* in which a Probabilistic Polynomial-Time (PPT) adversary $\mathcal{A}$ (*i.e.* a malicious cloud server) intends to frame a verifier $\mathcal{V}$. $\mathcal{A}$'s goal is to create a situation where verifier $\mathcal{V}$ is convinced that data owner $\mathcal{O}$'s file is retrievable, with an overwhelming probability, while that file is actually not. Adversarial corruption can consist in either modification or deletion of file blocks. This section also defines the file extractor algorithm, denoted $\mathcal{E}$.

To formally capture the capabilities of adversary $\mathcal{A}$, we assume that she has access to the following oracles:

$\mathcal{O}_{\mathsf{Encode}}$ This oracle takes as inputs a file F and data owner's key K, and returns a file identifier fid and a verifiable version $\hat{F}$ of F that will be outsourced by $\mathcal{A}$.

$\mathcal{O}_{\mathsf{Challenge}}$ On input of a file identifier fid and data owner's key K, this oracle returns a POR query chal to adversary $\mathcal{A}$.

$\mathcal{O}_{\mathsf{Verify}}$ When queried with data owner's key K, a file identifier fid, a POR query chal and a proof of retrievability $\mathcal{P}$, this oracle outputs a bit b such that $b = 1$ if $\mathcal{P}$ is a valid proof of retrievability and $b = 0$ otherwise.

In the soundness game we define, adversary $\mathcal{A}$ accesses the aforementioned oracles in two phases: a **learning** phase and a **challenge** phase.

Learning. Adversary $\mathcal{A}$ can call oracles $\mathcal{O}_{\mathsf{Encode}}$, $\mathcal{O}_{\mathsf{Challenge}}$ and $\mathcal{O}_{\mathsf{Verify}}$ for a polynomial number of times in any interleaved order as depicted in Algorithm 1. In particular, adversary $\mathcal{A}$ can make three types of oracle queries:

Encode query: $\mathcal{A}$ queries $\mathcal{O}_{\mathsf{Encode}}$ for a file F of her choice, given key K. $\mathcal{O}_{\mathsf{Encode}}$ outputs the corresponding retrievable version $\hat{F}$ of F together with a generated file identifier fid, and sends them to $\mathcal{A}$.

Challenge query: Given a file identifier fid associated with some file F chosen by adversary $\mathcal{A}$ and given key K, $\mathcal{A}$ queries $\mathcal{O}_{\mathsf{Challenge}}$ to generate a random challenge chal. On reception of chal, adversary $\mathcal{A}$ produces a proof of retrievability $\mathcal{P}$, either arbitrarily or by executing algorithm $\mathsf{ProofGen}$.

Verify query: Adversary $\mathcal{A}$ calls $\mathcal{O}_{\mathsf{Verify}}$ to check the proof $\mathcal{P}$ based on challenge chal and file identifier fid and gets decision bit b.

At the end of this learning phase, adversary $\mathcal{A}$ chooses a file identifier fid^* among all the file identifiers she has obtained throughout the phase. We denote F^* the corresponding file.

▼ **Algorithm 1:** Learning phase of the soundness game

```
// $\mathcal{A}$ executes the following in any interleaved order for a polynomial number of times
$(\mathsf{fid}, \hat{F}) \leftarrow \mathcal{O}_{\mathsf{Encode}}(F, K)$;
$\mathsf{chal} \leftarrow \mathcal{O}_{\mathsf{Challenge}}(K, \mathsf{fid})$;
$\mathcal{P} \leftarrow \mathcal{A}$;
$b \leftarrow \mathcal{O}_{\mathsf{Verify}}(K, \mathsf{fid}, \mathsf{chal}, \mathcal{P})$;
// $\mathcal{A}$ outputs a file identifier $\mathsf{fid}^*$
$\mathsf{fid}^* \leftarrow \mathcal{A}$;
```

Challenge. The goal of adversary $\mathcal{A}$ is to generate γ valid proofs of retrievability $\mathcal{P}_j^*$ for file F^* whose file identifier is fid^* (cf. Algorithm 2). To this end, $\mathcal{A}$ calls oracle $\mathcal{O}_{\mathsf{Challenge}}$ that supplies her with γ challenges chal_j^*. Next, adversary $\mathcal{A}$ generates γ proofs $\mathcal{P}_j^*$ for each of the challenges. Finally, on input of data owner key K, file identifier fid^*, challenges chal_j^* and proofs $\mathcal{P}_j^*$ $(1 \leq j \leq \gamma)$, oracle $\mathcal{O}_{\mathsf{Verify}}$ outputs γ decision bits b_j^*.

Adversary $\mathcal{A}$ is deemed successful if $b^* = \bigwedge_{j=1}^{\gamma} b_j^* = 1$. In other terms, adversary $\mathcal{A}$ succeeds in producing γ proofs of retrievability $\mathcal{P}_j^*$ that are accepted by $\mathcal{O}_{\mathsf{Verify}}$.

Thereafter, we formalize the notion of the file extractor algorithm, $\mathcal{E}$, that uses adversary $\mathcal{A}$ to retrieve file F^*:

▼ Algorithm 2: Challenge phase of the soundness game

```
for $j = 1$ to $\gamma$ do
    $\mathsf{chal}^*_j \leftarrow \mathcal{O}_{\mathsf{Challenge}}(K, \mathsf{fid}^*)$;
    $\mathcal{P}^*_j \leftarrow \mathcal{A}$;
    $b^*_j \leftarrow \mathcal{O}_{\mathsf{Verify}}(K, \mathsf{fid}^*, \mathsf{chal}^*_j, \mathcal{P}^*_j)$;
end
$b^* = \bigwedge_{j=1}^{\gamma} b^*_j$;
```

▷$\mathcal{E}(K, \mathsf{fid}^*) \to F^*$**:** The extractor algorithm takes as input data owner's key K and file identifier fid^*. $\mathcal{E}$ is allowed to initiate a polynomial number of POR executions by interacting with adversary $\mathcal{A}$ for file F^*. $\mathcal{E}$ is also allowed to rewind adversary $\mathcal{A}$. This suggests in particular that extractor $\mathcal{E}$ can execute the challenge phase of the soundness game a polynomial number of times, while the state of adversary $\mathcal{A}$ remains unchanged.

Intuitively, a POR scheme is sound, if for any adversary $\mathcal{A}$ that wins the soundness game described above with a non-negligible probability δ, there exists a file extractor algorithm $\mathcal{E}$ that succeeds in retrieving challenge file F^* with an overwhelming probability. We say that a probability is overwhelming if it is equal to $1 - \varepsilon$, where ε is negligible.

Definition 4 (Soundness). *A POR scheme* (KeyGen, Encode, $\mathsf{Challenge}$, $\mathsf{ProofGen}$, Verify) *is said to be* (δ, γ)*-sound, if for every adversary* $\mathcal{A}$ *that provides* γ *valid proofs of retrievability in a row (*i.e. *succeeds in the soundness game described above) with a non-negligible probability* δ*, there exists an extractor algorithm* $\mathcal{E}$ *such that:*

$$Pr[\mathcal{E}(K, \mathsf{fid}^*) \to F^* \mid \mathcal{E}(K, \mathsf{fid}^*) \rightleftharpoons \mathcal{A}] \leq 1 - \varepsilon,$$

where $\rightleftharpoons$ *symbolizes the interaction between extractor* $\mathcal{E}$ *and adversary* $\mathcal{A}$*, and* ε *is a negligible function in security parameter* κ*.*

Informally, if verifier $\mathcal{V}$ issues a number of queries larger than γ and if cloud server $\mathcal{S}$ correctly responds to them, then $\mathcal{V}$ can ascertain that $\mathcal{S}$ is still storing a retrievable version of file F^* with high probability. While γ characterizes the number of *valid* proofs of retrievability that extractor $\mathcal{E}$ has to receive to assert that file F^* is retrievable, δ quantifies the number of operations that $\mathcal{E}$ has to execute and the amount of data that it has to download to first declare that F^* is retrievable and then to extract it.

2.2 StealthGuard

2.2.1 Intuition behind StealthGuard

$\mathcal{S}$tealthGuard pursues an idea originally proposed by Juels and Kaliski [107] which relies on the insertion, in random positions in the data to be outsourced, of some special (pre-computed) random blocks, called *watchdogs*. Then, the proofs of retrievability consist in checking that some of these watchdogs are still intact in the outsourced data. This method is based on the random sampling technique: at each instance of the POR protocol, the verifier samples some of the watchdogs and checks their integrity. In other terms, it offers a probabilistic retrievability guarantee while being computationally very light.

In a nutshell, to prepare a retrievable version of a data file F, data owner $\mathcal{O}$ first encrypts her data and embeds some pseudo-randomly generated watchdogs in it. Here, encryption guarantees the indistinguishability of the watchdogs from a real data block. Once the data is outsourced to cloud server $\mathcal{S}$, a verifier $\mathcal{V}$ issues queries for some watchdogs, in order to check that they are intact in the data. Cloud server $\mathcal{S}$ responds to these queries by generating a proof for the targeted watchdogs. If unintentional or adversarial corruption affects the data, then with high probability, it would also impact the watchdogs. Thus, cloud server $\mathcal{S}$ would unlikely return a valid proof of retrievability. Besides, in order to protect the data from small corruptions, $\mathcal{S}$tealthGuard applies an ECC that enables the recovery of the corrupted data. In other words, $\mathcal{S}$tealthGuard satisfies the requirement of **extractability**.

Our proposal differs from the solution of [107] in the way server $\mathcal{S}$ generates the proofs of retrievability. In [107], verifier $\mathcal{V}$ selects a collection of watchdogs and sends their positions to server $\mathcal{S}$ which responds with the blocks located at the requested positions. Thereafter, verifier $\mathcal{V}$ checks that the returned blocks are really the requested watchdogs. Although this solution is efficient in terms of computational complexity, it does not meet the **unbounded number of POR queries** requirement. Indeed, when verifier $\mathcal{V}$ challenges server $\mathcal{S}$, she discloses the positions of the watchdogs. Hence, the challenged watchdogs cannot be used for further verifications since $\mathcal{S}$ knows that these particulars blocks are *watchdogs* and thus can discard the blocks that correspond to the file content. Juels and Kaliski [107] envisioned to use a Private Information Retrieval (PIR) algorithm to retrieve the watchdogs in a privacy-preserving way. This technique would allow to re-use them for an unbounded number of verifications. However, the authors did not investigate on this idea since they perceived the application of a PIR algorithm on the entire data as being computationally prohibitive.

To cope with this concern, $\mathcal{S}$tealthGuard leverages a Privacy-Preserving Word Search (PPWS) scheme combined with watchdogs inserted in the outsourced data. By selecting an *efficient* PPWS solution and by inserting an *optimal* number of watchdogs (this number will be determined in Section 2.3.2), $\mathcal{S}$tealthGuard is efficient while satisfying the **unbounded number of verification** requirement. The privacy preserving property of the search ensures that cloud server $\mathcal{S}$ cannot discover which watchdogs were targeted by the search queries. As a result, verifier $\mathcal{V}$ can launch an unbounded number of POR queries, even for the same watchdog, without the need to update the data with new watchdogs, thus realizing the desired POR design requirement. In addition, the search results are obfuscated thanks to the underlying PPWS scheme. The only way that cloud server $\mathcal{S}$ can convince verifier $\mathcal{V}$ about the retrievability of a targeted file is by returning valid search results, that is by storing the file in its entirety and executing the PPWS correctly.

2.2.2 StealthGuard Phases

In this section, we consider the following scenario: Data owner $\mathcal{O}$ outsources to cloud server $\mathcal{S}$ some of her files F. At some point of time[53], verifier $\mathcal{V}$ (be it the data owner herself or any authorized party who is delegated the challenge and verification capabilities) checks the retrievability of file F using $\mathcal{S}$tealthGuard.

In accordance with the definition of a POR scheme stated in Section 1.2, our protocol comprises three phases:

- **Setup.** During this phase, $\mathcal{O}$ performs some transformations over F and inserts a certain number of watchdogs to F. The resulting file $\hat{F}$ is sent to server $\mathcal{S}$.
- **Challenge.** This phase corresponds to the Challenge phase described in Definition 2. It consists in searching for some watchdog w in a privacy-preserving manner. Hence, $\mathcal{V}$ prepares and sends a privacy-preserving lookup query for w; $\mathcal{S}$, in turn, processes $\hat{F}$ to generate a correct response to the search and returns the output to verifier $\mathcal{V}$.
- **Verification.** Verifier $\mathcal{V}$ checks the validity of the received response and makes the decision about the existence of watchdog w in the outsourced file.

[53]In practice, a verifier will audit the cloud storage in a periodic fashion based on a contract, or SLA.

According to the POR soundness definition we gave in Definition 4, verifier $\mathcal{V}$ must receive at least γ (γ must be larger than a threshold determined in Section 2.3) correct responses from $\mathcal{S}$ to decide that $\mathcal{O}$'s file F is retrievable. On the other hand, if $\mathcal{V}$ receives one response that is not valid, then she is convinced that F is either corrupted or lost.

2.2.3 Building Blocks

Before detailing the phases of $\mathcal{S}$tealthGuard, we start by introducing the different building blocks that are used in our solution.

2.2.3.1 Pseudo-Random Functions and Permutations

Pseudo-Random Function (PRF). Informally, a pseudo-random function, introduced by Goldreich *et al.* [95], is a function $\Phi : \mathcal{K} \times \mathcal{X} \to \mathcal{Y}$, where $\mathcal{K}$, $\mathcal{X}$ and $\mathcal{Y}$ respectively denote the set of keys, the set of possible inputs and the set of possible outputs, such that:

- Given a key $K \in \mathcal{K}$ and an input $X \in \mathcal{X}$, there exists an *efficient* algorithm to compute $Y = \Phi(K, X) \in \mathcal{Y}$;
- Φ is indistinguishable from a random function, selected from the uniform distribution of all functions $f : \mathcal{X} \to \mathcal{Y}$.

In $\mathcal{S}$tealthGuard, the watchdogs are generated using an appropriate PRF, see Section 2.2.4.1.

Pseudo-Random Permutation (PRP). In a nutshell, a pseudo-random permutation is a function $\Pi : \mathcal{K} \times \mathcal{X} \to \mathcal{X}$, where $\mathcal{K}$ and $\mathcal{X}$ respectively represent the set of keys and the set of possible inputs, such that:

- Given a key $K \in \mathcal{K}$ and an input $X \in \mathcal{X}$, there exists an *efficient* algorithm to evaluate $Y = \Pi(K, X) \in \mathcal{X}$;
- For any key $K \in \mathcal{K}$, Π is a bijection from $\mathcal{X}$ to $\mathcal{X}$. Namely there exists an *efficient* inverse function $\Pi^{-1} : \mathcal{K} \times \mathcal{X} \to \mathcal{X}$ such that, for any key $K \in \mathcal{K}$ and any $X \in \mathcal{X}$, $X = \Pi^{-1}(K, \Pi(K, X))$;
- Π is indistinguishable from a random permutation, selected from the uniform distribution of all permutations $\pi : \mathcal{X} \to \mathcal{X}$.

$\mathcal{S}$tealthGuard employs such a PRP to randomize the position of data blocks and the location of the watchdogs in the data. A cryptographically secure keyed block cipher such as Advanced Encryption Standard (AES) [68] is a good PRP candidate.

2.2.3.2 Error-Correcting Codes

An $[n, k, d]$-error correcting code (ECC) denotes a code which transforms a k-bit word into a n-bit codeword, where d denotes the minimal Hamming distance[54] between codewords. Any ECC algorithm would suit to $\mathcal{S}$tealthGuard. As a matter of example, we can employ Reed-Solomon error-correcting codes. Reed-Solomon codes are an $[n, k, n-k+1]$-ECC over a finite field $\mathbb{F}_p$ of prime order $p > n$. The encoded unit, called a *symbol*, is generally a byte or any symbol of 2^i bits. As the Reed-Solomon codes are known to be block-based codes, they encode data in blocks, where the block length is defined by n: a block is constituted of n symbols. Reed-Solomon codes are also systematic, in the sense that the original data is

[54] The Hamming distance of two sequences of bits is the number of bits which differ from the two sequences. If we denote $d(a, b)$ the Hamming distance between two bit strings $a = (a_i)_{i \in [0, n-1]}$ and $b = (b_i)_{i \in [0, n-1}$ with $n \in \mathbb{N}$, then $d(a, b) = \sum_{i=1}^{n} a_i \oplus b_i$.

left unchanged and the blocks generated during the encoding are just appended at the end of the data. We call these extra blocks “parity” blocks or “redundant” blocks or even “ECC” blocks[55]. The Reed-Solomon decoding processes each block of data and attempts to detect and correct the errors that can be included in the data. Such a decoder can detect and correct up to $\frac{d}{2} = \frac{n-k+1}{2}$ errors.

2.2.3.3 Semantically-Secure Encryption

The notion of a semantically secure encryption SSE scheme, introduced by Goldwasser and Micali [96], characterizes a probabilistic encryption scheme in which, given the ciphertext of some message chosen from any distribution of plaintexts, any *passive* Probabilistic Polynomial-Time (PPT) adversary cannot distinguish which of these plaintexts correponds to the given ciphertext. In other terms, semantic security guarantees that the adversary cannot feasibly derive any significant information about the plaintext from the ciphertext (and the message’s length).

In $\mathcal{S}$tealthGuard, the watchdogs are pseudo-randomly generated (with the use of a PRF), and inserted in the data encrypted with a SSE algorithm. Hence, no adversary can distinguish watchdogs from original data blocks.

2.2.3.4 Privacy-Preserving Word Search

As mentioned above, $\mathcal{S}$tealthGuard enables the cloud server to operate a search for the watchdogs over the (encrypted) data it stores, in a privacy-preserving fashion. By relying on a Privacy-Preserving Word Search (PPWS) algorithm that preserves both search queries and search results privacy, our solution ensures that the cloud server cannot discover which watchdogs are targeted by the search queries and does not learn any information on whether these watchdogs are found or not. As a result, the PPWS algorithm allows to reuse the watchdogs for future queries.

In this work, we resort to a *simplified* version[56] of an existing solution called Prism [37]. Prism transforms the search problem into several parallel efficient Private Information Retrieval (PIR) instances. The PIR mechanism inherited by Prism is based on the Trapdoor Group PIR scheme[57] proposed by Trostle and Parrish [176] that is depicted in Figure 2.1. In this scheme, the data outsourced in the cloud is represented by a matrix and the PIR procedure allows a user to obtain one row from this matrix without letting the cloud determine which row the user is querying. The words are mapped to a binary matrix and unique position for each word. Therefore, a user can perform a word search on her outsourced data as follows:

- A user issues a search query for a particular word ω using the underlying $\mathsf{PIRquery}$ algorithm (cf. Figure 2.1);
- The cloud creates an *index* matrix $\mathcal{M}$ of “0” and “1” on-the-fly, where each matrix position is assigned to a word in the data and where each matrix element stores a one-bit witness calculated via a hash over the word assigned to the corresponding position;
- The cloud runs $\mathsf{PIRprocess}$, described in Figure 2.1, with inputs matrix $\mathcal{M}$ and the PIR query on word ω. It should be highlighted that Prism is not another PIR algorithm. Indeed, algorithm $\mathsf{PIRprocess}$ does not retrieve the word itself, but it retrieves some very short

[55]Same terminology applies for symbols and bits. Thus we might mention redundant symbols or parity bits.

[56]However, any efficient PPWS scheme assuring the confidentiality of both the query and the result can fit in our framework.

[57]A security breach in the PIR algorithm developed by Trostle and Parrish [176] have been found by Lepoint and Tibouchi [119], shortly after the design of $\mathcal{S}$tealthGuard. Nevertheless, for a matter of illustration, we describe in this section the mechanism behind this PIR protocol. In our protocol, the PIR mechanism operates as a black box: any secure (computationally-)PIR solution that guarantees the confidentiality of the PIR query and PIR reply can be applied to our protocol

information (i.e the witness) that enables the user to decide about the presence or absence of the searched word.

• The user analyzes the output of the PIRanalysis (see Figure 2.1) and gets the answer of the search result (either word ω is present, or absent in the targeted data).

Figure 2.1: Private Information Retrieval of Trostle and Parrish [176]

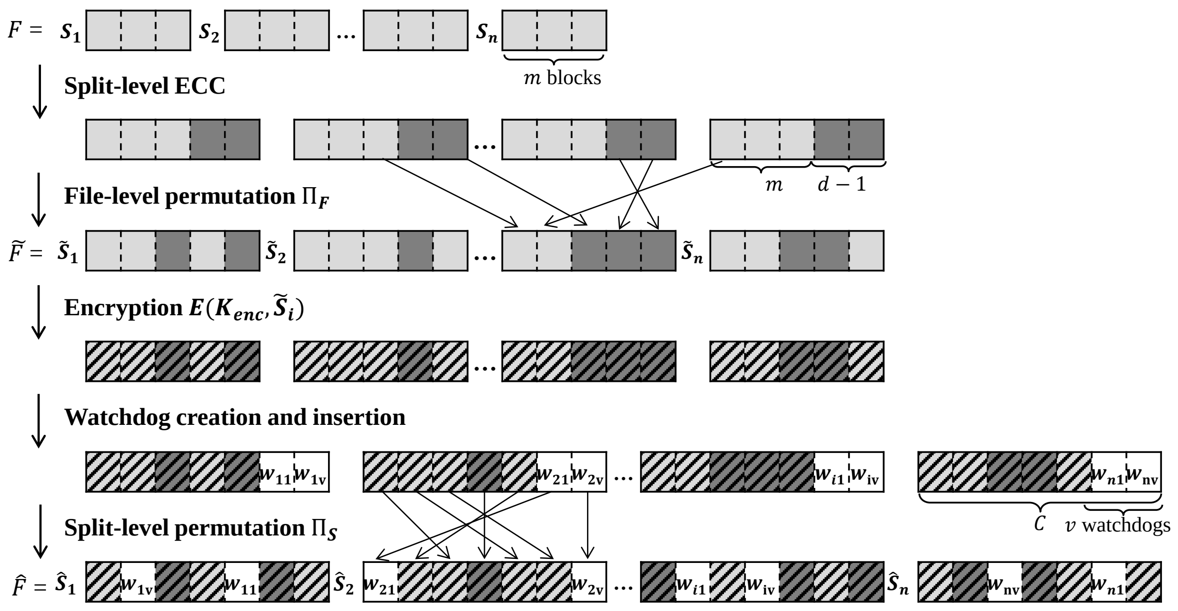
```
/* The data outsourced at the cloud is represented by a $(s,t)$ matrix $\mathcal{M}$ of elements in $\mathbb{Z}_2$ where $s \cdot t$ is the size of the data. */
/* A user is interested in retrieving an element $\epsilon \in \{0,1\}$ at position $(x,y)$ in $\mathcal{M}$. */
▼ Algorithm: $\vec{u} \leftarrow$ PIRquery$(x,y)$
# Executed by the user
  1. Select a group $\mathbb{Z}_p$ where $p$ is prime;
  2. Select a random generator $u \in \mathbb{Z}_p$ and $s$ random values $a_i \in \mathbb{Z}_p$;
  3. Compute $e_x = 1 + 2 \cdot a_x$ and $\forall\ i \neq x$, $e_i = 2 \cdot a_i$;
  4. Compute $u_i = u \cdot e_i \pmod p$ for all $1 \leq i \leq s$;
  5. Return $\vec{u} = (u_1, u_2, ..., u_s)$;

▼ Algorithm: $\vec{v} \leftarrow$ PIRprocess$(\vec{u}, \mathcal{M})$
# Run by the cloud server
  1. Compute the matrix-vector product $\vec{v} = (v_1, v_2, ..., v_t) = \mathcal{M}\vec{u}$;
  2. Return $\vec{v}$;

▼ Algorithm: $\epsilon \leftarrow$ PIRanalysis$(\vec{v}, y)$
# Executed by the user
  1. Compute the value $z_y = v_y \cdot u^{-1} \pmod p$;
  2. Compute the retrieved element $\epsilon = z_y \pmod 2$;
  3. Return $\epsilon$;
```

2.2.4 Description of the Entire Protocol

As stated in Section 2.2.2, $\mathcal{S}$tealthGuard consists of of the three phases: *Setup*, *Challenge* and *Verification*. Throughout the description of the protocol, the reader may refer to Table 2.1 that lists the symbols used in $\mathcal{S}$tealthGuard.

Table 2.1: List of notations in $\mathcal{S}$tealthGuard

Index	**Description**	**Range**
n	Number of splits S_i in file F	-
m	Number of blocks in a split S_i	-
D	Number of blocks in an encoded split $\tilde{S}_i$	-
v	Number of watchdogs in one split S_i	-
C	Number of blocks in a split $\hat{S}_i$ with watchdogs	-
i	Index of a split S_i	$[\![1, n]\!]$
k	Index of a block in split $\hat{S}_i$	$[\![1, C]\!]$
j	Index of a watchdog	$[\![1, v]\!]$
l	Size of a block (in bits)	-
L	Size of a split (in bits)	$L = m \cdot l$
p	Index of a block in $\tilde{F}$	$[\![1, n \cdot D]\!]$
q	Number of cloud matrices	-
r	Index of a cloud matrix	$[\![1, q]\!]$
(s,t)	Size of cloud matrices	-
(x,y)	Coordinates in a cloud matrix	$[\![1, s]\!] \times [\![1, t]\!]$



$F = S_1$ S_2 S_n
m blocks
Split-level ECC
m $d-1$
File-level permutation Π_F
$\tilde{F} = \tilde{S}_1$ $\tilde{S}_2$ $\tilde{S}_n$
Encryption $E(K_{enc}, \tilde{S}_i)$
Watchdog creation and insertion
w_{11} w_{1v} w_{21} w_{2v} w_{i1} w_{iv} w_{n1} w_{nv}
Split-level permutation Π_S
C v watchdogs
$\hat{F} = \hat{S}_1$ $\hat{S}_2$ $\hat{S}_n$

Figure 2.2: Setup phase in $\mathcal{S}$tealthGuard

2.2.4.1 Setup

The *Setup* phase prepares a verifiable version $\hat{F}$ of file F.

KeyGen: Data owner O first runs algorithm KeyGen to generate the master secret key K. In addition to this master key, this algorithm derives $n+3$ additional keys used for further operations in algorithm Encode. These keys are computed with dedicated cryptographic hash functions $H_{\mathsf{enc}}, H_{\mathsf{wdog}}, H_{\mathsf{perm}F}, H_{\mathsf{perm}S_i}$ that can be the Secure Hash Algorithm (SHA)-256 algorithm [131]:

- K_{enc}: This key serves as a data encryption key and is computed as $K_{\mathsf{enc}} = H_{\mathsf{enc}}(K)$, where H_{enc} is a cryptographic hash function;
- K_{wdog}: To generate the watchdogs, $\mathcal{S}$tealthGuard resorts to a PRF whose input key is $K_{\mathsf{wdog}} = H_{\mathsf{wdog}}(K)$;
- $K_{\mathsf{perm}F}$: Algorithm Encode requires to permute all blocks in a file using a PRP (see below), thus algorithm KeyGen computes $K_{\mathsf{perm}F} = H_{\mathsf{perm}F}(K)$;
- $K_{\mathsf{perm}S_i}$: For all $1 \leq i \leq n$, algorithm Encode inserts the generated watchdogs in random positions in split S_i, thus algorithm KeyGen generates the keys $K_{\mathsf{perm}S_i} = H_{\mathsf{perm}S_i}(K)$.

Encode: Once all the keying material is generated, data owner O runs algorithm Encode which first generates a pseudo-random and unique file identifier fid for file F, segments F into n splits $\{S_1, S_2, ..., S_n\}$ of equal size of L bits, and each split S_i into m blocks $\{b_{i1}, b_{i2}, ..., b_{im}\}$ of l bits[58], *i.e.* $L = m \cdot l$, and then processes F as described below.

1. **Error correcting**: The Error Correcting Code (ECC) assures the protection of the file against small corruptions. This step applies to each split S_i an ECC that operates over l-bit symbols. It uses an $[m+d-1, m, d]$-ECC. Each split is expanded with $d-1$ blocks of redundancy. Thus, the new splits are made of $D = m + d - 1$ blocks.
2. **File-level permutation**: $\mathcal{S}$tealthGuard applies a Pseudo-Random Permutation (PRP) to permute all the blocks in the file. This operation conceals within a

[58]If necessary, F will be padded to a multiple of L.

Figure 2.3: $\mathcal{S}$tealthGuard's Encode algorithm

```
▼ Algorithm: $(\mathsf{fid}, \hat{F}) \leftarrow \mathsf{Encode}(K, F)$
      Generate fid;
      Divide $F$ into $n$ equal-sized splits $S_i$ of $m$ blocks;
  1.  Error-correcting code
      For $1 \leq i \leq n$ do
          Apply ECC on split $S_i \in F$;
          # Split $S_i$ is expanded with $d-1$ blocks of redundancy
          # The size of ECC-encoded split $S_i$ is $D = m + d - 1$
      End
  2.  File block permutation
      Compute $K_{\mathsf{perm}F} = H_{\mathsf{perm}F}(K)$;
      For $1 \leq p \leq n \cdot D$ do
          # $\Pi_F : \{0,1\}^\kappa \times [\![1, n \cdot D]\!] \rightarrow [\![1, n \cdot D]\!]$ is a PRP
          Permute block at current position $p$ to the position $\Pi_F(K_{\mathsf{perm}F}, p)$
      End
      # Permuted file is denoted $\tilde{F}$
      Divide $\tilde{F}$ into $n$ equal-sized splits $\tilde{S}_i$ of $D$ blocks;
  3.  Encryption
      Compute $K_{\mathsf{enc}} = H_{\mathsf{enc}}(K)$;
      For $1 \leq i \leq n$ do
          # $E$ is an SSE scheme
          Encrypt split $\tilde{S}_i$ as $E(K_{\mathsf{enc}}, \tilde{S}_i)$
      End
  4.  Watchdog creation and insertion
      Compute $K_{\mathsf{wdog}} = H_{\mathsf{wdog}}(K)$;
      For $1 \leq i \leq n$ do
          # $\Phi : \{0,1\}^\kappa \times [\![1, n]\!] \times [\![1, v]\!] \times \{0,1\}^* \rightarrow \{0,1\}^l$ is a PRF
          For $1 \leq j \leq v$ do
              Generate watchdog $w_{ij}$ as $\Phi(K_{\mathsf{wdog}}, i, j, \mathsf{fid})$;
              Append watchdog $w_{ij}$ to split $i$;
          End
          # Split number $i$ contains $C = D + v$ blocks
          Compute $K_{\mathsf{perm}S_i} = H_{\mathsf{perm}S_i}(K)$;
          For $D \leq k \leq C$ do
              # $\Pi_S : \{0,1\}^\kappa \times [\![1, C]\!] \rightarrow [\![1, C]\!]$ is a PRP
              Permute block at current position $k$ to the position $\Pi_F(K_{\mathsf{perm}S_i}, k)$;
          End
          # Final split augmented with watchdogs is denoted $\hat{S}_i$
      End
  5.  Return $(\mathsf{fid}, \hat{F} = \{\hat{S}_1.\hat{S}_2, ..., \hat{S}_n\})$;
```

split the dependencies between the original data blocks and the corresponding redundancy blocks. We explain at the end of the description of the *Setup* phase the importance of such a permutation.

Let $\Pi_F : \{0,1\}^\kappa \times [\![1, n \cdot D]\!] \rightarrow [\![1, n \cdot D]\!]$ be a PRP: for each $p \in [\![1, n \cdot D]\!]$, the block at current position p will be at position $\Pi_F(K_{\mathsf{perm}F}, p)$ in the permuted file that we denote $\tilde{F}$. $\tilde{F}$ is then divided into n splits $\{\tilde{S}_1, \tilde{S}_2, ..., \tilde{S}_n\}$ of equal size D.

3. **Encryption**: $\mathcal{S}$tealthGuard uses a Semantically-Secure Encryption (SSE) E that operates over l-bit blocks[59] to encrypt the data. Encryption E is applied to each block of $\tilde{F}$ using K_{enc}.

4. **Watchdog creation**: For each split of encrypted blocks, v l-bit watchdogs are generated using a Pseudo-Random Function (PRF) $\Phi : \{0,1\}^\kappa \times [\![1, n]\!] \times [\![1, v]\!] \times \{0,1\}^* \rightarrow \{0,1\}^l$. Hence, for $j \in [\![1, v]\!]$, $w_{ij} = \Phi(K_{\mathsf{wdog}}, i, j, \mathsf{fid})$. Since the watchdogs are pseudo-randomly generated and the blocks in the split are encrypted, a malicious cloud cannot distinguish watchdogs from data blocks.

[59]Practically, l will be 128 or 256 bits.

5. **Watchdog insertion**: The v watchdogs are appended to each split. Let $C = D+v$ be the size (in blocks) of the new splits. A split-level PRP $\Pi_S : \{0,1\}^\kappa \times [\![1, C]\!] \to [\![1, C]\!]$ is then applied to the blocks within the same split in order to randomize the location of the watchdogs: for S_i, such that $i \in [\![1, n]\!]$, the block at current position k will be at position $\Pi_S(K_{\mathsf{perm}S_i}, k)$ in the permuted split. We denote $\hat{S}_i$, $i \in [\![1, n]\!]$, the permuted split and $\hat{b}_{ik}$, $k \in [\![1, C]\!]$ its blocks.

Encode file $\hat{F}$ together with fid. Data owner $\mathcal{O}$ uploads $(\mathsf{fid}, \hat{F} = \{\hat{S}_1, \hat{S}_2, ..., \hat{S}_n\})$ to cloud server $\mathcal{S}$. We stress the fact that the only information stored at data owner $\mathcal{O}$ is master secret key K. The operations of algorithm Encode are depicted in Figure 2.2 and Figure 2.3.

Discussion on the block permutation step (step 2). As mentioned earlier, this operation conceals within a split the dependencies between the original data blocks and the corresponding redundancy blocks. Without this permutation, the corresponding redundancy blocks are just appended to the split, since Reed-Solomon codes are systematic codes. Hence, without such a permutation, a malicious cloud could for instance delete all the redundancy blocks located at the end of a split and a single data block from this split and thus render the file irretrievable. Such an attack would not easily be detected since the malicious server could still be able to respond with valid proofs to a given POR query if the watchdogs are not impacted by this attack or if the query targets other splits in the file. The permutation prevents this attack since data blocks and redundancy blocks are mixed up among all splits.

2.2.4.2 Challenge

Once data is uploaded, verifier $\mathcal{V}$ wants to check the retrievability of file F stored at cloud server $\mathcal{S}$. Hence, $\mathcal{V}$ issues search queries for randomly selected watchdogs. Such a query applies to a single watchdog in a particular split. $\mathcal{S}$ processes these queries without knowing what the values of the watchdogs are and where they are located in the targeted split. We propose WDSearch, a Privacy-Preserving Word Search (PPWS) solution derived from Prism [37] which is based on a Private Information Retrieval (PIR) protocol. Our proposal is a simpler version of Prism and improves its performance in the particular context of $\mathcal{S}$tealthGuard.

To process a query in the framework of WDSearch, cloud server $\mathcal{S}$ constructs q (s,t)-binary matrices such that $s \cdot t = C$. Each element in the matrices is filled with the witness (a one-bit information on the existence of the watchdog) of the corresponding block in the split. Based on the PIR query sent by the verifier, the server retrieves in the matrices the witnesses corresponding to the requested watchdogs. We insist on the fact that WDSearch is not a PIR solution: the server does not retrieve the watchdog itself but only the witness.

WDSearch, depicted in Figure 2.5, consists of two steps, corresponding to the challenge-response protocol inherent to any POR scheme: Challenge in which verifier $\mathcal{V}$ prepares the challenge, *i.e.* the POR query to be sent to cloud server $\mathcal{S}$ and Response where cloud server $\mathcal{S}$ processes the received POR query to issue a POR response to be sent to verifier $\mathcal{V}$.

Challenge: Verifier $\mathcal{V}$ executes algorithm Challenge to generate a challenge chal that is transmitted to cloud server $\mathcal{S}$. Challenge takes as input master key K and file identifier fid and it is executed in three phases:

1. It randomly selects a split index i and a watchdog index j ($i \in [\![1, n]\!]$ and $j \in [\![1, v]\!]$), and computes the position pos_j of the watchdogs w_{ij} in the split $\hat{S}_i$ by applying PRP Π_S $\mathsf{pos}_j = \Pi_S(K_{\mathsf{perm}S_i}, D+j)$. Then, Challenge maps the position pos_j to its unique position (x_j, y_j) in an (s,t)-matrix (*i.e.* the ones created by cloud server $\mathcal{S}$ during the Response phase):

$$x_j = \lceil \frac{\mathsf{pos}_j}{t} \rceil \qquad\qquad y_j = \mathsf{pos}_j - \lceil \frac{\mathsf{pos}_j}{t} \rceil \times t + t$$

2. Given (x_j, y_j), Challenge computes a PIR query $\vec{\mu} = \mathsf{PIRquery}(x_j, y_j)$, to retrieve the witness at position (x_j, y_j) in an (s, t)-matrix.
3. Challenge generates a nonce R. This nonce will be used by cloud server $\mathcal{S}$ when creating matrices to guarantee the freshness of its responses. Then Challenge outputs challenge $\mathsf{chal} = (\vec{\mu}, R, i, j)$.

Eventually, $\mathcal{V}$ sends challenge chal and file identifier fid to $\mathcal{S}$.

Response: Upon receiving challenge $\mathsf{chal} = (\vec{\mu}, R, i, j)$ and file identifier fid, $\mathcal{S}$ runs ProofGen to process the POR query in two phases:

1. $\mathcal{S}$ creates q binary matrices[60] of size (s, t). For each block $\hat{b}_{ik}$ in split $\hat{S}_i$, $\mathcal{S}$ computes $h_{ik} = H(\hat{b}_{ik}, R)$, where $k \in [\![1, C]\!]$. Here, H denotes a cryptographic hash function. Thanks to R, $\mathcal{S}$ cannot drop the block in order to only store the hash and respond to the query using that hash.
   Let $h_{ik}|_q$ be the first q bits of h_{ik}. For $r \in [\![1, q]\!]$, let $\mathcal{M}_r$ be one of the matrices created by cloud server $\mathcal{S}$. $\mathcal{S}$ fills the r^{th} matrix with the r^{th} bit of $h_{ik}|_q$ as algorithm FillMatrix in Figure 2.4 shows. It should be noted that according to the assignment process described in Figure 2.4, the witness at position (x_j, y_j) in $\mathcal{M}_r$ is associated with watchdog w_{ij}: it is the r^{th} bit of $H(w_{ij}, R)$.
2. Once all the q matrices are filled, $\mathcal{S}$ processes PIR query $\vec{\mu}$ by executing PIRprocess (cf. Figure 2.1) that retrieves one row from each matrix $\mathcal{M}_r$, $r \in [\![1, q]\!]$. We denote $\vec{\sigma}_r$ the output of PIRprocess for matrix $\mathcal{M}_r$.
3. Algorithm ProofGen outputs $\mathcal{P}$, *i.e.* the proof of retrievability which consists in the set $\mathcal{P} = \{\vec{\sigma}_1, \vec{\sigma}_2 ..., \vec{\sigma}_q\}$. In a nutshell, vectors $\vec{\sigma}_r$ are the rows of coordinate x_j and size t, retrieved in a privacy-preserving manner from matrix $\mathcal{M}_r$.
   $\mathcal{S}$ sends proof $\mathcal{P}$ to verifier $\mathcal{V}$.

Figure 2.4: StealthGuard's FillMatrix procedure

```
▼ Algorithm: $\mathcal{M}_r \leftarrow \mathsf{FillMatrix}(\mathcal{M}_r, \{h_{ik}\}_{1 \le k \le C})$
  1. k=1;
  2. For $1 \le x \le s$ do
        For $1 \le y \le t$ do
            $\mathcal{M}_r[x, y] \leftarrow r^{th}$ bit of $h_{ik}$;
            $k = k + 1$;
        End
     End
  3. Return $\mathcal{M}_r$;
```

(a) FillMatrix

$C = 6$ is the number of blocks in one split $\hat{S}_i$
$s = 2$ is the number of rows in a matrix
$t = 3$ is the number of columns in a matrix
$q = 4$ is the number of matrices

Matrix elements are indexed as $\begin{smallmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{smallmatrix}$
Hashes of blocks truncated to the first $q = 4$ bits:
$h_{i1} = \underline{1}\,\underline{0}\,\underline{1}\,\underline{0}$; $h_{i2} = 1100$; $h_{i3} = 0001$;
$h_{i4} = 1001$; $h_{i5} = 1111$; $h_{i6} = 1101$

$$\underset{\mathcal{M}_1}{\begin{bmatrix} \underline{1} & 1 & 0 \\ 1 & 1 & 1 \end{bmatrix}} \quad \underset{\mathcal{M}_2}{\begin{bmatrix} \underline{0} & 1 & 0 \\ 0 & 1 & 1 \end{bmatrix}} \quad \underset{\mathcal{M}_3}{\begin{bmatrix} \underline{1} & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix}} \quad \underset{\mathcal{M}_4}{\begin{bmatrix} \underline{0} & 0 & 1 \\ 1 & 1 & 1 \end{bmatrix}}$$

(b) FillMatrix toy example

2.2.4.3 Verification

The last phase of StealthGuard consists in the *Verification* phase in which verifier $\mathcal{V}$ runs algorithm Verify to analyze proof $\mathcal{P}$ she received from cloud server $\mathcal{S}$.

Algorithm Verify takes as inputs master key K, proof $\mathcal{P}$, challenge chal (from which it extracts split index i and watchdog index j) and file identifier fid, and outputs bit $b = 1$ if proof $\mathcal{P}$ is valid or $b = 0$ otherwise.

As shown in Figure 2.6, algorithm Verify operates in three phases:

1. It first processes the q vectors $\vec{\sigma}_r$, $1 \le r \le q$, contained in proof $\mathcal{P}$ using procedure PIRanalysis (depicted in Figure 2.1) with inputs: vector $\vec{\sigma}_r$ and coordinate y_j. For

[60] q defines a tunable parameter of our system, that quantifies the size of the witness retrieved in StealthGuard. Typically, we select $|q| = 80$ bits. The value of q is discussed in Section 2.3.2.

Figure 2.5: $\mathcal{S}$tealthGuard's WDSearch algorithm

```
▼ Algorithm: $\mathsf{chal} \leftarrow \mathsf{Challenge}(K, \mathsf{fid})$
# Executed by verifier $\mathcal{V}$
    1. Pick a random split index $i$, watchdog index $j$;
       Compute $\mathsf{pos}_j = \Pi_S(K_{\mathsf{perm}S_i}, D + j)$;
       Determine $x_j = \lceil \frac{\mathsf{pos}_j}{t} \rceil$ and $y_j = \mathsf{pos}_j - \lceil \frac{\mathsf{pos}_j}{t} \rceil \times t + t$;
    2. Generate query $\vec{\mu} = \mathsf{PIRquery}(x_j, y_j)$;
    3. Pick a random number $R$;
    4. Return $\mathsf{chal} = (\vec{\mu}, R, i, j)$;

▼ Algorithm: $\mathcal{P} \leftarrow \mathsf{ProofGen}(\mathsf{fid}, \mathsf{chal})$
# Run by cloud server $\mathcal{S}$
       Parse $\mathsf{chal} = (\vec{\mu}, R, i, j)$
    1. Initialize $q$ binary matrices $\mathcal{M}_r$ of size $(s, t)$;
       For $1 \leq k \leq C$ do
           # Loop over the blocks in split $\hat{S}_i$
           Compute $h_{ik} = H(\hat{b}_{ik}, R)$;
       End
       For $1 \leq r \leq q$ do
           # Loop over the matrix indices
           Execute procedure $\mathcal{M}_r = \mathsf{FillMatrix}(\mathcal{M}_r)$   # cf. Figure 2.4;
       End
    2. For $1 \leq r \leq q$ do
           Compute $\vec{\sigma}_r = \mathsf{PIRprocess}(\vec{\mu}, \mathcal{M}_r)$   # cf. Figure 2.1;
       End
    3. Return $\mathcal{P} = (\vec{\sigma}_1, \vec{\sigma}_2, ..., \vec{\sigma}_q)$;
```

each $\vec{\sigma}_r$, PIRanalysis outputs the queried bit ϵ_r from matrix $\mathcal{M}_r$ at position (x_j, y_j), for $1 \leq r \leq q$. Let h denote the bit string $\epsilon_1\epsilon_2...\epsilon_r...\epsilon_q$.

2. We recall that verifier $\mathcal{V}$ queried watchdog w_{ij} in split $\hat{S}_i$ and that by having access to master key K, verifier $\mathcal{V}$ can recompute the value of $w_{ij} = \Phi(K_{\mathsf{wdog}}, i, j, \mathsf{fid})$ and its position in split $\hat{S}_i$, $\mathsf{pos}_j = \Pi_S(K_{\mathsf{perm}S_i}, D + j)$. Thereafter, algorithm Verify computes the hash of the watchdog $h_{i,\mathsf{pos}_j} = H(w_{ij}, R)$, with the same nonce R chosen in the WDSearch phase and considers the q first bits of h_{i,pos_j}.
3. Based on the value of h and h_{i,pos_j}, algorithm Verify checks whether $h = h_{i,\mathsf{pos}_j}|_q$. If it is the case, then verifier $\mathcal{V}$ returns 1 and judges that with overwhelming probability the watchdog is correct in the split. Otherwise, verifier $\mathcal{V}$ outputs 0 and it interprets the invalid proof as the occurrence of an attack.

Figure 2.6: $\mathcal{S}$tealthGuard's Verify algorithm

```
▼ Algorithm: $b \leftarrow \mathsf{Verify}(K, \mathsf{fid}, \mathsf{chal}, \mathcal{P})$
       Parse $\mathcal{P} = (\vec{\sigma}_1, \vec{\sigma}_2, ..., \vec{\sigma}_q)$;
    1. For $1 \leq r \leq q$ do
           Evaluate $\epsilon_r = \mathsf{PIRanalysis}(\vec{\sigma}_r, y_j)$;
       End
       # $h = \epsilon_1\epsilon_2..\epsilon_q$
    2. Compute $w_{ij} = \Phi(K_{\mathsf{wdog}}, i, j, \mathsf{fid})$;
       Compute $\mathsf{pos}_j = \Pi_S(K_{\mathsf{perm}S_i}, D + j)$;
       Compute $h_{i,\mathsf{pos}_j} = H(w_{ij}, R)$;
    3. Check whether $h = h_{i,\mathsf{pos}_j}|_q$;
       If this check fails then return $b = 0$ else return $b = 1$;
```

As mentioned in Section 2.2.2, in order to acknowledge the retrievability of file F, verifier $\mathcal{V}$ needs to initiate at least γ POR queries[61] from randomly selected splits in order to either ascertain that file F is retrievable or detect a corruption attack: if verifier $\mathcal{V}$ receives γ valid

[61]The value of γ is determined in Section 2.3.

POR responses, then she can conclude that $\mathcal{S}$ stores a retrievable version of the file, otherwise, she concludes that $\mathcal{S}$ has corrupted part of the file.

2.3 Security Analysis of our Protocol

In this section, we state and prove the two security theorems of completeness and soundness satisfied by $\mathcal{S}$tealthGuard.

2.3.1 Completeness

Theorem 1 (Completeness). *$\mathcal{S}$tealthGuard is complete.*

Proof of Theorem 1. Without loss of generality, we assume that the honest verifier $\mathcal{V}$ runs a $\mathcal{S}$tealthGuard instance for a file F. To this end, $\mathcal{V}$ sends a challenge $\mathsf{chal} = (\vec{\mu}, R, i, j)$ for watchdog w_{ij}, and file identifier fid of F. Upon receiving challenge chal and file identifier fid, cloud server $\mathcal{S}$ generates a proof of retrievability $\mathcal{P}$ for file F.

According to the description of $\mathcal{S}$tealthGuard, the verification of POR $\mathcal{P}$ consists in first retrieving the first q bits of a hash h_{i,pos_j}, then verifying whether $h_{i,\mathsf{pos}_j}|q$ corresponds to the first q bits of the hash $H(w_{ij}, R)$. Since $\mathcal{S}$ is honest, then this entails that it stores w_{ij} and therewith, can always compute $h_{i,\mathsf{pos}_j} = H(w_{ij}, R)$.

Consequently, $\mathsf{Verify}(K, \mathsf{fid}, \mathsf{chal}, \mathcal{P}) = 1$.

2.3.2 Soundness

We recall that we consider the scenario where data owner $\mathcal{O}$ outsources the storage of her file $F = \{S_1, S_2, ..., S_n\}$ to cloud server $\mathcal{S}$. In the following, we enunciate the soundness theorem for $\mathcal{S}$tealthGuard.

Theorem 2 (Soundness). *Let κ be the security parameter of $\mathcal{S}$tealthGuard and let $\rho = \frac{d}{2D}$ denote the error rate of the ECC, δ be the probability that an adversary $\mathcal{A}$ wins the POR soundness game and γ be the number of valid proofs of retrievability provided by adversary $\mathcal{A}$ in a row.*

$\mathcal{S}$tealthGuard is (δ, γ)-sound in the Random Oracle Model (ROM)*, if $\delta > \delta_{\mathsf{neg}}$ and $\gamma \geq \gamma_{\mathsf{neg}}$, where*

$$\begin{aligned} \delta_{\mathsf{neg}} &= \frac{1}{2^\kappa} \\ \gamma_{\mathsf{neg}} &= \left\lceil \frac{\ln(2)\kappa}{\rho_{\mathsf{neg}}} \right\rceil \\ \left(1 - \frac{\rho}{\rho_{\mathsf{neg}}}\right)^2 \rho_{\mathsf{neg}} &= \frac{3\ln(2)\kappa}{D} \\ \rho_{\mathsf{neg}} &\leq \rho. \end{aligned}$$

Accordingly, if $\gamma \geq \gamma_{\mathsf{neg}}$, then there exists an extractor $\mathcal{E}$ that recovers a file F with a probability $1 - \frac{n}{2^\kappa}$, such that n is the number of splits in F, by interacting with an adversary $\mathcal{A}$ against $\mathcal{S}$tealthGuard who succeeds in the soundness game with a probability $\delta > \frac{1}{2^\kappa}$.

Interpretation. If $\mathcal{V}$ issues $\gamma \geq \gamma_{\mathsf{neg}}$ POR queries for some file F to which $\mathcal{S}$ responds correctly, then $\mathcal{V}$ can declare F as retrievable with probability $1 - \frac{n}{2^\kappa}$. In addition, since an instance of $\mathcal{S}$tealthGuard's protocol executed for F consists c obliviously fetching a witness for a watchdog from the encoding $\hat{F}$ of that file, $\mathcal{O}$ must insert at least γ_{neg} watchdogs in file F, to ensure a security level of $\frac{1}{2^\kappa}$. That is, if file F comprises n splits, then $nv \geq \gamma_{\mathsf{neg}}$ where v is the number of watchdogs per split.

Proof of Theorem 2. We assume there is an adversary $\mathcal{A}$ that corrupts on average ρ_{adv} fraction of the outsourced file, and succeeds in the soundness game, depicted in Section 2.1.2, with some probability δ. In the following, we show that if $\delta > \delta_{\mathsf{neg}} = \frac{1}{2^\kappa}$, then there exists an extractor algorithm $\mathcal{E}$ that retrieves the challenge file $F^* = \{S_1^*, S_2^*, ..., S_n^*\}$ by interacting with adversary $\mathcal{A}$ and by controlling a *random oracle* $\mathcal{H}$.

Proof overview. For ease of exposition, the proof of Theorem 2 is broken down into four consecutive steps:

1- Computation of the probability δ of success in the soundness game: This step quantifies probability δ according to which adversary $\mathcal{A}$ succeeds in the soundness game when it corrupts on *average* ρ_{adv} fraction of outsourced file $\hat{F}^*$. We find that $\delta \simeq (1-\rho_{\mathsf{adv}})^\gamma$.

2- Computation of corruption threshold ρ_{neg} above which file F^* is irretrievable: In this step, we determine threshold ρ_{neg} for the probability ρ_{adv} of corruption by adversary $\mathcal{A}$ such that:

- If $\rho_{\mathsf{adv}} \geq \rho_{\mathsf{neg}}$, then the file F^* is irretrievable by extractor $\mathcal{E}$.
- If $\rho_{\mathsf{adv}} < \rho_{\mathsf{neg}}$, then the file F^* is retrievable by extractor $\mathcal{E}$ with an overwhelming probability.

3- Computation of γ_{neg} above which F^* is said retrievable when $\mathcal{A}$ wins the game: Here we derive a bound γ_{neg} for value γ, such that if $\gamma \geq \gamma_{\mathsf{neg}}$ and:

- If $\rho_{\mathsf{adv}} \geq \rho_{\mathsf{neg}}$, then $\delta = (1-\rho_{\mathsf{adv}})^{\gamma_{\mathsf{neg}}} \leq \delta_{\mathsf{neg}} = \frac{1}{2^\kappa}$. This ensures that if adversary $\mathcal{A}$ corrupts more than ρ_{neg} fraction of the file F^*, then $\mathcal{A}$ will be detected with an overwhelming probability (adversary $\mathcal{A}$ wins the soundness game with a negligible probability).
- If $\rho_{\mathsf{adv}} < \rho_{\mathsf{neg}}$, then $\delta = (1-\rho_{\mathsf{adv}})^{\gamma_{\mathsf{neg}}} > \delta_{\mathsf{neg}}$, however, the file F^* is still retrievable.

4- Construction of a file extractor $\mathcal{E}$: File extraction can be performed whenever adversary $\mathcal{A}$ succeeds in the soundness game with a probability $\delta > \delta_{\mathsf{neg}}$. For that purpose, extractor $\mathcal{E}$ controls a random oracle $\mathcal{H}$ that simulates and keeps track of the output of the hash function H which is used in $\mathcal{S}$tealthGuard to generate and verify the proofs of retrievability (cf. Section 2.2.4).

Proof details. In the soundness game depicted in Section 2.1.2, $\mathcal{A}$ enters the **learning** phase and makes a polynomial number of *encode* queries to oracle $\mathcal{O}_{\mathsf{Encode}}$, *challenge* queries to oracle $\mathcal{O}_{\mathsf{Challenge}}$ and *verify* queries to oracle $\mathcal{O}_{\mathsf{Verify}}$. At the end of this phase, $\mathcal{A}$ selects a file identifier fid^* from the ones output by oracle $\mathcal{O}_{\mathsf{Encode}}$.

After the **learning** phase, $\mathcal{A}$ gets into the **challenge** phase of the soundness game and provides γ proofs of retrievability for the challenged file F^* to oracle $\mathcal{O}_{\mathsf{Verify}}$. Adversary $\mathcal{A}$ succeeds in the soundness game if all the proofs given to oracle $\mathcal{O}_{\mathsf{Verify}}$ are valid. Without loss of generality, we assume that fid^* labels file $F^* = \{S_1^*, S_2^*, ..., S_n^*\}$ and that oracle

$\mathcal{O}_{\mathsf{Encode}}$ returns the pair $(\mathsf{fid}^*, \hat{F}^*)$ such that $\hat{F}^* = \{\hat{S}_1^*, \hat{S}_2^*, ..., \hat{S}_n^*\}$. In the following we detail the four parts of the proofs.

1- Computation of the probability δ of success in the soundness game.
We determine the probability δ as a function of adversarial corruption fraction ρ_{adv}. For this purpose, we model the corruption pattern: for each split $\hat{S}_i^*$, let X_{ik} denote the random variable that corresponds to the event in which adversary $\mathcal{A}$ corrupts the k^{th} block of split $\hat{S}_i^*$. Namely,

$$X_{ik} = \begin{cases} 1 & \text{if } \mathcal{A} \text{ corrupts the } k^{th} \text{ block of the split } \hat{S}_i^* \\ 0 & \text{otherwise.} \end{cases}$$

We assume for sake of simplicity that for all $1 \leq i \leq n$ and all $1 \leq k \leq C$, random variable X_{ik} follows a *Bernouilli process* of parameter ρ_{adv}, *i.e.* $Pr(X_{ik} = 1) = \rho_{\mathsf{adv}}$ and $Pr(X_{ik} = 0) = 1 - \rho_{\mathsf{adv}}$. This implies that random variables X_{ik} are independent identical distributed binary random variables. In other words, the probability that a block in split $\hat{S}_i^*$ is corrupted by adversary $\mathcal{A}$ is *the same for all blocks* in file $\hat{F}^*$. This model is valid in the case of $\mathcal{S}$tealthGuard: Indeed, the use of a secure PRP and a SSE in algorithm Encode (cf. Section 2.2.4.1) ensures that adversary $\mathcal{A}$ sees block values as uniformly distributed.

It follows that adversary $\mathcal{A}$ succeeds in providing a valid proof of retrievability for some challenge chal_j^* $(1 \leq j \leq \gamma)$ generated by oracle $\mathcal{O}_{\mathsf{Challenge}}$ (cf. Algorithm 2) according to the following two cases:

- The watchdog associated with challenge chal_j^* (we assume it is located at position k in split $\hat{S}_i^*$) is not affected by the adversarial corruption. This event occurs with probability $Pr(X_{ik} = 0) = 1 - \rho_{\mathsf{adv}}$.
- $\mathcal{A}$ does corrupt this watchdog but she is still able to provide the q-bit witness associated with that watchdog (cf. Figure 2.5 and Figure 2.6) as expected by oracle $\mathcal{O}_{\mathsf{Verify}}$. The probability of such an event is $Pr(X_{ik} = 1) \times \left(\frac{1}{2}\right)^q = \frac{\rho_{\mathsf{adv}}}{2^q}$.

Let $P^{\mathcal{A}}_{(\mathsf{Success},j)}$ denote the probability that adversary $\mathcal{A}$ succeeds in providing a valid proof of retrievability for challenge chal_j^*. Accordingly, $P^{\mathcal{A}}_{(\mathsf{Success},j)} = 1 - \rho_{\mathsf{adv}} + \frac{\rho_{\mathsf{adv}}}{2^q}$.

As mentioned in the soundness definition enunciated in Section 2.1.2, adversary $\mathcal{A}$ succeeds in the **challenge** phase of the soundness game if she succeeds in supplying oracle $\mathcal{O}_{\mathsf{Verify}}$ with γ valid proofs of retrievability. Therefore, the probability that adversary $\mathcal{A}$ succeeds in the soundness game is:

$$\delta = \prod_{j=1}^{\gamma} P^{\mathcal{A}}_{(\mathsf{Success},j)} = (1 - \rho_{\mathsf{adv}})^{\gamma} + \underbrace{\frac{\gamma \rho_{\mathsf{adv}} (1 - \rho_{\mathsf{adv}})^{\gamma - 1}}{2^q} + o\left(\frac{1}{2^q}\right)}_{\text{denoted } \zeta}.$$

We recall that q corresponds to the number of bits that a verifier has to retrieve from the cloud server in $\mathcal{S}$tealthGuard. Hence, we note that if q is large enough, for example $q = 80$ bits, then ζ is negligible. Therefore, to simplify, we assume $q \geq 80$ and $\delta \simeq (1 - \rho_{\mathsf{adv}})^{\gamma}$. This result can be interpreted as follows: to win the soundness game, adversary $\mathcal{A}$ has to send the q-bit witness that corresponds to the watchdog targeted by the challenge generated by oracle $\mathcal{O}_{\mathsf{Challenge}}$. The bigger q, the harder for $\mathcal{A}$ to guess the exact sequence of q bits. Thus, $\mathcal{A}$ may not corrupt the watchdog to provide valid proofs of retrievability, which yields probability δ to reach $(1 - \rho_{\mathsf{adv}})^{\gamma}$ (*i.e.* the probability that γ watchdogs are not corrupted by adversary $\mathcal{A}$).

2- Computation of corruption threshold ρ_{neg} above which file F^* is irretrievable. Now that we have quantified probability δ according to which adversary $\mathcal{A}$ wins the soundness game, we determine the threshold ρ_{neg} for the corruption probability ρ_{adv}. As specified above, if $\rho_{\mathsf{adv}} \leq \rho_{\mathsf{neg}}$ then the challenged file F^* is retrievable by file extractor $\mathcal{E}$ with overwhelming probability.

Before computing this threshold, we introduce a simplified *Chernoff bound* lemma that we will use later to bound the probability that a split $\hat{S}_i^*$ is corrupted in a manner that prevents file extractor $\mathcal{E}$ from retrieving S_i^*, and therewith prevents $\mathcal{E}$ from retrieving file F^*.

Lemma 1 (Simplified Chernoff Bound). *Let $X_1, X_2, ..., X_D$ be independent random variables that follow a Bernouilli process of parameter π, i.e $Pr(X_i = 1) = \pi$ for $1 \leq i \leq C$. Then for $X = \sum_{i=1}^{D} X_i$, $\mu = E[X] = D\pi$, and for any $\alpha \in]0,1]$, we have*

$$Pr(X > (1+\alpha)\mu) \leq e^{-\frac{\alpha^2\mu}{3}}$$
$$\textit{and } Pr(X < (1-\alpha)\mu) \leq e^{-\frac{\alpha^2\mu}{2}}.$$

We leverage Lemma 1 in the following lines. File extractor $\mathcal{E}$ fails at recovering file F^* if there is a split $\hat{S}_i^*$ of the encoding F^* damaged by more than $\frac{d}{2} = \rho D$ errors (by definition of an Error Correcting Code (ECC)). This assertion can be interpreted in terms of probability: Let $P^{\mathcal{E}}_{(\mathsf{Fail},j)}$ denote the probability that split $\hat{S}_i^*$ is damaged with more than ρD errors. In particular, it expresses the probability that extractor $\mathcal{E}$ fails in recovering split $\hat{S}_i^*$. Thus $P^{\mathcal{E}}_{(\mathsf{Fail},j)} = Pr(\sum_{k=1}^{D} X_{ik} > \rho D)$, where $\sum_{k=1}^{D} X_{ik}$ quantifies the adversarial corruptions in split $\hat{S}_i^*$, that is the number of errors in that split. By applying Lemma 1, we find the bound $P^{\mathcal{E}}_{(\mathsf{Fail},j)} \leq e^{-\frac{\alpha^2\mu}{3}}$ where $\mu = E[\sum_{k=1}^{D} X_{ik}] = \rho_{\mathsf{adv}} D$ (indeed, ρ_{adv} is the parameter of the Bernoulli process followed by the random variables X_{ik}) and $\alpha = \frac{\rho}{\rho_{\mathsf{adv}}} - 1$. Written in another way, this bound corresponds to:

$$P^{\mathcal{E}}_{(\mathsf{Fail},j)} \leq e^{-\frac{\rho_{\mathsf{adv}} D}{3}(1-\frac{\rho}{\rho_{\mathsf{adv}}})^2}.$$

We notice that probability $P^{\mathcal{E}}_{(\mathsf{Fail},j)}$ is negligible, i.e $P^{\mathcal{E}}_{(\mathsf{Fail},j)} \leq \frac{1}{2^\kappa}$ where κ is the security parameter in $\mathcal{S}$tealthGuard, for any ρ_{adv} that satisfies the inequality $(1 - \frac{\rho}{\rho_{\mathsf{adv}}})^2 \rho_{\mathsf{adv}} \geq \frac{3\ln(2)\kappa}{D}$. In particular, $P^{\mathcal{E}}_{(\mathsf{Fail},j)}$ is negligible for any $\rho_{\mathsf{adv}} \leq \rho_{\mathsf{neg}}$ where ρ_{neg} is defined as

$$(1 - \frac{\rho}{\rho_{\mathsf{neg}}})^2 \rho_{\mathsf{neg}} = \frac{3\ln(2)\kappa}{D}$$
$$\text{and } \rho_{\mathsf{neg}} < \rho.$$

3- Computation of γ_{neg} above which F^* is said retrievable when $\mathcal{A}$ wins the game. In order to ensure that file F^* is retrievable whenever adversary $\mathcal{A}$ succeeds in the soundness game, γ, the number of challenges that have to be issued and sent to adversary $\mathcal{A}$, must pass a threshold value denoted γ_{neg}, such that if adversary $\mathcal{A}$ corrupts more than ρ_{neg} fraction of the encoded file $\hat{F}^*$, she will be detected by extractor $\mathcal{E}$ with an overwhelming probability. In other words, we want to assure that if $\gamma \geq \gamma_{\mathsf{neg}}$ and probability of corruption ρ_{adv} is larger than probability ρ_{neg} (above which file F^* is irretrievable by extractor $\mathcal{E}$), then probability δ that adversary $\mathcal{A}$ succeeds in the soundness game is negligible. Precisely,

$$\delta = (1 - \rho_{\mathsf{adv}})^\gamma \text{ as shown in step 1 of this proof}$$

$$\delta \leq (1-\rho_{\mathsf{adv}})^{\gamma_{\mathsf{neg}}} \text{ since } \gamma \geq \gamma_{\mathsf{neg}}.$$

We want to assure that:

$$\delta \leq \delta_{\mathsf{neg}} = \frac{1}{2^{\kappa}}.$$

Therefore, from the last two lines, we deduce that γ_{neg} should be greater than $\frac{-\ln(2)\kappa}{\ln(1-\rho_{\mathsf{adv}})}$. To fulfill the above condition whenever $\rho_{\mathsf{adv}} \geq \rho_{\mathsf{neg}}$, it suffices to have $\gamma_{\mathsf{neg}} \geq \frac{-\ln(2)\kappa}{\ln(1-\rho_{\mathsf{neq}})} \geq \frac{\ln(2)\kappa}{\rho_{\mathsf{neg}}}$. As a result, we can define γ_{neg} as:

$$\gamma_{\mathsf{neg}} = \left\lceil \frac{\ln(2)\kappa}{\rho_{\mathsf{neg}}} \right\rceil.$$

Summary of the findings. Before detailing the last step of the proof, we give here a brief summary of the above findings. If $\gamma \geq \gamma_{\mathsf{neg}}$:

- If $\rho_{\mathsf{adv}} \geq \rho_{\mathsf{neg}}$, that is, when adversary $\mathcal{A}$ corrupts a fraction of file F^* that is larger than the limit for which file extractor $\mathcal{E}$ can recover the file, then the probability that adversary $\mathcal{A}$ wins the soundness game is $\delta \leq \delta_{\mathsf{neg}} = \frac{1}{2^{\kappa}}$, which is negligible.
- If $\rho_{\mathsf{adv}} < \rho_{\mathsf{neg}}$, then the probability that adversary $\mathcal{A}$ succeeds in the soundness game is $\delta > \delta_{\mathsf{neg}}$. Nevertheless, file F^* is retrievable, as shown in part 4 of the proof, with a probability larger than $1 - \frac{n}{2^{\kappa}}$, where n is the number of splits in file F^*.

4- Construction of a file extractor $\mathcal{E}$. In this last part of the proof of Theorem 2, we show that there exists a file extractor $\mathcal{E}$ that can recover challenge file F^* whenever adversary $\mathcal{A}$ succeeds in the soundness game with probability $\delta > \frac{1}{2^{\kappa}}$.

For this purpose, we require that such an extractor simulates the output of hash function H by controlling a random oracle $\mathcal{H}$, as depicted in the following lines. We recall that this hash function H was used in algorithm $\mathsf{WDSearch}$ to construct and verify the proofs of retrievability (cf. Figure 2.5).

Simulation of random oracle $\mathcal{H}$. To respond to the queries of the random oracle $\mathcal{H}$, extractor $\mathcal{E}$ keeps a table T_H of tuples $(\beta, H(\beta))$ as follows:

On a query $H(\beta)$, extractor $\mathcal{E}$ checks:

1. If there is a tuple $(\beta, H(\beta))$ that corresponds to β, then $\mathcal{E}$ returns $H(\beta)$.
2. If β has never been queried before, then $\mathcal{E}$ picks a random number h, and returns $H(\beta) = h$.

We assume for the rest of the proof that $\mathcal{A}$ succeeds in the soundness game with probability $\delta > \delta_{\mathsf{neg}}$. Below, we show that if $\gamma \geq \gamma_{\mathsf{neg}}$ then extractor $\mathcal{E}$ is able to recover file F^* with an overwhelming probability. We denote $\Pi^{\mathcal{E}}_{\mathsf{Success}}$ the probability that extractor $\mathcal{E}$ recovers file F^* by interacting with adversary $\mathcal{A}$.

We highlight the fact that if $\gamma \geq \gamma_{\mathsf{neg}}$, then succeeding in the soundness game implies that $\mathcal{A}$ corrupts less than ρ_{neg} fraction of the encoded file $\hat{F}^*$. This means that the probability that the ECC-encoded split S^*_i receives more that $\rho D = \frac{d}{2}$ errors is negligible, and so is probability $P^{\mathcal{E}}_{(\mathsf{Fail},i)}$ that extractor $\mathcal{E}$ fails in recovering split S^*_i.

In what follows, we show how extractor $\mathcal{E}$ recovers file F^*:

- $\mathcal{E}$ simulates oracle $\mathcal{O}_{\mathsf{Challenge}}$ to issue a challenge $\mathsf{chal} = (\vec{\mu}, R, i)$ for challenged file F^*, where R is the random number that is used by adversary $\mathcal{A}$ to generate its POR response and i is the index of split $\hat{S}^*_i$ that extractor $\mathcal{E}$ wants to extract. Here, $\vec{\mu}$ serves to retrieve the witnesses corresponding to the blocks composing $\hat{S}^*_i$. Without loss of generality, we assume that extractor $\mathcal{E}$ is interested in retrieving the k^{th} block of split $\hat{S}_i^*$ (*i.e.*, $\hat{b}_{ik}^*$). Accordingly, if the proof sent by adversary $\mathcal{A}$ for challenge chal is valid, extractor $\mathcal{E}$ will be able to recover the q-bit string $h = \epsilon_1\epsilon_2...\epsilon_q$ (see Figure 2.6) corresponding to the q first bits of $H(\hat{b}_{ik}, R)$.

- Provided with h, extractor $\mathcal{E}$ identifies the block $\beta \in T_H$ for which $H(\beta, R)|q = h$ if there is any. If it is the case, extractor $\mathcal{E}$ outputs $\hat{b}_{ik} = \beta$. Otherwise, extractor $\mathcal{E}$ declares block $\hat{b}_{ik}$ as missing.

Extractor $\mathcal{E}$ repeats the above procedure until retrieving the n splits $\hat{S}_i^*$ of file F^*. To this end, $\mathcal{E}$ issues nC (C is the number of blocks composing $\hat{S}_i^*$) challenges to the adversary $\mathcal{A}$ which is rewound before each challenge. Once the n splits $\hat{S}_i^*$ are retrieved, extractor $\mathcal{E}$ uses secret key K to decrypt the splits, then uses the ECC to correct the errors in the splits if there are any.

Note that extractor $\mathcal{E}$ fails in retrieving file F^* if she does not succeed in retrieving at least one of the splits S_i^*. The probability of this event is $\Pi^{\mathcal{E}}_{\mathsf{Fail}} \leq \sum_{i=1}^{n} P^{\mathcal{E}}_{(\mathsf{Fail},i)}$. Hence, $\mathcal{E}$ recovers file F^* with the following probability:

$$\Pi^{\mathcal{E}}_{\mathsf{Success}} = 1 - \Pi^{\mathcal{E}}_{\mathsf{Fail}} \geq 1 - \sum_{i=1}^{n} P^{\mathcal{E}}_{(\mathsf{Fail},i)}$$

Since adversary $\mathcal{A}$ corrupts less than ρ_{neg} fraction of file $\hat{F}^*$, the probability that a split S_i^* in the file F^* is irretrievable is negligible, namely, $P^{\mathcal{E}}_{(\mathsf{Fail},i)} \leq \frac{1}{2^\kappa}$, and therefore:

$$\Pi^{\mathcal{E}}_{\mathsf{Success}} \geq 1 - \underbrace{\frac{n}{2^\kappa}}_{\mathsf{negligible}}$$

Conclusion. $\mathcal{S}$tealthGuard is a (δ, γ)-sound proof of retrievability for any $\delta > \delta_{\mathsf{neg}} = \frac{1}{2^\kappa}$ and $\gamma \geq \gamma_{\mathsf{neg}} = \lceil \frac{\ln(2)\kappa}{\rho_{\mathsf{neg}}} \rceil$, where ρ_{neg} fulfills the following conditions:

$$(1 - \frac{\rho}{\rho_{\mathsf{neg}}})^2 \rho_{\mathsf{neg}} = \frac{3\ln(2)\kappa}{D} \text{ and } \rho_{\mathsf{neg}} \leq \rho.$$

2.4 Performance Analysis of $\mathcal{S}$tealthGuard

In this section we discuss the efficiency and the choice of the parameters in $\mathcal{S}$tealthGuard. We therefore give an example of parametrization and draw important conclusions for $\mathcal{S}$tealthGuard's implementation. We finally report some experimental results issued from the construction and the test of a $\mathcal{S}$tealthGuard prototype.

2.4.1 Discussion on Efficiency

To discuss the efficiency of $\mathcal{S}$tealthGuard, please refer to the notations listed in Table 2.1. Table 2.2 sums up the computational, storage and communication complexities of our $\mathcal{S}$tealthGuard.

2.4.1.1 Storage

At the end of the *Setup* phase, data owner $\mathcal{O}$ is only required to store master secret key K.

On the other hand, cloud server $\mathcal{S}$ must store the retrievable version $\hat{F}$ of the outsourced file F which amounts to $n \cdot C \cdot l$ bits, where n is the number of splits in $\hat{F}$, C is the number of blocks in each split $\hat{S}_i$ of $\hat{F}$ and l in the size in bits of a block. This value includes the storage overhead induced by the application of ECC - $n \cdot (d-1) \cdot l$ bits - and the insertion of watchdogs- $n \cdot v \cdot l$ bits.

2.4.1.2 Computation

The computational costs of algorithm KeyGen consists in first invoking a Pseudo-Random Number Generator (PRNG) to issue master secret key K. Thereafter, it computes $n+3$ keys by computing hash functions.

Algorithm Encode operates an ECC and a SSE over each split. Moreover, it performs nD *file-level* PRP and nv *split-level* PRP. Besides, to generate the watchdogs, this algorithm computes nv PRF. To check the retrievability of some file F composed of n splits and obtain a security level of $\frac{1}{2^\kappa}$, where κ is the security parameter, $\mathcal{S}$tealthGuard requires data owner $\mathcal{O}$ to generate $v > \frac{\gamma_{\mathsf{neg}}}{n}$ watchdogs per split where γ_{neg} (computed in Section 2.3.2) is the threshold of the number of queries that verifier $\mathcal{V}$ should issue. As shown in Theorem 2, this threshold does not depend on the size of data (in bytes). Instead, γ_{neg} is defined solely by the security parameter κ, the number $D = m + d - 1$ of blocks (including those generated by the application of an ECC) per split and the rate $\rho = \frac{d}{2D}$ of errors that the underlying ECC can correct. Namely, γ_{neg} is inversely proportional to both D and ρ. This means that by increasing the number of blocks D per split or the *correctable* error rate ρ, the number of queries that data owner $\mathcal{O}$ should issue decreases. However, having a large ρ would increase the size of data that $\mathcal{O}$ has to outsource to $\mathcal{S}$, which can be inconvenient for the data owner. Besides, increasing D leads to an increase in the number of blocks $C = s \cdot t$ per split $\hat{S}_i$ which has a direct impact on the communication cost and the computation load *per query* at both verifier $\mathcal{V}$ and cloud server $\mathcal{S}$. It follows that when defining the parameters of $\mathcal{S}$tealthGuard, one should consider the trade-off between the affordable storage cost and the computation and communication complexity per POR query.

It should also be noticed that the permutations applied during the execution of algorithm Encode are without doubt the most computationally-intensive steps in $\mathcal{S}$tealthGuard. Indeed, permutations require a non-negligible amount of random accesses which slow down the performance of the overall *Setup* phase. Notwithstanding their substantial costs, we highlight the fact that algorithm Encode is carried out only once for a theoretical unlimited number of POR verifications. In other terms, $\mathcal{S}$tealthGuard adopts the *amortized model* as we explained in Section 1.3: $\mathcal{O}$ performs a one-time expensive pre-processing operation that is amortized over an unlimited number of verifications. Indeed, as we can see in Table 2.2, the costs induced by algorithm Verify are light compared to the ones of Encode which can be potentially expensive due to the application of the pseudo-random permutation.

To illustrate the computation performances of $\mathcal{S}$tealthGuard, we take into consideration the **Trapdoor Group PIR**[62], proposed in [176] to implement the PIR procedure in WDSearch. This PIR mechanism enables verifier $\mathcal{V}$ who runs algorithm Challenge to fetch a row from an (s,t) matrix (representing a split) without revealing to cloud server $\mathcal{S}$, who executes algorithm ProofGen, which row verifier $\mathcal{V}$ is querying. One important feature of this PIR scheme is that it only involves random number generations, additions and multiplications in $\mathbb{Z}_p$ (where p is a prime of size $|p| = 200$ bits) which are not computationally intensive and could be performed by a lightweight verifier.

[62]As mentioned in 37, this PIR has been recently proven not to be secure. For illustration we mention this protocol but in practice, any efficient PIR algorithm that preserve confidentiality of queries and results would fit in our model.

Table 2.2: Complexities of $\mathcal{S}$tealthGuard

Storage	*$\lvert p \rvert$ refers to the size (in bits) of elements in $\mathbb{Z}_p$.*				
Data owner	$*$				
Server	$n \cdot C \cdot l$				
Included ECC	$n \cdot (d-1) \cdot l$				
Included watchdogs	$n \cdot v \cdot l$				
Communication					
Outbound	$\gamma \cdot (s \cdot \lvert p \rvert)$				
Inbound	$\gamma \cdot q \cdot (t \cdot \lvert p \rvert)$				
Operations	KeyGen	Encode	Challenge	ProofGen	Verify
PRNG	1	-	$\gamma \cdot (s+2)$	-	-
Hash computations	$n+3$	-	-	$\gamma \cdot C$	γ
ECC	-	n	-	-	-
PRP	-	$n \cdot D + n \cdot v$	γ	-	-
Encryption	-	n	-	-	-
PRF	-	$n \cdot v$	-	-	γ
Additions	-	-	3γ	$\gamma \cdot q \cdot (s-1) \cdot t$	-
Multiplications	-	-	$\gamma \cdot (s^2+2)$	$\gamma \cdot q \cdot s \cdot t$	$\gamma \cdot q$

2.4.1.3 Communication

The communication complexity only depends on the underlying PIR algorithm used to query and perform the watchdog search. We emphasize that PIR in $\mathcal{S}$tealthGuard is not employed to retrieve a watchdog, but rather a q-bit hash of the watchdog (typically $q = 80$), and that it is not performed on the entire file, but it is instead executed over a split. When employing **Trapdoor Group PIR**[176], the communication cost of $\mathcal{S}$tealthGuard is minimal when $s \simeq \sqrt{Cq}$ and $t \simeq \sqrt{\frac{C}{q}}$. This results in a communication complexity (per query) at verifier $\mathcal{V}$ of $\mathcal{O}(\sqrt{Cq})$ and a communication complexity at the server of $\mathcal{O}(\sqrt{Cq})$.

2.4.2 Example of Parameterization

To illustrate the efficiency of $\mathcal{S}$tealthGuard, we provide in this section an example of parameterization. Table 2.3 exposes the chosen parameters and the computed efficiency figures.

Let us consider a file F of 4 GB divided into $n = 32768$ splits $F = \{S_1, S_2, ..., S_n\}$, and each split S_i is composed of $m = 4096$ blocks of size $l = 256$ bits. The choice of block size $l = 256$ is obviously correlated to the block size operated by AES. Besides, this value ensures that the watchdogs are large enough to be impacted by any adversarial corruption (thus to be detected by an instance of $\mathcal{S}$tealthGuard) and to prevent a malicious cloud server from guessing watchdogs values by a brute-force attack. As it is, $\mathcal{S}$tealthGuard applies an ECC that corrects up to 228 corrupted blocks. In other terms, it operates the (D, m, d)-Reed-Solomon code of error rate $\rho = 5\%$ and where $d = 456$ and $D = m + d - 1 = 4551$. Thereafter, $\mathcal{S}$tealthGuard inserts $v = 8$ watchdogs per split.

We obtain thus $\hat{F} = \{\hat{S}_1, \hat{S}_2, ..., \hat{S}_n\}$, where $\hat{S}_i$ is composed of $C = 4559$ blocks of size $l = 256$ bits. This results in a redundancy of $\simeq 11.3\%$, where 11.1% redundancy is due to the use of ECC, and 0.20% redundancy is caused by the use of watchdogs.

Furthermore, we select optimum values for $s \simeq \sqrt{Cq}$ and $t \simeq \sqrt{\frac{C}{q}}$ such that $C = s \cdot t$ and $q = 80$. It appears that $(s, t) = (570, 8)$ is an optimal choice. Besides, if $\mathcal{S}$tealthGuard implements the Trapdoor Group PIR [176] where $\lvert p \rvert = 200$ bits, then verifier $\mathcal{V}$'s query will be of size $\simeq 13.9$ KB, whereas cloud server $\mathcal{S}$'s response will be of size $\simeq 15.6$ KB. In addition, if cloud server $\mathcal{S}$ still stores the file $\hat{F}$, then verifier $\mathcal{V}$ will declare file F as retrievable with probability $1 - \frac{n}{2^{60}} \simeq 1 - \frac{1}{2^{45}}$ by executing the POR protocol $\gamma = 1719$ times; namely, by sending γ POR queries which amounts to 23.4 MB and by downloading 26.2 MB which corresponds to 0.64% of the size of the original file F.

Table 2.3: Parameterization of $\mathcal{S}$tealthGuard

Param.	**Description**	**Values**
n	Number of splits S_i in file F	32768 splits
m	Number of blocks in a split S_i	4096 blocks
l	Size of a block	256 bits
L	Size of a split S_i	1048576 bits
d	Parameter of the ECC	456 blocks
D	Number of blocks in an encoded split $\tilde{S}_i$	4551 blocks
ρ	Error rate of the ECC	5%
v	Number of watchdogs in one split S_i	8 watchdogs
C	Number of blocks in a split $\hat{S}_i$ with watchdogs	4559 blocks
q	Number of cloud matrices (or size of witness)	80 matrices (or bits)
s	Number of rows in a cloud matrix	570
t	Number of columns in a cloud matrix	8
γ	Number of POR queries	1719

Table 2.4: Efficiency of $\mathcal{S}$tealthGuard's example

Metrics	**Costs**
Storage overhead (at server)	4.45 GB
Redundancy	11.3%
Due to ECC	11.1%
Due to watchdogs	0.20%
Communication cost for $\gamma = 1719$ queries and responses	
Outbound	23.4 MB
Inbound	26.2 MB

2.4.3 Comparison with Related Work

Table 2.5 and Table 2.6 depict the performance results of $\mathcal{S}$tealthGuard and compares it with previous work. In particular, we compare our solution with other POR schemes, namely [17, 107, 165, 189] within the same setting. This comparison considers a file F of size 4 GB and a POR assurance of $1 - \frac{1}{2^{45}}$ (as computed in Section 2.4.2).

We assume that all the compared schemes have three initial operations in the *Setup* phase: The application of an ECC, a SSE and a file-level PRP over all the blocks in file F. Since these three initial operations have comparable costs for all the schemes, we omit them in Table 2.5. Computational costs are represented with `exp` for exponentiation, `mul` for multiplication, `PRF` for pseudo-random function or `PRP` for pseudo-random permutation. The row corresponding to $\mathcal{S}$tealthGuard is filled with the values provided in Section 2.4.2. As for the other schemes, all initial parameters derive from the respective papers. An exception is made for [165] where no information about the number of blocks in a split is given. Therefore, we choose the same value as in [189].

Setup. In $\mathcal{S}$tealthGuard, data owner $\mathcal{O}$ computes $n{\cdot}v \approx 2.6{\times}10^5$ PRF and $\approx 2.6{\times}10^5$ PRP for the generation and the insertion of watchdogs. One of the advantages of $\mathcal{S}$tealthGuard is to provide a more lightweight *Setup* phase when $\mathcal{O}$ preprocesses large files. Indeed, the *Setup* phase in most of previous work [17, 165, 170, 189] requires $\mathcal{O}$ to compute an authentication tag for each block of data in the file which is computationally demanding in the case of large files.

Storage Overhead. The insertion of watchdogs in $\mathcal{S}$tealthGuard induces a smaller storage overhead compared to other schemes that generate authentication tags as in [17, 165, 189].

Proof Generation and Verification. As specified in Section 2.4.2, we consider the PIR operations as multiplications of elements in $\mathbb{Z}_p$ where $|p| = 200$ bits. To determine the server and verifier computational costs of existing work, we rely on the parameters and the bounds given in their respective papers. In particular, we compute the number of requested blocks in one challenge to obtain a probability of $1 - \frac{1}{2^{45}}$ to declare the file as retrievable: 764 blocks in [17], 1719 sentinels in [107], 45 blocks in [165] and 1639 blocks in [189]. As depicted in Table 2.6, it appears that $\mathcal{S}$tealthGuard induces high cost compared to existing work but is still acceptable.

Proof Generation and Verification. Even if its communication cost is relatively low compared to $\mathcal{S}$tealthGuard, the solution in [107] (JK POR) suffers from the limited number of challenges, that causes the data owner to download the whole file to regenerate new sentinels. Although we realize that $\mathcal{S}$tealthGuard's communication cost is much higher than [17, 165, 189], such schemes would induce additional cost at the file retrieval step, as mentioned in Section 1.5.

Summary. $\mathcal{S}$tealthGuard trades off between light computation at the data owner, small storage overhead at the cloud and significant but still acceptable communication cost. Nevertheless, we believe that $\mathcal{S}$tealthGuard's advantages pay off when processing large files. The difference between the costs induced by existing schemes and those induced by $\mathcal{S}$tealthGuard may become negligible if the size of the outsourced file increases.

Table 2.5: Comparative table between $\mathcal{S}$tealthGuard and relevant existing work (Setup).

Protocols	Parameters	Data Owner	Storage
Robust PDP [17]	Block size: 2 KB Tag size: 128 B	4.4×10^6 `exp` 2.2×10^6 `mul`	Tags: 267 MB
JK POR [107]	Block size: 128 bits Nb sentinels: 2×10^6	2×10^6 `PRF`	Sentinels: 30.6 MB
Compact POR [165]	Block size: 80 bits Blocks/split: 160 Tag size: 80 bits	5.4×10^6 `PRF` 1.1×10^9 `mul`	Tags: 51 MB
Efficient POR [189]	Block size: 160 bits Blocks/split: 160	2.2×10^8 `mul` 1.4×10^6 `PRF`	Tags: 26 MB
$\mathcal{S}$tealthGuard	Block size: 256 bits Blocks/split: 4096	2.6×10^5 `PRF` 2.6×10^5 `PRP`	Watchdogs: 8 MB

Table 2.6: Comparative table (Challenge-Response and Verification).

Protocols	**Server**	**Verifier**		**Communication**	
		Challenge	Verify	**Out**	**In**
Robust PDP [17]	764 `PRP` 764 `PRF` 765 `exp` 1528 `mul`	1 `exp`	766 `exp` 764 `PRP`	168 B	148 B
JK POR [107]	$\perp$	1719 `PRP`	$\perp$	6 KB	26.9 MB
Compact POR [165]	7245 `mul`	1 `enc` 1 `MAC`	45 `PRF` 365 `mul`	1.9 KB	1.6 KB
Efficient POR [189]	160 `exp` 2.6×10^5 `mul`	$\perp$	2 `exp` 1639 `PRF` 1639 `mul`	36 KB	260 B
$\mathcal{S}$tealthGuard	6.3×10^8 `mul`	5.6×10^8 `mul` 1719 `PRP`	1.4×10^5 `mul` 1719 `PRF`	23.4 MB	26.2 MB

2.4.4 Experimental Analysis

This section presents the implementation of $\mathcal{S}$tealthGuard into a prototype to validate the theoretical performance analysis conducted in Section 2.4.1 as well as to evaluate its practicality.

Experimental Setup. We simulate data owner $\mathcal{O}$, cloud server $\mathcal{S}$ and verifier $\mathcal{V}$ in the same environment, on a machine with the following characteristics: Processor Intel Core i5-2500; CPU 3.30 GHz clock speed; 64 bit OS; RAM 16 GB. The prototype is mainly written in Python language, resorts to the pycrypto library[63] for cryptographic operations (AES, SHA-256) and consists of two scripts: one for the data owner (used by the verifier as well) and another for the server. The data owner's script implements algorithms KeyGen, Encode, Challenge and Verify. It also provides a function Decode to reverse the operations performed by Encode when the outsourced data is retrieved. The server's program enforces algorithm ProofGen.

Algorithm Encode involves the application of an Error Correcting Code (ECC) algorithm. For our prototype, we implement the Reed-Solomon codes [154] by means of a C++ library called Schifra[64] and which operates on 16-bit symbol for the encoding. The pseudo-random function is implemented by means of the SHA-256 algorithm provided by the pycrypto library. The pseudo-random permutation is performed via a permutation table randomly generated on-the-fly by the program. Our prototype is perhaps not optimized to render the experimental analysis in line with our efficiency expectations. Nevertheless, the current version of the program is still valuable in order to interpret some of the results. Further improvements in the code will be implemented as a future work.

We run the prototype over files of different sizes ranging from 100 MB to 1000 MB. Each file is composed of splits of $m = 2048$ blocks of $l = 256$ bits. Hence, each split is of size $L = 65535$ bytes and each file is made of $n = \frac{\mathsf{filesize}}{L}$ splits.

[63]The Python Cryptography Toolkit (pycrypto): https://pypi.python.org/pypi/pycrypto [Accessed: February 2, 2016].

[64]Schifra library: http://www.schifra.com/ [Accessed: February 2, 2016].

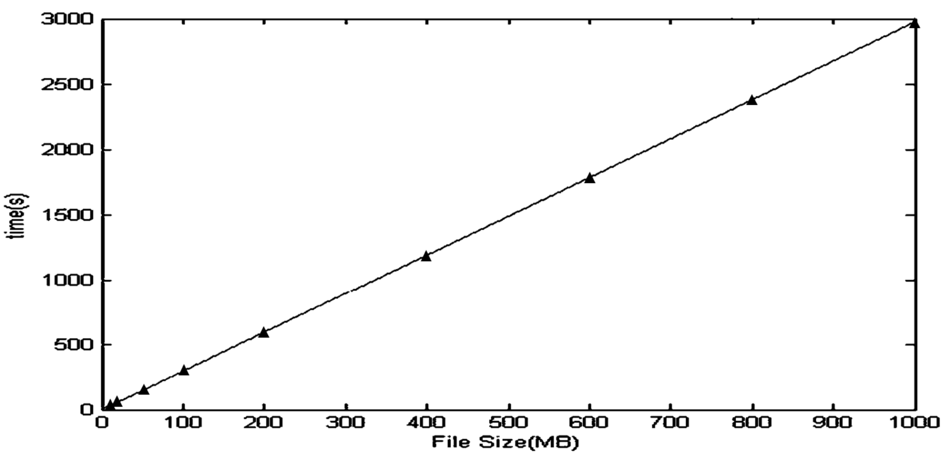


Figure 2.7: Error-Correcting Code encoding time

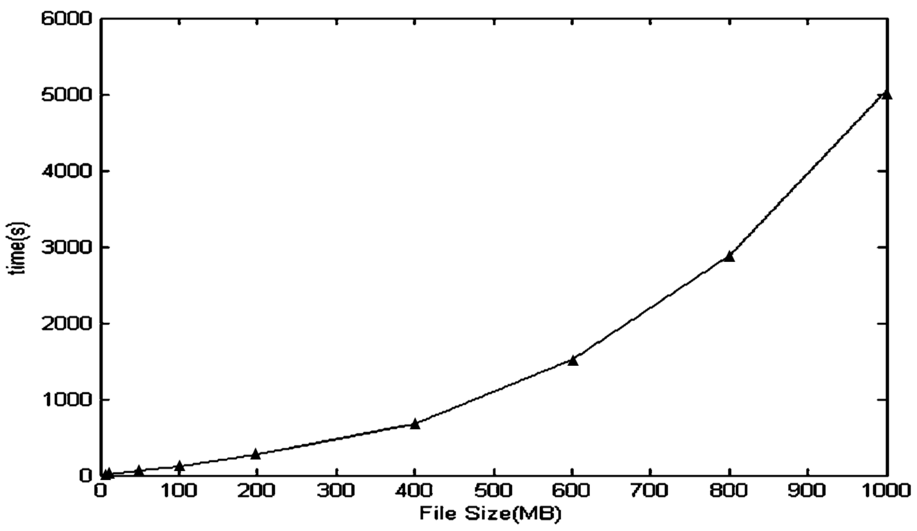


Figure 2.8: File-level permutation time

***Setup* phase.** The Setup phase of $\mathcal{S}$tealthGuard is relatively expensive as shown in Table 2.2. Indeed, algorithm Encode involves demanding operations such as the application of the ECC and the permutation of all the blocks in the data. Figure 2.7 shows the time needed to apply the ECC as a function of file size, Figure 2.8 displays the time required to perform the file-level permutation and Figure 2.9 depicts the total time needed by Encode.

While Figure 2.7 shows that the time required by the application of the ECC is linear in the file size (at the speed of approximately 340 KB/s), Figure 2.8 reveals that the file-level permutation time follows an exponential growth in the file size, which slows the overall performance of algorithm Encode, as shown in Figure 2.9. We explain this slow file-permutation operation by the fact that it requires $\mathcal{O}(nm)$ random accesses for permuting the file at the data owner side. Nonetheless, we highlight that this permutation is required only once by the Encode algorithm for multiple POR queries and verifications. Besides, our implementation of the permutation is not optimized.

The *Challenge* and *Verification* phases. We test our prototype to evaluate the performances of the generation and verification of POR for a file of 1GB whose splits contain $m = 2048$ blocks of $l = 256$ bits. Hence, the number of splits is $n = 16385$. The prototype applies an ECC that corrects up to 114 corrupted blocks, namely the ECC rate is $\rho = 5\%$, $d = 228$ and $D = m + d - 1 = 2275$. Besides, the test inserts $v = 3$ watchdogs in each split.

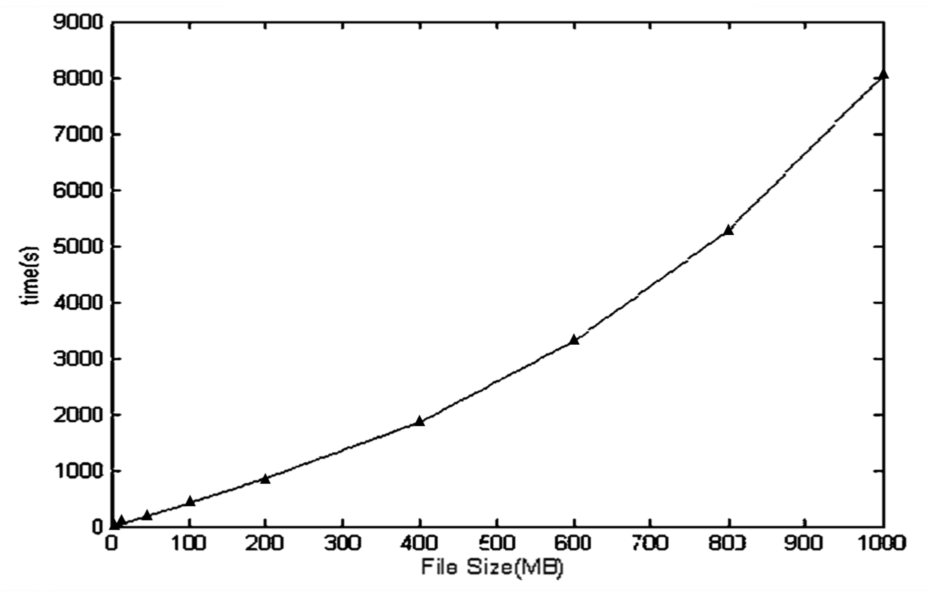


Figure 2.9: Encode total time

Table 2.7: *Challenge* and *Verification* phase times

Challenge (s)	ProofGen (s)	Verify (s)	TOTAL (s)
74.553	6.549	1.745×10^{-4}	81.102

Table 2.7 displays the average times to execute the three algorithms involved in the *Challenge* and *Verification* phases.

Challenge: This operation is the slowest among the three algorithms. As we can in see in Table 2.7 the average time to generate a single POR query is around 74.553 seconds. We argue that this value is non-negligible due to a non-optimized implementation of Challenge, and in particular of the permutation needed to generate the POR query.

ProofGen: On average, the time required by the server to respond to a single POR query, that is, to perform the privacy-preserving word search on a queried watchdog, is about 6.549 seconds. Measurements vary in the range $[0.535; 12.891]$ seconds. We highlight the fact that this computation is done on the server side.

Verify: The verification of the proof of retrievability takes approximately 1.745×10^{-4} second, which is is fast and negligible compared to the time required by other algorithms of our protocol.

Concluding thoughts. The above experimentations show that the overall protocol performance perceived by the data owner is impacted by (i) the application of the ECC to each split in the data to be outsourced; and (ii) the permutation of all the blocks in the data. Nonetheless, there exist mitigating aspects that should be considered. (i) Encoding the data is a one-time operation that allows an unrestricted number of POR verifications; and (ii) Our suboptimal implementation of the permutation operation does not reflect the efficiency of algorithm Encode and Challenge, but leaves open the way for improvements. The time to check the proofs of retrievability using algorithm Verify is shown to be affordable by users with limited resources. Finally, provided that the permutation operation is optimized in the prototype, $\mathcal{S}$tealthGuard is a viable POR solution.

2.5 Conclusion

Part I was devoted to the problem of proofs of storage. Well-suited to detect data losses in a remote storage service, proofs of storage have the advantage to be verifiable without the need to transfer the outsourced data in order to check its integrity.

We particularly focused on a particular kind of proofs of storage. Namely, proofs of retrievability not only ascertain the correct storage of outsourced data, but also prove that this data can be recovered if small corruptions affect it. We therefore design a new POR protocol, $\mathcal{S}$tealthGuard, that generates proofs of retrievability by combining randomly generated watchdogs with an efficient privacy-preserving word search mechanism. We showed that our technique is secure against malicious cloud servers who would be tempted to falsely claim that they correctly store the data. Besides, we evaluated $\mathcal{S}$tealthGuard's practicality by means of a prototype. Our solution perfectly answers Problem 1 on Verifiable Storage we expounded in the introduction of this thesis. If $\mathcal{S}$tealthGuard is deployed into a real-world cloud-based environment, the cloud users will be able to control that their data is correctly stored by the cloud service provider.

Future Work

A key direction for future research with respect to $\mathcal{S}$tealthGuard would address the difficulties of handling updates in the data while applying error-correcting codes. The protocol described in Chapter 2 does not consider update operations the data owner can perform over its data. Indeed, any update operation in the data, be it *split insertion, split deletion* or *split modification* has an impact on the security of the $\mathcal{S}$tealthGuard mechanism. For example, if the data owner modifies a same block several times then the cloud can discover that this particular block is not a watchdog. Additionally, updated splits should contain new watchdogs. Otherwise, if some of the watchdogs are not modified at each update, then the cloud may not execute the required operation, keep an old version of a split and hence it can still be successful in responding to POR queries, although it has not performed the update operation. Besides, updates in the data significantly alter the efficiency of $\mathcal{S}$tealthGuard. We recall that in $\mathcal{S}$tealthGuard, algorithm Encode first applies the ECC and then operates a permutation (Π_F) over all the blocks in the file (step 2 in Figure 2.3) to prevent the attack depicted in Section 2.2.4.1. In the case of data that are subject to updates, the permutation step renders any split update operation inefficient. Indeed, updating a split requires to update the corresponding ECC-blocks, resulting in the update of other splits and thus revealing to $\mathcal{S}$ the dependencies between the data blocks and the ECC-blocks. This knowledge allows the cloud to perform selective corruption attacks on the outsourced data. Therefore, the file permutation becomes ineffective. Some techniques are available to conceal these dependencies such as batching updates [170, 167] or Oblivious RAM (O-RAM) [53]. While O-RAM does not reveal to the cloud which blocks are blocks of ECC, this tool is computationally demanding and bandwidth consuming and has no practical relevance. Therefore, future work should be devoted to the design of an O-RAM-like mechanism that protects access pattern privacy, such that the updates can be securely performed with light costs.

Another direction for future research in the area of proofs of storage is to consider not only the cloud as malicious, but also the data owners and verifiers as untrusted. Indeed, we can envision that a cloud service would provide its users with financial compensation for any losses impacting their data. Malicious users would then be motivated to falsely claim that the cloud lost their data in order to fraudulently obtain this compensation. To cope with this concern, $\mathcal{S}$tealthGuard, and any POR protocols, should be enhanced with the *fairness* property. Fairness entails that a data owner cannot accuse an honest cloud server of having lost her data. Fairness also means that in case a dispute arises between the data owner and the server, a trusted third party (such as a judge in court) can mediate the two parties to reach a solution. Therefore, the server should be empowered with a mechanism to prove to this trusted third party that the data owner is lying, or to prove that the server itself is honest. A couple of papers [197, 116] considered the fairness property in the context of POS. A straightforward solution would be to require the data owner to sign the data before its outsourcing. However, in the case of dynamic POR protocol, this is more challenging since the data owner does not physically possess the data anymore. Downloading the data to update it and then compute the signature on the updated data cannot be considered in practice because it incurs prohibitive bandwidth consumption. A possible solution to circumvent this difficulty involves secure Chameleon signatures [114, 13]. The characteristic property of this kind of signature relies on the fact that it is hard to find a pair (S', r') such that $\mathsf{CH}(S', r') = \mathsf{CH}(S, r)$ (where CH denotes the Chameleon signature, S (resp. S') a split in an outsourced file (resp. an updated split) and r, r' two random numbers), but with the knowledge of a trapdoor, one can feasibly compute r' such that $\mathsf{CH}(S', r') = \mathsf{CH}(S, r)$, given $\mathsf{CH}(S, r)$ and S'. Integrating this signature for each split in the framework of $\mathcal{S}$tealthGuard would commit the data owner to the content of the data she outsourced. Hence, she cannot repudiate that she has stored all the signed splits. In case of dispute, the honest server can prove, thanks to the Chameleon signatures that a trusted third party can check, that it correctly stores the data. Besides, in case of updates, the data owner will be able to generate a new r' for the Chameleon signature of the updated split, without the need to download it. We will investigate more on $\mathcal{S}$tealthGuard with the fairness property via Chameleon signatures in the future.

Part II

Efficient Techniques for Verifiable Computation

Chapter 3

Charaterization of Verifiable Computation

3.1 Introduction to Verifiable Computation

The advent of cloud computing offers to individuals and organizations promising technologies for delegating not only the storage of their possibly huge amount of data (as analyzed in Part I), but also the execution of computationally demanding operations as investigated in the present part. However, outsourcing computation may jeopardize users' computation privacy and integrity, which dissuades a wide adoption of cloud technologies. As a matter of fact, remote servers, such as cloud providers, to whom control over data and computation is lent, cannot always be trusted. One of the main concerns deals with the problem of the outsourced computation integrity and is the matter of this part. In particular, we consider the following scenario: we assume a cloud user, be it an individual cloud-end user or a company (or any other kind of organization), wishes to delegate to the cloud, the execution of a computationally demanding operation $\mathfrak{f}$ so that she can later submit some input x of her choice and receive from the cloud the output $y = \mathfrak{f}(x)$. The question raised here, is: *how can the user be sure that y legitimately corresponds to the execution of function $\mathfrak{f}$ on input x?* In other terms, the cloud must not only correctly perform the requested computation, but also try to convince the user that the output is correct. One of the challenges here is that the user outsourced function $\mathfrak{f}$ to the cloud, thus relinquished the control over $\mathfrak{f}$ to the cloud. Therefore, the hypothetical and trivial solution having the user recompute $y^* = \mathfrak{f}(x)$ and then verify that $y = y^*$ cannot be considered. Besides, this trivial solution would have suffered from a major drawback: recomputing y^* and verifying $y = y^*$ is too expensive in terms of computational complexity and bandwidth, canceling out the advantage of outsourcing $\mathfrak{f}$ to the cloud. Hence, one of the goals in the present work is to devise solutions whereby the cloud can convince the user of the correctness of the computation in such a way that it still remains advantageous for the user to outsource that computation rather than executing it on her side.

To cope with the aforementioned challenges, Gennaro *et al.* [90] formalized the concept of Verifiable Computation (VC) in which a user delegates the execution of an operation to the cloud and further receives the result with some *cryptographic proofs* asserting the correct execution of the requested operation.

We describe in Section 3.2 a scenario that highlights the problem of Verifiable Computation. In Section 3.3, we give a characterization of a protocol in which a cloud server must provide the proofs of correct computation. Section 3.4 extends this initial definition to a special context where these proofs are *publicly verifiable*. Finally, we analyze prior art in Section 3.6.

3.2 Motivating Scenario

To concretely appreciate the problem and the challenges, the requirements and the features that we are dealing with, we elaborate the following scenario.

3.2.1 The Scenario

Let us consider an international space agency, such as the European Space Agency (ESA) or the National Aeronautics and Space Administration (NASA), that conducts research on Earth observation[65]. As a science organization, this space agency collects, produces and owns terabytes, even petabytes[66] of data such as space and aerial images or time-series data acquired from observing satellites. In other terms, the space agency has to deal with the so-called "big data" concept. The exploitation of this large amount of Earth observation data is essential for a wide range of applications: monitoring water quality in Africa[67], monitoring air pollution in Europe[68], monitoring the ice melting in the Arctic[69], or analyzing post-disaster effects[70] such as after the occurrence of earthquakes, tsunamis or volcanoes. Many stakeholders participate and collaborate in the various tasks related to Earth observation: the international space agency and its employees, outside researchers on Earth observation from all around the world, members of the space research community, educational users, and any other organizations to whom observation results are crucial for their business[71]. Notwithstanding their tremendous significance, the management of Earth observation data poses three major challenges to the space agency. First, it requires a large amount of space to store the tera, even petabytes of data. Furthermore, exploiting this data, that is, running data mining and information extraction operations in an automated way, is resource and time expensive. Finally, the data is envisioned to be efficiently shared among all the stakeholders cited above so as to enable collaborative research. We can also mention an additional challenge for the case of disaster monitoring: the Earth observation data should be processed nearly in real time to support emergency response teams after the occurrence of a hazard. Therefore, these challenges lead the international space agency to adopt cloud computing technologies, as it was the case for the ESA[72] and the NASA[73] in 2013. This migration to the cloud is expected to:

- reduce the agency's heavy investments in IT assets, in particular for its storage servers;
- offload to the cloud long and expensive data processing operations, such as data mining or image processing;
- make data access and sharing among researchers all around the world easier so that collaborative work is rendered efficient and effective.

One of the tasks that is incumbent upon our international space agency consists in processing and analyzing earth data such as space and aerial images or time-series data produced by observing satellites. Most image processing techniques leverage polynomial and matrix

[65]For more details on ESA's (respectively NASA's) Earth observation missions, see http://tiny.cc/2qdt8x [Accessed: February 3, 2016] (resp. http://tiny.cc/xrdt8x) [Accessed: February 3, 2016].

[66]The NASA's Earth Observing System Data and Information System metrics for 2014 shows that the system operates 9 petabytes of data. Source: http://tiny.cc/2sdt8x [Accessed: February 3, 2016].

[67]Water quality in Africa, ESA, http://tiny.cc/otdt8x [Accessed: February 3, 2016].

[68]Measurement of change in the ozone layer, ESA, http://tiny.cc/7udt8x [Accessed: February 3, 2016].

[69]Measurements of the volume of Arctic sea ice, ESA, http://tiny.cc/jvdt8x [Accessed: February 3, 2016].

[70]Hazard monitoring, ESA, http://tiny.cc/4xdt8x [Accessed: February 3, 2016].

[71]For instance, we can consider the case where, after the occurrence of a tsunami, monitoring its effects is essential for rescue services and civil defense authorities. Besides, insurance companies may be interested in mapping the extent of the tsunami. Therefore, both emergency response teams and insurance companies are "consumers" of Earth observation data.

[72]"European Space Agency delivers its SuperSites Exploitation Platform on Interoute Virtual Data Centre", Interoute, September 18, 2013, http://tiny.cc/w9dt8x [Accessed: February 3, 2016].

[73]Steve Cole, "NASA's Brings Earth Science "Big Data" to the Cloud with Amazon Web Services", NASA, November 12, 2013, http://tiny.cc/ceet8x [Accessed: February 3, 2016].

arithmetics. For example, polynomial evaluation can be used for contour detection [177], while matrices are often employed as a computational *mask*: Images are represented as a matrix of pixels that is multiplied by another matrix (the mask) which encodes an image processing operation (such as denoising or edge detection) [150]. Since the space images are remotely stored in the cloud, the latter is also requested to process the images (that is, to operate the underlying polynomial evaluations or matrix multiplications) on behalf of the space agency. In addition to image processing, data mining is fundamental in our scenario. For instance, the agency may desire to conduct some statistics on the images or on the annotations accompanying the images in order to classify them. In this line of research, keyword search is one of the most frequently used primitives for data mining. The agency may want to search its database for files that contain particular keywords, to classify the images[74]. As in the case of image processing, the cloud will be responsible for the keyword search on behalf of the space agency.

This computation outsourcing use case falls in the domain of **verifiable computation**, whereby the space agency wishes to receive cryptographic proofs generated by the cloud server attesting the correctness of the computation results. Namely, the agency must be convinced that the images are processed as expected (for example that the image -i.e the matrix- resulting from the application of the mask is correct) and that the keyword search returns the correct set of images that corresponds to the targeted keywords.

3.2.2 Requirements and Features for Verifiable Computation Protocols

From the above space agency scenario, we are able to extract the requirements and desired features for a VC scheme. The proofs of correct computation must satisfy two **security requirements**: they must be **correct** (if the cloud server is honest, the agency will always accept the proofs) and **sound** (a malicious cloud server cannot make the agency accept an incorrect result). Furthermore, it is essential that the verification of the results of outsourced computation must induce less costs than having the data owner process the space data locally. Otherwise, the migration to a cloud infrastructure would not be profitable for the space agency; on the contrary, it should help the agency save costs in processing. This illustrates another requirement: the **efficiency requirement**. Besides, the space agency would like to delegate to its researchers and to other collaborators (such as researchers from universities abroad, subcontractors, etc.) the capability to execute the outsourced operations (evaluate polynomial, compute matrix multiplication, search for keywords). These third-party users will also be empowered with the capability to verify the results of these operations. Consequently, the space agency needs a framework that ensures **public delegatability** and **public verifiability** of outsourced operations.

Efficiency Requirement. This requirement can be expressed in the following terms: In order not to waste the advantages of outsourcing the computation to the cloud, the cost for the user of submitting computation requests and verifying the cryptographic proofs must be less expensive than running the computation locally from scratch. This requirement imposes that the computational, storage and bandwidth overheads must be kept at minimum.

The efficiency requirement is also closely related to the concept of **amortized model**, introduced by Gennaro *et al.* [90]. This model authorizes the user to run a one-time expensive preprocessing operation that prepares the function before its outsourcing. This preprocessing can be as costly as computing the function from scratch. However, after this stage, the preprocessing is amortized over an *unlimited* number of fast verifications.

[74]For example, the concerned images are accompanied with a report that describes in words the elements that can be seen in the images.

Security Requirements. To capture the essence of verifiable computation, we highlight the two standard security requirements that they must satisfy:

Correctness: A honest server cannot be accused of deviating from the correct execution of the outsourced computation. Thus the verifier (i.e the entity who checks the validity of the proofs) will always be convinced by the server's correct behavior.

Soundness: If the server diverges from the correct execution of the computation, it cannot forge false proofs that would make the verifier accept the results.

Additional Features. In order to handle a setting similar to the one presented in the space agency scenario, a VC protocol should allow any user to request computation to the cloud and verify the returned result. Parno *et al.* [143] characterized the first public delegatable and public verifiable computation solution and gave the following definitions for the two desired properties of *public delegatability* and *public verifiability*:

Public delegatability: Anyone can submit inputs to the remote server to evaluate the outsourced function without any interaction with the user that outsourced the function. For ease of exposition, we will refer to *the querier* as the entity that submits the inputs. Therefore, the querier only needs to have access to a public key to request the computation to the cloud.

Public verifiability: Anyone (not only the one who submitted the inputs) can assess the correctness of the server's results. Thus, we will call this party *the verifier*. During the verification process, the verifier does not need any involvement of the querier but only a public verification key advertised by the querier. Note that in this framework, the verifier trusts the querier and hence trusts the verification key.

3.3 Definition of Verifiable Computation

This section formally defines a Verifiable Computation (VC) scheme. In a nutshell, a VC scheme is a two-party protocol in which a *data owner* outsources a computationally expensive function to a *cloud server*. Later on, the data owner submits some inputs of her choices to the server which is then required to evaluate the outsourced function on the requested inputs. The data owner finally verifies that the output returned by the server actually corresponds to a correct evaluation of the outsourced function on the provided inputs. In the following, we identify the players in a VC protocol and describe their capabilities. Then, we present the system model of such a protocol.

3.3.1 Parties Involved

A VC scheme comprises the following players:

Data owner $\mathcal{O}$: Data owner $\mathcal{O}$ outsources the computation of some (computationally demanding) function $\mathfrak{f}$ belonging to a family of functions $\mathcal{F}$ to a cloud server $\mathcal{S}$. Additionally, data owner $\mathcal{O}$ can provide cloud server $\mathcal{S}$ with some inputs x. The latter is required to compute $y = \mathfrak{f}(x)$ and tries to convince data owner $\mathcal{O}$ that y is indeed $\mathfrak{f}(x)$. $\mathcal{O}$ enjoys the capability to check that the result returned by $\mathcal{S}$ is correct.

Cloud Server $\mathcal{S}$: Often considered as potentially malicious, cloud server $\mathcal{S}$ is presumed to evaluate outsourced function $\mathfrak{f}$ on requested input x. Cloud server $\mathcal{S}$ also produces a proof that the output $\mathfrak{f}(x)$ is correct. Hence, we may refer to server $\mathcal{S}$ as the *prover*.

3.3.2 System Model

In this paragraph, we present the definition of a VC protocol. Without loss of generality, we assume that data owner $\mathcal{O}$ outsources the computation of a function $\mathfrak{f} \in \mathcal{F}$ to cloud server $\mathcal{S}$. Then $\mathcal{O}$ asks $\mathcal{S}$ to evaluate function $\mathfrak{f}$ on input $x \in \mathcal{D}_{\mathfrak{f}}$ (the domain of definition of function $\mathfrak{f}$) and checks the correctness of the computation result returned by server $\mathcal{S}$.

Definition 5 (VC Scheme). *A VC scheme consists of four polynomial-time algorithms* (Setup, ProbGen, Compute, Verify) *distributed across three phases:*

▶ **Setup.** *This phase only involves data owner $\mathcal{O}$. She runs algorithm* Setup *to produce the keying material required in the VC scheme and to process function $\mathfrak{f}$ before its outsourcing:*

▷$\mathsf{Setup}(1^\kappa, \mathfrak{f}) \rightarrow (\mathsf{param}, \mathsf{SK}_\mathfrak{f}, \mathsf{EK}_\mathfrak{f})$**:** *It is a randomized algorithm executed by data owner $\mathcal{O}$. It takes as input the security parameter 1^κ and a description of the function $\mathfrak{f}$ to be outsourced, and outputs a set of parameters* param*, a secret key* $\mathsf{SK}_\mathfrak{f}$ *that is kept by data owner $\mathcal{O}$, and an evaluation key* $\mathsf{EK}_\mathfrak{f}$ *that encodes function $\mathfrak{f}$ to be used by cloud server $\mathcal{S}$ to evaluate $\mathfrak{f}$.*

▶ **Computation.** *The Computation phase consists of two steps. Data owner $\mathcal{O}$ runs algorithm* ProbGen *that prepares an input x to be submitted to cloud server $\mathcal{S}$. In turn, the server invokes algorithm* Compute *that evaluates function $\mathfrak{f}$ on input x and generates a proof of correct computation.*

▷$\mathsf{ProbGen}(x, \mathsf{SK}_\mathfrak{f}) \rightarrow (\sigma_x, \mathsf{VK}_x)$**:** *Given an input x in the domain $\mathcal{D}_\mathfrak{f}$ of the outsourced function $\mathfrak{f}$ and secret key* $\mathsf{PK}_\mathfrak{f}$*, data owner $\mathcal{O}$ calls this algorithm to produce an encoding σ_x of input x that is transmitted to server $\mathcal{S}$, and a secret verification key* VK_x *kept by $\mathcal{O}$ and which will afterwards be used to check the correctness of the server's result.*

▷$\mathsf{Compute}(\sigma_x, \mathsf{EK}_\mathfrak{f}) \rightarrow \sigma_y$**:** *On input of the encoding σ_x and the evaluation key* $\mathsf{EK}_\mathfrak{f}$*, server $\mathcal{S}$ runs this algorithm to compute an encoding σ_y of $\mathfrak{f}$'s output $y = \mathfrak{f}(x)$.*

▶ **Verification.** *After receiving the encoding of the computation result from cloud server $\mathcal{S}$, data owner $\mathcal{O}$ executes algorithm* Verify *to check its validity.*

▷$\mathsf{Verify}(\sigma_y, \mathsf{VK}_x) \rightarrow \mathsf{out}_y$**:** *Data owner $\mathcal{O}$ operates this deterministic algorithm to check the correctness of the result σ_y supplied by server $\mathcal{S}$ on input σ_x. More precisely, this algorithm first decodes σ_y which yields a value y, and then uses the verification key* VK_x *associated with the encoding σ_x to decide whether y is equal to the expected output $\mathfrak{f}(x)$. If so,* Verify *outputs* $\mathsf{out}_y = y$ *meaning that $\mathfrak{f}(x) = y$; otherwise it outputs an error* $\mathsf{out}_y = \perp$.

Before formalizing the requirements for a VC scheme we have identified in Section 3.2.2, that is the efficiency and security requirements, we enhance the above model with the two additional features of *public delegatability* and *public verifiability.*

3.4 Definition of Publicly Verifiable Computation

The system model we detailed in Section 3.3 focuses on a *privately* verifiable computation scheme in the sense that only the data owner can request the cloud server to evaluate the outsourced function and only she can verify the result returned by the server. In our international space agency scenario, depicted in Section 3.2, we let the agency delegate the search and verification capabilities to any third party such as outside researchers. Therefore, we devise here the model for a VC scheme that enables public delegatability and public verifiability.

3.4.1 Parties Involved in a PVC Protocol

A Publicly Verifiable Computation (PVC) scheme comprises four players: a data owner and a cloud server (which are identical to the ones in a VC scheme, as well as a querier and a verifier, that are specific to the model of a PVC scheme.

Data owner $\mathcal{O}$: As in a privately VC protocol, data owner $\mathcal{O}$ outsources the computation of some computationally demanding function $\mathfrak{f}$ belonging to a family of functions $\mathcal{F}$ to a cloud server $\mathcal{S}$. She then produces an evaluation key $\mathsf{EK}_{\mathfrak{f}}$ used by server $\mathcal{S}$ to respond to any requested computation on function $\mathfrak{f}$. In addition, data owner $\mathcal{O}$ can delegate to anyone the capability to submit inputs and to verify outputs, to achieve public delegatability and public verifiability. To do so, she advertises a public key $\mathsf{PK}_{\mathfrak{f}}$ that will be used by anyone who wishes to request computations on $\mathfrak{f}$.

Cloud Server $\mathcal{S}$: $\mathcal{S}$ is presumed to evaluate outsourced function $\mathfrak{f}$ on a requested input x. Cloud server $\mathcal{S}$ also produces a proof that the output $\mathfrak{f}(x)$ is correct.

Querier $\mathcal{Q}$: Given public key $\mathsf{PK}_{\mathfrak{f}}$, querier $\mathcal{Q}$ requests cloud server $\mathcal{S}$ to evaluate the already outsourced function $\mathfrak{f}$ on some input x in the domain $\mathcal{D}_{\mathfrak{f}}$ of $\mathfrak{f}$. $\mathcal{Q}$ would like to obtain from $\mathcal{S}$ the assurance that the results that $\mathcal{S}$ returns are correct. Therefore, she generates a public verification key VK_x tied to input x. Note that querier $\mathcal{Q}$ can be not only data owner $\mathcal{O}$ herself but also any party that is interested in evaluating function $\mathfrak{f}$ (*public delegatability*).

Verifier $\mathcal{V}$: With the help of public verification key VK_x, this player checks that the result $\mathfrak{f}(x)$ returned by cloud server $\mathcal{S}$ given x is correct. The role of verifier can be played either by querier $\mathcal{Q}$ who has requested the computation or by any other entity that wants to verify the computation result on behalf of $\mathcal{Q}$ (*public verifiability*).

3.4.2 System Model

In this paragraph, we present the definition of a PVC protocol. Without loss of generality, we assume that data owner $\mathcal{O}$ outsources the computation of a function $\mathfrak{f} \in \mathcal{F}$ to cloud server $\mathcal{S}$. Then a querier $\mathcal{Q}$ asks $\mathcal{S}$ to evaluate function $\mathfrak{f}$ on input $x \in \mathcal{D}_{\mathfrak{f}}$ and a verifier $\mathcal{V}$ checks the correctness of the computation result returned by server $\mathcal{S}$.

Definition 6 (PVC Scheme). *A PVC scheme consists of four polynomial-time algorithms* (Setup, ProbGen, Compute, Verify) *distributed across three phases:*

▶ **Setup.** *This phase only involves data owner $\mathcal{O}$. She runs algorithm* Setup *to produce the keying material required in the PVC scheme and to process function $\mathfrak{f}$ before its outsourcing:*

▷$\mathsf{Setup}(1^\kappa, \mathfrak{f}) \to (\mathsf{param}, \mathsf{PK}_{\mathfrak{f}}, \mathsf{EK}_{\mathfrak{f}})$**:** *It is a randomized algorithm executed by data owner $\mathcal{O}$. It takes as input the security parameter 1^κ and a description of the function $\mathfrak{f}$ to be outsourced, and outputs a set of public parameters* param*, a public key $\mathsf{PK}_{\mathfrak{f}}$, and an evaluation key $\mathsf{EK}_{\mathfrak{f}}$ that will be used by subsequent algorithms.*

▶ **Computation.** *The Computation phase consists of two steps. Querier $\mathcal{Q}$ runs algorithm* ProbGen *that prepares an input x to be submitted to cloud server $\mathcal{S}$. In turn, the server invokes algorithm* Compute *that evaluates function $\mathfrak{f}$ on input x and generates a proof of correct computation.*

▷$\mathsf{ProbGen}(x, \mathsf{PK}_{\mathfrak{f}}) \rightarrow (\sigma_x, \mathsf{VK}_x)$**:** *Given an input x in the domain $\mathcal{D}_{\mathfrak{f}}$ of the outsourced function $\mathfrak{f}$ and public key $\mathsf{PK}_{\mathfrak{f}}$, querier $\mathcal{Q}$ calls this algorithm to produce an encoding σ_x of input x that is transmitted to server $\mathcal{S}$, and a public verification key VK_x that will afterwards be used by verifier $\mathcal{V}$ to check the correctness of the server's result.*

▷$\mathsf{Compute}(\sigma_x, \mathsf{EK}_{\mathfrak{f}}) \rightarrow \sigma_y$**:** *On input of the encoding σ_x and the evaluation key $\mathsf{EK}_{\mathfrak{f}}$, server $\mathcal{S}$ runs this algorithm to compute an encoding σ_y of $\mathfrak{f}$'s output $y = \mathfrak{f}(x)$.*

▶ **Verification.** *After receiving the result and the proof of computation from cloud server $\mathcal{S}$, verifier $\mathcal{V}$ executes algorithm* Verify *to check their validity.*

▷$\mathsf{Verify}(\sigma_y, \mathsf{VK}_x) \rightarrow \mathsf{out}_y$**:** *Verifier $\mathcal{V}$ operates this deterministic algorithm to check the correctness of the result σ_y supplied by server $\mathcal{S}$ on input σ_x. More precisely, this algorithm first decodes σ_y which yields a value y, and then uses the public verification key VK_x associated with the encoding σ_x to decide whether y is equal to the expected output $\mathfrak{f}(x)$. If so,* Verify *outputs $\mathsf{out}_y = y$ meaning that $\mathfrak{f}(x) = y$; otherwise it outputs an error $\mathsf{out}_y = \perp$.*

Before formalizing the *security* requirements, namely the notion of correctness and soundness of a PVC scheme, we provide here a characterization of the *efficiency* requirement and the desired properties of public delegatability and public verifiability.

We require for a *viable* PVC scheme that the costs of generating a computation request and to verify the output of this computation must be smaller than the costs of computing the function from scratch.

Requirement 1 (Efficiency). *We say that a PVC scheme is a viable solution if the problem generation and the verification algorithms are efficient. In particular, given $\mathsf{PK}_{\mathfrak{f}}$ and for any x and any σ_y the time to run $\mathsf{ProbGen}(x, \mathsf{PK}_{\mathfrak{f}})$ together with the time to execute $\mathsf{Verify}(\sigma_y, \mathsf{VK}_x)$ (where VK_x is generated by algorithm* ProbGen*) is $o(T)$, where T is the time needed to compute $\mathfrak{f}(x)$.*

Following the *amortized model*, the time required for Setup is omitted in this definition. In this model, algorithm Setup operates an expensive pre-processing operation that encodes the function to be outsourced in such a way that public delegatability and public verifiability are possible. Nonetheless, Setup is executed once for many computations of the same function on many different inputs. Therefore, it does not undermine the above consideration for a *viable* solution.

Additionally, we formalize in the following lines the notions of public delegatability and public verifiability, considered as optional requirements.

Requirement 2 (Public delegatability). *A VC scheme is publicly delegatable if, given a public key $\mathsf{PK}_{\mathfrak{f}}$, some input $x \in \mathcal{D}_{\mathfrak{f}}$, and some additional public parameters, then any third-party querier can execute algorithm* ProbGen *on inputs x and $\mathsf{PK}_{\mathfrak{f}}$, without the need of any secret information provided by the data owner that outsourced function $\mathfrak{f}$.*

Requirement 3 (Public verifiability). *A VC scheme is publicly verifiable if, given an encoding σ_y of the outsourced computation $y = \mathfrak{f}(x)$ for some input $x \in \mathcal{D}_\mathfrak{f}$, a public verification key VK_x, and some additional public parameters, then a third-party verifier can run algorithm* Verify *to check that σ_y is a valid encoding of $y = \mathfrak{f}(x)$, without knowing any secret information required from the data owner that outsourced function $\mathfrak{f}$ or from the querier who submitted input x.*

3.5 Adversary Model in Verifiable Computation

A PVC scheme must fulfill the basic security requirements of *correctness* and *soundness.* Succinctly, the *correctness* property states that if the server honestly evaluates the function based on the user's input (*i.e.* correctly executes algorithm Compute), then the verifier (who runs algorithm Verify) will always accept the server's result. On the other hand, the *soundness* requirement captures the fact that a malicious server cannot make algorithm Verify (and thus the verifier) accept a result that is not correctly computed via algorithm Compute.

3.5.1 Correctness

A publicly verifiable computation scheme for a family of functions $\mathcal{F}$ is deemed to be *correct*, if whenever an *honest* server $\mathcal{S}$ executes the algorithm Compute to evaluate a function $\mathfrak{f} \in \mathcal{F}$ on an input $x \in \mathcal{D}_\mathfrak{f}$, this algorithm *always* yields an encoding σ_y that will be accepted by a verifier $\mathcal{V}$ running algorithm Verify (*i.e.* $\mathsf{Verify}(\sigma_y, \mathsf{VK}_x) \to \mathfrak{f}(x)$, where VK_x is generated via ProbGen).

Definition 7 (Correctness). *A publicly verifiable computation scheme for a family of functions $\mathcal{F}$ is correct,* **iff** *for any function $\mathfrak{f} \in \mathcal{F}$ and any input $x \in \mathcal{D}_\mathfrak{f}$:*
If $\mathsf{ProbGen}(x, \mathsf{PK}_\mathfrak{f}) \to (\sigma_x, \mathsf{VK}_x)$ *and* $\mathsf{Compute}(\sigma_x, \mathsf{EK}_\mathfrak{f}) \to \sigma_y$, *then:*

$$\Pr(\mathsf{Verify}(\sigma_y, \mathsf{VK}_x) \to \mathfrak{f}(x)) = 1.$$

3.5.2 Soundness

A publicly verifiable computation scheme for a family of functions $\mathcal{F}$ is said to be *sound*, if for any $\mathfrak{f} \in \mathcal{F}$ and for any $x \in \mathcal{D}_\mathfrak{f}$, a server $\mathcal{S}$ cannot convince a verifier $\mathcal{V}$ to accept an incorrect result. Notably, a verifiable computation scheme is sound if it assures that the only way server $\mathcal{S}$ generates a result σ_y that will be accepted by verifier $\mathcal{V}$ (*i.e.* by algorithm Verify) as a valid encoding of the evaluation of some function $\mathfrak{f} \in \mathcal{F}$ on an input x, is by correctly computing σ_y (*i.e.* $\sigma_y \leftarrow \mathsf{Compute}(\sigma_x, \mathsf{EK}_\mathfrak{f})$).

We capture the adversarial capabilities of an adversary (*i.e.* malicious server) $\mathcal{A}$ against a publicly verifiable computation scheme for a family of functions $\mathcal{F}$ through a *soundness experiment*, depicted in Algorithm 3 and 4. In this experiment, adversary $\mathcal{A}$ accesses the output of algorithm Setup by calling oracle $\mathcal{O}_{\mathsf{Setup}}$. When queried with a security parameter 1^κ and a description of a function $\mathfrak{f} \in \mathcal{F}$, oracle $\mathcal{O}_{\mathsf{Setup}}$ returns the set of public parameters param, public key $\mathsf{PK}_\mathfrak{f}$, and evaluation key $\mathsf{EK}_\mathfrak{f}$. Adversary $\mathcal{A}$ also invokes an oracle $\mathcal{O}_{\mathsf{ProbGen}}$ that selects an input $x \in \mathcal{D}_\mathfrak{f}$ and that, given public key $\mathsf{PK}_\mathfrak{f}$, executes algorithm ProbGen. This algorithm outputs a pair of matching encoding σ_x and public verification key VK_x. Finally, adversary $\mathcal{A}$ generates an encoding σ_y and calls algorithm Verify on the pair $(\sigma_y, \mathsf{VK}_x)$.

The soundness game we develop involves two phases: a **learning** phase and a **challenge** phase:

Learning. Adversary $\mathcal{A}$ *adaptively* calls oracle $\mathcal{O}_{\mathsf{Setup}}$ with t distinct functions $\mathfrak{f}^{(k)}$ allowing $\mathcal{A}$ to receive for each function $\mathfrak{f}^{(k)}$ the corresponding public parameters $\mathsf{param}^{(k)}$, the public key $\mathsf{PK}_{\mathfrak{f}}^{(k)}$ and the evaluation key $\mathsf{EK}_{\mathfrak{f}}^{(k)}$. Then, given an input $x^{(k)}$ and public key $\mathsf{PK}_{\mathfrak{f}}^{(k)}$ associated with file $\mathfrak{f}^{(k)}$, adversary $\mathcal{A}$ queries $\mathcal{O}_{\mathsf{ProbGen}}$ to generate an encoding $\sigma_x^{(k)}$ and a verification key $\mathsf{VK}_x^{(k)}$ associated with $x^{(k)}$ and file $\mathfrak{f}^{(k)}$. On reception of this encoding, $\mathcal{A}$ produces an encoding $\sigma_y^{(k)}$, either arbitrarily or by executing algorithm $\mathsf{Compute}$. Then, adversary $\mathcal{A}$ invokes algorithm Verify on inputs $\mathsf{VK}_x^{(k)}$ and $\sigma_y^{(k)}$.

▼ **Algorithm 3:** Learning Phase of the Soundness Experiment

```
for $k := 1$ to $t$ do
    $\mathcal{A} \to \mathfrak{f}^{(k)}$;
    $(\mathsf{param}^{(k)}, \mathsf{PK}_{\mathfrak{f}}^{(k)}, \mathsf{EK}_{\mathfrak{f}}^{(k)}) \leftarrow \mathcal{O}_{\mathsf{Setup}}(1^\kappa, \mathfrak{f}^{(k)})$;
    $\mathcal{A} \to x^{(k)}$;
    $(\sigma_x^{(k)}, \mathsf{VK}_x^{(k)}) \leftarrow \mathcal{O}_{\mathsf{ProbGen}}(x^{(k)}, \mathsf{PK}_{\mathfrak{f}}^{(k)})$;
    $\mathcal{A} \to \sigma_y^{(k)}$;
    $\mathsf{out}_y^{(k)} \leftarrow \mathsf{Verify}(\sigma_y^{(k)}, \mathsf{VK}_x^{(k)})$;
end
```

Challenge. Afterwards, adversary $\mathcal{A}$ selects function $\mathfrak{f}^*$ and the corresponding public key $\mathsf{PK}_{\mathfrak{f}^*}$ from the set of public keys $\{\mathsf{PK}_{\mathfrak{f}}^{(k)}\}_{1 \leq k \leq t}$ she has received during the **learning** phase. Then, $\mathcal{A}$ outputs a challenge input $x^* \in \mathcal{D}_{\mathfrak{f}^*}$ and executes algorithm $\mathsf{ProbGen}$ with public key $\mathsf{PK}_{\mathfrak{f}^*}$ and x^* to get the matching pair of encoding σ_{x^*} and public verification key VK_{x^*}.

Finally, adversary $\mathcal{A}$ generates an encoding σ_{y^*} and runs algorithm Verify on the pair $(\sigma_{y^*}, \mathsf{VK}_{x^*})$.

▼ **Algorithm 4:** Challenge Phase of the Soundness Experiment

```
$\mathcal{A} \to (\mathsf{param}^*, \mathsf{PK}_{\mathfrak{f}^*}, \mathsf{EK}_{\mathfrak{f}^*})$;
$\mathcal{A} \to x^*$;
$(\sigma_{x^*}, \mathsf{VK}_{x^*}) \leftarrow \mathcal{O}_{\mathsf{ProbGen}}(x^*, \mathsf{PK}_{\mathfrak{f}^*})$;
$\mathcal{A} \to \sigma_{y^*}$;
$\mathsf{out}_{y^*} \leftarrow \mathsf{Verify}(\sigma_{y^*}, \mathsf{VK}_{x^*})$;
```

Let out_{y^*} denote the output of algorithm Verify at the end of the experiment. We say that adversary $\mathcal{A}$ succeeds in the soundness experiment of publicly verifiable computation if $\mathsf{out}_{y^*} \neq \perp$ and $\mathsf{out}_{y^*} \neq \mathfrak{f}^*(x^*)$.

Definition 8. *Let $\Pi_{\mathcal{A},\mathfrak{f}^*}$ denote the probability that adversary $\mathcal{A}$ succeeds in the soundness experiment of publicly verifiable computation (*i.e. $\mathsf{out}_{y^*} \neq \perp \ \wedge \ \mathsf{out}_{y^*} \neq \mathfrak{f}^*(x^*)$*).*

A publicly verifiable computation scheme for a family of functions $\mathcal{F}$ is sound, **iff***: For any adversary $\mathcal{A}$ and for any $\mathfrak{f} \in \mathcal{F}$, $\Pi_{\mathcal{A},\mathfrak{f}} \leq \varepsilon$ and $\varepsilon(\kappa)$ is a negligible function in the security parameter κ.*

3.6 State of the Art in Verifiable Computation

An important body of research recently emerged to address the problem of verifiable outsourced computation. Especially, the advent of cloud computing stimulated a large number of research work proposing *cryptographic* solutions for a *verifier* to efficiently verify the execution by a remote untrusted server, the *prover*, of computationally demanding operations. However, the problem of VC is not new: decades ago, many solutions were proposed to respond to concerns on how to put more trust on results computed by a remote server.

This section attempts to provide an in-depth review of existing solutions for verifiable computation. At the time of writing this dissertation, no such a review has been carried out, or published. We can only refer to a relevant survey conducted by Walfish and Blumberg [181]. Nevertheless the authors only focus on *general-purpose* solutions that provide answers to the problem of VC for arbitrary functions, discarding those solutions that apply to specific computations. The rationale behind the lack of an extensive review may lie in the fact that multiple approaches have been adopted and that the rapid and growing plebiscite for cloud computing reawakened the interest for the problem of VC, yielding multitudinous solutions that may be difficult to compare. Our review endeavors to compile and to propose a possible classification for a list of relevant existing works.

In the following, we establish a first categorization: non-proof based solutions (Section 3.6.1) and proof-based solutions (Section 3.6.2). As their name implies, solutions in the former category do not require the generation and verification of a proof stating that the computation result is correct. Thereafter, we organize solutions from the second category in several subcategories depending on whether they satisfy particular properties. Indeed, we identify the following characteristics that proof-based solutions may satisfy:

Generality: Some existing solutions (Section 3.6.2) provide protocols to verify the correct computation of arbitrary functions while other proposals (Section 3.6.3) target a restricted class of functions, exploiting special properties of these functions.

Interactivity: Most early work on VC build an Interactive Proofs (IP) system in which the prover and the verifier engage in an "exchange" to convince the latter about the result correctness (Section 3.6.2.1). However, an increasing number of solutions are non-interactive and only require a challenge-response phase between the prover and the verifier (Section 3.6.2.2).

Practicality: Early solutions to the problem of VC were mainly theoretical, with poor practicality, since they were too inefficient for actual implementation. The breakthroughs not only in cloud computing but also in mobile devices (smartphones, tablets, laptops, etc.) made crucial the design of practical solutions that spare both the prover and verifier's work.

Security model: Schemes for VC can be based on several assumptions: For example, early theoretical work on IP assume an *all-powerful* prover whereas work on *arguments* [110] presumes a *polynomial-time* prover. Some work are based on complexity-theory assumptions only while others rely on cryptographic assumptions.

Additional features: In a nutshell, depending on the scenario in which solutions for VC apply, one may desire the features of *public verifiability* or *public delegatability*, notions that are defined in Section 3.2.2 and Section 3.4. Some other solutions focus on privacy of verifiable computation, in the sense that the outsourced data over which the verifiable computations are performed are encrypted, such that the prover cannot infer any information from the data, the computations and their outputs. Note that in our international space agency scenario, we do not require the data to be encrypted nor the computation to be kept private. However, for a deep survey on VC protocols we also mention the existence of the privacy-preserving property.

Based on these identified characteristics and the requirements for a PVC we outlined in Section 3.1, we provide here our review of existing work. We highlight the fact that we use the term "verifiable computation" for all the schemes discussed in the following. This naming may appear improper since Gennaro *et al.* [90] were the first to formalize the notion of "verifiable computation". Nevertheless, we regroup under the same term all prior work that aims at verifying the outcome of a computation.

3.6.1 Non-Proof-based and Hardware-based Solutions

A first answer to the problem of VC is the replication of the outsourced computation among multiple servers [7, 6, 51]. As described in Section 1.1.2, the SETI@Home project [7] replicates an instance of the same computation to different (untrusted) nodes through the BOINC middleware [6]. This middleware sends a copy of the computation to several computers and compares the results output by these computers (located in different nodes). If the results match, then, with high probability, they are considered correct. Otherwise, the middleware sends other replicas to other computers. This process is repeated until a majority of matching results is obtained. Therefore, this replication system requires that a *majority* of computers, among all the computers where the computation is replicated, behaves honestly. Canetti *et al.* [51] reduce this assumption to a single server: the computation is still replicated to several servers, but it suffices that only a single server is honest[75].

Other techniques [142, 157, 158] rely on trusted computing and attestation procedures based on trusted hardware, such as the Trusted Platform Module (TPM), embedded at the server side. For example, Parno *et al.* [142] suggest to combine code identity recordings (like computing a cryptographic hash on the binary of the software performing the outsourced computation) and remote attestation that checks the code signature. However, as the TPM is under the (physical) control of the server, this solution appears to put a lot of trust in the module. Sadeghi *et al.* [157] propose to combine a trusted-hardware token (such as a cryptographic coprocessor) with techniques for secure function evaluation (such as Fully-Homomorphic Encryption (FHE) [92] or Yao's garbled circuits [191]) to compute outsourced arbitrary functions on encrypted data (this scheme guarantees input privacy but not output privacy). The token consists in a tamper-proof coprocessor attached to the remote server and which executes operations within a shielded environment, on behalf of the user that requests the computation. Yet, again in this proposal, the verifier has to trust the token's manufacturer.

3.6.2 Proof-based General-Purpose Solutions

As mentioned before, the majority of solutions to the problem of VC follows the *proof-based* approach: the VC protocol enables the prover to return the results of an outsourced function along with a proof that the result is correct. The review of the state of the art identifies two types of solutions. This section inspects general-purpose solutions which define a proof system for arbitrary functions. They make few or no assumptions on the outsourced function and enable to verify the correctness of the evaluation of that function. We survey the second category of proof-based solutions in Section 3.6.3. This class of solutions targets specific operations. In a nutshell, they exploit the particular *structure* and *properties* of these operations to enable efficient verification.

Designing a proof system for general computation first spurred the interest in the theoretical computer science community: Researchers crafted interactive solutions that were mostly impractical for an actual deployment. With time, more and more effort was put on the design of solutions that bring the theory into more efficient and thus more practical solutions.

[75] Besides, the protocol by Canetti *et al.* [51] allows to detect cheating and honest servers.

3.6.2.1 Interactive solutions

Interactive Proofs (IP) systems for verifiable computation were first proposed by Babai [19] and Goldwasser *et al.* [97]. IP systems, as its name implies, define proofs that are:

Interactive: the prover and the verifier engage a randomized "dialog" where the prover tries to convince the verifier that she has performed the computation correctly;

Probabilistic: the verifier is legitimately convinced of the correctness of the proofs with "very high probability", and falsely convinced with "very small probability" [97].

In such a system, the prover is considered to have infinite power, while the verifier is polynomial-time. Besides, Goldwasser *et al.* [97] introduced the concept of *correct* proof and *sound* proof (*i.e.* it is not possible to return a proof for an incorrect computation) as well as the notion of *efficiency* (that is, the verifier's work should be less demanding than the prover's). Nevertheless, such a system is believed hard to bring to practice.

Muggle proofs. Goldwasser *et al.* [98] brought the concept of IP systems to closer practicality in a real scenario: the verifier is "super-efficient" (i.e runs in quasi-linear time), the honest prover is efficient (i.e runs in polynomial time) and a dishonest prover is unbounded. The authors accordingly named this efficient prover a "Muggle" in comparison to "Merlin" which represents the all-powerful prover in [19]. In the Muggle IP system, there exists a pre-processing phase in which the outsourced computation is translated into a Boolean circuit[76]. Then, the verifier and the prover interact, at the same time the prover evaluates the circuit on some inputs, such that the verifier verifies each step of the computation, that is the output of each gate in the circuit. However, Muggle proofs are efficient only for small circuits and functions that can be parallelized: The work of Goldwasser *et al.* [98] yet suffers from still being impractical. Therefore it has been extended in [66, 173], achieving better performance for the prover (so closer to practicality) by putting some further restrictions on the circuit (parallelization, "sufficiently regular" wiring pattern). Note that the Muggle proof system is publicly delegatable, but not publicly verifiable.

Probabilistically Checkable Proofs. As a consequence of the breakthrough in IP systems, Arora and Safra [10] introduced Probabilistically Checkable Proofs (PCP). In this setting, a proof (say of size n) is encoded such that a verifier can be convinced of its correctness (with a high confidence level) only by querying and checking (via interactions with the prover) a constant number of randomly selected locations in the encoding (of size polynomial in n, thus much longer than the proof). The authors stated the PCP theorem [10, 11] that stipulates that it is not necessary to query and verify the entire proof (which can be very long and hard) for a verifier to be convinced of its correctness. However, while theoretically interesting for the problem of verifiable computation, PCPs are not practical since the encoded proofs are very long yielding huge amount of work for the prover to construct these encodings and for the verifier to check sufficient random locations in the encodings. Consecutive efforts were performed in [28, 29] to construct shorter PCPs, but the generation and verification of such proofs were still not deployable in "the real world". Besides, in an actual deployment of PCP, the soundness of PCPs can be violated: the prover might be tempted to change values of queried locations in the proof so as to answer the verifier's later queries, while speciously matching earlier queries. Therefore, the proof computed by the prover must be predetermined so as to answer all verifier's queries.

Arguments. Following the work on IP and PCP systems, Kilian [110] introduced the concept of *efficient arguments* [110, 111]. While IP systems consider an all-powerful prover and the Muggle setting assumes a polynomial-time honest prover but unbounded (polynomial time) dishonest provers, the work on *efficient argument* systems develop interactive protocols

[76] In a nutshell, a Boolean circuit is a computation model representing a function into a graph whose vertices are logical gates (AND, OR, NOT), whose leaves are the function's inputs and whose roots are the function's outputs.

that are sound against computationally-bounded dishonest provers. In a nutshell, this proof system based on efficient arguments combines PCP with standard *cryptographic primitives* (namely, bit commitments[77] and cryptographic hash functions in [110], as well as Merkle trees [125] in the subsequent publication [111]): the prover produces a (large) encoding of a proof and commits to the bits of this encoding (using a Merkle tree whose leaves correspond to the bits of the encoded proof); the verifier queries a certain number of locations in this encoding and its commitment; the prover reveals the bits at these locations and the verifier checks that they are correct (using the Merkle tree root and the authentication paths). The commitment on the proof (or its encoding) is the key contribution of [110, 111] to circumvent the violation of PCP soundness we mentioned above. However, as it is, the cost of checking a single bit is logarithmic in the proof size. Nevertheless, it is important to stress that the work on efficient arguments was the first to leverage cryptographic assumptions for solutions to the problem of VC. Many proposals ensued from this new setting, opening the way to more practical VC schemes that are efficient in terms of computation and communication complexities. In particular, Ishai *et al.* [104] developed an efficient argument system that considers PCP as a linear function. For a verifier, querying the proof implies requesting the prover to evaluate the function at some inputs selected by the verifier. The scheme then employs a linearly-homomorphic encryption scheme (such as Paillier [138] or El-Gamal [76] cryptosystems) to issue a *(linear) commitment* of the proof: the prover commits a linear function (representing the PCP) to some points chosen by the verifier; then the latter submits inputs to the function (corresponding to the PCP queries); and the prover outputs the results (depicting the PCP responses). The verifier finally checks that the prover's responses are consistent with the commitment. Nonetheless, this protocol requires an expensive pre-processing phase for the verifier; though this phase is amortized over many verification instances (thus, the protocol adopts the amortized model we defined in Section 3.4).

Towards practicality. Practicality is one of the major concerns for designing a viable solution for the problem of verifiable computation. In other words, research work on VC aims at building systems that have practical performance and are simple to implement, while fulfilling the correctness and soundness properties specific to any VC solution. However, the bulk of work we described so far in Section 3.6.2 remains theoretical and no implementations were performed in their respective publications. Starting from the work by Setty *et al.* [161], a collection of research papers proposes actual deployment (with some implementation-oriented refinements) of some of the theories mentioned above. Accordingly, Setty et al. [161, 163] proposed Pepper which implements the protocol of Ishai *et al.* [104] in a real scenario. In their prototype, to verify the evaluation of an arbitrary function $\mathfrak{f}$ by a remote prover, the verifier first encodes $\mathfrak{f}$ into an *arithmetic* circuit that she sends to the prover. Such a kind of circuit is similar to Boolean circuits, except that (i) input and output wires are elements (or variables x, y) of a field; and (ii) gates represent the add operation (*sum gates*) or the multiply operation between two input wires (*product gates*). During a computation instance, the verifier sends input x to the prover who sends back y as the evaluation $\mathfrak{f}(x)$. Also, the prover computes a linear commitment to an encoded proof based on the evaluation of the arithmetic circuit. Then the verifier sends queries to the encoded proof (as in PCP) and the prover responds to them by evaluating the linear function at the verifier-selected points. However, while the verifier's asymptotic costs are efficient, the verifier should still operate a "constant" expensive pre-processing phase (encoding the outsourced function in an arithmetic circuit), that is substantially larger than computing the outsourced function itself. This cost is nevertheless amortized over several instances of the same computation. Another drawback of this protocol concerns "small" computations (in [161], the authors illustrate this drawback with a specific function: matrix multiplication): the computation request and verification

[77]A bit commitment protocol involves two parties, a sender and a receiver. The sender commits to the receiver to a bit b, such that the receiver does not know the value of b. Besides, the sender has no mean to change b after it was committed. Later on, the sender reveals bit b and the receiver can verify that b is really the committed bit.

procedures are more expensive for the verifier than performing the computation locally. As a follow-up to [161], Setty *et al.* [164] describe another improved scheme named Ginger that mainly cuts the costs for the verifier by compressing proof queries and making those queries reusable. While the authors of [163, 164] claim that their solutions apply to the general-purpose verifiable computation scenario, their schemes nonetheless put restrictions on the class of computations that can be verified. Indeed, the encoding of the outsourced function into an arithmetic circuit assumes that that function is "encodable" in an efficient way (namely, this assumption applies to function that are decomposable into several subcomputations such as matrix multiplication). Their consecutive proposal, Zaatar [162] eliminates those restrictions by generating an arithmetic representation of the outsourced function (i.e encoding the function into a quadratic arithmetic program - QAP), as proposed by Gennaro *et al.* [91] (see below, in Section 3.6.2.2). Using QAP also yields shorter proofs in Zaatar than in Ginger. Nevertheless, the encoding into an algebraic representation induces high costs, substantially higher than evaluating the outsourced function. Besides, the amortization of the initial costs by the verifier is only done via batching verifications for several inputs. Subsequent implementations built upon Pepper, Ginger and Zaatar enhance the previous implementations with additional aspects: Allspice [179] attempts to reduce the dependence on cryptographic machinery; Pantry [46] considers a setting where the verifiers do not have the entire input to the outsourced function and leverages the parallelization of the MapReduce paradigm[78] [70]; and Buffet [180] improves Pantry's computational costs.

3.6.2.2 Non-Interactive solutions.

Unlike interactive solutions, schemes that are claimed to work without interaction between the prover and the verifier output the computation result and the corresponding proof in the same message.

CS proofs. Micali [126] was the first to make the transition from interactive to non-interactive solutions for the problem of VC, and proposes Computationally Sound (CS) proofs. Informally, the author combines the rationale behind PCP with efficient argument systems and eliminates the interaction between the prover and the verifier by invoking a random oracle and applying the Fiat and Shamir heuristic [83]. This heuristic, initially applied for a digital signature scheme, works as follows: In an interactive protocol, the prover sends an initial message (representing a proof) to the verifier; then the latter sends a random query to which the prover responds with another message, enabling the verifier to check the proof. In the non-interactive version of the same protocol, the Fiat and Shamir heuristic removes the interaction by having the prover query a random oracle (in practice, evaluate a collision-free hash function) on the initial message, in lieu of the verifier sending the random query. In other terms, the random oracle *simulates* the verifier's query. Basing his work on this approach, Micali [126] introduces the CS proofs. As in PCP [10], the prover in a CS proof system computes a proof of computation π for some function and encodes this proof into a longer and samplable version τ. As in the efficient argument model [110, 111], the bits of encoding τ are then stored in the leaves of a Merkle tree, built using a random oracle. This yield a root value σ. Now recall that in a PCP system, the verifier interacts with the prover to query and check some randomly-sampled bits of encoded proof τ. In the CS proof setting, the prover uses a second random oracle to generate the randomly-sampled locations, and issue the responses of these locations. The CS proof therefore consists in the root value σ, the sampled bits in the encoding τ and their respective authentication paths[79]. The verifier receives the proof and can verify it using the same oracle. The CS proofs are publicly delegatable and

[78]In a nutshell, MapReduce is a programming paradigm enabling parallel and distributed processing of big data in the cloud. The rationale is to divide the processing into small subtasks undertaken by several computing nodes (the *map* phase) and then to aggregate the resulting subtasks' outputs into other outputs which are step by step reduced to the final result.

[79]See Section 6.4.3 for more details on Merkle tree and authentication paths.

verifiable. However, they rely on the random oracle model.

SNARKs. The work by Bitansky *et al.* [36] introduce the concept of Succinct Non-Interactive ARgument of Knowledges (SNARKs). A SNARK protocol removes the use of random oracles in CS proofs and employs instead a concept the authors introduce, namely Extractable Collision-Resistant Hash Functions (ECRH), that rely on the *non-falsifiable* assumption[80] that, given an image of the ECRH there exists an extractor that computes a pre-image. Furthermore, the SNARK approach combines the theory behind CS proofs [126] with an instantiation of a Private Information Retrieval (PIR) protocol (as it was also suggested in [73]): The verifier sends "encrypted" PIR queries on the encoded proof (at randomly selected locations) and the prover responds by executing the PIR on the encoded proof. The SNARK-based protocol defined by Bitansky *et al.* [36] is publicly delegatable but relies on non-standard and non-falsifiable assumptions. Concerning this kind of hardness assumptions, Gentry and Wichs [93] show that it exists an intrinsic limitation on solutions based on SNARKs: they cannot rely on falsifiable assumptions.

Pinocchio. Gennaro *et al.* [91] construct a publicly verifiable computation scheme based on Quadratic Span Program (QSP) and Quadratic Arithmetic Program (QAP), upon which they propose a new SNARK protocol. Indeed, the authors introduce the concept of QSP to convert any Boolean circuit and QAP to translate any arithmetic circuit. As in previous schemes [98, 161, 163, 164], the Boolean or arithmetic circuit encodes any arbitrary function. The choice between QSP and QAP depends on the outsourced function; each of the encoding may yield different performance, according to the "structure" of the function. The rationale behind QAP is as follows (similar considerations apply to QSP): each gate in an arithmetic circuit encoding an outsourced function is replaced by a quadratic polynomial which is in turn encoded in the exponent (such that the resulting encoding belongs to a bilinear group). The set of all encoded polynomials represents the QAP outsourced to the prover along with the function. Then, given the verifier's (encrypted) inputs, the prover evaluates the circuit and produces a proof based on QAP's polynomials. All polynomial evaluations (i.e gate computations) are then (succintly, i.e using constant amount of time) verified in the exponent, using bilinear pairings. The PVC scheme deriving from the above idea shows great promising practical performance. Indeed, this work was brought into practicality with a prototype called Pinocchio [144], which implements the QAP-based scheme. In particular, benchmarks on Pinocchio shows that the verifier (more precisely the user who outsources the function) operates a one-time pre-processing phase, that enables an unbounded number of verifications. But the workload at the prover is still substantial. Compared to Zaatar [162] that also implements an instantiation of a QAP [91], Pinocchio induces more expensive costs, since it uses more cryptographic machinery, but has the advantage to be non-interactive and publicly verifiable (which Zaatar is not). Despite its close-to-practicality property, the scheme by Gennaro *et al.* [91] and Pinocchio [144] rely on a *non-falsifiable* assumption (namely, the power knowledge of exponent), that is a non-standard assumption, and thus on which we have not a high confidence on it.

Use of FHE. Gennaro *et al.* [90] formalize the notion of non-interactive verifiable computation in the *amortized model*, whereby the verifier must execute a one-time expensive pre-processing operation to allow an unbounded number of efficient verifications. Their solution combines the use of garbled Boolean circuits [191] with FHE [92]: During the pre-processing phase, the function to be outsourced is encoded into a Boolean circuit using Yao's garbled

[80]Gentry and Wichs [93] gave a definition of falsifiable assumptions (thus giving a characterization of non-falsifiable ones). In essence, falsifiable assumptions capture the idea that there exists an "efficient process to test whether an adversarial strategy falsifies (*i.e.* breaks) the assumption" [93]. This idea is modeled as an interactive game between a challenger and an adversary as in the soundness experiment we described in Section 3.5. Under falsifiable assumptions (such as RSA, CDH or Decisional Diffie-Hellman (DDH)), the probability that an adversary wins the game is negligible. In contrast, non-falsifiable assumptions are considered to be non-standard and "harder to reason about" [93]. Thus they appear to be stronger assumptions than falsifiable ones [141]. Variants of the knowledge-based assumption (such as the Knowledge of Exponent Assumption) fall into the category of non-falsifiable assumptions.

circuit construction [191], which associates random labels to each wire in the circuit. A computation request generates the labels associated with an input, and as a response, the prover computes the labels associated with the output, based on the garbled circuit and the input labels. The verifier will then verify that the output labels correspond to a correct evaluation of the outsourced function. Since a computation request reveals the labels to the prover, they cannot be used for subsequent verifications. The application of a FHE scheme to the labels enables to protect those labels and hence reuse the same circuit for multiple verifications. A similar idea to combine circuits and FHE was proposed in [64, 65]. However, at the time of writing this thesis, computational overhead of FHE is still prohibitive, limiting its practicality for actual implementation. Besides, these solutions only allow private verifications, that is, they are not publicly delegatable nor publicly verifiable.

Using Attribute-Based Encryption (ABE). Parno *et al.* [143] propose a solution for public delegation and public verification of computation using ABE [99]. In an ABE scheme, a (secret) decryption key is associated with a Boolean function $\mathfrak{f}$. This key can decrypt a ciphertext that results from the encryption of a message m under an attribute x, if and only if $\mathfrak{f}(x) = 1$. Based on this concept of ABE, Parno *et al.* [143] design a VC scheme for arbitrary (Boolean) functions. In a nutshell, a user requests the prover to evaluate some Boolean function $\mathfrak{f}$ on some input x. The computation query consists of the encryption of a random message m using the underlying ABE technique that associates the resulting ciphertext with input x. In other terms, x becomes an attribute of the encryption of m. This means that m can be decrypted using the (secret) decryption key associated with function $\mathfrak{f}$ if and only if $\mathfrak{f}(x) = 1$. The prover who is given the key and the ciphertext that encrypts m can now prove that $\mathfrak{f}(x) = 1$ if he can return the decrypted message m. Public delegatability and verifiability are therefore achieved with the use of ABE. However, this scheme is limited to the computation of Boolean functions that output a single bit. For functions with more than one output bit, the verification has to be repeated (for each output bit).

Homomorphic MACs and signatures. Another line of work designs homomorphic message authenticators [89, 2] or homomorphic signatures [106, 39, 57]; the former allows private verification (i.e only the holder of a secret key can verify the authenticators) while the latter enables public verifiability. Such homomorphic primitives have been first considered in the context[81] of *linear network coding* [3] where several authenticators are linearly combined into a single tag that proves the correct linear combination of the underlying messages. Subsequent work have addressed a larger class of functions (other than linear operations as it was the case for network coding).

Homomorphic MACs. Gennaro and Wichs [89] define a new primitive referred as *fully homomorphic message authenticators.* With this construction, the prover can perform arbitrary computations (in particular, Boolean functions) on authenticated data and (homomorphically) generate an unforgeable tag (i.e a succinct authenticator) that certifies the correctness of the computation, without resorting to any secret key. Such a secret is only used to verify the tag, in order to validate the result of the computation over the authenticated data. Notwithstanding, the verifier (who was given the secret key used to compute the tag by the entity that outsourced the data) is able to verify the authenticators without knowing the data itself. The construction and verification of the tag are based on a FHE scheme, such as the one proposed by Gentry [92], thus requiring heavy cryptographic computations. In particular, as mentioned by Gennaro and Wichs [89], the verification of fully homomorphic message authenticators is not more efficient that evaluating the function. The authors suggest to outsource the verification step to the prover using existing protocols for verifiable computation such as CS proofs [126] or SNARKs [36]. A new construction for homomorphic MAC was proposed by Catalano and Fiore [55] for functions that can be encoded into an arithmetic circuit (whose gates are either additive or multiplicative). This new homomorphic MAC generates

[81] Homormorphic authenticators also appear in the context of proofs of retrievability and provable data possession [14, 165]. See Part I for more details.

for each gate a tag as a 1-degree polynomial $P_m(X) = aX + m$ whose constant term m is the authenticated data, and which evaluates to a pseudo-random number (generated by a PRF) on some secret input s; that is, $\mathsf{HomMAC}(m, s) = P_m(s) = as + m = r$, where HomMAC is the homomorphic MAC on data m and $r = \mathsf{PRF}(s)$. The natural homomorphism property of these polynomials ensures the homomorphism of the MAC. Therefore, when evaluating the circuit, the prover can produce a tag at each internal gate without knowing the secret key used to authenticate the underlying data. Gate by gate, the prover obtains a single tag for the entire circuit evaluation. Hence the verifier checks that the resulting tag is a correctly produced tag. This new solution is less general than the one in [89] in the sense that it applies only to polynomially-bounded arithmetic circuits (as opposed to Boolean functions in [89]), but it does not rely on the burden of FHE and presents a more efficient (thus practical) verification process. However, the size of the MAC heavily depends on the degree of the arithmetic circuit. Indeed, multiplicative gates in the circuit increases the degree of the "polynomial" tag. Consequently to this work, Catalano *et al.* [56], Backes *et al.* [20] and Zhang and Safavi-Naini [192] propose other frameworks for fully-homomorphic MAC that ensure succinct authenticators and efficient verification. In particular, the homomorphic MAC propounded by Backes *et al.* [20] resort to Algebraic Pseudo-Random Functions (aPRF)[82] to generate the pseudo-random number used as the evaluation of the 1-degree polynomial $P_m(X)$ on the secret input s [55]. This strategy makes the verification process more efficient than in [55]. However, the framework introduced by Catalano *et al.* [56] and Zhang and Safavi-Naini [192] rely on the multilinear map abstraction, and corresponding hardness assumptions, which are not yet straightforwardly practical [118, 87]. Besides, the MAC of Backes *et al.* [20] only apply to a restricted type of homomorphism (namely, evaluation of arithmetic circuits of degree up to 2, i.e encoding quadratic polynomials).

Homomorphic signatures. Homomorphic signatures [39, 57] are the "public" version of homomorphic MACs in the sense that anyone having access to the verifying public key can verify a homomorphic signature. Similarly to homomorphic MACs, the prover can compose such signatures into a single signature without knowledge of the secret signing key. However, to the best of our knowledge, homomorphic signatures have been essentially developed for the verification of *polynomial* functions on authenticated data. Boneh and Freeman [39] devise polynomially-homomorphic signatures based on lattices. The verification procedure is as costly as computing the function from scratch. This scheme is proven secure under the Random Oracle Model (ROM). Catalano *et al.* [57] eliminate the ROM assumption and design new polynomially-homomorphic signatures that are secure under a more standard model (in particular, they are based on problems in groups that admit multilinear maps). In addition, the verification of a signature homomorphically computed for a function $\mathfrak{f}$ of authenticated data is more efficient than the computation of $\mathfrak{f}$. As a matter of fact, this property is verified in the the amortized model: after a one-time pre-computation of function $\mathfrak{f}$, the verifier can check an unbounded number of signatures on $\mathfrak{f}$ efficiently, that is, check the evaluation of $\mathfrak{f}$ on any data efficiently.

AD-SNARKs. One of the most recent techniques proposed for the problem of VC, and which has retained our attention, combines ideas stemming from QAPs and their implementation in Pinocchio [144] with the rationale of (linearly) homomorphic MACs [55]. AD-SNARKs [21] (SNARKs for Authenticated Data) extends Pinocchio with the privacy-preserving feature. Indeed, AD-SNARKs support operations where the prover gets the inputs form a trusted source that produce and authenticate them. A third-party verifier, that has no knowledge of the data, can request and verify computations but learn nothing but the outputs and their correctness. To render Pinocchio privacy-preserving, Backes *et al.* [21] propose to shift to the prover some parts of the verification procedure that involve linear computations of the inputs (as in QAPs). Hence, linearly-homomorphic MACs are computed to authenticate this part of the verification procedure to the verifier. However, as it was the case for Pinocchio,

[82]See Section 3.6.3.1 for more details on aPRF.

AD-SNARKs must rely on non-falsifiable assumptions.

3.6.3 Proof-based Function-Specific Solutions

All the proposals we describe in the previous section realize solutions to the VC problem for a broad range of computations, or perhaps for arbitrary functions satisfying some restrictive properties (parallelizable functions as in [98] or Boolean functions as in [143]). Nevertheless, some effort has been devoted to protocols that offer solutions for a specific class of functions. As a matter of fact, these protocols exploit the peculiar properties and structures of some functions to enable tailored efficient delegation and verification solutions. Among others, polynomial evaluation [30, 85, 193, 141], matrix computation [85, 195, 193], set operations [52, 140] and keyword search [30, 198] have received significant interest from the research community. Linear algebra (such as matrix inversion or computing the rank and determinant of a matrix or solving a linear equation system) have also given rise to notable publications. The investment in those specific functions can be explained by the fact that they can be used as primitives for broader problems, such as outsourced image recognition or outsourced data mining. Benabbas *et al.* [30] pioneered a new body of research for *practical* VC protocols. Adopting the amortized model approach introduced by Gennaro *et al.* [90], they initiate the concept of Algebraic Pseudo-Random Functions (aPRF) that enables efficient verifications, at the cost of an expensive one-time pre-processing operation that prepares the function to be outsourced. In particular, the authors removed the burden of relying on heavy cryptographic tools, such as FHE as in [90, 64] or on expensive circuit encoding and evaluation as in [91, 144].

3.6.3.1 Polynomial evaluation.

In the verifiable polynomial evaluation problem, a data owner outsources a polynomial of large degree to a server and then requests the server to evaluate that polynomial for several inputs, such that a verifier can verify the correctness of the result.

In the polynomial evaluation scheme of Benabbas *et al.* [30], the verifier outsources a polynomial A of degree d and coefficients $a_i \in \mathbb{F}_p$ $(0 \leq i \leq d-1)$ where p is a large prime, to the prover. She also sends a vector of elements of the form $g^{\alpha a_i + r_i}$, where the r_i's are the coefficients of a polynomial R, of degree d and α is randomly selected from $\mathbb{F}_p$. On input of some point x selected by the verifier, the prover returns the evaluation $y = A(x)$ and a proof of correct evaluation $\pi = g^{\alpha A(x) + R(x)}$. The verifier accepts the result if $\pi = g^{\alpha y + R(x)}$. It is obvious that R should be generated in such a way that the verification is efficient. Otherwise, if R was completely random, the verification would require the same amount of computation as evaluating A directly, thus outsourcing would be useless. The idea to optimize the verifier's work is to produce polynomial R with the help of an aPRF F_K: F_K is a pseudo-random function that has a special property called *closed form efficiency* such that anyone who knows the secret key K can efficiently (namely sub-linearly in d) compute polynomial $R(x)$, where the coefficients of R are defined as $r_i = F_K(i)$. This solution, however, only works in the symmetric-key setting; thus it does not enable public verifiability nor delegatability. In the same line of work, Fiore and Gennaro [85] devise new aPRFs and combine them with bilinear pairings to develop a publicly verifiable protocol for polynomial evaluation. As a follow-up to the work of Fiore and Gennaro [85], Zhang and Safavi-Naini [193] propose a solution that trades off storage at the prover and the computational costs for the verifier. Indeed, in these two schemes [85, 193], outsourcing polynomial A together with the auxiliary polynomial R doubles the storage overhead for the prover. Therefore, to reduce such costs, the authors in [193] leverage the aPRF for publicly verifiable polynomial evaluation introduced in [85] and break the delegated evaluation into several sub-computations, all of them verifiable in a single proof.

Another solution for public verification considers Signatures of Correct Computation (SCC) [141]. As its name implies, this solution applies to the public-key setting and allows public verifiability. Besides, public delegatability is possible since the scheme does not require any secret to submit a computation query to the prover. SCC employ polynomial commitments [108] to construct the signatures. In a nutshell, the authors in [108] observe that the polynomial $A - A(\alpha)$ is divisible by the polynomial $X - \alpha$ (for any polynomial $A \in \mathbb{F}_p[X]$ and $\alpha \in \mathbb{F}_p$). We can then find a polynomial W such that $A(x) - A(\alpha) = (x - \alpha)W(\alpha)$. Based on this property, the prover in [108] constructs a witness by encoding polynomial W to the exponent in some group that admits a bilinear pairing; the verifier only needs to verify that the equation involving $A(x)$ and $W(x)$ holds in the exponent. The work by Papamanthou *et al.* [141] extends the above solution for multivariate polynomials.

3.6.3.2 Matrix Multiplication.

In this setting a data owner outsources a large matrix M to a server and then requests the server to multiply matrix M with an input vector $\vec{x}$. A verifier can then check the validity of the output returned by the server. This vector-matrix multiplication can generalize to matrix-matrix multiplication by applying the mentioned scenario to each column of the input matrix.

To our best knowledge, Atallah and Frikken [12] were the first to specifically tackle the problem of verifiable matrix multiplication. The authors give an insight of an idea later formalized by Fiore and Gennaro [85]. In [12], the actual matrix is outsourced together with an auxiliary matrix, called "random noise" that is used to verify the correct matrix multiplication. In [85], a data owner outsources a $n \times m$ matrix M of elements $M_{i,j}$ to the prover together with an auxiliary matrix N of the form $N = \alpha M + R$, where α is a random number and R is a $n \times m$ random matrix. The data owner wishes to compute the product $\vec{y} = M\vec{x}$ for an m-sized vector $\vec{x}$. The prover produces the proof $\vec{\pi} = N\vec{x}$. The verifier can then check that $\vec{\pi} = \alpha\vec{y} + R\vec{x}$. All these operations are performed in the exponent; namely elements of N are of the form $N_{i,j} = g^{\alpha M_{i,j} + R_{i,j}}$ to provide secrecy of random matrix R. This scheme provides public verifiability through the use of bilinear pairings that encode a public verification key to the form $e(g,g)^{R\vec{x}}$. As in the polynomial case described above, Fiore and Gennaro [85] suggest to generate the auxiliary random matrix R using dedicated aPRFs to allow the verifier to efficiently check the proof. As in the polynomial case, Zhang and Safavi-Naini [193] revisit the scheme of Fiore and Gennaro [85] to reduce the storage overhead induced by the outsourcing of the matrices M and R. However, public delegatability is not supported by these two protocols. On the other hand, Zhang and Blanton [195] propose a publicly delegatable and verifiable scheme that does not employ aPRF but instead leverage basic matrix properties, with comparable costs than the ones induced by the scheme of Fiore and Gennaro [85]. Nonetheless, in spite of the additional feature of public delegatability, the work by Zhang and Blanton [195] relies on the non-standard multiple Decisional Diffie-Hellman (m-DDH) assumption. Besides, the generation of the computation request (ProbGen) is costly and not amortized over multiple verifications. Indeed, the setup of this scheme does not depend on a particular matrix, so the generation of the matrix multiplication request should be repeated each time a user outsources that computation to the prover.

The work of Mohassel [127], also mentioned in [85, 193, 195], also addresses the problem of verifiable matrix multiplication. The author gives a general approach in which a verifier that wishes to outsource the multiplication of two matrices A and B, precomputes and stores $\vec{y} = B\vec{x}$ and $\vec{y}' = A\vec{y}$ (where $\vec{x}$ is a random vector). Later, she receives from the prover the product $C = AB$. To verify that $C = AB$, the verifier checks that $\vec{y}^* = C\vec{x} = \vec{y}'$. In this approach, the verifier costs are induced by the computation of $\vec{y}$, $\vec{y}'$ and $\vec{y}^*$ which is substantially less than the computation of AB. Besides, this scheme ensures privacy of the matrices and their product by employing a homomorphic encryption scheme (such as Paillier's [138], El-Gamal's [76], etc.) that encrypts each element of the input matrices. The verifier is then able to verify the correctness of the computation without the need to decrypt its result. However, this protocol is not very efficient for the verifier who first has to encrypt the matrices, and computing the vectors $\vec{y}$, $\vec{y}'$ and $\vec{y}^*$.

Note finally that the work by Thaler [173] applies his *interactive* proof system (see Section 3.6.2.1) to the parallelizable problem of matrix multiplication. The operation is encoded into an arithmetic circuit whose evaluation is efficient for the prover.

3.6.3.3 (Conjunctive) Keyword Search.

In this setting, a data owner stores to a server a large amount of data and delegates to that server the keyword search operation, such that the verification of the search is less costly that the operation of search for the data owner.

The problem of verifiable keyword search finds some early solutions in the domain of verifiable polynomial evaluation as explained in [30, 85]: A document is encoded as a polynomial P whose roots are exactly the keywords contained in the document. To prove whether a keyword ω is present in the document, the prover returns $P(\omega)$ to the verifier along with the proof of the correct evaluation of P on ω (using a verifiable polynomial evaluation scheme). Thereafter, the verifier runs the underlying verification procedure: If $P(\omega) = 0$ and the proof is valid, then the verifier acknowledges that the keyword exists in the document; if $P(\omega) \neq 0$ and the proof is valid, then the verifier is convinced that the keyword does not appear in the document; otherwise (when the proof is not valid), the verifier decides that the evaluation (thus the search) is not correct. Although this solution is convenient to test whether a single keyword appears in a document, it does not efficiently support conjunctive keyword search. Besides, to search for a single keyword in multiple documents, the prover has to search for each document, one by one, and thus generate a proof for each document. The cost of such a scheme is linear in the number of documents outsourced to the server.

The problem of verifiable *conjunctive* keyword search is closely related to the ones of verifiable set operations (and in particular set intersection). Indeed, many search algorithms consider a collection of keywords organized into an *inverted index* data structure [22]: Each keyword in a database is mapped to the set of all the database files that include this keyword. A query for keywords ω_1 and ω_2 targets the records that are present in both sets mapping ω_1 and ω_2 respectively, namely the intersection of these two sets. Therefore, a verifier should be able to verify that the prover outputs the correct intersection, yielding the correct search result.

Verifiable Set Intersection for Keyword Search. Morselli *et al.* [129] first explore the concept of verifiable set operations and design a tool based on RSA accumulators[83] [31] and Counting Bloom Filters (CBFs) [81]: a data owner produces a digest of her sets by (i) creating for each set a CBF of the elements included in the considered set; (ii) generating an RSA accumulator for each set. Both filters and accumulators are then signed. When the prover receives an intersection query, she returns the elements of the intersection, the CBF of each set, their accumulators, the RSA witnesses (proving that the intersection is included in each intersected set) and the signatures on the filters and accumulators. The verifier then checks that the signatures and the witnesses are correct (that is, they satisfy a particular equation involving the witnesses, the RSA accumulators and the intersection). However, this scheme incurs linear verification costs for the verifier. In addition, while it supports public verification, it does not offer public delegatability.

Papamanthou *et al.* [140] also tackle the problem of verifiable set intersection and claim that their solution is suitable for the problem of verifiable keyword search. Their scheme exploits the primitive referred as *polynomial-based accumulators*[84] [132] and devises a new authenticated data structure, called *accumulation tree*, built upon these accumulators that allow to efficiently query and update the elements in the sets. Here also, the keywords contained in a collection of outsourced database records are viewed as an inverted index: To each keyword ω_i corresponds a set S_i whose elements are pointers to documents where

[83]This primitive is briefly explained in Section 6.4.2.

[84]See Section 6.4.2 for more details.

keyword ω_i is present. To query a conjunction of keywords $\omega_1, .., \omega_k$, the prover should return the intersection $I = \bigcap_{i=1}^{k} S_i$ (i.e I contains the pointers to documents that contain all keywords in the conjunction) along with a proof of the correct intersection. In a nutshell, the authors reduce the problem of verifiable set intersection into two subproblems: (i) *subset containment* which allows to prove that I is contained in each S_i; and (ii) *set disjointness* which proves that $\bigcap_{i=1}^{k} S_i \backslash I = \emptyset$. Besides, the solution proposed by Papamanthou *et al.* [140] encodes each set involved in the intersection with a polynomial of degree equal to the cardinal of the set and whose roots are the elements in the set. This polynomial is then encoded to the exponent: this corresponds to the polynomial-based accumulator. The homomorphic characteristics of this kind of accumulators enable the verification of relations between the sets thanks to arithmetic properties between the accumulators. Therefore, the subset containment property can be expressed in terms of polynomial: for a set S_i, its encoding polynomial divides the polynomial encoding the intersection I. Hence, to verify that the intersection is contained in each intersected set S_I, the verifier is required to verify the divisibility criterion of the corresponding polynomials, using their respective accumulators and using bilinear maps. This criterion can be publicly and efficiently verified in the exponent. Similarly, the set disjointness property can be reduced to checking that the polynomials' encoding $S_i \backslash I$ are mutually co-prime, such that they satisfy the Bézout identity[85]. The verifier can publicly and efficiently check this identity in the exponent using the corresponding polynomial-based accumulators and bilinear maps. Thus, this solution for verifiable set intersection can be exploited for conjunctive keyword search. The work of Papamanthou *et al.* [140] has been extended by Canetti *et al.* [52] in order to compose multiple set operations while having the same security guarantees as defined in [140][86]. Sun *et al.* [172] leverage the techniques of Papamanthou *et al.* [140] and Canetti *et al.* [52] to propose a verifiable conjunctive keyword search on dynamic encrypted data, allowing to update the outsourced data collection.

Kosba *et al.* [113] introduce TrueSet, a protocol for verifiable set operations based on circuit that is applicable to the problem of keyword search as the authors mention. In a nutshell, Kosba *et al.* [113] introduces the concept of Quadratic Polynomial Programs (QPP) analogous to QAP defined in [91]. QQPs encode a *set circuit* in order to derive efficient SNARK. A set circuit is a circuit whose gates implement set operations such as union, difference and intersection. This kind of circuit can be represented as a polynomial circuit where every wire is a polynomial and every gate defines either polynomial addition or polynomial multiplication. The scheme offers an elegant technique that combines QPPs and SNARK, but as these building blocks, TrueSet relies on a non-falsifiable assumption (the power knowledge of exponent assumption).

Verifiable Conjunctive Keyword Search on Encrypted Data. Several works [198, 59, 112, 122, 171] focus on verifiable search on encrypted data. These proposals aim at satisfying three features for outsource databases: (i) Privacy protection on the data via encryption; (ii) Keyword search over encrypted data; and (iii) Verifiability of the search. Zheng *et al.* [198] develop the concept of Verifiable Attribute-Based Keyword Search (VABKS) which allows the owner of a database to grant a user (a verifier) satisfying an access control policy the right to query a keyword over the owner's outsourced encrypted files and to verify the search result returned by the remote server (the prover). This protocol makes use of basic cryptographic primitives such as ABE, Bloom filters and digital signature and create a new building block called *attribute-based keyword search.* This new primitive organizes the keywords according to their respective access control policies and authorizes the user satisfying an access control policy to conduct keyword search over the encrypted data. However, VABKS does not allow

[85]The Bézout identity states that if D is the GCD of two polynomials A and B then there exist two polynomials U and V such that $AU + BV = D$.

[86]Note that these papers [140, 52] also propose techniques for set union and difference. But in the context of verifiable keyword search only intersections are of interest.

public delegatability or verifiability since the verifier and the user who searches for some keywords should satisfy an access control policy. Besides, VABKS does not efficiently enable the search for a conjunction of keywords.

Cheng et al. [62] propose a protocol for verifiable conjunctive keyword search that leverages a combination of a searchable symmetric encryption scheme with an *indistinguishability obfuscation circuit* (iO circuit) realizing the search operation. While public verifiability is achieved by means of another (public) iO circuit representing the verification function, public delegatability is not addressed in this work. Nevertheless, it is worth considering generating an additional iO circuit to realize the publicly delegatable property. Still, the generation and obfuscation of such circuits induce substantial costs that the authors in [62] barely mention.

In the same line of work, Chai and Gong [59] consider a prover that may correctly operate only a fraction of search queries to save both computation and bandwidth. This prover is referred as *semi-honest-but-curious*[87]. The scheme relies on a MAC-based *prefix tree* that allows the prover to perform the search on encrypted data and the verifier to verify the correctness of the result. As it relies on MAC to authenticate the responses, this solution offers neither public verifiability nor public delegatability. This work was extended in [112] for verifiable phrase search. This solution boils down to performing a single keyword search for each keyword in the targeted phrase and to computing the intersection of the server's responses.

Verifiable SQL databases. A certain number of work focuses on systems that enable a verifier to perform verifiable SQL queries over relational databases [168, 139, 190, 196]. One of the most recent proposal on that field is IntegriDB, proposed by Zhang *et al.* [196]. It is based on the accumulator-based verifiable set operations described in [140, 52]. Besides, the authors suggest to create another tree, called interval tree (a sort of Merkle tree that has the properties of a traditional search tree) in order to enable verifiable join and range queries. Unfortunately, this tree as to be constructed for each pair of columns of each table of the outsourced database. Namely, if the database consists of k tables with n rows, then the data owner has to compute $k \cdot \binom{n}{2}$ interval trees. Besides, each tree involves the computation of an accumulator of all the row values for a given column. Since the database is large, the number of rows might be very important. Thus the computation of the interval tree is computationally demanding.

3.6.4 Conclusions of the State of the Art Analysis

From the above survey, we can draw the following conclusions on the gap that exists between the requirements identified in Section 3.2.2 and existing work. Table 3.1 gathers and compares the results of our analysis of the state of the art.

1. Interactive solutions for general-purpose functions are far from being practical. Indeed, they require substantial computation and yield large-sized proofs.

2. Some related work on non-interactive proofs for arbitrary functions either are secure in the Random Oracle Model (ROM) (CS proofs [126]) or rely on non-falsifiable assumptions (SNARK [36]). Moreover, solutions that are based on FHE [90, 64] are not practical yet and not publicly delegatable nor publicly verifiable. The scheme by Parno *et al.* [143] is not fully applicable to any function, since the authors put restrictions on Boolean functions only.

3. Proposals for homomorphic MACs and signatures appear to not fulfill the efficiency requirement since verifying the MAC or the signature is as costly as executing the function locally.

[87] In [59], a *semi-honest-but-curious* server refers to a server that "may execute only a fraction of search operations honestly and/or return a fraction of search outcome honestly".

4. Function-specific solutions are believed to be more efficient than the solutions for arbitrary computations. However, no function-specific solutions are publicly delegatable and publicly verifiable while fully satisfying the efficiency requirement at the same time.

Therefore, we advocate for solutions that are:

- Efficient for the verifier (while we allow the execution of a one-time pre-processing operation, following the amortized model)
- Non-interactive
- Based on falsifiable assumptions
- Publicly delegatable and publicly verifiable.

		Efficiency							**Features**		
		General	Specific	Interactive	Non-Interactive	Amortized model	Practical	Implementation	Public Delegatability	Public Verifiability	Privacy-Preserving
Interactive proofs	Babai [19]	✓		✓							
	Goldwasser *et al.* [97]	✓		✓							
	Goldwasser *et al.* [98]	✓		✓					✓		
	Arora and Safra [10]	✓		✓		✓					
	Kilian [110]	✓		✓							
	Ishai *et al.* [104]	✓		✓		✓					
	Setty *et al.* [161]	✓		✓		✓	✓	✓			
	Setty *et al.* [163]	✓		✓		✓	✓	✓			
	Setty *et al.* [164]	✓		✓		✓	✓	✓			
	Setty *et al.* [162]	✓		✓		✓	✓	✓			
Non-interactive proofs	Micali [126]	✓			✓				✓	✓	
	Bitansky *et al.* [36]	✓			✓				✓	✓	
	Gennaro *et al.* [91]	✓			✓	✓	✓	✓		✓	
	Parno *et al.* [144]	✓			✓	✓	✓	✓		✓	
	Gennaro *et al.* [90]	✓			✓	✓					✓
	Parno *et al.* [143]				✓		✓		✓	✓	
	Gennaro and Wichs [89]	✓			✓						
	Backes *et al.* [20]	✓			✓	✓					
	Boneh and Freeman [39]	✓			✓					✓	
	Catalano *et al.* [57]	✓			✓					✓	
	Backes *et al.* [21]	✓			✓		✓	✓		✓	✓
Polyn.	Benabbas *et al.* [30]		✓		✓	✓	✓				
	Fiore and Gennaro [85]		✓		✓	✓	✓			✓	
	Zhang and Safavi-Naini [193]		✓		✓	✓	✓			✓	
	Papamanthou *et al.* [141]		✓		✓		✓		✓	✓	
Matrices	Fiore and Gennaro [85]		✓		✓	✓	✓			✓	
	Zhang and Safavi-Naini [193]		✓		✓	✓	✓			✓	
	Zhang and Blanton [195]		✓		✓		✓		✓	✓	
	Mohassel [127]		✓		✓	✓	✓				✓
Search	Morselli *et al.* [129]		✓		✓		✓	✓		✓	
	Papamanthou *et al.* [140]		✓		✓		✓		✓	✓	
	Kosba *et al.* [113]		✓		✓	✓	✓	✓		✓	
	Zheng *et al.* [198]		✓		✓	✓	✓	✓	AC	AC	✓
	Chai and Gong [59]		✓		✓						✓

AC means for users satisfying an access control policy.

Table 3.1: Comparison of relevant existing solutions for Verifiable Computation

Chapter 4

Verifiable Polynomial Evaluation

4.1 Introduction to Verifiable Polynomial Evaluation

In this chapter, we focus on the public delegatability and verifiability of a specific family of functions: namely large-degree polynomial evaluation.

Let us recall our space agency scenario. The agency outsources to a cloud server a considerable collection of high-resolution (digital) space images. As we mentioned in Section 3.2, these images can be represented as a matrix where each element stores a pixel of the image. For example, to date, the highest resolution for an image produced by the Hubble Telescope is 1.5 billion pixels, which can be represented by a 60,000-by-22,000 matrix[88]. Thus, owing to its huge size, the processing of this kind of images will also be offload to the cloud, since the cloud can afford computationally demanding operations.

In particular, many techniques for image processing employ polynomial evaluation as a primitive. Among others, we can cite some contour detection or background estimation techniques that require to evaluate a (possible high-degree) polynomial in each pixel of the image. For example, in [177], the authors estimate and remove background from space images using polynomial-based techniques: Each pixel belonging to the background is fit to a polynomial (using a least-square polynomial fitting method[89]). The background is then removed from the considered pixel by evaluating the estimated polynomial on the coordinates of this pixel and subtracting the resulting estimate from the considered pixel. Due to their possible heavy costs, the international space agency outsources such polynomial evaluations to the cloud and wants to be ensured of the correctness of their results.

This scenario fits clearly into the VC model where data owner $\mathcal{O}$ represents the space agency. It wishes to delegate to cloud server $\mathcal{S}$ the computation of a polynomial A of large degree d that encodes an expensive polynomial-evaluation-based image processing operation. Precisely, if we apply Definition 6, we let $\mathfrak{f} = A$ and data owner $\mathcal{O}$ executes algorithm Setup; cloud server $\mathcal{S}$ runs $\mathsf{Compute}$ and returns the result to $\mathcal{O}$. Additionally to the outcome of the computation, data owner $\mathcal{O}$ should receive some cryptographic proofs of correct polynomial evaluation from cloud server $\mathcal{S}$. Furthermore, the space agency solicits public delegatability and verifiability of the computation to enhance collaborative work within universities and research centers all around the world. For instance, any space researcher can submit an input to that polynomial and another researcher may verify the results returned by the cloud. Hence, the first space researcher corresponds to querier $\mathcal{Q}$ in Definition 6. She runs algorithm

[88]NASA, ESA, J. Dalcanton (University of Washington, USA), B. F. Williams (University of Washington, USA), L. C. Johnson (University of Washington, USA), the PHAT team, and R. Gendler, “Sharpest ever view of the Andromeda Galaxy”, ESA/Hubble Media Newsletter, January 5, 2015, http://tiny.cc/uift8x [Accessed: February 3, 2016].

[89]This method uses a polynomial $P(x) = a_0 + a_1 x + ... + a_n x^n$ to approximate a series of data of the form (x_i, y_i), where $y = f(x_i)$ for some function f $(1 \leq i \leq m)$, with estimation $(x_i, \hat{y}_i)$, where $\hat{y}_i = P(x_i)$. The least-square fit ensures that the coefficients of P are selected such that the fitting error $\sum_i (y_i - \hat{y}_i)^2$ is minimized.

ProbGen on some input x. The second researcher acts as verifier $\mathcal{V}$. She receives result y and proof π from cloud server $\mathcal{S}$ and invokes algorithm Verify to check whether the returned value y actually equals $A(x)$.

4.2 Protocol Overview

Having set the above motivating scenario, we will devise in the following sections a protocol for publicly delegatable and verifiable evaluation of polynomials.

The solution we propose draws upon the basic properties of *Euclidean division* of polynomials: for any pair of polynomials $A \neq 0$ and $B \neq 0$ of degrees d and 2 respectively, the Euclidean division of A by B yields a unique pair of polynomials Q and R such that: (i) $A = QB + R$; and (ii) the degree of *quotient* polynomial Q equals $d-2$, whereas the *remainder* polynomial R has a degree less than 1.

Now, data owner $\mathcal{O}$, who would like to outsource the evaluation of a polynomial A of degree d to cloud server $\mathcal{S}$, runs algorithm Setup (as defined in Definition 6) which first defines a polynomial $B(X) = X^2 + b_0$ for a randomly chosen b_0, and divides A by B to get the quotient polynomial $Q(X) = \sum_{i=0}^{d-2} q_i X^i$ and the remainder polynomial $R(X) = r_1 X + r_0$. Next, data owner $\mathcal{O}$ outsources polynomial A together with quotient polynomial Q to server $\mathcal{S}$ and publishes the *public* key $\mathsf{PK}_A = (g^{b_0}, g^{r_1}, g^{r_0})$, where g is the generator of a well-defined cyclic group. Consequently, whenever a querier $\mathcal{Q}$ wants to evaluate polynomial A at point x, she invokes algorithm ProbGen which first computes and advertises the *public* verification key $\mathsf{VK}_x = (g^{B(x)}, g^{R(x)})$, and then transmits x to server $\mathcal{S}$. The latter in turn calls algorithm Compute which returns $y = A(x)$ and generates the proof $\pi = Q(x)$. Given the server's output (y, π), a verifier $\mathcal{V}$ checks whether $g^y = (g^{B(x)})^{\pi} g^{R(x)}$.

This protocol meets the efficiency requirement defined in Section 3.4. Indeed, the efficiency of the verification in the solution stems from the fact that B and R are small-degree polynomials. Indeed, to verify the correctness of a result (y, π) provided by server $\mathcal{S}$ on an input x, algorithm Verify performs a small and constant number of computations as opposed to carrying out the $\mathcal{O}(d)$ exponentiations that are required to evaluate polynomial A.

It is clear that to meet the soundness requirement stipulated in Section 3.5, the description of polynomials B and R must remain secret. However since B is a two-degree polynomial, the secrecy of these two polynomials can be easily compromised by disclosing the quotient polynomial Q. To remedy this shortcoming, the client encodes polynomial Q using an *additively homomorphic one-way* encoding. Namely, each coefficient q_i of polynomial Q is encoded as h^{q_i}. In this manner, we allow server $\mathcal{S}$ to compute the proof $\pi = h^{Q(x)}$ of correct execution (where h is the generator of a group) while ensuring the confidentiality of polynomials B and R. Finally, we use bilinear pairings to let verifier $\mathcal{V}$ assess the correctness of the server's results. Accordingly, we show that our solution is sound under the $\lfloor d/2 \rfloor$-*Strong Diffie-Hellman* ($\lfloor d/2 \rfloor$-SDH) assumption.

To sum things up, we will describe in the following lines our proposal for verifiable polynomial evaluation that is:

Efficient: We propose a solution that is non-interactive and practical. We will show that our protocol induces constant costs for algorithms ProbGen and Verify, that are independent of the degree of the outsourced polynomial, and which are much less expensive that the cost of evaluating the polynomial.

Amortized: Algorithm Setup requires heavy exponentiations to prepare the outsourced polynomial. However, these operations are performed only once for an unlimited number of verifications for the same polynomial.

Publicly delegatable: The data owner publishes public key PK_A that enables anyone to submit input to the server.

Publicly verifiable: The querier generates a public verification key VK_x, tied to input x, enabling any verifier to check the result returned by the server.

Secure: As we will demonstrate in Section 4.5, our protocol is correct and sound.

4.3 Building Blocks

Before describing our protocol in full detail, we recall the definitions of bilinear pairings and the SDH assumption.

4.3.1 Bilinear Pairings

Let $\mathbb{G}_1$, $\mathbb{G}_2$ and $\mathbb{G}_T$ be three cyclic groups of the same finite order p.

A bilinear pairing is a map e: $\mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$, with the following properties:

1. e is *bilinear*: $\forall\ \alpha, \beta \in \mathbb{Z}_p$, $g \in \mathbb{G}_1$ and $h \in \mathbb{G}_2$, $e(g^\alpha, h^\beta) = e(g,h)^{\alpha\beta}$;

2. e is *non-degenerate*: If g is a generator of $\mathbb{G}_1$ and h is a generator of $\mathbb{G}_2$, then $e(g,h)$ is a generator of $\mathbb{G}_T$;

3. e is *computable*: There is an efficient algorithm to compute $e(g,h)$ for any $(g,h) \in \mathbb{G}_1 \times \mathbb{G}_2$.

4.3.2 D-Strong Diffie-Hellman Assumption

Definition 9 (D-SDH Assumption). *Let $\mathbb{G}_1$, $\mathbb{G}_2$ and $\mathbb{G}_T$ be three cyclic groups of the same finite prime order p such that there exists a bilinear pairing $e : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$.*

*We say that the D-**Strong Diffie-Hellman assumption** (D-SDH) holds, if given the tuple $(g, g^\alpha, h, h^\alpha, ..., h^{\alpha^D}) \in \mathbb{G}_1^2 \times \mathbb{G}_2^{D+1}$ for some randomly chosen $\alpha \in \mathbb{F}_p^*$, the probability to produce a pair $(\beta, h^{1/(\beta+\alpha)}) \in \mathbb{F}_p \backslash \{-\alpha\} \times \mathbb{G}_2$ is negligible.*

4.4 Protocol Description

We assume here that data owner $\mathcal{O}$ wants to outsource the evaluation of a d-degree polynomial $A(X) = \sum\limits_{i=0}^{d} a_i X^i$ with coefficients $a_i \in \mathbb{F}_p$ where p is a large prime. Our protocol for verifiable polynomial evaluation follows the system model introduced in Section 3.4. Indeed, the protocol operates in three phases: the *Setup* phase in which data owner $\mathcal{O}$ prepares polynomial A to outsource; the *Computation* phase where a querier $\mathcal{Q}$ crafts a computation request based on an input x and where a server $\mathcal{S}$ replies to that request with $A(x)$ and a proof of correct evaluation; and the *Verification* phase during which a verifier $\mathcal{V}$ checks that the result is correct. This scheme satisfies the requirements of correctness and soundness as defined in Section 3.5 while meeting the efficiency property. Moreover, it is publicly delegatable and verifiable since at the end of the *Setup* phase, data owner $\mathcal{O}$ publishes a public key that can be used by querier $\mathcal{Q}$ to request the evaluation of A on x. Besides this request, $\mathcal{Q}$ advertises a public verification key so that verifier $\mathcal{V}$ checks the returned results. Figure 4.1 illustrates the details of the different algorithms involved in our protocol for verifiable polynomial evaluation.

4.4.1 Setup

In this phase, data owner $\mathcal{O}$ runs algorithm Setup which, on input of security parameter κ and polynomial A, prepares A to enable publicly verifiable polynomial evaluation as follows:

Public parameters generation: Algorithm Setup chooses two cyclic groups $\mathbb{G}_1$ and $\mathbb{G}_2$ of prime order p that admit a bilinear pairing $e : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$. Then it picks a generator g and a generator h of $\mathbb{G}_1$ and $\mathbb{G}_2$ respectively, and defines the set of public parameters as:

$$\mathsf{param} = (p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, e, g, h).$$

Next, algorithm Setup randomly selects $b_0 \in \mathbb{F}_p^*$ such that polynomial $B(X) = X^2 + b_0$ does not divide polynomial A and performs the Euclidean division of polynomial A by polynomial B in $\mathbb{F}_p[X]$. We denote the resulting quotient polynomial by $Q(X) = \sum_{i=0}^{d-2} q_i X^i$ and the resulting remainder polynomial by $R(X) = r_1 X + r_0$. Notice that R is a polynomial of degree at most 1, *i.e.* r_1 could be 0.

Public key computation: Algorithm Setup computes the public key

$$\mathsf{PK}_A = (\mathbf{b}_0, \mathbf{r}_1, \mathbf{r}_0) = (g^{b_0}, h^{r_1}, h^{r_0})$$

Evaluation key computation: To compute evaluation key EK_A algorithm Setup computes $\mathbf{q}_i = h^{q_i} \in \mathbb{G}_2$ for all $0 \leq i \leq d-2$, and lets

$$\mathsf{EK}_A = (A, \mathbf{q}_0, \mathbf{q}_1, ..., \mathbf{q}_{d-2})$$

Algorithm Setup concludes its execution by outputting the tuple $(\mathsf{param}, \mathsf{PK}_A, \mathsf{EK}_A)$.

4.4.2 Computation

In this phase, a querier $\mathcal{Q}$ requests cloud server $\mathcal{S}$ to evaluate outsourced polynomial A on point $x \in \mathbb{F}_p$ and to return the result of this evaluation. To that effect, querier $\mathcal{Q}$ calls ProbGen that takes x and public key PK_A and returns encoding σ_x and verification key VK_x. In turn, server $\mathcal{S}$ performs the evaluation by invoking algorithm Compute with inputs σ_x and EK_A. Compute outputs an encoding of the evaluation result σ_y.

Algorithm ProbGen and Compute operate as following:

$\mathsf{ProbGen}(x, \mathsf{PK}_A)$: On input of $x \in \mathbb{F}_p$ and public key $\mathsf{PK}_A = (\mathbf{b}_0, \mathbf{r}_1, \mathbf{r}_0)$, algorithm ProbGen first computes

$$\mathsf{VK}_{(x,B)} = \mathbf{b_0} g^{x^2}$$
$$\mathsf{VK}_{(x,R)} = \mathbf{r}_1^x \mathbf{r}_0$$

and then outputs the public encoding $\sigma_x = x$ and the public verification key $\mathsf{VK}_x = (\mathsf{VK}_{(x,B)}, \mathsf{VK}_{(x,R)})$.

$\mathsf{Compute}(\sigma_x, \mathsf{EK}_A)$: Given $\sigma_x = x$ and evaluation key $\mathsf{EK}_A = (A, \mathbf{q}_0, \mathbf{q}_1, ..., \mathbf{q}_{d-2})$, algorithm Compute evaluates $y = A(x) = \sum_{i=0}^{d} a_i x^i \mod p$, generates the proof $\pi = \prod_{i=0}^{d-2} \mathbf{q}_i^{x^i}$, and outputs the encoding $\sigma_y = (y, \pi)$.

4.4.3 Verification

On reception of the polynomial evaluation result, verifier $\mathcal{V}$ checks the correctness of server $\mathcal{S}$'s response by running algorithm Verify:

$\mathsf{Verify}(\sigma_y, \mathsf{VK}_x)$**:** Provided with $\sigma_y = (y, \pi)$ and verification key $\mathsf{VK}_x = (\mathsf{VK}_{(x,B)}, \mathsf{VK}_{(x,R)})$, algorithm Verify checks whether the following equation holds:

$$e(g, h^y) = e(\mathsf{VK}_{(x,B)}, \pi) e(g, \mathsf{VK}_{(x,R)}) \tag{4.1}$$

If so, then Verify outputs y meaning that $A(x) = y$; otherwise it outputs $\perp$.

Figure 4.1: Verifiable Polynomial Evaluation

```
▼ Algorithm: $\{\mathsf{param}, \mathsf{PK}_A, \mathsf{EK}_A\} \leftarrow \mathsf{Setup}(1^\kappa, A)$
1. Parameter generation
      Pick $\mathsf{param} = \{p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, g, h, e\}$;
2. Public key computation
      Pick random $b_0 \in \mathbb{F}_p^*$ and set $B(X) = X^2 + b_0$;
      # Make sure $B$ does not divide $A$
      # Euclidean division of $A$ by $B$:
      Compute polynomials $(Q, R)$ such that $A = BQ + R$;
      # $Q(X) = \sum_{i=0}^{d-2} q_i X^i$ and $R(X) = r_1 X + r_0$
      Compute $\mathsf{PK}_A = (\mathbf{b}_0, \mathbf{r}_1, \mathbf{r}_0) = (g^{b_0}, h^{r_1}, h^{r_0})$;
3. Evaluation key computation
      For $0 \leq i \leq d-2$ do
         | Compute $\mathbf{q}_i = h^{q_i}$;
      End
      Set $\mathsf{EK}_A = (A, \mathbf{q}_0, \mathbf{q}_1, ..., \mathbf{q}_{d-2})$;
 Return $(\mathsf{param}, \mathsf{PK}_A, \mathsf{EK}_A)$;

▼ Algorithm: $\{\sigma_x, \mathsf{VK}_x\} \leftarrow \mathsf{ProbGen}(x, \mathsf{PK}_A)$
1. Compute $\mathsf{VK}_{(x,B)} = \mathbf{b_0} g^{x^2}$;
2. Compute $\mathsf{VK}_{(x,R)} = \mathbf{r}_1^x \mathbf{r}_0$;
3. Set $\sigma_x = x$ and $\mathsf{VK}_x = (\mathsf{VK}_{(x,B)}, \mathsf{VK}_{(x,R)})$;
4. Return $(\sigma_x, \mathsf{VK}_x)$;

▼ Algorithm: $\sigma_y \leftarrow \mathsf{Compute}(\sigma_x, \mathsf{EK}_A)$
1. Compute $y = A(x) = \sum_{i=0}^{d} a_i x^i \mod p$;
2. Compute $\pi = \prod_{i=0}^{d-2} \mathbf{q}_i^{x^i}$;
3. Return $\sigma_y = (y, \pi)$;

▼ Algorithm: $\mathsf{out}_y \leftarrow \mathsf{Verify}(\sigma_y, \mathsf{VK}_x)$
1. Parse $\sigma_y = (y, \pi)$ and $\mathsf{VK}_x = (\mathsf{VK}_{(x,B)}, \mathsf{VK}_{(x,R)})$;
2. Verify $e(g, h^y) \overset{?}{=} e(\mathsf{VK}_{(x,B)}, \pi) e(g, \mathsf{VK}_{(x,R)})$;
3. If it verifies then return $\mathsf{out}_y = y$ else return $\mathsf{out}_y = \perp$;
```

4.5 Security Analysis

In this section, we state and prove the two security theorems pertaining to our protocol for verifiable polynomial evaluation.

4.5.1 Correctness

Theorem 3 (Correctness). *Our scheme for publicly verifiable polynomial evaluation is correct.*

Proof of Theorem 3. If on input $\sigma_x = x \in \mathbb{F}_p$, server $\mathcal{S}$ executes algorithm $\mathsf{Compute}$ correctly, then the latter's output will correspond to

$$\sigma_y = (y, \pi) = (A(x), h^{Q(x)})$$

Indeed, we have:

$$\pi = \prod_{i=0}^{d-2} \mathbf{q}_i^{x^i} = \prod_{i=0}^{d-2} h^{q_i x^i} = h^{\sum_{i=0}^{d-2} q_i x^i} = h^{Q(x)}$$

Given that $A = QB + R$ in $\mathbb{F}_p[X]$ and that the order of $e(g, h)$ is equal to p, we get:

$$\begin{aligned} e(g,h)^{A(x)} &= e(g,h)^{Q(x)B(x)+R(x)} \\ &= e(g, h^{Q(x)})^{B(x)} e(g,h)^{R(x)} \end{aligned}$$

As $y = A(x)$ and $\pi = h^{Q(x)}$ we have:

$$\begin{aligned} e(g,h)^y &= e(g,\pi)^{B(x)} e(g,h)^{R(x)} \\ &= e(g^{B(x)}, \pi) e(g, h^{R(x)}) \end{aligned}$$

Since

$$\mathsf{VK}_{(x,B)} = \mathbf{b}_0 g^{x^2} = g^{b_0 + x^2} = g^{B(x)}$$

and

$$\mathsf{VK}_{(x,R)} = \mathbf{r}_1^x \mathbf{r}_0 = h^{r_1 x + r_0} = h^{R(x)}$$

we conclude that

$$e(g,h)^y = e(\mathsf{VK}_{(x,B)}, \pi) e(g, \mathsf{VK}_{(x,R)})$$

and that Verify outputs "$y = A(x)$".

4.5.2 Soundness

Theorem 4. *The scheme proposed for publicly verifiable polynomial evaluation is sound under the $\lfloor d/2 \rfloor$-SDH assumption.*

Proof of Theorem 4. Assume there is an adversary $\mathcal{A}$ that breaks the soundness of our protocol for publicly verifiable polynomial evaluation with a non-negligible advantage ϵ. We demonstrate in what follows that there exists another adversary $\mathcal{B}$ that breaks the $\lfloor d/2 \rfloor$-SDH assumption with a non-negligible advantage ϵ.

The proof of soundness of our solution for publicly verifiable polynomial evaluation involves two games:

Game 0. This game corresponds to the soundness experiment (cf. Section 3.5.2) of our protocol for verifiable polynomial evaluation.

Game 1. The goal of adversary $\mathcal{B}$ in this game is to break the $\lfloor d/2 \rfloor$-SDH assumption using adversary $\mathcal{A}$.

Let $\mathcal{O}_{\mathsf{SDH}}$ be an oracle which when queried returns the pair (g, g^{α}) in $\mathbb{G}_1$ and the tuple $(h, h^{\alpha}, h^{\alpha^2}, ..., h^{\alpha^{\lfloor d/2 \rfloor}})$ in $\mathbb{G}_2$ for randomly generated α in $\mathbb{F}_p^*$.

To break $\lfloor d/2 \rfloor$-SDH, adversary $\mathcal{B}$ first calls oracle $\mathcal{O}_{\mathsf{SDH}}$ to obtain a tuple $(g, g^{\alpha}, h, h^{\alpha}, ..., h^{\alpha^{\lfloor d/2 \rfloor}})$; then simulates the soundness experiment to adversary $\mathcal{A}$.

Namely, adversary $\mathcal{A}$ enters the **learning** phase of the soundness experiment depicted in Algorithm 3: Adversary $\mathcal{A}$ calls oracle $\mathcal{O}_{\mathsf{Setup}}$ with t different polynomials $A^{(k)}(X) = \sum_{i=0}^{d} a_i^{(k)} X^i$ in $\mathbb{F}_p[X]$, where $1 \leq k \leq t$. Adversary $\mathcal{B}$ then simulates $\mathcal{O}_{\mathsf{Setup}}$'s response as follows:

1. Adversary $\mathcal{B}$ defines the public parameters

$$\widehat{\mathsf{param}} = (p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, e, g, h)$$

2. For $1 \leq k \leq t$, to compute the evaluation key $\widehat{\mathsf{EK}}_A^{(k)} = (A^{(k)}, \widehat{\mathbf{q}}_0^{(k)}, ..., \widehat{\mathbf{q}}_{d-2}^{(k)})$, adversary $\mathcal{B}$ proceeds as follows:

   - $\mathcal{B}$ lets $\widehat{\mathbf{q}}_{d-2}^{(k)} = h^{a_d^{(k)}}$ and $\widehat{\mathbf{q}}_{d-3}^{(k)} = h^{a_{d-1}^{(k)}}$;
   - For each $2 \leq l \leq d-2$, adversary $\mathcal{B}$ computes

   $$\widehat{\mathbf{q}}_{d-2-l}^{(k)} = \prod_{i=0}^{\lfloor l/2 \rfloor} h^{a_{d-l+2i}^{(k)} (-1)^i \alpha_k^i}$$

   where $\alpha_k = \alpha\mu_k + \gamma_k$ and μ_k, γ_k are randomly selected from $\mathbb{F}_p^*$. Note that adversary $\mathcal{B}$ can compute $h^{\alpha_k^i}$ and thus $\widehat{\mathbf{q}}_{d-2-l}^{(k)}$ without the knowledge of α using the Binomial Theorem: $h^{\alpha_k^i} = h^{(\alpha\mu_k+\gamma_k)^i} = \prod_{j=0}^{i} (h^{\alpha^j})^{\binom{i}{j}\mu_k^j \cdot \gamma_k^{i-j}}$, where h^{α^j} is given by the $\lfloor d/2 \rfloor$-SDH tuple returned by $\mathcal{O}_{\mathsf{SDH}}$.

3. $\mathcal{B}$ computes the public key $\widehat{\mathsf{PK}}_A^{(k)} = (\widehat{\mathbf{b}}_0^{(k)}, \widehat{\mathbf{r}}_1^{(k)}, \widehat{\mathbf{r}}_0^{(k)})$ as follows:

$$\widehat{\mathbf{b}}_0^{(k)} = g^{\alpha_k}$$
$$\widehat{\mathbf{r}}_0^{(k)} = \prod_{i=0}^{\lfloor d/2 \rfloor} h^{a_{2i}^{(k)} (-1)^i \alpha_k^i}$$
$$\widehat{\mathbf{r}}_1^{(k)} = \prod_{i=0}^{\lfloor (d-1)/2 \rfloor} h^{a_{2i+1}^{(k)} (-1)^i \alpha_k^i}$$

   Note that $\mathcal{B}$ can efficiently compute $\widehat{\mathbf{b}}_0^{(k)}$ as $g^{\alpha_k} = (g^{\alpha})^{\mu_k} g^{\gamma_k}$, where g^{α} is given by the $\lfloor d/2 \rfloor$-SDH tuple returned by $\mathcal{O}_{\mathsf{SDH}}$. If $(\widehat{\mathbf{r}}_0^{(k)}, \widehat{\mathbf{r}}_1^{(k)}) = (1, 1)$, then adversary $\mathcal{B}$ stops the experiment.

4. Otherwise, $\mathcal{B}$ returns public parameters $\widehat{\mathsf{param}}$, evaluation key $\widehat{\mathsf{EK}}_A^{(k)}$ and public key $\widehat{\mathsf{PK}}_A^{(k)}$ to adversary $\mathcal{A}$.

It can easily be shown that if adversary $\mathcal{B}$ does not stop the experiment, then the distribution of the tuple $(\widehat{\mathsf{param}}, \widehat{\mathsf{PK}}_A^{(k)}, \widehat{\mathsf{EK}}_A^{(k)})$ returned by adversary $\mathcal{B}$ is *statistically indistinguishable* from the distribution of $(\mathsf{param}, \mathsf{PK}_A, \mathsf{EK}_A)$ in *Game 0*. As a matter of fact, if we denote for all $0 \leq i \leq d-2$, $\widehat{\mathbf{q}}_i^{(k)} = h^{q_i^{(k)}}$ and if we let $(\widehat{\mathbf{r}}_0^{(k)}, \widehat{\mathbf{r}}_1^{(k)}) = (h^{r_0^{(k)}}, h^{r_1^{(k)}})$, then we can easily verify that:

- $a_d^{(k)} = q_{d-2}^{(k)} \mod p$ and $a_{d-1}^{(k)} = q_{d-3}^{(k)} \mod p$;
- for all $2 \leq i \leq d-2$, $a_i^{(k)} = \alpha_k q_i^{(k)} + q_{i-2}^{(k)} \mod p$;
- $a_1^{(k)} = \alpha_k q_1^{(k)} + r_1^{(k)} \mod p$ and $a_0^{(k)} = \alpha_k q_0^{(k)} + r_0^{(k)} \mod p$;
- $(r_0^{(k)}, r_1^{(k)}) \neq (0,0)$.

This entails that the polynomials defined as $Q^{(k)}(X) = \sum_{i=0}^{d-2} q_i^{(k)} X^i$, $B^{(k)}(X) = X^2 + \alpha_k$ and $R^{(k)}(X) = r_1^{(k)} X + r_0^{(k)}$ verify the following equality: $A^{(k)} = B^{(k)} Q^{(k)} + R^{(k)}$ with $R^{(k)} \neq 0$.

Therefore we can safely conclude (i) that polynomial $B^{(k)}$ does not divide polynomial $A^{(k)}$; (ii) that each $\widehat{\mathbf{q}}_i^{(k)}$ correctly encodes the i^{th} coefficient of the quotient polynomial $Q^{(k)}$ that results from the Euclidean division of polynomial $A^{(k)}$ by polynomial $B^{(k)}$; (iii) that the pair $(\widehat{\mathbf{r}}_0^{(k)}, \widehat{\mathbf{r}}_1^{(k)})$ correctly encodes the corresponding remainder polynomial $R^{(k)}$.

Following the **learning** phase of the soundness game, adversary $\mathcal{A}$ selects a challenge value $x^{(k)} \in \mathbb{F}_p$ and calls oracle $\mathcal{O}_{\mathsf{ProbGen}}$ with the pair $(x^{(k)}, \widehat{\mathsf{PK}}_A^{(k)})$. Accordingly, adversary $\mathcal{B}$ computes the response of oracle $\mathcal{O}_{\mathsf{ProbGen}}$ and returns verification key

$$\mathsf{VK}_x^{(k)} = (\mathsf{VK}_{(x,B)}^{(k)}, \mathsf{VK}_{(x,R)}^{(k)}) = (\widehat{\mathbf{b}}_0^{(k)} g^{x^2}, \widehat{\mathbf{r}}_0^{(k)} \widehat{\mathbf{r}}_1^{(k)x}).$$

Finally, adversary $\mathcal{A}$ returns a pair $(y^{(k)}, \pi^{(k)})$ and invokes algorithm Verify which outputs $\mathsf{out}_y^{(k)}$.

In the **challenge** phase of the soundness game (cf. Algorithm 4), adversary $\mathcal{A}$ selects a polynomial A from the ones challenged to $\mathcal{O}_{\mathsf{Setup}}$ during the **learning** phase (For more readibility, we do not pursue the notation with the * as in Algorithm 4). Without loss of generality, we assume polynomial A is associated with public key $\widehat{\mathsf{PK}}_A$ and evaluation key $\widehat{\mathsf{EK}}_A$. In particular, $\widehat{\mathsf{PK}}_A$ and $\widehat{\mathsf{EK}}_A$ are computed on the basis of $\hat{\alpha} = \alpha\hat{\mu} + \hat{\gamma}$, where $\hat{\mu}, \hat{\gamma} \in \mathbb{F}_p^*$ are ones of the t values μ_k, γ_k selected in the **learning** phase. Adversary $\mathcal{A}$ selects a challenge value $x \in \mathbb{F}_p$ and calls oracle $\mathcal{O}_{\mathsf{ProbGen}}$ with the pair $(x, \widehat{\mathsf{PK}}_A)$. Accordingly, adversary $\mathcal{B}$ computes the response of oracle $\mathcal{O}_{\mathsf{ProbGen}}$ and returns verification key

$$\mathsf{VK}_x = (\mathsf{VK}_{(x,B)}, \mathsf{VK}_{(x,R)}) = (\widehat{\mathbf{b}}_0 g^{x^2}, \widehat{\mathbf{r}}_0 \widehat{\mathbf{r}}_1^x).$$

Finally, adversary $\mathcal{A}$ returns a pair (y, π) such that $y \neq A(x)$ and (y, π) is accepted by algorithm Verify with a non-negligible advantage ϵ.

Consequently, to break $\lfloor d/2 \rfloor$-SDH, adversary $\mathcal{B}$ first computes $A(x)$ and the proof

$$\pi^* = \prod_{i=0}^{d-2} (\widehat{\mathbf{q}}_i)^{x^i}.$$

Since the pair (y, π) passes the verification, it satisfies Equation 4.1, namely:

$$e(g,h)^y = e(\widehat{\mathbf{b}}_0 g^{x^2}, \pi) e(g, \widehat{\mathbf{r}}_0 (\widehat{\mathbf{r}}_1)^x) = e(g^{x^2+\hat{\alpha}}, \pi) e(g, \widehat{\mathbf{r}}_0 (\widehat{\mathbf{r}}_1)^x). \tag{4.2}$$

Furthermore, by construction:

$$e(g,h)^{A(x)} = e(g^{x^2+\hat{\alpha}}, \pi^*)e(g, \widehat{\mathbf{r}}_0(\widehat{\mathbf{r}}_1)^x), \tag{4.3}$$

where π^* denotes the proof of correct evaluation $A(x)$. By dividing Equation 4.2 by 4.3, we obtain:

$$e(g,h)^{(y-A(x))} = e\left(g^{x^2+\hat{\alpha}}, \frac{\pi}{\pi^*}\right)$$

Since $y \neq A(x)$, the above equation implies:

$$e(g,h) = e\left(g^{x^2+\hat{\alpha}}, \left(\frac{\pi}{\pi^*}\right)^{(y-A(x))^{-1}}\right).$$

This entails:

$$\begin{aligned} h &= \left(\frac{\pi}{\pi^*}\right)^{(x^2+\hat{\alpha})(y-A(x))^{-1}} \\ &= \left(\frac{\pi}{\pi^*}\right)^{(x^2+\alpha\hat{\mu}+\hat{\gamma})(y-A(x))^{-1}} && \text{since } \hat{\alpha} = \alpha\hat{\mu} + \hat{\gamma} \text{ with } \hat{\mu}, \hat{\gamma} \in \mathbb{F}_p^* \\ &= \left(\frac{\pi}{\pi^*}\right)^{\hat{\mu}(\alpha+\frac{x^2+\hat{\gamma}}{\hat{\mu}})(y-A(x))^{-1}} && . \end{aligned}$$

Hence if adversary $\mathcal{B}$ does not stop the experiment, then she will be able to break the $\lfloor d/2 \rfloor$-SDH assumption by outputting the pair

$$(\beta, h^{1/(\beta+\alpha)}) = \left(\frac{x^2+\hat{\gamma}}{\hat{\mu}}, \left(\frac{\pi}{\pi^*}\right)^{\hat{\mu}(y-A(x))^{-1}}\right).$$

Now if adversary $\mathcal{B}$ aborts the experiment *i.e.* when $(\widehat{\mathbf{r}}_0, \widehat{\mathbf{r}}_1) = (1,1)$, then adversary $\mathcal{B}$ can conclude that B divides A. As $\mathcal{B}$ knows that $B = X^2 + \hat{\alpha}$, $\mathcal{B}$ can break the $\lfloor d/2 \rfloor$-SDH assumption as follows:

1. $\mathcal{B}$ factorizes polynomial A into a product of irreducible polynomials in $\mathbb{F}_p[X]$.
2. $\mathcal{B}$ discards all the irreducible polynomials of degree above 2 and considers only polynomials of degree 2 of the form X^2+b and polynomials of degree 1 that when multiplied are under the form X^2+b. For instance, if one of the irreducible factor is X^2+X+1, it is discarded, but since $X+1$ and $X-1$ give $(X+1)(X-1) = X^2-1$, polynomial X^2-1 is retained.
3. Among all polynomials (irreducible or combined) of the form X^2+b that were selected in the previous step, one is equal to B. Therefore, for each of those polynomials, $\mathcal{B}$ tests whether $g^b = g^{\hat{\alpha}}$. If it is the case, then adversary $\mathcal{B}$ determines $\hat{\alpha}$ and thus learns α (since $\hat{\alpha} = \alpha\hat{\mu}+\hat{\gamma}$). With knowledge of α, $\mathcal{B}$ can easily break the $\lfloor d/2 \rfloor$-SDH assumption.

Thus, we deduce that if there is an adversary $\mathcal{A}$ that breaks the soundness of our protocol for publicly verifiable polynomial evaluation with a non-negligible advantage ϵ, then there is an adversary $\mathcal{B}$ that breaks the $\lfloor d/2 \rfloor$-SDH assumption with a non-negligible advantage $\geq \epsilon$.

Finally, we want to highlight the fact that if $B(X) = X^\delta + b_0$, then using a similar argument as the one above, we can easily show that our protocol for verifiable polynomial evaluation is secure under the D-SDH assumption for $D \geq \lfloor d/\delta \rfloor$.

4.6 Performance Analysis

This section evaluates the theoretical performance of our verifiable polynomial evaluation scheme. We study the storage overhead induced by our protocol as well as its communication and computation complexities. We will show that while adopting the amortized model defined by Gennaro *et al.* [90], our protocol meets the efficiency requirement defined in Section 3.4.

4.6.1 Storage

Data owner $\mathcal{O}$ is required to store and publish the public key $(\mathbf{b}_0, \mathbf{r}_1, \mathbf{r}_0) \in \mathbb{G}_1 \times \mathbb{G}_2^2$. Server $\mathcal{S}$ however keeps the $d+1$ coefficients $a_i \in \mathbb{F}_p$ of polynomial A and the $d-1$ encodings $\mathbf{q}_i \in \mathbb{G}_2$. Table 4.1 lists the storage complexity of our protocol.

4.6.2 Communication

In terms of communication complexity, our verifiable polynomial evaluation solution requires constant bandwidth consumption. Indeed, at the end of the execution of algorithm $\mathsf{ProbGen}$, querier $\mathcal{Q}$ sends to cloud server $\mathcal{S}$ encoding σ_x and verification key VK_x, requiring $\mathcal{O}(1)$ space. Similarly, server $\mathcal{S}$ returns encoding $\sigma_y = (y, \pi)$ which amounts to $\mathcal{O}(1)$ space. Table 4.1 sums up the bandwidth complexity.

4.6.3 Computation

Algorithm Setup first generates a random coefficient $b_0 \in \mathbb{F}_p^*$ to construct polynomial B and conducts an Euclidean division of polynomial A by polynomial B. The latter operation consists of d multiplications and additions, where d is the degree of polynomial A. Once the Euclidean division is performed, algorithm Setup performs one exponentiation in $\mathbb{G}_1$ to derive $\mathbf{b}_0$, and $d+1$ exponentiations in $\mathbb{G}_2$ to compute $\mathbf{r}_0$, $\mathbf{r}_1$ and $\mathbf{q}_i$. Although computationally expensive, algorithm Setup is executed only once by the client. Besides, its computational cost is *amortized* over the large number of verifications that third-party verifiers can carry out.

On the other hand, $\mathsf{ProbGen}$ computes the verification key $\mathsf{VK}_x = (\mathsf{VK}_{(x,B)}, \mathsf{VK}_{(x,R)})$ which demands a constant number of operations that does not depend on the degree of polynomial A. More precisely, $\mathsf{ProbGen}$ consists of computing x^2 in $\mathbb{F}_p$, performing one exponentiation and one multiplication in $\mathbb{G}_1$ to get $\mathsf{VK}_{(x,B)} = g^{B(x)}$, and running one exponentiation and one multiplication in $\mathbb{G}_2$ to obtain $\mathsf{VK}_{(x,R)} = h^{R(x)}$.

Furthermore, algorithm $\mathsf{Compute}$ runs in two steps: (i) the evaluation of polynomial A at point x which requires at most d additions and multiplications in $\mathbb{F}_p$ if the server uses *Horner's rule*; and (ii) the generation of the proof π which involves $d-3$ multiplications in $\mathbb{F}_p$ and $d-1$ exponentiations and $d-2$ multiplications in $\mathbb{G}_2$.

Finally, the work at verifier $\mathcal{V}$ only consists of one exponentiation and one division in $\mathbb{G}_2$ and the computation of 2 bilinear pairings (indeed, we can rephrase Equation 4.1 as $e(g, \frac{h^y}{\mathsf{VK}_{(x,R)}}) = e(\mathsf{VK}_{(x,B)}, \pi)$).

Summary. The reader may refer to Table 4.1 for a summary of the computational performances of our protocol. We can conclude from the above that our solution meets the requirement on efficiency. Indeed, as Table 4.1 shows, the combined costs of algorithms $\mathsf{ProbGen}$ and Verify are negligible compared to the complexity of evaluating the polynomial. As a matter of fact, the time required to compute $\mathsf{ProbGen}$ and Verify are constant and independent of the degree of the outsourced polynomial. Moreover, the asymptotic cost of $\mathsf{Compute}$ is kept linear in d, which is substantially the same as evaluating the polynomial. In other terms, the complexity of generating the proof of computation does not influence the overall asymptotic complexity of $\mathsf{Compute}$. The complexity of algorithm Setup is admittedly linear in the degree of the outsourced polynomial, however, it is amortized over an unlimited number of efficient verifications. Furthermore, our protocol is efficient in terms of communication complexity but also efficient in terms of storage for the data owner.

Table 4.1: Costs of our Verifiable Polynomial Evaluation scheme

Storage	*$\lvert G\rvert$ refers to the size (in bits) of elements in set G.*			
Data owner	$\mathcal{O}(1)$	$1\cdot\lvert\mathbb{G}_1\rvert + 2\cdot\lvert\mathbb{G}_2\rvert$ bits		
Server	$\mathcal{O}(d)$	$(d+1)\cdot\lvert\mathbb{F}_p\rvert + (d-1)\cdot\lvert\mathbb{G}_2\rvert$ bits		
Communication				
Outbound	$\mathcal{O}(1)$	$1\cdot\lvert\mathbb{F}_p\rvert + 2\cdot\lvert\mathbb{G}_1\rvert$ bits		
Inbound	$\mathcal{O}(1)$	$1\cdot\lvert\mathbb{F}_p\rvert + 1\cdot\lvert\mathbb{G}_2\rvert$ bits		
Operations	Setup	ProbGen	Compute	Verify
PRNG	1	-	-	-
Additions in $\mathbb{F}_p$	d	-	d	-
Multiplications in $\mathbb{F}_p$	d	1	$2d-3$	-
Multiplications in $\mathbb{G}_1$	-	1	-	-
Multiplications in $\mathbb{G}_2$	-	1	$d-2$	1
Exponentiations in $\mathbb{G}_1$	1	1	-	-
Exponentiations in $\mathbb{G}_2$	$d+1$	1	$d-1$	1
Pairings	-	-	-	2

4.6.4 Comparison with Related Work

We compare our solution with two relevant existing techniques for verifiable polynomial evaluation. Fiore and Gennaro [85] devise Algebraic Pseudo-Random Functions (aPRF), also used by Zhang and Safavi-Naini [193], to develop publicly verifiable solutions. Compared to these two solutions, our protocol induces the same amount of computational costs but with the additional property of public delegatability. Another solution for public verification considers signatures for correct computation [141], and uses polynomial commitments [108] to construct these signatures. Besides public verifiability, this solution implements public delegatability. However, the construction in [141] relies on the d-SBDH assumption, whereas our solution is secure under a weaker assumption that is the $\lfloor d/2\rfloor$-SDH. It is worth mentioning that our protocol can be changed to rely on the $\lfloor d/\delta\rfloor$-SDH assumption, where δ is the degree of the divisor polynomial and therefore our scheme can accommodate higher-degree polynomials. Table 4.2 compares our verifiable polynomial evaluation solution with the work described by Fiore and Gennaro [85] and Papamanthou *et al.* [141].

4.6.5 Experimental Results

We developed a prototype of our verifiable polynomial evaluation scheme in Python, using the Charm-Crypto library[90] which implements cryptographic primitives such as elliptic curves and bilinear pairings. We experimented on a machine with the following characteristics: Processor Intel Core i5-2500; CPU@3.80GHz clock speed; 64 bit OS; RAM 16 GB. All reported times are computed as the average of the times measured for a total of 20 executions of our protocol.

Figure 4.2 depicts the time (reported in Table 4.3) needed by each of the four algorithms of our protocol in function of the degree of the outsourced polynomial. As expected, the costs induced by algorithms Setup and Compute grow linearly with the degree of the polynomial. We highlight the fact that in accordance with the theoretical cost analysis we conducted at

[90]Charm-Crypto library, http://jhuisi.github.io/charm/ [Accessed: February 3, 2016].

Table 4.2: Comparison with related work

	Setup	ProbGen	Compute	Verify	Hardness Assumptions	Public Delegatability
Fiore and Gennaro [85]	1 pairing $2(d+1)$ exp in $\mathbb{G}_1$ 1 exp in $\mathbb{G}_T$	1 pairing 1 exp in $\mathbb{G}_1$	$d+1$ exp in $\mathbb{G}_1$	1 pairing 1 exp in $\mathbb{G}_T$	co-CDH DLin	No
Papamanthou et al. [141]	*Polynomial preparation* $2d+1$ exp in $\mathbb{G}_1$		$d+1$ exp in $\mathbb{G}_1$	2 pairing 2 exp in $\mathbb{G}_1$	d-SBDH	Yes
Our scheme	$d+1$ exp in $\mathbb{G}_2$ 1 exp in $\mathbb{G}_1$	1 exp in $\mathbb{G}_1$ 1 exp in $\mathbb{G}_2$	$d-1$ exp in $\mathbb{G}_2$	2 pairings 1 exp in $\mathbb{G}_2$	$\lfloor d/2 \rfloor$-SDH	Yes

Table 4.3: Average times of our protocol and amortization

d	Setup (s)	ProbGen (s)	Compute (s)	Verify (s)	$\text{Compute}_{\text{Local}}$ (s)	**Amortization**
5	0.011	0.0031	0.005	0.0032	2.174×10^{-5}	×
50	0.103	0.0030	0.070	0.0031	1.245×10^{-4}	×
500	0.728	0.0029	0.723	0.0031	0,001	×
5000	7.205	0.0030	7.22	0.0031	0,012	**1195**
50000	72.58	0.0036	72.22	0.0032	0,127	**602**
500000	796.9	0.0043	724.9	0.0032	1,324	**606**

the beginning of Section 4.6, algorithms ProbGen and Verify generate light costs, independent from the degree of the outsourced polynomial. We also compute in the last column of Table 4.3 the breakeven point from which the expensive cost of Setup is amortized over multiple verifications. To interpret these values, we introduce the following criterion, called *outsourceability*.

Definition 10 (Outsourceability - computation). *The criterion x of outsourceability for a verifiable computation scheme is determined by a parameter $x \geq 0$, according to which the time to pre-process the function f to be outsourced is amortized over x verifications of results returned by a remote server. Stated differently, x is such that:*

$$t_{\mathsf{Setup}} + x \cdot (t_{\mathsf{ProbGen}} + t_{\mathsf{Verify}}) \leq x \cdot t_{\mathsf{Compute}_{\mathsf{Local}}}$$

where t_{algo} is the time required to execute algorithm algo.

Hence, x is defined by the relation:

$$x = \left\lceil \frac{t_{\mathsf{Setup}}}{t_{\mathsf{Compute}_{\mathsf{Local}}} - (t_{\mathsf{ProbGen}} + t_{\mathsf{Verify}})} \right\rceil$$

Table 4.3 shows that for degrees $d \leq 5000$, outsourcing the evaluation of the polynomials is not an interesting strategy because it would be more costly for the data owner to outsource the polynomial and verify the correctness of the results than evaluating it locally. However, for polynomials with larger degrees $d \geq 5000$, outsourcing is a winning strategy. Namely, if we consider the case where $d = 500000$, the data owner should better make the choice to outsource the polynomial to the cloud, if at least $x = 606$ polynomial evaluations are requested to the server (naturally, for the same polynomial). Besides, it is worth noticing that we run our benchmarks on a machine that has 16 GB of RAM. Modern smartphones[91] have between 1 and 4 GB of RAM, latest laptops[92] have no more than 8 GB of RAM.

[91] Gareth Beavis, "Best Phone 2016: The 10 Best Smartphones We've Tested", TechRadar, January 25, 2016, http://tiny.cc/w4ft8x [Accessed: February 3, 2016].

[92] Joel Santo Domingo, Laarni Almendrala Ragaza, "The 10 Best Laptops of 2016", PC Magazine, January

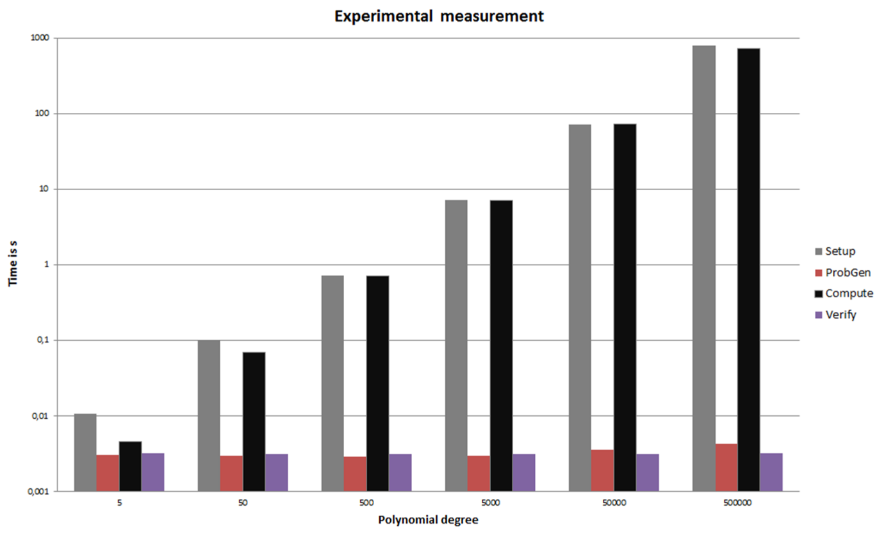


Figure 4.2: Experimental measurements

Therefore, we can extrapolate the analysis of outsourceability of our verifiable polynomial solution to the real world. It may take greater time on smartphones and laptops to evaluate a polynomial with large degree. Hence, even for polynomial with degree $d \leq 5000$, the best strategy for users of these devices would be to outsource these polynomials to the cloud, in order to save computation resources.

4.7 Conclusion to Verifiable Polynomial Evaluation

We presented in this chapter our publicly delegatable and publicly verifiable polynomial evaluation scheme. Exploiting the basic properties of Euclidean division for polynomials, our solution is provably secure according to the correctness and soundness definitions we highlighted in Section 3.5, without relying on heavy cryptographic operations or non-falsifiable assumptions. In addition, our protocol satisfies the efficiency requirement for any VC scheme while adopting the amortized model approach.

In comparison to prior art, we are the first to propose a verifiable polynomial protocol that takes advantage of the Euclidean division and which does not rely on algebraic PRFs. The strength of our approach is that there is no need to design such specific algebraic PRFs to verify any polynomial evaluation while keeping the cost of the verification lightweight. While some of the related work do not support public delegatability and verifiability, our proposal enables third-party queriers and verifiers to submit inputs to the server and check the results. Besides, our scheme can accommodate higher-degree polynomials than most of existing work, thanks to the $\lfloor d/2 \rfloor$-SDH assumption.

7, 2016, http://tiny.cc/gdgt8x [Accessed: February 3, 2016].

Chapter 5

Verifiable Matrix Multiplication

5.1 Introduction to Verifiable Matrix Multiplication

As we explained in our motivating scenario, matrix multiplications are a key primitive that can be employed by the international space agency for several operations such as image processing. For instance, the Haar Wavelet Transform technique is widely used for image compressing or edge detection [54, 156]. This method employs a large matrix, called the Haar matrix, that is multiplied with each row or column of the matrix encoding the image. Since the images produced by the agency are represented as large matrices, it requires considerable computational resources to process them via matrix multiplications, such as the ones performed in the Haar Wavelet Transform. Hence, the space agency also delegates these matrix-based processing operations to the cloud. As a consequence, the cloud must generate a proof to convince the agency that the outcome of these operations is valid.

Obviously, this scenario implies the need for a verifiable protocol for matrix multiplication that fits into the VC model we presented in Section 3.4. In this setting, similarly to the verifiable polynomial evaluation scheme we introduced in Chapter 4, data owner $\mathcal{O}$ corresponds to our international space agency, delegating to a cloud server $\mathcal{S}$ a matrix M of size $n \times m$, that can, for example, encode a large Haar matrix. If we refer to the model discussed in Section 3.4, our context imposes that $\mathfrak{f}$ is the operation of multiplying a vector with M. Besides, data owner $\mathcal{O}$, i.e the space agency, executes algorithm Setup that performs a one-time expensive operation that prepares matrix M to enable an unlimited number of verifiable multiplications on the same matrix M for different inputs. $\mathcal{O}$ also lets anyone submit inputs to $\mathcal{S}$, in order to provide public delegatability. Namely, the space agency allows any collaborating researcher around the world to process the images that it owns. If we consider the example of the Haar Wavelet Transform used to detect edges[93] in the images, a researcher, denoted as querier $\mathcal{Q}$ (as specified in the PVC model in Definition 6), can provide a vector of pixels of size m (that is, a column of the image matrix), denoted $\vec{x}$, to server $\mathcal{S}$ by calling algorithm $\mathsf{ProbGen}$. In turn, $\mathcal{S}$ runs algorithm $\mathsf{Compute}$ that returns $\vec{y} = M\vec{x}$, containing n elements, along with a proof π that certifies the correctness of the multiplication. Furthermore, to support the public verifiability feature desired in our scenario, the space agency lets anyone verify the outcome of algorithm $\mathsf{Compute}$. Therefore a collaborating researcher, which can be different from the one who submitted vector $\vec{x}$, is able to run algorithm Verify on $\vec{y}$ and π that checks that $\vec{y}$ actually equals $M\vec{x}$.

In the next sections, we devise a new protocol for verifiable matrix multiplication that is efficient and provably secure. It adopts the PVC model presented in Definition 6 and provides a solution to the scenario illustrated above.

[93]We recall that our space agency's mission is to observe Earth from space images and we assume that edge extraction is a technique that can be used for such a task.

5.2 Protocol Overview

The protocol we introduce in this section relies on the intuition already expressed in [85], which states that in order to verify that server $\mathcal{S}$ correctly multiplies an (n, m)-matrix M of elements M_{ij} with some column vector $\vec{x} = (x_1, x_2, ..., x_m)^\intercal$, it suffices that data owner $\mathcal{O}$ randomly picks a *secret* (n, m)-matrix R of elements R_{ij}, and supplies server $\mathcal{S}$ with the (n, m)-matrix M and an auxiliary (n, m)-matrix $\mathcal{N}$ such that $\mathcal{N}_{ij} = \tilde{g}^{M_{ij}} g^{R_{ij}}$ (where $\tilde{g} = g^\delta$ for some randomly generated δ and g is a generator of a well-defined group). Consequently, when a querier $\mathcal{Q}$ runs algorithm ProbGen that prompts server $\mathcal{S}$ to multiply matrix M with vector $\vec{x}$, the latter returns vector $\vec{y} = (y_1, y_2, ..., y_n)^\intercal$ and proof $\vec{\pi} = (\pi_1, \pi_2, ..., \pi_n)^\intercal$, such that $\pi_i = \tilde{g}^{y_i} g^{\sum_{j=1}^{m} R_{ij} x_j}$ if the server is honest. If we denote $\pi_i = g^{\gamma_i}$ and $\vec{\gamma} = (\gamma_1, \gamma_2, ..., \gamma_n)^\intercal$, then the verification process induced by algorithm Verify consists of checking whether $\vec{\gamma} = \delta\vec{y} + R\vec{x}$.

Now, to transform this intuition into a viable solution, that is, to fulfill the efficiency requirement expounded in Requirement 1, one must ensure that the execution of both algorithms ProbGen and Verify is much less computationally demanding than performing the matrix multiplication $M\vec{x}$ for all vectors $\vec{x}$. Besides the efficiency requirement, we desire the properties of public delegatability and verifiability. Thus, we cannot directly apply the idea of Fiore and Gennaro [85], that suggest to generate the secret matrix R using dedicated *algebraic PRFs* that optimize the multiplication $R\vec{x}$. As a matter of fact, this method does not offer the desired features.

We tackle this issue by observing that for any vector $\vec{\lambda} = (\lambda_1, \lambda_2, ..., \lambda_n)$, the verification whether $\vec{\lambda}\vec{\gamma} = \delta\vec{\lambda}\vec{y} + \vec{\lambda}(R\vec{x})$ takes $\mathcal{O}(n)$ time if the vector $\vec{\lambda}R$ is computed beforehand. Therefore, we define the public key by an exponent encoding of $\vec{\lambda}R$, and the verification key for vector $\vec{x}$ by an exponent encoding of $(\vec{\lambda}R)\vec{x}$.

More concretely, we generate the elements in the auxiliary matrix $\mathcal{N}$ as $\mathcal{N}_{ij} = \tilde{g}_i^{M_{ij}} g_i^{R_{ij}}$ for $g_i = g^{\lambda_i}$, we let the public key PK_M be a vector of m components $\mathsf{PK}_j = \prod_{i=1}^{n} g_i^{R_{ij}}$, and we compute the verification key for vector $\vec{x}$ as $\mathsf{VK}_x = \prod_{j=1}^{m} \mathsf{PK}_j^{x_j}$. This modus operandi requires an expensive execution of algorithm Setup, but our system model allows such a requirement, since Setup will be amortized by several instances of the verification procedure for different inputs.

As a result, the proposed solution does not only offer public delegatability, but also is sound under the assumption of co-Computation Diffie-Hellman (co-CDH).

Definition 11 (Co-Computational Diffie-Hellman Assumption). *Let $\mathbb{G}_1$, $\mathbb{G}_2$ and $\mathbb{G}_T$ be three cyclic groups of the same finite prime order p such that there exists a bilinear pairing $e : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$.*

We say that the co-CDH holds in $\mathbb{G}_1$, if given $g, g^\alpha \in \mathbb{G}_1$ and $h, h^\beta \in \mathbb{G}_2$ for random $\alpha, \beta \in \mathbb{F}_p^*$, the probability to compute $g^{\alpha\beta}$ is negligible.

To summarize, the protocol for verifiable matrix multiplication, that we describe in the following sections, is:

Efficient: While being non-interactive and practical, our scheme requires less costs to operate algorithms ProbGen and Verify, than executing the matrix multiplication. In particular, algorithms ProbGen and Verify run in $\mathcal{O}(m)$ and $\mathcal{O}(n)$ time respectively, whereas the matrix multiplication takes $\mathcal{O}(nm)$ time.

Amortized: Algorithm Setup requires heavy exponentiations and bilinear pairings to prepare the outsourced matrix. However, these operations are performed only once for an unlimited number of verifications for the same matrix.

Publicly delegatable: The data owner publishes public key PK_M that enables anyone to submit input to the server.

Publicly verifiable: The querier generates a public verification key VK_x, tied to input vector $\vec{x}$, enabling any verifier to check the result returned by the server.

Secure: As we will demonstrate in Section 5.4, our protocol is correct and sound.

5.3 Protocol Description

Without loss of generality, we assume that data owner $\mathcal{O}$ outsources to a cloud server $\mathcal{S}$ the multiplication operations involving an (n, m)-matrix M of elements $M_{ij} \in \mathbb{F}_p$ ($1 \leq i \leq n$ and $1 \leq j \leq m$) with p being a large prime. Adopting the VC model introduced in Definition 6 our protocol for verifiable matrix multiplication comprises three phases *Setup*, *Computation* and *Verification.* We give in what follows the details of the idea expressed in Section 5.2.

5.3.1 Setup

In this phase, data owner $\mathcal{O}$ runs Setup which, on input of security parameter 1^κ and matrix M, prepares M as follows:

Parameters generation: Algorithm Setup chooses two cyclic groups $\mathbb{G}_1$ and $\mathbb{G}_2$ of prime order p that admit a bilinear pairing $e : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$. It then selects a generator h of group $\mathbb{G}_2$ and computes $\tilde{h} = h^\delta$ for a randomly selected δ in $\mathbb{F}_p^*$. Thereafter, it randomly picks n generators g_i of $\mathbb{G}_1$, for all $1 \leq i \leq n$. Without loss of generality, we can assume that $g_i = g^{\lambda_i}$ for λ_i in $\mathbb{F}_p^*$. Subsequently, algorithm Setup defines the public parameters associated with matrix M as:

$$\mathsf{param} = (p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, e, \{g_i\}_{1 \leq i \leq n}, h, \tilde{h}).$$

Evaluation key computation: Algorithm Setup selects an (n, m)-random matrix R of elements R_{ij} in $\mathbb{F}_p^*$ and derives another (n, m)-matrix $\mathcal{N}$ of elements $\mathcal{N}_{ij} = g_i^{\delta M_{ij} + R_{ij}}$, $\forall\ 1 \leq i \leq n,\ 1 \leq j \leq m$. Finally, Setup sets the evaluation key to $\mathsf{EK}_M = (M, \mathcal{N})$.

Public key computation: Setup generates m keys $\mathsf{PK}_j = e(\prod_{i=1}^{n} g_i^{R_{ij}}, h)$, $1 \leq j \leq m$ and sets $\mathsf{PK}_M = (\mathsf{PK}_1, \mathsf{PK}_2, ..., \mathsf{PK}_m)$.

At the end of its execution, algorithm Setup outputs public parameters param, evaluation key EK_M and public key PK_M.

5.3.2 Computation

We assume in this phase that a querier $\mathcal{Q}$ requests cloud server $\mathcal{S}$ to multiply outsourced matrix M with vector $\vec{x}$ of her choice and to return the result of this computation. To that effect, querier $\mathcal{Q}$ calls $\mathsf{ProbGen}$ that takes vector $\vec{x}$ and public key PK_M as inputs and returns encoding σ_x and verification key VK_x. In turn, server $\mathcal{S}$ performs the multiplication of $\vec{x}$ by M by invoking algorithm $\mathsf{Compute}$ with inputs encoding $\vec{x}$ and evaluation key EK_M. $\mathsf{Compute}$ outputs an encoding of the multiplication result σ_y. Algorithm $\mathsf{ProbGen}$ and $\mathsf{Compute}$ operate as follows:

$\mathsf{ProbGen}(\vec{x}, \mathsf{PK}_M)$**:** On input of column vector $\vec{x} = (x_1, x_2..., x_m)^\intercal \in \mathbb{F}_p^m$ and public key $\mathsf{PK}_M = (\mathsf{PK}_1, \mathsf{PK}_2, ..., \mathsf{PK}_m)$ associated with matrix M, this algorithm derives verification key $\mathsf{VK}_x = \prod_{j=1}^{m} \mathsf{PK}_j^{x_j}$ and returns encoding $\sigma_x = \vec{x}$ and verification key VK_x.

$\mathsf{Compute}(\sigma_x, \mathsf{EK}_M)$**:** Provided with encoding $\sigma_x = \vec{x} = (x_1, x_2, ..., x_m)^\intercal$ and evaluation key $\mathsf{EK}_M = (M, \mathcal{N})$, algorithm $\mathsf{Compute}$ multiplies matrix M with vector $\vec{x}$, which yields a column vector $\vec{y} = (y_1, y_2, ..., y_n)^\intercal$. Then, $\mathsf{Compute}$ evaluates the product: $\pi = \prod_{i=1}^{n} \prod_{j=1}^{m} \mathcal{N}_{ij}^{x_j}$ and outputs encoding $\sigma_y = (\vec{y}, \pi)$.

5.3.3 Verification

Upon reception of computation result, verifier $\mathcal{V}$ checks the correctness of server $\mathcal{S}$'s response by running algorithm Verify:

$\mathsf{Verify}(\sigma_y, \mathsf{VK}_x)$**:** Given $\sigma_y = (\vec{y}, \pi)$ and verification key VK_x, it checks whether the following equality holds:

$$e(\pi, h) \stackrel{?}{=} e(\prod_{i=1}^{n} g_i^{y_i}, \tilde{h})\mathsf{VK}_x \tag{5.1}$$

If so, algorithm Verify outputs $\vec{y}$ meaning that $M\vec{x} = \vec{y}$; otherwise it outputs $\perp$.

Figure 5.1: Verifiable Matrix Multiplication

```
▼ Algorithm: $\{\mathsf{param}, \mathsf{PK}_M, \mathsf{EK}_M\} \leftarrow \mathsf{Setup}(1^\kappa, M)$
1. Parameter generation
      Select prime $p$ and groups $\mathbb{G}_1, G_2$ of prime order $p$ that admits a pairing $e : \mathbb{G}_1 \times \mathbb{G}_2 \rightarrow G_T$;
      Pick $n$ generators $g_i$ of $\mathbb{G}_1$ and a generator $h$ of $\mathbb{G}_2$;
      Select random $\delta \in \mathbb{F}_p^*$;
      Compute $\tilde{h} = h^\delta$;
      Set $\mathsf{param} = \{p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, e, \{g_i\}_{1 \leq i \leq n}, h, \tilde{h}\}$;
2. Evaluation key computation
      Select random $(n, m)$-matrix $R$ of element in $\mathbb{F}_p^*$;
      For $1 \leq i \leq n$, $1 \leq j \leq m$ do
         Compute $\mathcal{N}_{ij} = g_i^{\delta M_{ij} + R_{ij}}$;
      End
      Set $\mathsf{EK}_M = (M, \mathcal{N})$;
3. Public key computation
      For $1 \leq j \leq m$ do
         Compute $\mathsf{PK}_j = e(\prod_{i=1}^{n} g_i^{R_{ij}}, h)$;
      End
      Set $\mathsf{PK}_M = (\mathsf{PK}_1, \mathsf{PK}_2, ..., \mathsf{PK}_m)$;
   Return $(\mathsf{param}, \mathsf{PK}_M, \mathsf{EK}_M)$;

▼ Algorithm: $\{\sigma_x, \mathsf{VK}_x\} \leftarrow \mathsf{ProbGen}(\vec{x}, \mathsf{PK}_M)$
1. Compute $\mathsf{VK}_x = \prod_{j=1}^{m} \mathsf{PK}_j^{x_j}$;
2. Set $\sigma_x = \vec{x}$;
3. Return $(\sigma_x, \mathsf{VK}_x)$;

▼ Algorithm: $\sigma_y \leftarrow \mathsf{Compute}(\sigma_x, \mathsf{EK}_M)$
1. Compute $\vec{y} = (y_1, y_2, ..., y_n)^\intercal = M\vec{x}$;
2. Compute $\pi = \prod_{i=1}^{n} \prod_{j=1}^{m} \mathcal{N}_{ij}^{x_j}$;
3. Return $\sigma_y = (\vec{y}, \pi)$;

▼ Algorithm: $\mathsf{out}_y \leftarrow \mathsf{Verify}(\sigma_y, \mathsf{VK}_x)$
1. Parse $\sigma_y = (\vec{y}, \pi)$;
2. Verify $e(\pi, h) \stackrel{?}{=} e(\prod_{i=1}^{n} g_i^{y_i}, \tilde{h})\mathsf{VK}_x$;
3. If it verifies then return $\mathsf{out}_y = \vec{y}$ else return $\mathsf{out}_y = \perp$;
```

5.4 Security Analysis

We state and formally prove in this section the two security properties of correctness and soundness satisfied by our scheme. In particular, we adapt the adversary model described in Section 3.5 to the specific scenario of matrix multiplication.

5.4.1 Correctness

Theorem 5 (Correctness). *Our scheme for publicly verifiable matrix multiplication is correct.*

Proof of Theorem 5. If when queried with vector $\vec{x} = (x_1, x_2, ..., x_n)^\intercal$, server $\mathcal{S}$ correctly operates algorithm $\mathsf{Compute}$, then Equation 5.1 always holds.

In that case, σ_y corresponds to the pair $(\vec{y}, \Pi)$ such that $\vec{y} = (y_1, y_2, ..., y_n)^\intercal = M\vec{x}$ and $\Pi = \prod_{i=1}^{n}\prod_{j=1}^{m} \mathcal{N}_{ij}^{x_j}$. This implies that for all $1 \le i \le n$: $y_i = \sum_{j=1}^{m} M_{ij}x_j \mod p$, and as the order of g_i is p, it also implies that:

$$\begin{aligned}
\Pi &= \prod_{i=1}^{n}\prod_{j=1}^{m} \mathcal{N}_{ij}^{x_j} \\
&= \prod_{i=1}^{n}\prod_{j=1}^{m} \left(g_i^{\delta M_{ij} + R_{ij}}\right)^{x_j} \\
&= \prod_{i=1}^{n}\prod_{j=1}^{m} \left(g_i^{\delta M_{ij}x_j + R_{ij}x_j}\right) \\
&= \prod_{i=1}^{n}\prod_{j=1}^{m} g_i^{\delta M_{ij}x_j} g_i^{R_{ij}x_j} \\
&= \prod_{i=1}^{n} g_i^{\delta \sum_{j=1}^{m} M_{ij}x_j} \prod_{i=1}^{n}\prod_{j=1}^{m} g_i^{R_{ij}x_j} \\
&= \prod_{i=1}^{n} g_i^{\delta y_i} \prod_{i=1}^{n}\prod_{j=1}^{m} g_i^{R_{ij}x_j}.
\end{aligned}$$

Therefore, we have:

$$\begin{aligned}
e(\Pi, h) &= e(\prod_{i=1}^{n} g_i^{\delta y_i} \prod_{i=1}^{n}\prod_{j=1}^{m} g_i^{R_{ij}x_j}, h) \\
&= e(\prod_{i=1}^{n} g_i^{y_i}, h^\delta) e(\prod_{i=1}^{n}\prod_{j=1}^{m} g_i^{R_{ij}x_j}, h) \text{ thanks to the bilinearity of } e \\
&= e(\prod_{i=1}^{n} g_i^{y_i}, h^\delta) \prod_{j=1}^{m} e(\prod_{i=1}^{n} g_i^{R_{ij}}, h)^{x_j}.
\end{aligned}$$

As $\tilde{h} = h^\delta$ and $\mathsf{VK}_x = \prod_{j=1}^{m} \mathsf{PK}_j^{x_j}$, where $\mathsf{PK}_j = e(\prod_{i=1}^{n} g_i^{R_{ij}}, h)$, we get:

$$e(\Pi, h) = e(\prod_{i=1}^{n} g_i^{y_i}, \tilde{h})\mathsf{VK}_x.$$

We can conclude that Verify outputs $\vec{y} = M\vec{x}$.

5.4.2 Soundness

Theorem 6 (Soundness). *Our solution for publicly verifiable matrix multiplication is sound under the co-CDH assumption in $\mathbb{G}_1$, provided that $e : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$ is a bilinear pairing of type 2. Namely, there exists a homomorphism $\phi : \mathbb{G}_2 \to \mathbb{G}_1$ such that $\phi(h) = g$, where g (respectively h) is a generator of $\mathbb{G}_1$ (respectively $\mathbb{G}_2$).*

Proof of Theorem 6. Assume there is an adversary $\mathcal{A}$ that breaks the soundness of our protocol for publicly verifiable delegation of matrix multiplication with a non-negligible advantage ϵ. In other terms, we assume that adversary $\mathcal{A}$ convinces a verifier to accept an incorrect result. Using the soundness definition outlined in Section 3.5.2, we show in the following via a proof-by-reduction that this assumption is not valid. In particular, we prove that if adversary $\mathcal{A}$ (efficiently) breaks our protocol, that is, returns an incorrect result to the matrix multiplication, we can build another adversary $\mathcal{B}$ that uses adversary $\mathcal{A}$ to break the co-CDH assumption in $\mathbb{G}_1$ with a non-negligible advantage $\epsilon' \simeq \epsilon$.

The proof of the soundness of our protocol for publicly verifiable matrix multiplication comprises two games:

Game 0. This corresponds to the soundness experiment of the protocol described in Algorithm 3 and Algorithm 4.

Game 1. In this game, adversary $\mathcal{B}$ would like to break the co-CDH assumption in $\mathbb{G}_1$ with the help of adversary $\mathcal{A}$.

Let $\mathcal{O}_{\mathsf{co-CDH}}$ be an oracle which, when invoked, returns the co-CDH pairs $(g, g^{\alpha}) \in \mathbb{G}_1^2$ and $(h, h^{\beta}) \in \mathbb{G}_2^2$, for some $\alpha, \beta \in \mathbb{F}_p^*$.

To break co-CDH, adversary $\mathcal{B}$ first calls oracle $\mathcal{O}_{\mathsf{co-CDH}}$ which randomly chooses $\alpha, \beta \in \mathbb{F}_p^*$ and outputs the co-CDH pairs (g, g^{α}) and (h, h^{β}).

When adversary $\mathcal{A}$ enters the **learning** phase of the soundness experiment depicted in Algorithm 3 in Section 3.5.2, $\mathcal{A}$ calls oracle $\mathcal{O}_{\mathsf{Setup}}$ with t different (n, m)-matrices $M^{(k)}$ of elements $M_{ij}^{(k)}$ in $\mathbb{F}_p$, where $1 \leq k \leq t$, $1 \leq i \leq n$ and $1 \leq j \leq m$. As a matter of fact, adversary $\mathcal{B}$ simulates $\mathcal{O}_{\mathsf{Setup}}$ as algorithm Setup in *Game 0* except for the following:

1. For $1 \leq k \leq t$, adversary $\mathcal{B}$ selects $\mu_k, \eta_k, \varphi_k, \psi_k$ uniformly at random in $\mathbb{F}_p^*$ and computes the pairs (g, g^{α_k}) and (h, h^{β_k}), where $\alpha_k = \alpha\mu_k + \eta_k$ and $\beta_k = \beta\varphi_k + \psi_k$. Note that adversary $\mathcal{B}$ can compute these pairs without the knowledge of α and β by just having access to co-CDH pairs (g, g^{α}) and (h, h^{β}): $g^{\alpha_k} = (g^{\alpha})^{\mu_k} g^{\eta_k}$ and $h^{\beta_k} = (h^{\beta})^{\varphi_k} h^{\psi_k}$.

2. Now, for all $1 \leq k \leq t$, adversary $\mathcal{B}$ sets $\hat{g}_k = g^{\alpha_k}$ and $\hat{h}_k = (h^{\beta_k})^{\delta}$, for some $\delta \in \mathbb{F}_p^*$. $\mathcal{B}$ then computes for all $1 \leq k \leq t$ and $1 \leq i \leq n$, $\hat{g}_{ki} = \hat{g}_k^{\lambda_i}$ for some randomly chosen $\lambda_i \in \mathbb{F}_p^*$. Hereafter, $\mathcal{B}$ sets the public parameters to

$$\widehat{\mathsf{param}}_k = (p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, e, \{\hat{g}_{ki}\}_{\substack{1 \leq i \leq n \\ 1 \leq k \leq t}}, h, \hat{h}_k),$$

   where e is a bilinear pairing that maps an element from $\mathbb{G}_1 \times \mathbb{G}_2$ to an element in $\mathbb{G}_T$ and there exists a homomorphism $\phi : \mathbb{G}_2 \to \mathbb{G}_1$ such that $\phi(h) = g$.

3. $\mathcal{B}$ generates an (n, m)-random matrix $\widehat{\mathcal{N}}^{(k)}$ of elements $\widehat{\mathcal{N}}_{ij}^{(k)} \in \mathbb{G}_1$;

4. Afterwards, $\mathcal{B}$ computes, for all $1 \leq k \leq t$, and for all $1 \leq j \leq m$,

$$\widehat{\mathsf{PK}}_j^{(k)} = \frac{e(\prod_{i=1}^{n} \widehat{\mathcal{N}}_{ij}^{(k)}, h)}{e(\prod_{i=1}^{n} \hat{g}_{ki}^{M_{ij}^{(k)}}, \hat{h}_k)}; \tag{5.2}$$

5. $\mathcal{B}$ defines the public key associated with matrix $M^{(k)}$ as $\widehat{\mathsf{PK}}_M^{(k)} = (\widehat{\mathsf{PK}}_1^{(k)}, ..., \widehat{\mathsf{PK}}_m^{(k)})$;

6. Finally, $\mathcal{B}$ sets the corresponding evaluation key to $\widehat{\mathsf{EK}}_M^{(k)} = (M^{(k)}, \widehat{\mathcal{N}}^{(k)})$.

Adversary $\mathcal{B}$ concludes its simulation of oracle $\mathcal{O}_{\mathsf{Setup}}$ by outputting public parameters $\widehat{\mathsf{param}}_k$, public key $\widehat{\mathsf{PK}}_M^{(k)}$ and evaluation key $\widehat{\mathsf{EK}}_M^{(k)}$.

Note that the simulated output of oracle $\mathcal{O}_{\mathsf{Setup}}$ in the game is statistically indistinguishable from the distribution of the output of algorithm Setup in *Game 0.* Namely, the following is true:

- The statistical distribution of matrix $\widehat{\mathcal{N}}^{(k)}$ is identical to the distribution of matrix $\mathcal{N}$ generated by algorithm Setup, which is of the form $\mathcal{N}_{ij} = g_i^{\delta M_{ij}} g_i^{R_{ij}}$, where R_{ij} are elements of a random matrix R, for $1 \leq i \leq n$ and $1 \leq j \leq m$ (see Section 5.3.1). Since, each $\widehat{\mathcal{N}}^{(k)}$ are randomly generated, then for each $\widehat{\mathcal{N}}_{ij}^{(k)}$, we can always find a random $\hat{g}_{ki}^{R_{ij}}$ such that $\widehat{\mathcal{N}}_{ij}^{(k)} = \hat{g}_{ki}^{\delta M_{ij}} \hat{g}_{ki}^{R_{ij}}$.

- For all vectors $\vec{x} = (x_1, ..., x_m)^\intercal \in \mathbb{F}_p^m$ and $\vec{y} = (y_1, ..., y_n)^\intercal = M^{(k)}\vec{x}$, the simulated public key $\widehat{\mathsf{PK}}_M^{(k)} = (\widehat{\mathsf{PK}}_1^{(k)}, ..., \widehat{\mathsf{PK}}_m^{(k)})$ verifies this equation:

$$e\left(\prod_{i=1}^{n}\prod_{j=1}^{m}(\widehat{\mathcal{N}}_{ij}^{(k)})^{x_j}, h\right) = e\left(\prod_{i=1}^{n} \hat{g}_{ki}^{y_i}, \hat{h}\right) \prod_{j=1}^{m} \widehat{\mathsf{PK}}_j^{(k)x_j}.$$

  Therefore, we can conclude that the distribution of public key $\widehat{\mathsf{PK}}_M^{(k)}$ is the same as the distribution of $\mathsf{PK}_M = (\mathsf{PK}_1, ..., \mathsf{PK}_m)$ produced by algorithm Setup.

In the rest of the **learning** phase, adversary $\mathcal{A}$ picks a challenge vector $\vec{x}^{(k)} = (x_1^{(k)}, x_2^{(k)}, ..., x_m^{(k)})^\intercal$ and queries oracle $\mathcal{O}_{\mathsf{ProbGen}}$ with the pair $(\vec{x}^{(k)}, \widehat{\mathsf{PK}}_M^{(k)})$. As a result, adversary $\mathcal{B}$ simulates oracle $\mathcal{O}_{\mathsf{ProbGen}}$ and outputs the pair $(\vec{x}^{(k)}, \widehat{\mathsf{VK}}_x^{(k)})$ with $\widehat{\mathsf{VK}}_x^{(k)} = \prod_{j=1}^{m} \widehat{\mathsf{PK}}_j^{(k)x_j}$. Afterwards, adversary $\mathcal{A}$ returns a response $\sigma_y^{(k)} = (\vec{y}^{(k)}, \Pi^{(k)})$ and invokes algorithm Verify which outputs $\mathsf{out}_y^{(k)}$.

In the **challenge** phase of the soundness game (cf. Algorithm 4), adversary $\mathcal{A}$ selects a matrix M from the ones challenged to $\mathcal{O}_{\mathsf{Setup}}$ during the **learning** phase. Without loss of generality, we assume matrix M is associated with public parameters $\widehat{\mathsf{param}} = (p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, e, \{\hat{g}_i\}_{1 \leq i \leq n}, h, \hat{h})$, where for $1 \leq i \leq n$, $\hat{g}_i = \hat{g}^{\lambda_i} = g^{\hat{\alpha}\lambda_i}$, with $\hat{\alpha} = \alpha\hat{\mu} + \hat{\eta}$ and $(\hat{\mu}, \hat{\eta})$ is one of the pair of coefficients selected by adversary $\mathcal{B}$ when she simulated oracle $\mathcal{O}_{\mathsf{Setup}}$ to adversary $\mathcal{A}$ at the beginning of Game 1. Similarly, $\hat{h} = h^{\hat{\beta}}$ where $\hat{\beta} = \beta\hat{\varphi} + \hat{\psi}$. In addition, matrix M is associated with public key $\widehat{\mathsf{PK}}_M$ and evaluation key $\widehat{\mathsf{EK}}_M$, both generated during the simulation of $\mathcal{O}_{\mathsf{Setup}}$ by adversary $\mathcal{B}$. Adversary $\mathcal{A}$ also picks a challenge vector $\vec{x} = (x_1, x_2, ..., x_m)^\intercal$ and queries oracle $\mathcal{O}_{\mathsf{ProbGen}}$ with the pair $(\vec{x}, \widehat{\mathsf{PK}}_M)$.

Adversary $\mathcal{B}$ simulates oracle $\mathcal{O}_{\mathsf{ProbGen}}$ and outputs the pair $(\vec{x}, \widehat{\mathsf{VK}}_x)$ with $\widehat{\mathsf{VK}}_x = \prod_{j=1}^{m} \widehat{\mathsf{PK}}_j^{x_j}$. Afterwards, adversary $\mathcal{A}$ returns a response $\sigma_y = (\vec{y}, \pi)$ such that $\vec{y} \neq M\vec{x}$. In the remainder of this proof, we denote $\vec{y}^* = (y_1^*, y_2^*, ..., y_n^*)^\intercal = M\vec{x}$.

To break the co-CDH assumption in $\mathbb{G}_1$, adversary $\mathcal{B}$ first fetches the vector $\vec{\lambda} = (\lambda_1, \lambda_2, ..., \lambda_n)$ used to compute the powers $\hat{g}_i = \hat{g}^{\lambda_i}$ and verifies whether $\vec{\lambda}\vec{y} = \vec{\lambda}\vec{y}^* \mod p$. If so, adversary $\mathcal{B}$ aborts the game. We will later provide Lemma 2 stating that, under the hardness of discrete logarithm, the probability that $\vec{\lambda}\vec{y} = \vec{\lambda}\vec{y}^* \mod p$ is negligible. If $\vec{\lambda}\vec{y} \neq \vec{\lambda}\vec{y}^* \mod p$, $\mathcal{B}$ breaks co-CDH as follows:

Since $\sigma_y = (\vec{y}, \pi)$ passes the verification, then this implies that the following equation holds:

$$e(\pi, h) = e(\prod_{i=1}^{n} \hat{g}_i^{y_i}, \hat{h})\widehat{\mathsf{VK}}_x \tag{5.3}$$

Also given Equation 5.2, we have:

$$e(\prod_{i=1}^{n}\prod_{j=1}^{m} \widehat{\mathcal{N}}_{ij}^{x_j}, h) = e(\prod_{i=1}^{n} \hat{g}_i^{y_i^*}, \hat{h})\widehat{\mathsf{VK}}_x \tag{5.4}$$

By dividing Equation 5.3 with Equation 5.4, we obtain:

$$\begin{aligned} e\left(\frac{\pi}{\prod_{i=1}^{n}\prod_{j=1}^{m} \widehat{\mathcal{N}}_{ij}^{x_j}}, h\right) &= e\left(\prod_{i=1}^{n} \hat{g}_i^{y_i - y_i^*}, \hat{h}\right) \\ &= e\left(\prod_{i=1}^{n} \hat{g}^{\lambda_i(y_i - y_i^*)}, \hat{h}\right) \\ &= e\left(\hat{g}^{\sum_{i=1}^{n}\lambda_i(y_i - y_i^*)}, \hat{h}\right) \\ &= e\left(\hat{g}^{\vec{\lambda}(\vec{y} - \vec{y}^*)}, \hat{h}\right) \end{aligned}$$

As $\hat{g} = g^{\hat{\alpha}}$ and $\hat{h} = h^{\hat{\beta}\delta}$, where $\hat{\alpha}, \hat{\beta}, \delta$ correspond to the challenged matrix M, we deduce that

$$\begin{aligned} e\left(\frac{\pi}{\prod_{i=1}^{n}\prod_{j=1}^{m} \widehat{\mathcal{N}}_{ij}^{x_j}}, h\right) &= e\left(g^{\hat{\alpha}\vec{\lambda}(\vec{y} - \vec{y}^*)}, h^{\hat{\beta}\delta}\right) \\ &= e\left(g^{\hat{\alpha}}, h^{\hat{\beta}}\right)^{\delta\vec{\lambda}(\vec{y} - \vec{y}^*)} \end{aligned}$$

To make the notation less cluttered, we denote for the rest of this proof: $\Lambda = \delta\vec{\lambda}(\vec{y} - \vec{y}^*)$. Hence:

$$e\left(\frac{\pi}{\prod_{i=1}^{n}\prod_{j=1}^{m} \widehat{\mathcal{N}}_{ij}^{x_j}}, h\right) = e\left(g^{\hat{\alpha}}, h^{\hat{\beta}}\right)^{\Lambda}.$$

Since $\hat{\alpha} = \alpha\hat{\mu} + \hat{\eta}$ and $\hat{\beta} = \beta\hat{\varphi} + \hat{\psi}$, we can write:

$$e\left(\frac{\pi}{\prod_{i=1}^{n}\prod_{j=1}^{m} \widehat{\mathcal{N}}_{ij}^{x_j}}, h\right) = e\left((g^{\alpha})^{\hat{\mu}}g^{\hat{\eta}}, h^{\hat{\beta}}\right)^{\Lambda}$$

$$
\begin{aligned}
&= e\left((g^{\alpha})^{\hat{\mu}}, h^{\hat{\beta}}\right)^{\Lambda} e\left(g^{\hat{\eta}}, h^{\hat{\beta}}\right)^{\Lambda} \\
&= e\left((g^{\alpha})^{\hat{\mu}}, (h^{\beta})^{\hat{\varphi}} h^{\hat{\psi}}\right)^{\Lambda} e\left(g^{\hat{\eta}}, h^{\hat{\beta}}\right)^{\Lambda} \\
&= e\left((g^{\alpha})^{\hat{\mu}}, (h^{\beta})^{\hat{\varphi}}\right)^{\Lambda} e\left((g^{\alpha})^{\hat{\mu}}, h^{\hat{\psi}}\right)^{\Lambda} e\left(g^{\hat{\eta}}, h^{\hat{\beta}}\right)^{\Lambda} \\
&= e\left(g^{\alpha\beta}, h\right)^{\hat{\mu}\hat{\varphi}\Lambda} e\left((g^{\alpha})^{\hat{\mu}\hat{\psi}\Lambda} g^{\hat{\eta}\hat{\beta}\Lambda}, h\right)
\end{aligned}
$$

Since $\phi(h) = g$, where $\phi : \mathbb{G}_2 \to \mathbb{G}_1$ is a homomorphism, we have:

$$
e\left(\frac{\pi}{\prod_{i=1}^{n}\prod_{j=1}^{m} \widehat{\mathcal{N}}_{ij}^{x_j}}, h\right) = e\left(g^{\alpha\beta}, h\right)^{\hat{\mu}\hat{\varphi}\Lambda} e\left((g^{\alpha})^{\hat{\mu}\hat{\psi}\Lambda} \phi(h^{\hat{\beta}})^{\hat{\eta}\Lambda}, h\right)
$$

Again, to reduce the amount of notation, we set $\Theta = (g^{\alpha})^{\hat{\mu}\hat{\psi}} \phi(h^{\hat{\beta}})^{\hat{\eta}}$. Note that adversary $\mathcal{B}$ can easily compute Θ since she knows (g, g^{α}) (that she received from $\mathcal{O}_{\mathsf{co-CDH}}$), $\hat{\mu}, \hat{\psi}$ and $\hat{\eta}$ (that she selected herself) and $h^{\hat{\beta}}$ (she computed during the **learning** phase). Therefore, we can conclude:

$$
\begin{aligned}
e\left(\frac{\pi}{\prod_{i=1}^{n}\prod_{j=1}^{m} \widehat{\mathcal{N}}_{ij}^{x_j}}, h\right) &= e\left(g^{\alpha\beta}, h\right)^{\hat{\mu}\hat{\varphi}\Lambda} e\left(\Theta^{\Lambda}, h\right) \\
e\left(\frac{\pi}{\Theta^{\Lambda}\prod_{i=1}^{n}\prod_{j=1}^{m} \widehat{\mathcal{N}}_{ij}^{x_j}}, h\right) &= e\left(g^{\alpha\beta}, h\right)^{\hat{\mu}\hat{\varphi}\Lambda} \\
e\left(\left(\frac{\pi}{\Theta^{\Lambda}\prod_{i=1}^{n}\prod_{j=1}^{m} \widehat{\mathcal{N}}_{ij}^{x_j}}\right)^{(\Lambda\hat{\mu}\hat{\varphi})^{-1}}, h\right) &= e\left(g^{\alpha\beta}, h\right)
\end{aligned}
$$

As a result, if $\vec{\lambda}(\vec{y} - \vec{y}^*) \neq 0 \mod p$, then $\delta\vec{\lambda}(\vec{y} - \vec{y}^*) \neq 0 \mod p$ ($\delta \in \mathbb{F}_p^*$) and $\Lambda \neq 0$. Furthermore, since $\hat{\mu}, \hat{\varphi} \in \mathbb{F}_p^*$, $\hat{\mu}\hat{\varphi} \neq 0$. We finally have

$$
g^{\alpha\beta} = \left(\frac{\pi}{\Theta^{\Lambda}\prod_{i=1}^{n}\prod_{j=1}^{m} \widehat{\mathcal{N}}_{ij}^{x_j}}\right)^{(\Lambda\hat{\mu}\hat{\varphi})^{-1}},
$$

where $\Theta = (g^{\alpha})^{\hat{\mu}\hat{\psi}} \phi(h^{\hat{\beta}})^{\hat{\eta}}$ and $\Lambda = \delta\vec{\lambda}(\vec{y} - \vec{y}^*)$.

Hence, adversary $\mathcal{B}$ breaks the co-CDH assumption in $\mathbb{G}_1$ as long as $\vec{\lambda}\vec{y} \neq \vec{\lambda}\vec{y}^* \mod p$. Fortunately, under the hardness of discrete logarithm, the probability that $\vec{\lambda}\vec{y} = \vec{\lambda}\vec{y}^* \mod p$ is negligible, as shown in Lemma 2.

To summarize, if there is an adversary $\mathcal{A}$ that breaks the soundness of our protocol for publicly verifiable matrix multiplication with a non-negligible advantage ϵ, then there exists an adversary $\mathcal{B}$ that breaks the co-CDH assumption in $\mathbb{G}_1$ with a non-negligible advantage $\epsilon' \simeq \epsilon$.

Lemma 2. *If adversary $\mathcal{A}$ outputs $\vec{y}$ such that $\vec{\lambda}\vec{y} = \vec{\lambda}\vec{y}^* \mod p$, then adversary $\mathcal{B}$ can break the Discrete Log (DL) assumption in $\mathbb{G}_1$.*

Proof of Lemma 2. Assume there is an adversary $\mathcal{A}$ that outputs a vector $\vec{y} = (y_1, y_2, ..., y_n)^\intercal$ verifying the property above with a non-negligible advantage ϵ. We show that there is another adversary $\mathcal{B}$ which uses adversary $\mathcal{A}$ to break the DL assumption in $\mathbb{G}_1$ with a non-negligible advantage greater than $\frac{\epsilon}{n}$.

Assume that adversary $\mathcal{B}$ receives $\widehat{g} \in \mathbb{G}_1$ and is required to output $\lambda \in \mathbb{F}_p$ such that $\widehat{g} = g^\lambda$.

To this effect, adversary $\mathcal{B}$ simulates the soundness experiment. More precisely, upon receipt of an (n, m)-matrix M, it simulates the output of $\mathcal{O}_{\mathsf{Setup}}$ exactly as in *Game 0* except for the following:

- It selects k randomly in $\{1, 2, ..., n\}$ and lets $\widehat{g}_k = \widehat{g}$;
- for all $1 \leq i \leq n$, $i \neq k$, it randomly selects $\lambda_i \in \mathbb{F}_p^*$ and sets $\widehat{g}_i = \widehat{g}^{\lambda_i}$;
- it sets the public parameters to $\widehat{\mathsf{param}} = (p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, e, \{\widehat{g}_i\}_{1\leq i\leq n}, h, \tilde{h})$.

Adversary $\mathcal{A}$ eventually returns a pair of vectors $\vec{x} = (x_1, x_2, ..., x_m)^\intercal$ and $\vec{y} = (y_1, y_2, ..., y_n)^\intercal$ that verify $\vec{y} \neq M\vec{x}$ and $\vec{\lambda}\vec{y} = \vec{\lambda}M\vec{x} \mod p$, whereby $\vec{\lambda} = (\lambda_1, \lambda_2, ..., \lambda, \lambda_{k+1}, ..., \lambda_n)$.

If we denote $\vec{y}^* = (y_1^*, y_2^*, ..., y_n^*)^\intercal = M\vec{x}$, then the above equality entails that

$$\lambda = \frac{\sum\limits_{i=1, i\neq k}^{n} \lambda_i (y_i^* - y_i)}{y_k - y_k^*}$$

as long as $y_k \neq y_k^*$.

Since $\vec{y} \neq \vec{y}^*$, then there is at least one index $1 \leq j \leq n$ such that $y_j \neq y_j^*$. Since k is randomly chosen from $\{1, ..., n\}$, the probability that $y_k \neq y_k^*$ is at least $1/n$, and consequently, adversary $\mathcal{B}$ will be able to break the DL assumption with advantage $\geq \epsilon/n$.

5.5 Performance Analysis

We discuss here the performance of our solution for verifiable matrix multiplication.

5.5.1 Storage

In terms of storage complexity, server $\mathcal{S}$ is required to keep the (n, m)-matrix M of elements $M_{ij} \in \mathbb{F}_p$ and the (n, m)-matrix $\mathcal{N}$ of elements $\mathcal{N}_{ij} \in \mathbb{G}_1$. The storage complexity is equal to $nm \cdot |\mathbb{F}_p|$ bits and $nm \cdot |\mathbb{G}_1|$ bits where $|\mathbb{F}_p|$ (resp $|\mathbb{G}_1|$) designates the size (in bits) of an element in $\mathbb{F}_p$ (resp in $\mathbb{G}_1$).

On the other hand, data owner $\mathcal{O}$ is required to store and publish the public parameters which are of size $n \cdot |\mathbb{G}_1|$ bits and the public key PK_M whose size is $m \cdot |\mathbb{G}_T|$ bits. We highlight the fact that the public parameters' size can be made constant: Instead of advertising the set $\{g_i\}_{1\leq i\leq n}$, the client can select a hash function $\mathcal{H} : \mathbb{F}_p^* \to \mathbb{G}_1 \setminus \{1\}$ and compute the generators g_i as $\mathcal{H}(i)$, for all $1 \leq i \leq n$. On the downside, this optimization makes our scheme secure only in the random oracle model.

The reader is provided with a summary of this storage complexity in Table 5.1.

5.5.2 Communication

As regard communication complexities, querier $\mathcal{Q}$ sends encoding σ_x and verification key VK_x which consists of a bandwidth consumption of $\mathcal{O}(m)$ space, since m is the dimension of $\sigma_x = \vec{x}$. On the other hand, server $\mathcal{S}$ sends an encoding of the matrix multiplication result $\vec{y}$ which represents $\mathcal{O}(n)$ space ($\vec{y}$ amounts to $n \cdot |\mathbb{F}_p|$ bits). Table 5.1 summarizes the communication complexity of our protocol for verifiable matrix multiplication.

5.5.3 Computation

Algorithm Setup generates the (n, m)-random matrix R which requires the generation of nm random numbers in $\mathbb{F}_p$. To compute the elements $\mathcal{N}_{ij}$ of matrix $\mathcal{N}$ as $g_i^{\delta M_{ij}+R_{ij}}$, algorithm Setup performs nm multiplications and $n(m-1)$ nm additions in $\mathbb{F}_p$, and nm exponentiations in $\mathbb{G}_1$. Furthermore, the generation of public key PK_M demands $m(n-1)$ multiplications in $\mathbb{G}_1$, nm exponentiations in $\mathbb{G}_1$ and m pairings. It should be noted that while algorithm Setup involves expensive operations such as exponentiations and pairings, it is executed by the data owner only once, and consequently, its cost is *amortized* over the large number of verifications that a verifier can perform.

To multiply a vector $\vec{x} = (x_1, x_2, ..., x_m)^\intercal$ with matrix M, algorithm $\mathsf{ProbGen}$ computes $\mathsf{VK}_x = \prod_{j=1}^{m} \mathsf{PK}_j^{x_j}$. This involves $m-1$ multiplications and m exponentiations in $\mathbb{G}_T$.

Moreover, algorithm $\mathsf{Compute}$ consists of two operations: (i) the matrix multiplication $\vec{y} = M\vec{x}$ which requires nm multiplications and additions in $\mathbb{F}_p$; and (ii) the generation of the proof Π which involves nm exponentiations and $(n-1)(m-1)$ multiplications in $\mathbb{G}_1$.

Finally, algorithm Verify evaluates two bilinear pairings, $(n-1)$ multiplications and n exponentiations in $\mathbb{G}_1$, and one multiplication in $\mathbb{G}_T$. We summarize in Table 5.1 the computational complexity of our protocol for verifiable matrix multiplication.

Table 5.1: Costs of our Verifiable Matrix Multiplication solution

Storage	*$\lvert G\rvert$ refers to the size (in bits) of elements in set G.*			
Data owner	$\mathcal{O}(n+m)$	$n \cdot \lvert\mathbb{G}_1\rvert + m \cdot \lvert\mathbb{G}_T\rvert$ bits		
Server	$\mathcal{O}(nm)$	$nm \cdot \lvert\mathbb{F}_p\rvert + nm \cdot \lvert\mathbb{G}_1\rvert$ bits		
Communication				
Outbound	$\mathcal{O}(m)$	$m \cdot \lvert\mathbb{F}_p\rvert + 1 \cdot \lvert\mathbb{G}_T\rvert$ bits		
Inbound	$\mathcal{O}(n)$	$n \cdot \lvert\mathbb{F}_p\rvert + 1 \cdot \lvert\mathbb{G}_1\rvert$ bits		
Operations	Setup	$\mathsf{ProbGen}$	$\mathsf{Compute}$	Verify
PRNG	nm	-	-	-
Additions in $\mathbb{F}_p$	nm	-	$n(m-1)$	-
Multiplications in $\mathbb{F}_p$	nm	-	nm	-
Multiplications in $\mathbb{G}_1$	$m(n-1)$	-	$(n-1)(m-1)$	$n-1$
Multiplications in $\mathbb{G}_2$	-	-	-	-
Multiplications in $\mathbb{G}_T$	-	$m-1$	-	-
Exponentiations in $\mathbb{G}_1$	$2nm$	-	nm	n
Exponentiations in $\mathbb{G}_2$	-	-	-	-
Exponentiations in $\mathbb{G}_T$	-	m	-	-
Pairings	m	-	-	2

5.5.4 Comparison with Related Work

We analyze two relevant prior works on verifiable matrix multiplication and compare them with our solution. Fiore and Gennaro [85] (extended by Zhang and Safavi-Naini [193]) exploit Algebraic Pseudo-Random Functions (aPRF) for publicly verifiable matrix multiplications.

	Setup	ProbGen	Compute	Verify	Public Delegatability
Fiore and Gennaro [85]	$3nm$ exp in $\mathbb{G}_1$	n pairings $2(n+m)$ exp in $\mathbb{G}_1$	nm exp in $\mathbb{G}_1$	n pairings n exp in $\mathbb{G}_T$	No
Zhang and Blanton [195]	1 pairing	n exp in $\mathbb{G}_1$ m exp in $\mathbb{G}_2$ $(n+1)$ exp in $\mathbb{G}_T$	nm exp in $\mathbb{G}_2$	n pairings $(n+1)$ exp in $\mathbb{G}_T$	Yes
Our scheme	$2nm$ exp in $\mathbb{G}_1$ m pairings	m exp in $\mathbb{G}_T$	nm exp in $\mathbb{G}_1$	2 pairings n exp in $\mathbb{G}_1$	Yes

Table 5.2: Comparison with related work

However, only the data owner can submit input vectors to the outsourced multiplication, hence their constructions do not meet the public delegatability requirement. Besides, even though our Setup is more expensive than in [85], our algorithms ProbGen and Verify require less computational resources. Zhang and Blanton [195] present a construction for publicly delegatable and verifiable outsourcing of matrix multiplication that uses mathematical properties of matrices instead of aPRFs. Unlike our work, the public verifiable scheme suggested in [195] does not transfer the matrix M to the server during Setup (whose purpose is reduced to generating the public parameters). Instead, ProbGen prepares the matrix and the input vector for the delegation. The solution proposed by Zhang and Blanton [195] does not follow the amortized model. Hence, ProbGen has to be repeated for each matrix to outsource, inducing possible expensive costs. This construction is secure under the multiple decisional Diffie-Hellman (M-DDH) and the eXternal Diffie-Hellman (XDH) assumptions, which are stronger than the co-CDH assumption we rely on in our solution. Table 5.2 depicts a comparison of our proposal for matrix multiplication with the solution proposed by Fiore and Gennaro [85] and Zhang and Blanton [195].

5.5.5 Experimental Results

We benchmark our verifiable matrix multiplication solution by means of a prototype implemented in Python, with the help of the Charm-Crypto library. We experiment on the same machine as in the polynomial case (Processor Intel Core i5-2500; CPU@3.80GHz clock speed; 64 bit OS; RAM 16 GB). The various measurements reported in this section correspond to an average time of 20 trials of our prototype. We run two types of benchmarks:

Impact of m: For several values of n (the number of rows in the matrix), we tune m (the number of columns) to appreciate the effects of this parametrization over the overall time of our prototype. Results are listed in Table 5.3 and plotted in Figure 5.2.

Impact of n: Symmetrically, given some values of m, we test, for different values of n, the impact of the number of rows in the times required during the execution of our implementation. Table 5.4 shows the measured time for each algorithm of our protocol whereas Figure 5.3 depicts their evolution.

In accordance with Table 5.1, the experimental results show that:

1. Setup is the most expensive operation. The time required to execute this algorithm is linear both in m (as shown in Figure 5.2) and n (as depicted in Figure 5.3).

2. Similarly, the time for algorithm Compute linearly increases with n and m.

3. Table 5.3 and Table 5.4 reveal that algorithm ProbGen only depends on m, according to which the times to compute a computation request linearly grows.

4. As we can see in Table 5.3 and Table 5.4, algorithm Verify only depends on the value of n and grows linearly with it.

Table 5.3: Average times of our protocol and amortization (impact of number of columns)

n	m	Setup (s)	ProbGen (s)	Compute (s)	Verify (s)	Compute$_{\text{Local}}$ (s)
10	10	0.307	0.001	0.139	0.016	$6{,}072\times 10^{-5}$
	100	2.987	0.013	1.359	0.015	$4{,}891\times 10^{-4}$
	1000	29.728	0.125	13.500	0.015	0,004
	10000	297.037	1.245	134.987	0.015	0,043
	100000	2978.320	12.263	1343.951	0.015	0,472
100	10	2.899	0.001	1.359	0.138	$6{,}209\times 10^{-4}$
	100	28.987	0.013	13.567	0.138	0,005
	1000	288.171	0.125	134.938	0.137	0,043
	10000	2882.103	1.247	1350.279	0.137	0,436
1000	10	28.739	0.001	13.484	1.351	0,008
	100	287.223	0.013	134.935	1.353	0,049
	1000	2874.599	0.125	1350.250	1.354	0,461
10000	10	287.380	0.001	135.307	13.523	0,079
	100	2871.961	0.012	1350.648	13.499	0,477

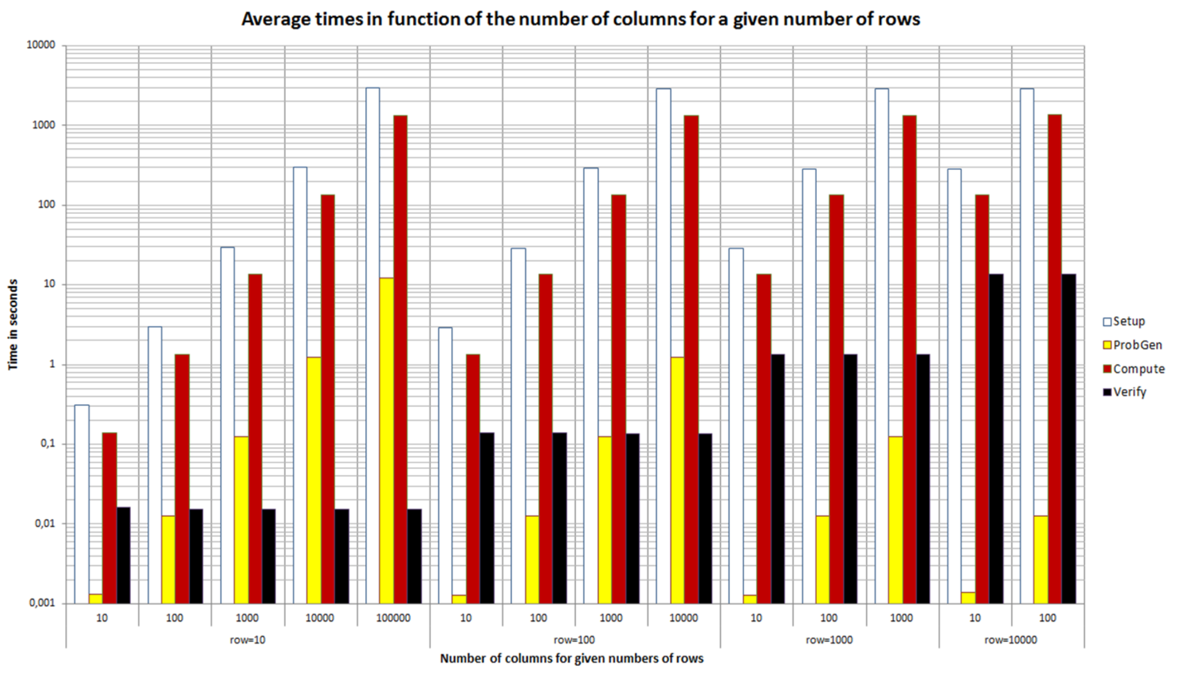


Figure 5.2: Experimental measurements in function of the number of columns

5. Algorithms ProbGen and Verify generate light costs as shown in Table 5.4 and Table 5.3, which makes the matrix multiplicaiton outsourcing practical.

In the last column of Table 5.3, we report the time it takes to compute the matrix multiplication locally by the data owner (that is, without resorting to our verifiable matrix multiplication scheme). The values show that, according to our criterion of outsourceability presented in Definition 10, the expensive Setup is not amortized over multiple verifications. Indeed, this observation comes from the fact that computing the matrix multiplication only involves $\mathcal{O}(nm)$ multiplications in $\mathbb{F}_p$ while algorithm ProbGen requires $\mathcal{O}(m)$ exponentiations in $\mathbb{G}_T$ and algorithm Verify $\mathcal{O}(n)$ exponentiations in G_1. Thus, the time to execute these two algorithms is greater than the time to perform the matrix multiplication. Nonetheless, we can still acknowledge that matrix multiplication protocol is outsourceable for two reasons.

First, our scheme applies to scenarios that involve very large matrices. If we consider our space agency scenario, high resolution space images, which are represented as matrices of pixels, can be very large. For example, the Mars Reconnaissance Orbiter's HiRISE (High

Table 5.4: Average times of our protocol and amortization (impact of number of rows)

m	n	Setup (s)	ProbGen (s)	Compute (s)	Verify (s)
10	10	0,307	0,001	0,139	0,016
	100	2,899	0,001	1,359	0,138
	1000	28,739	0,001	13,485	1,351
	10000	287,380	0,001	135,307	13,523
	100000	2875,865	0,004	1338,877	134,072
100	10	2,987	0,013	1,359	0,016
	100	28,987	0,013	13,567	0,138
	1000	287,223	0,013	134,935	1,353
	10000	2871,961	0,012	1350,648	13,499
1000	10	29,728	0,124	13,500	0,015
	100	288,171	0,125	134,938	0,137
	1000	2874,599	0,125	1350,250	1,354
10000	10	297,037	1,244	134,987	0,015
	100	2882,103	1,247	1350,279	0,137

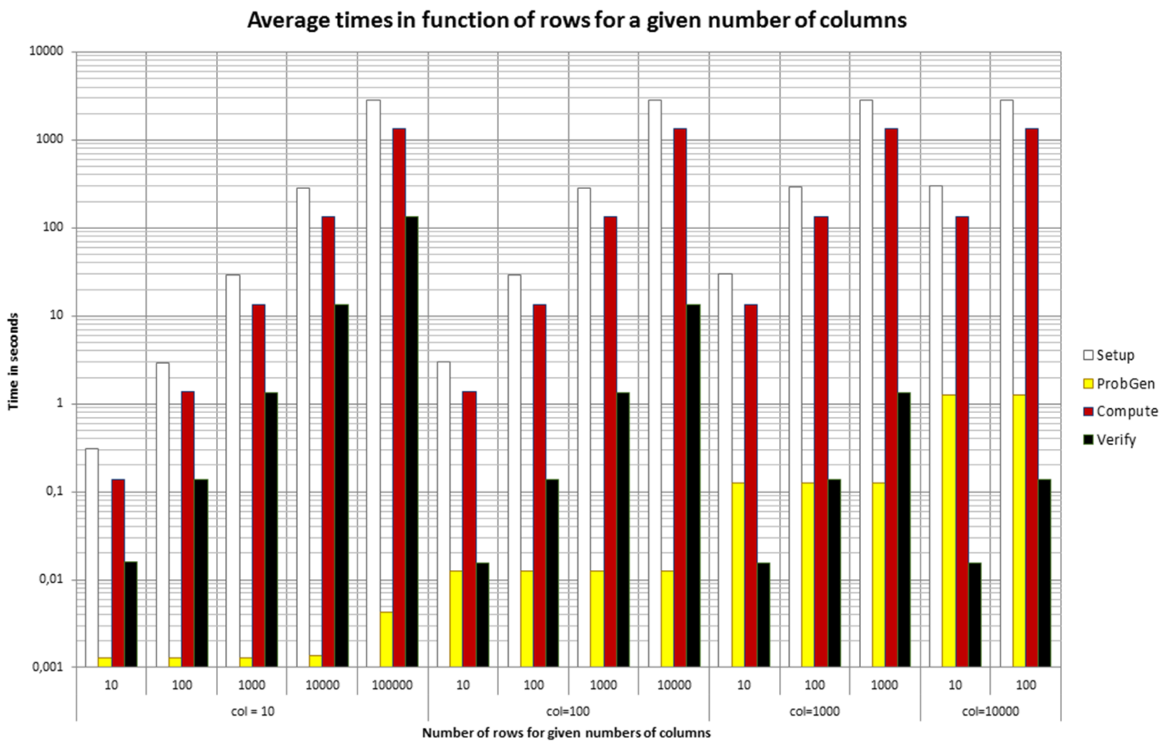


Figure 5.3: Experimental measurements in function of the number of rows

Resolution Imaging Science Experiment) camera takes pictures of around 2.52 gigapixels [151]. In memory, this kind of images amounts to store matrices of 14 GB, if each pixel contains 16 millions of colors, encoded on 6 bytes. It makes sense that the owner of these images may not afford to store this amount of memory. Thus her best strategy is: outsourcing the storage of these images to the cloud and requesting the server to process them, using the proposed verifiable matrix multiplication protocol. This scenario is realistic: Powell *et al.* [151] use cloud computing technologies to perform gigapixel image processing, which involves matrix multiplication. To summarize, storage efficiency defines another criterion for the amortized model and outsourceability.

Furthermore, another advantage of our protocol is the public delegatability and verifiability properties. Namely, for an outsourced matrix, multiple users can request the cloud to multiply this matrix by an input vector of their choices. We can thus define a third criterion with respect to bandwidth efficiency for outsourceability. This criterion is particularly suited for verifiable computation solutions that are publicly delegatable and verifiable.

Definition 12 (Outsourceability - bandwidth). *Outsourceability for a publicly delegatable and publicly verifiable computation scheme is determined by a parameter $N \geq 0$, according to which the bandwidth consumption between data owner $\mathcal{O}$ and N third-party queriers $\mathcal{Q}$ is greater than the bandwidth consumed by $\mathcal{O}$ to outsource the function to the cloud server $\mathcal{S}$ plus the bandwidth consumed between $\mathcal{S}$ and the N third-party queriers $\mathcal{Q}$.*

Stated differently, x is such that:

$$N \cdot \mathsf{BW}(\mathcal{O} \to \mathcal{Q}) \geq \mathsf{BW}(\mathcal{O} \to \mathcal{S}) + N \cdot \mathsf{BW}(\mathcal{S} \leftrightarrows \mathcal{Q})$$

where BW *is the bandwidth consumed between two parties.*

Let us consider the scenario where the data owner stores the large matrix locally. To enable N third-party users to perform multiplications on this matrix, the data owner needs to transmit the matrix to them, which amounts to a bandwidth complexity in $\mathcal{O}(Nnm)$ (namely, $Nnm \cdot |\mathbb{F}_p|$ bits have to be transmitted). On the other hand, outsourcing the matrix will lead to a communication complexity in $\mathcal{O}(nm + N(m+n))$ (namely, the data owner transmits $nm \cdot |\mathbb{F}_p|$ bits to the cloud, N ProbGen transmit $Nm \cdot |\mathbb{F}_p| + N \cdot |\mathbb{G}_T|$ bits and N Verify send $Nn \cdot |\mathbb{F}_p| + N \cdot |\mathbb{G}_1|$ bits, as shown in Table 5.1). Let us calculate these figures for the space agency scenario. We consider that a 1 gigapixel image (6 GB in memory) is a matrix of approximately $n = 32000$ rows and $m = 32000$ columns and that $N = 1000$ third-party researchers will multiply this matrix with some inputs. The matrix contains pixels of $|\mathbb{F}_p| = 6$ bytes and for simplicity $|\mathbb{G}_1| = |\mathbb{G}_T| = 160$ bits[94]. If the space agency locally stores this matrix and transmits it to third-party researchers for multiplication then the bandwidth consumption amounts to approximately 6 TB. If the space agency outsources the matrix and uses our publicly delegatable and verifiable matrix multiplication protocol, then the agency only needs to transmit the matrix of 6 GB to the cloud. The interaction between the cloud and the third parties who request matrix multiplication only amounts to 0.4 GB in total.

5.6 Conclusion to Verifiable Matrix Multiplication

We introduced in this chapter a new non-interactive publicly delegatable and publicly verifiable matrix multiplication scheme. While many existing solutions leverage algebraic PRF, our solution only relies on algebraic properties of matrices. Compared to prior work, our solution reduces the cost of the multiplication of a random matrix R with the input vector $\vec{x}$ by introducing a random row vector $\vec{\lambda}$. This vector is left-multiplied with $R\vec{x}$ as a projection of $R\vec{x}$ on $\vec{\lambda}$, which shrinks the cost of verification from quadratic to linear. Our solution is secure against the correctness and soundness definitions we stated in Section 3.5, without relying on heavy cryptographic operations, nor on non-falsifiable assumptions, as it is the case for some existing work. Furthermore, our protocol meets the efficiency requirement that we highlighted in Section 3.2.2 and follows the amortized model approach with respect to an expensive setup operation. The strength of our approach is that public delegatability and verifiability are ensured without sacrificing efficiency.

[94]This is the case for Charm-Crypto library.

Chapter 6

Verifiable Conjunctive Keyword Search

6.1 Introduction to Verifiable Conjunctive Keyword Search

In this chapter, we focus on the problem of verifiable conjunctive keyword search. Cloud servers are the best candidates to undertake the operation of search in large datasets. Indeed, they have means to store huge amount of data and own the necessary computational resources to process and analyze huge datasets. Keyword search is one of the most frequently used primitives in data mining. Therefore, we consider this operation and design a solution that assures the correctness of the search result.

More specifically, let us consider again the international space agency scenario. For the sake of their Earth observation mission, the agency collects and generates huge amount of data, be it space and aerial images or time-series data. To save IT investments, the agency outsources this *public* dataset to a cloud server. To enable the processing of this data, the space agency annotates each document with information relevant for their identification, classification, search or retrieval. Hence, an image can be annotated with information related to:

- the name of the satellite that acquired the image (LandSat[95], Sentinel-1[96], Ikonos[97], etc.)
- the mission or topic the image was captured for (forest, ocean, erosion, landslide, volcano, etc.)
- the places on Earth the image depicts (Nice, Haiti, Lake Baikal, Gobi Desert, World Trade Center New-York, etc.)
- the date of acquisition, etc.

As searching for keyword in the database of images might be costly, the space agency also delegates the search operation to the cloud. Furthermore, the agency wants to empower third-party researchers collaborating in its Earth observation mission (i) to issue conjunctive keyword search queries to the database and (ii) to efficiently verify the correctness of the results returned by the cloud. As an example, a collaborator from a research center can search for keywords such as "LandSat AND hurricane AND Florida" to search whether there exist, in the agency's public database, images that were produced by the LandSat satellite and showing a hurricane in Florida[98]. If such an image does not exist, the cloud server should return a proof that the search yields no result. Otherwise, it returns the identifiers of the images that satisfy the search query, that is, the images whose annotations contain keywords

[95]LandSat Image Gallery, February 4, 2016, http://tiny.cc/m9gu8x [Accessed: February 4, 2016].

[96]Sentinel-1 Missions, ESA, http://tiny.cc/a9gu8x [Accessed: February 4, 2016].

[97]Ikonos Satellite Images, Satellite Imaging Corporation, http://tiny.cc/qahu8x [Accessed: February 4, 2016].

[98]Such images do exist, for example see the image produced by satellite Landsat of hurricane Jeanne above Florida in September 2004. LandSat Image Gallery, http://tiny.cc/rchu8x [Accessed: February 4, 2016].

"LandSat AND hurricane AND Florida". Along with this result, the server should produce a proof stating that it operated the search correctly and returned the valid set of image identifiers corresponding to the search. In other terms, the server proves that the result it returned does not contain images that do not match the keyword search nor omit any images matching the search query.

It becomes obvious that this scenario fits into the PVC model defined in Section 3.4, where data owner $\mathcal{O}$ corresponds to our space agency. It delegates to a cloud server $\mathcal{S}$ the search operation on a large dataset $\mathcal{F} = \{F_1, F_2, ..., F_n\}$. Namely, data owner $\mathcal{O}$ executes algorithm Setup to outsource the set of files $\mathcal{F}$ and to enable verifiable conjunctive keyword search. Cloud server $\mathcal{S}$ undertakes the search operation by executing a slightly modified version of algorithm Compute, that we name Search. This algorithm also generates the proof that the search results are correct. Additionally, we mentioned that the space agency delegates the search and verification capabilities to third parties: (i) a querier $\mathcal{Q}$ will be able to run a modified version of algorithm ProbGen, named QueryGen for more expressiveness, to submit conjunctive search queries to server $\mathcal{S}$; and (ii) a verifier $\mathcal{V}$ who runs algorithm Verify can check the validity of the search results returned by cloud server $\mathcal{S}$.

In the two next sections, we adapt the system model of publicly verifiable computation presented in Section 3.4 to the problem of publicly verifiable conjunctive keyword search (Section 6.2). We also customize in Section 6.2.3 the adversary model introduced in Section 3.5.

6.2 Definition of Publicly Verifiable Conjunctive Keyword Search

In the following, we tailor the definition of a PVC scheme to the problem of Publicly Verifiable Conjunctive Keyword Search (PVCKS).

6.2.1 System Model

As discussed previously, publicly verifiable conjunctive keyword search enables a data owner $\mathcal{O}$ to outsource a set of files $\mathcal{F}$ to a server $\mathcal{S}$, while ensuring the properties of

• **Public delegatability**: A querier $\mathcal{Q}$ can issue *conjunctive keyword search queries* of the form $\mathcal{W} = \{\omega_1, \omega_2, ..., \omega_k\}$ to server $\mathcal{S}$ for outsourced files $\mathcal{F}$. Server $\mathcal{S}$ responds to this search query by returning the subset of files $\mathcal{F}_\mathcal{W} \subset \mathcal{F}$ containing all words in $\mathcal{W}$.

• **Public verifiability**: A verifier $\mathcal{V}$ can assess the correctness of the results returned by server $\mathcal{S}$, that is, verify whether the search result output by $\mathcal{S}$ for a collection of words $\mathcal{W}$ is correct. Namely, if we denote CKS the function which on inputs of files $\mathcal{F}$ and a collection of words $\mathcal{W}$ returns the files containing all keywords in $\mathcal{W}$, then verifier $\mathcal{V}$ checks that $\mathcal{F}_\mathcal{W}$ actually corresponds to $\mathsf{CKS}(\mathcal{F}, \mathcal{W})$.

In more formal terms, we adapt the PVC model for the search operation and define thereafter publicly verifiable conjunctive keyword search. In this setting, function $\mathfrak{f}$ in the PVC definition corresponds to the set of files $\mathcal{F}$ to be outsourced. Algorithm ProbGen becomes QueryGen that takes as input the conjunction of keywords $\mathcal{W}$ ant outputs an encoded search query $\mathcal{E}_Q$. Algorithm Compute from Definition 6 is appropriately renamed Search whereas algorithm Verify in PVCKS handles the same function as in PVC. Furthermore, PVCKS consists of three phases: *Setup*, *Search* and *Verification*.

Definition 13 (Publicly Verifiable Conjunctive Keyword Search Scheme). *A PVCKS scheme consists of four polynomial-time algorithms* (Setup*,* QueryGen*,* Search*,* Verify)*, distributed across three phases:*

▶ **Setup.** *This phase only involves data owner $\mathcal{O}$. She runs algorithm* Setup *to produce the keying material required in the PVCKS scheme and to process files $\mathfrak{f}$ before their outsourcing:*

▷Setup$(1^\kappa, \mathcal{F}) \to (\mathsf{PK}_\mathcal{F}, \mathsf{LK}_\mathcal{F})$: *Data owner $\mathcal{O}$ executes this randomized algorithm whenever it wishes to outsource a set of files $\mathcal{F} = \{F_1, F_2, ...\}$. On input of a security parameter 1^κ and files $\mathcal{F}$, algorithm* Setup *outputs the pair of public key* $\mathsf{PK}_\mathcal{F}$ *and lookup key* $\mathsf{LK}_\mathcal{F}$ *(i.e. search key, we use the terms lookup key and search key interchangeably).*

▶ **Search.** *The Search phase consists of two steps. Querier $\mathcal{Q}$ runs algorithm* QueryGen *that prepares a search query $\mathcal{W}$ to be submitted to cloud server $\mathcal{S}$. In turn, the server invokes algorithm* Search *that search keywords belonging to $\mathcal{W}$ in set of files $\mathfrak{f}$ and generates a proof of search.*

▷QueryGen$(\mathcal{W}, \mathsf{PK}_\mathcal{F}) \to (\mathcal{E}_Q, \mathsf{VK}_Q)$: *Given a collection of words $\mathcal{W} = \{\omega_1, \omega_2, ...\}$ and public key* $\mathsf{PK}_\mathcal{F}$*, querier $\mathcal{Q}$ calls algorithm* QueryGen *which outputs an encoded conjunctive keyword search query $\mathcal{E}_Q$ and the corresponding public verification key* VK_Q.

▷Search$(\mathsf{LK}_\mathcal{F}, \mathcal{E}_Q) \to \mathcal{E}_R$: *Provided with search key* $\mathsf{LK}_\mathcal{F}$ *and the encoded search query $\mathcal{E}_Q$, server $\mathcal{S}$ executes this algorithm to generate an encoding $\mathcal{E}_R$ of the search result $\mathcal{F}_\mathcal{W} = \mathsf{CKS}(\mathcal{F}, \mathcal{W})$.*

▶ **Verification.** *After receiving the search results and the proof of search from cloud server $\mathcal{S}$, verifier $\mathcal{V}$ executes algorithm* Verify *to check their validity.*

▷Verify$(\mathcal{E}_R, \mathsf{VK}_Q) \to \mathsf{out}$: *Verifier $\mathcal{V}$ invokes this deterministic algorithm to check the integrity of the server's response $\mathcal{E}_R$. Notably, algorithm* Verify *first converts $\mathcal{E}_R$ into a search result $\mathcal{F}_\mathcal{W}$, then uses verification key* VK_Q *to decide whether $\mathcal{F}_\mathcal{W}$ is equal to $\mathsf{CKS}(\mathcal{F}, \mathcal{W})$. Accordingly, algorithm* Verify *outputs* $\mathsf{out} = \mathcal{F}_\mathcal{W}$ *if it believes that $\mathcal{F}_\mathcal{W} = \mathsf{CKS}(\mathcal{F}, \mathcal{W})$, and in this case we say that verifier $\mathcal{V}$ accepts the server's response. Otherwise, algorithm* Verify *outputs* $\mathsf{out} = \perp$*, and we say that verifier $\mathcal{V}$ rejects the server's result.*

6.2.2 Definition of a Publicly Dynamic Verifiable Conjunctive Keyword Search protocol

In Definition 14, we formalize three algorithms that extends the initial definition of a publicly verifiable conjunctive keyword search (Definition 13) to the case where the data is subject to updates. Namely, we introduce algorithm UpdateQuery run by the data owner and which requests the server to perform the update using a new algorithm Update. This algorithm also produces a proof of correct update that is verified by the data owner who invokes the new algorithm VerifyUpdate.

Definition 14 (Publicly Dynamic Verifiable Conjunctive Keyword Search). *A Publicly Dynamic Verifiable Conjunctive Keyword Search is a PVCKS scheme which efficiently handles updates in the outsourced data. Additionally to the algorithms related to the definition of a PVCKS scheme, a Publicly Dynamic Verifiable Conjunctive Keyword Search solution includes these three algorithms:*

▷UpdateQuery$(F_j, \mathcal{W}_j, \mathsf{op}) \to (\mathcal{U}_Q, \mathsf{VK}_Q)$: *Given a file F_j to be updated (either F_j is an existing file to be modified or deleted, or $F_j = F_{n+1}$ is a new file to insert in the outsourced database), given $\mathcal{W}_j$ the collection of keywords that have to be updated, and given the operation* $\mathsf{op} \in \{\mathsf{modify}, \mathsf{delete}, \mathsf{insert}\}$ *to be performed, data owner*

$\mathcal{O}$ calls algorithm $\mathsf{UpdateQuery}$ *which outputs an encoded update query $\mathcal{U}_Q$ and the corresponding public verification key VK_Q.*

▷$\mathsf{Update}(\mathsf{PK}_\mathcal{F}, \mathsf{LK}_\mathcal{F}, \mathcal{U}_Q) \to (\Pi_{\mathsf{upd}}, \mathsf{LK}'_\mathcal{F})$**:** *Provided with public key $\mathsf{PK}_\mathcal{F}$, search key $\mathsf{LK}_\mathcal{F}$ and the encoded update query $\mathcal{E}_Q$, server $\mathcal{S}$ executes this algorithm to generate a new search key $\mathsf{LK}'_\mathcal{F}$ and a proof of update Π_{upd}.*

▷$\mathsf{VerifyUpdate}(\mathsf{VK}_Q, \Pi_{\mathsf{upd}}) \to \{\mathsf{accept}, \mathsf{reject}\}$**:** *Provided with verification key VK_Q, and the proof of update Π_{upd}, this algorithm checks that algorithm* Update *correctly modified $\mathsf{LK}'_\mathcal{F}$. If it is the case, then algorithm* $\mathsf{VerifyUpdate}$ *returns* accept. *Otherwise, it outputs* reject.

6.2.3 Adversary Model

We tailor the adversary model proposed for verifiable computation in Section 3.5 to our problem of PVCKS.

A conjunctive keyword search must fulfill the two security of *correctness* and *soundness*. We briefly recall that correctness means that a response generated by an *honest* server will always be accepted by the verifier; *soundness* implies that a verifier accepts a response of a (potentially malicious) server **if and only if** that response is the outcome of a *correct* execution of the Search algorithm. Since we adopt new notations for the case of search, compared to the general model presented in Section 3.4, we give in the following paragraphs a clear characterization of the correctness and soundness properties, specifically to the problem of verifiable keyword search.

6.2.3.1 Correctness

A verifiable conjunctive keyword search scheme is said to be correct, if whenever server $\mathcal{S}$ operates algorithm Search *correctly* on the input of some encoded search query $\mathcal{E}_Q$, it always obtains an encoding $\mathcal{E}_R$ that will be accepted by verifier $\mathcal{V}$ who runs algorithm Verify.

Definition 15. *A verifiable conjunctive keyword search is correct,* **iff** *for any set of files $\mathcal{F}$ and collection of words $\mathcal{W}$:*

If $\mathsf{Setup}(1^\kappa, \mathcal{F}) \to (\mathsf{PK}_\mathcal{F}, \mathsf{LK}_\mathcal{F})$, $\mathsf{QueryGen}(\mathcal{W}, \mathsf{PK}_\mathcal{F}) \to (\mathcal{E}_Q, \mathsf{VK}_Q)$ *and* $\mathsf{Search}(\mathsf{LK}_\mathcal{F}, \mathcal{E}_Q) \to \mathcal{E}_R$, *then:*

$$\Pr(\mathsf{Verify}(\mathcal{E}_R, \mathsf{VK}_Q) \to \mathsf{CKS}(\mathcal{F}, \mathcal{W})) = 1$$

6.2.3.2 Soundness

We say that a scheme for publicly verifiable conjunctive keyword search is sound, if for any set of files $\mathcal{F}$ and for any collection of words $\mathcal{W}$, server $\mathcal{S}$ cannot convince a verifier $\mathcal{V}$ to accept an incorrect search result.

To formalize the soundness of verifiable conjunctive keyword search, we adapt the soundness experiment presented in Section 3.5.2 with the notations we adopt in this chapter. Algorithm 5 depicts the learning phase of the soundness experiment, whereas Algorithm 5 details its challenge phase.

Let out^* denote the output of algorithm Verify on input $(\mathcal{E}_R^*, \mathsf{VK}_Q^*)$. Adversary $\mathcal{A}$ succeeds in the soundness experiment if: (i) $\mathsf{out}^* \neq \perp$ and (ii) $\mathsf{out}^* \neq \mathsf{CKS}(\mathcal{F}^*, \mathcal{W}^*)$, where $\mathcal{F}^*$ is the set of files associated with public key $\mathsf{PK}_\mathcal{F}^*$.

▼ **Algorithm 5:** Learning Phase of the Soundness Experiment

```
for $k := 1$ to $t$ do
    $\mathcal{A} \to \mathcal{F}_k$;
    $(\mathsf{PK}_{\mathcal{F}_k}, \mathsf{LK}_{\mathcal{F}_k}) \leftarrow \mathcal{O}_{\mathsf{Setup}}(1^\kappa, \mathcal{F}_k)$;
    $\mathcal{A} \to \mathcal{W}_k$;
    $(\mathcal{E}_{Q,k}, \mathsf{VK}_{Q,k}) \leftarrow \mathcal{O}_{\mathsf{QueryGen}}(\mathcal{W}_k, \mathsf{PK}_{\mathcal{F}_k})$;
    $\mathcal{A} \to \mathcal{E}_{R,k}$;
    $\mathsf{out}_k \leftarrow \mathsf{Verify}(\mathcal{E}_{R,k}, \mathsf{VK}_{Q,k})$;
end
```

▼ **Algorithm 6:** Challenge Phase of the Soundness Experiment

```
$\mathcal{A} \to (\mathsf{PK}^*_{\mathcal{F}}, \mathsf{LK}^*_{\mathcal{F}})$;
$\mathcal{A} \to \mathcal{W}^*$;
$(\mathcal{E}^*_Q, \mathsf{VK}^*_Q) \leftarrow \mathcal{O}_{\mathsf{QueryGen}}(\mathcal{W}^*, \mathsf{PK}^*_{\mathcal{F}})$;
$\mathcal{A} \to \mathcal{E}^*_R$;
$\mathsf{out}^* \leftarrow \mathsf{Verify}(\mathcal{E}^*_R, \mathsf{VK}^*_Q)$;
```

Definition 16. *Let $\mathcal{A}dv_{\mathcal{A}}$ denote the advantage of adversary $\mathcal{A}$ in succeeding the soundness game, i.e., $\mathcal{A}dv_{\mathcal{A}} = \Pr(\mathsf{out}^* \neq \perp \wedge \mathsf{out}^* \neq \mathsf{CKS}(\mathcal{F}^*, \mathcal{W}^*))$.*

A publicly verifiable conjunctive keyword search is sound, **iff** *for any adversary $\mathcal{A}$, $\mathcal{A}dv_{\mathcal{A}} \leq \epsilon$ and ϵ is a negligible function in the security parameter 1^κ.*

6.3 Protocol Overview

Our PVCKS solution relies on **polynomial-based accumulators** (*i.e.* bilinear pairing accumulators) defined in [132, 69] to represent the keywords present in files $\mathcal{F} = \{F_1, F_2, ..., F_n\}$. By definition, a polynomial-based accumulator maps a set to a unique polynomial such that each root of the polynomial corresponds to an element in the set. Hence, polynomial-based accumulators allow efficient **verifiable test of membership** which can be tailored for Verifiable Keyword Search (VKS).

A naive approach to accommodate polynomial-based accumulators to VKS would be to represent the words in each file $F_j \in \mathcal{F}$ with a single accumulator. To check whether a word ω is in file F_j, querier $\mathcal{Q}$ first sends a search query to server $\mathcal{S}$, upon which the latter generates a proof of membership if word ω is present in F_j; and a proof of non-membership otherwise.

This solution however is not efficient: (i) Given the mathematical properties of polynomial-based accumulators, the resulting complexity of keyword search in a file F_j is linear in the number of keywords in that file; (ii) additionally, to identify which files F_j contain a word, the user must search all files in $\mathcal{F}$ one by one.

To avoid these pitfalls, we combine polynomial-based accumulators with **Merkle trees** [125] to build an authenticated index of the keywords in files in $\mathcal{F}$ such that the keyword search at the server runs in *logarithmic time*. More specifically, data owner $\mathcal{O}$ first organizes the keywords in all files in $\mathcal{F}$ into an index $\mathcal{I}$ (*i.e.* hash table) where each entry corresponds to a bucket B storing at most d keywords. To construct an efficient index $\mathcal{I}$, data owner $\mathcal{O}$ employs the **Cuckoo hashing** algorithm introduced in [74] which guarantees a constant lookup time and minimal storage requirements. Later, data owner $\mathcal{O}$ authenticates index $\mathcal{I}$ as follows: (i) For each bucket B, it computes an accumulator of the keywords assigned to B; (ii) and it builds a binary Merkle tree TW that authenticates the resulting accumulators. Files in $\mathcal{F}$ are then outsourced together with Merkle tree TW to server $\mathcal{S}$. Hence, when server $\mathcal{S}$ receives

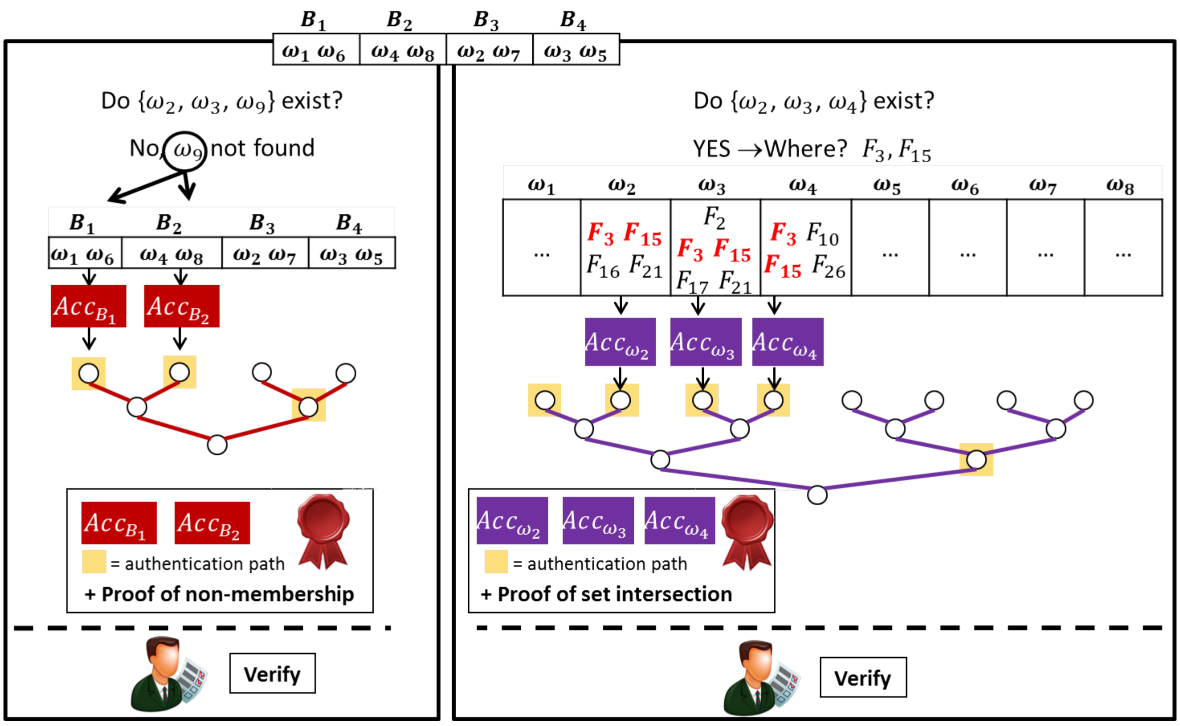

B_1 B_2 B_3 B_4
$\omega_1\ \omega_6$ $\omega_4\ \omega_8$ $\omega_2\ \omega_7$ $\omega_3\ \omega_5$
Do $\{\omega_2, \omega_3, \omega_9\}$ exist?
No, ω_9 not found
Acc_{B_1} Acc_{B_2}
= authentication path
+ Proof of non-membership
Verify
Do $\{\omega_2, \omega_3, \omega_4\}$ exist?
YES → Where? F_3, F_{15}
ω_1 ω_2 ω_3 ω_4 ω_5 ω_6 ω_7 ω_8
$F_3\ F_{15}\ F_{16}\ F_{21}$
$F_2\ F_3\ F_{15}\ F_{17}\ F_{21}$
$F_3\ F_{10}\ F_{15}\ F_{26}$
Acc_{ω_2} Acc_{ω_3} Acc_{ω_4}
= authentication path
+ Proof of set intersection
Verify

Figure 6.1: Overview of our protocol for Verifiable Conjunctive Keyword Search

a search query for a word ω, it finds the buckets corresponding to ω in index $\mathcal{I}$, retrieves the corresponding accumulator, generates a proof of membership (or non-membership), and authenticates the retrieved accumulator using the Merkle tree TW. Therefore, anyone holding the root of TW can verify the server's response.

The solution sketched above still does not identify which files exactly contain a word ω nor supports Verifiable Conjunctive Keyword Search (VCKS). Thus, data owner $\mathcal{O}$ constructs another Merkle tree TF whereby each leaf is mapped to a single keyword and associated with the polynomial-based accumulator of the subset of files containing that keyword. Data owner $\mathcal{O}$ then uploads files $\mathcal{F}$ and Merkle trees TW and TF to server $\mathcal{S}$. Given the root of TF, querier $\mathcal{Q}$ will be able to identify which subset of files contain a word ω. In addition, since polynomial-based accumulators allow efficient **verifiable set intersection**, querier $\mathcal{Q}$ will also be able to perform VCKS on files $\mathcal{F}$. Figure 6.1 depicts the steps of the search operation.

To summarize, the protocol for verifiable conjunctive keyword search is:

Efficient: Since our protocol is non-interactive and does not rely on heavy cryptographic mechanisms, it is efficient and practical. Namely algorithms QueryGen and Verify require less costs than executing algorithm Search.

Amortized: Algorithm Setup is a one-time expensive operation that pre-processes the files to be outsourced. However, Setup is executed only once for an unlimited number of search verifications.

Publicly delegatable: The data owner publishes public key $\mathsf{PK}_{\mathcal{F}}$ that enables anyone to submit conjunctive search queries to the server.

Publicly verifiable: The querier generates a public verification key $\mathsf{VK}_{\mathcal{Q}}$, tied to search query $\mathcal{W}$, enabling any verifier to check the result returned by the server.

Secure: We prove in Section 6.6 that our protocol is correct and sound.

6.4 Building Blocks

As mentioned in Section 6.3, our protocol for VCKS operates several building blocks to elaborate an efficient search scheme. We provide here a brief description of the tools underpinning our solution: Cuckoo hashing (Section 6.4.1), polynomial-based accumulators (Section 6.4.2) that allow verifiable test of membership and verifiable set intersection, and Merkle trees (Section 6.4.3). Note that our protocol will employ *symmetric* bilinear pairings. Symmetric bilinear pairings are bilinear pairings as defined in Section 4.3.1 with the characteristic that $\mathbb{G}_1 = \mathbb{G}_2 = \mathbb{G}$.

6.4.1 Cuckoo Hashing

Our VCKS solution builds an authenticated index of the keywords contained in the outsourced files. For this purpose, we employ the Cuckoo hashing approach, introduced by Pagh and Rodler [137].

This technique belongs to the multiple choice hashing techniques. In the seminal work by Pagh and Rodler [137], an object can be stored in one of two possible buckets, each located in two different hash tables, corresponding to two hash functions. Note that these functions are not *collision*-free hash functions, in the sense that two different objects can be assigned by the hash functions to the same bucket. To add a new object, the first hash function, thus the first table, is used. If the bucket assigned to this new object is full in the first table (this event is called a *collision*), then the object currently occupying this location is "kicked out", making possible to place the new item. The removed item is moved to the other bucket, using the second hash function that gives a position in the second table. This move may encounter another collision, thus requiring another element to be kicked out from its location. This insertion procedure is repeated until all objects find a free spot, or the number of insertion attempts reaches a predefined threshold to declare an insertion failure. In the worst case, the insertion procedure runs into a cycle, making it impossible to insert a new object. If this case happens, the method of *rehashing* should be performed: we choose two new hash functions and try to insert all the elements back into the tables using these new functions. Multiple rehash operations may be operated to succeed in inserting all the elements. To look for an item, it suffices to examine two locations, each in one of the two tables.

In our protocol, we leverage a variant proposed by Dietzfelbinger and Weidling [74]: Their solution inserts N elements using two independent and random hash functions $\mathcal{H}_1, \mathcal{H}_2 : \{0,1\}^* \rightarrow [1, m]$ into a single index $\mathcal{I}$ with m buckets B_i, such that: $m = \frac{1+\varepsilon}{d} N$, for $\varepsilon > 0$, and each bucket B_i stores at most d elements. As depicted in Figure 6.2, a lookup operation for a particular element x requires the evaluation of the two hash functions $\mathcal{H}_1(x)$ and $\mathcal{H}_2(x)$. In this Cuckoo hashing variant, the insertion of a new element x follows a random walk in the index, as suggested in [74]. In this approach, when a collision occurs in the first bucket B_1 pointed by $\mathcal{H}_1$, then item x is redirected to bucket B_2 given by $\mathcal{H}_2$. If this results in another collision, then an item y is randomly selected from one of the $2d$ elements stored in the buckets determined by $\mathcal{H}_1$ and $\mathcal{H}_2$. Then item y is kicked out from its location and replaced by x. Finally, the insertion procedure is repeated for y, until all elements can be stored in one bucket. Dietzfelbinger and Weidling [74] prove that the expected time to insert x is bounded by $(1/\varepsilon)^{\mathcal{O}(\log d)}$.

6.4.2 Polynomial-based Accumulators and Applications

Cryptographic accumulators, first introduced by Benaloh and De Mare [31], are authentication primitives that allow (i) combining the elements of a given set in a *constant-size value*, representing a short and secure description of that set; (ii) efficiently proving, thanks to the existence of a *constant-size witness*, whether a particular element is member of that set. Note that a unique witness is computed for each accumulated element. Original accumula-

Figure 6.2: Cuckoo Hashing Algorithms

```
▼ Algorithm: CuckooInsert($\mathcal{I}, \mathcal{H}_1, \mathcal{H}_2, x$)
# Insert $x$ in index $\mathcal{I}$ using hash functions $\mathcal{H}_1, \mathcal{H}_2 : \{0,1\}^* \rightarrow [1, m]$
  1. Compute $i_1 = \mathcal{H}_1(x)$ and $i_2 = \mathcal{H}_2(x)$;
  2. If bucket $B_{i_1}$ is not full then
         Insert $x$ in $B_{i_1}$;
         Return;
     End
  3. If bucket $B_{i_2}$ is not full then
         Insert $x$ in $B_{i_2}$;
         Return;
     End
  4. If buckets $B_{i_1}$ and $B_{i_2}$ both full then
         Randomly choose $y$ from the $2d$ elements in $B_{i_1} \cup B_{i_2}$;
         Remove $y$;
         CuckooInsert($\mathcal{I}, \mathcal{H}_1, \mathcal{H}_2, x$);
         CuckooInsert($\mathcal{I}, \mathcal{H}_1, \mathcal{H}_2, y$);
         Return;
     End

▼ Algorithm: {true, false} ← CuckooLookup($\mathcal{I}, \mathcal{H}_1, \mathcal{H}_2, x$)
# Search for $x$ in index $\mathcal{I}$
  1. Compute $i_1 = \mathcal{H}_1(x)$ and $i_2 = \mathcal{H}_2(x)$;
  2. Return $(x \in B_{i_1}) \vee (x \in B_{i_2})$;
```

tors were designed for membership proofs. They have later been extended for the purpose of non-membership proofs: such accumulators provide non-membership witnesses that can certify, using the accumulator computed for the considered set, that a given element is not present in that set. Definition 17 details our definition of such an accumulator.

Definition 17 (Accumulators). *An accumulator is a tuple of four PPT algorithms (*KeyGenAcc*,* ComputeAcc*,* GenerateWitness*,*VerifyMembership*) defined as follows:*

▷KeyGenAcc(1^κ) → (SK, PK)**:** *This probabilistic algorithm takes as input security parameter 1^κ and outputs a secret key* SK *stored by the data owner and public key* PK *that allows the server to respond to membership queries and enables any third party to verify membership query responses.*

▷ComputeAcc(SK, S) → $\mathcal{A}cc(S)$**:** *The data owner runs this probabilistic algorithm that takes as input set $S = \{h_1, h_2, ..., h_n\}$ and secret key* SK *and computes the public accumulation value $\mathcal{A}cc(S)$.*

▷GenerateWitness($h, S,$ PK) → $w_{S,h}$**:** *Executed by the server, this algorithm takes as input public key* PK*, set S and a target element h used to test membership, that is, test whether h belongs to set S. It outputs a witness $w_{S,h}$ for membership or non-membership of h with respect to set S.*

▷VerifyMembership($h, \mathcal{A}cc(S), w_{S,h},$ PK) → $\{h \in S, h \notin S,$ Reject$\}$**:** *A third-party verifier executes this algorithm that takes as input public key* PK*, accumulator $\mathcal{A}cc(S)$, target element h and the corresponding witness $w_{S,h}$. If witness $w_{S,h}$ is valid, then this algorithm outputs the result of the membership test: either $h \in S$ or $h \notin S$. Otherwise, the algorithm outputs* Reject*.*

Accumulators were first based on RSA exponentiation and exploited in some research work for membership tests [31, 24, 49] or non-membership proofs [120]. An alternative to the RSA-based construction was proposed by Nguyen [132] to allow efficient *dynamic* accumulators, whereby insertions and deletions of elements in the set result in efficient updates of the accumulators and witnesses. These accumulators are computed by means of polynomials. RSA-based dynamic accumulators were suggested by Camenisch and Lysyanskaya [49] but their solution requires the knowledge of some trapdoor information to perform updates. The dynamic accumulators introduced by Nguyen [132] rely on the sole properties of polynomials to support membership proofs. These polynomial-based accumulators [132] were then extended by Damgård and Triandopoulos [69] with the functionality of non-membership proofs.

Polynomial-based accumulators (also referred as bilinear accumulators) are a powerful tool: not only have they been leveraged for verifiable tests of membership [132, 69] but they have been employed for verifiable set intersections [140, 52], as we will explain in the following lines. We apply and tailor these primitives to our protocol for VCKS. Succinctly, our scheme uses polynomial-based accumulators to map a set of keywords to a unique polynomial such that each keyword in the set corresponds to a root of the polynomial. Accordingly, a verifiable test of membership relying on polynomial-based accumulators accommodates the question on whether a particular keyword is present in the set. On the other hand, our scheme computes additional accumulators: For each keyword, we compute an accumulator of the files that contain that particular keyword. Hence, to support verifiable conjunctive keyword search, a verifiable set intersection solution based on these accumulators supplies effective information to identify all the files that contain all the searched keywords.

In the rest of this section, we give an overview polynomial-based accumulators and the two protocols for verifiable test of membership and verifiable set intersection.

6.4.2.1 Polynomial-based accumulators.

Figure 6.3: Accumulator Computation

```
▼ Algorithm: $(\mathsf{SK}, \mathsf{PK}) \leftarrow \mathsf{KeyGenAcc}(1^\kappa)$
# Generates the keying material
    1. Select random $\alpha \in \mathbb{F}_p^*$;
    2. Set $\mathsf{SK} = \alpha$;
    3. For $0 \le i \le D$ do
           Compute $g^{\alpha^i}$;
       End
    4. Set $\mathsf{PK} = \{g, g^\alpha, g^{\alpha^2}, ..., g^{\alpha^D}\}$;
    5. Return $(\mathsf{SK}, \mathsf{PK})$;

▼ Algorithm: $\mathcal{Acc}(S) \leftarrow \mathsf{ComputeAcc}(\mathsf{SK}, S)$
# Compute the accumulator value of set $S = \{h_1, h_2, ..., h_n\}$
    1. Parse $\mathsf{SK} = \alpha$;
    2. Compute $P_S(\alpha) = \prod_{h_i \in S}(\alpha - h_i)$;
    3. Compute $\mathcal{Acc}(S) = g^{P_S(\alpha)}$;
    4. Return $\mathcal{Acc}(S)$;
```

The authors of [86], and later the ones of [140, 132], introduced the notion of *characteristic polynomial*, according to which a set $S = \{h_1, ..., h_n\}$ of elements in $\mathbb{F}_p$ can be encoded by its characteristic polynomial $P_S(X) = \prod_{h_i \in S}(X - h_i)$ (here X is the formal variable).

Let g be a random generator of a bilinear group $\mathbb{G}$ of prime order p. Given the public tuple $(g, g^\alpha, g^{\alpha^2}, ..., g^{\alpha^D})$, where α is randomly chosen in $\mathbb{F}_p^*$ and $D \ge n$, Nguyen [132] defines the public accumulator of the elements in S:

$$\mathcal{Acc}(S) = g^{P_S(\alpha)} \in \mathbb{G}$$

Figure 6.3 gives the detailed instructions of the two first algorithms presented in Definition 17, namely KeyGenAcc and ComputeAcc (the two other algorithms are considered in Section 6.4.2.2).

The different values that come into play in the accumulator computation can be interpreted as follows: The value D imposes an upper-bound on the number of elements to be accumulated in the set; α coincides with the accumulator secret key and the tuple $(g, g^{\alpha}, g^{\alpha^2}, ..., g^{\alpha^D})$ represents the corresponding public key. It should be mentioned that the accumulator value can be computed by any party who has access to this tuple using Fast Fourier Transform (FFT) interpolation, as explained in [52].

We indicated above that polynomial-based accumulators accommodate verifiable test of membership and set intersections. In the following paragraphs, we give an overview of these two protocols.

6.4.2.2 Verifiable Test of Membership.

To prove that a particular keyword is present or absent from some outsourced files, our VCKS solution operates the scheme for test of membership proposed by Damgård and Triandopoulos [69]. The authors observe that:

- any element $h \in \mathbb{F}_p$ is in set S iff $P_S(h) = 0$;
- for all element $h \in \mathbb{F}_p$, there exists a unique polynomial $Q_{S,h}$ such that $P_S(X) = (X - h) \cdot Q_{S,h}(X) + P_S(h)$.

In particular, for any h, the accumulator can be written as

$$\mathcal{A}cc(S) = g^{P_S(\alpha)} = g^{(\alpha-h)\cdot Q_{S,h}(\alpha)+P_S(h)} = \Omega_{S,h}^{(\alpha-h)} g^{P_S(h)}.$$

The value $\Omega_{S,h} \overset{\mathsf{def}}{=} g^{Q_{S,h}(\alpha)}$ defines the witness of h with respect to $\mathcal{A}cc(S)$: It constitutes a *membership* witness if $h \in S$, a *non-membership* witness otherwise. Following these observations, the authors in [69] define a verifiable test of membership depicted in Figure 6.4. This test is secure under the D-Strong Diffie-Hellman (D-SDH) assumption.

Definition 18 (D-Strong Diffie-Hellman Assumption). *Let $\mathbb{G}$ be a cyclic group of prime order p generated by g. We say that the D-SDH holds in $\mathbb{G}$ if, given the tuple $(g, g^{\alpha}, g^{\alpha^2}, ..., g^{\alpha^D}) \in \mathbb{G}^{D+1}$, for some randomly chosen $\alpha \in \mathbb{F}_p^*$, no PPT algorithm $\mathcal{A}$ can find a pair $(x, g^{\frac{1}{\alpha+x}}) \in \mathbb{F}_p^* \times \mathbb{G}$ with a non-negligible advantage.*

Figure 6.4: Verifiable Test of Membership

```
▼ Algorithm: $w_{S,h} \leftarrow \mathsf{GenerateWitness}(h, S, \mathsf{PK})$
# Compute the proof of (non-) membership of h with respect to set S
  1. Compute the value $P_S(h) = \prod_{h_i \in S}(h - h_i)$;
  2. Determine polynomial $Q_{S,h}$ such that $P_S(X) = (X - h) \cdot Q_{S,h}(X) + P_S(h)$;
  3. Compute the witness $\Omega_{S,h} = g^{Q_{S,h}(\alpha)}$;
  4. Return $w_{S,h} = (P_S(h), \Omega_{S,h})$;

▼ Algorithm: $\{h \in S, h \notin S, \mathsf{Reject}\} \leftarrow \mathsf{VerifyMembership}(h, \mathcal{A}cc(S), w_{S,h}, \mathsf{PK})$
# Verify the proof and output the result of the test of membership
  1. Parse $w_{S,h} = (P_S(h), \Omega_{S,h})$;
  2. Verify $e(\Omega_{S,h}, g^{\alpha} \cdot g^{-h}) e(g^{P_S(h)}, g) \overset{?}{=} e(\mathcal{A}cc(S), g)$.
     If it fails then return Reject;
  3. If $P_S(h) = 0$ then return $h \in S$ else return $h \notin S$;
```

Figure 6.5: Verifiable Set Intersection

```
▼ Algorithm: $(I, \Pi_I) \leftarrow \mathsf{ProveIntersection}(S_1, ..., S_k)$
# Generate the proof for the intersection of the k sets $S_1, ..., S_k$
    1. Compute $I = S_1 \cap ... \cap S_k$ and its characteristic polynomial $P$;
    2. Compute the polynomials $U_i = \frac{P_i}{P}$ and the values $\Delta_i = g^{U_i(\alpha)}$;
    3. Compute the polynomials $V_i$ such that $\sum_i U_i V_i = 1$;
    4. Compute the values $\Gamma_i = g^{V_i(\alpha)}$;
    5. Define $\Pi_I = \{(\Delta_1, \Gamma_1), ..., (\Delta_k, \Gamma_k)\}$;
    6. Return $(I, \Pi_I)$;

▼ Algorithm: $\{\mathsf{Accept}, \mathsf{Reject}\} \leftarrow \mathsf{VerifyIntersection}(I, \Pi_I, \{\mathcal{Acc}(S_i)\}_{1 \leq i \leq k})$
# Verifiy the proof for I, the intersection of the sets $S_1, ..., S_k$
    1. Parse $\Pi_I = \{\{\Delta_i, \Gamma_i\}_{1 \leq i \leq k}\}$;
    2. Verify the following equalities:
        – $e(\mathcal{Acc}(I), \Delta_i) \stackrel{?}{=} e(\mathcal{Acc}(S_i), g)$        – $\prod_i e(\Delta_i, \Gamma_i) \stackrel{?}{=} e(g, g)$
        # Check $I \subseteq S_i$ for $1 \leq i \leq k$          # Check $\bigcap_i (S_i \backslash I) = \emptyset$
       If any of the checks fails then return Reject else return Accept;
```

6.4.2.3 Verifiable Set Intersection.

To support conjunctive keyword search queries, our VCKS solution exploits the scheme for verifiable set intersection proposed by Canetti *et al.* [52]. In particular, this scheme is used to determine which files contain all the keywords of a conjunctive search query.

In the verifiable set intersection protocol, we consider k sets S_i and their respective characteristic polynomials P_i. If we denote $I = \bigcap_i S_i$ and P the characteristic polynomial of I then $P = \gcd(P_1, P_2, .., P_k)$. It follows that the k polynomials $U_i = \frac{P_i}{P}$ identify the sets $S_i \backslash I$. Since $\bigcap_i (S_i \backslash I) = \emptyset$, $\gcd(U_1, U_2, ..., U_k) = 1$. Therefore, according to Bézout's identity[99] [35], there exist polynomials V_i such that $\sum_i U_i V_i = 1$.

Based on these observations, Canetti et al. [52] define a protocol for verifiable set intersection described in Figure 6.5 that involves the accumulators of each set S_i and of their intersection. In particular, using these accumulators and the properties of the characteristic polynomials we just pointed out, the protocol checks the following relations satisfied by the intersection:

Subset: For all set S_i, $I \subseteq S_i$. In other words, all elements found in the intersection actually appear in all sets S_i.

Complement Disjointness: $\bigcap_i (S_i \backslash I) = \emptyset$. It means that no element that must belong to the intersection is omitted.

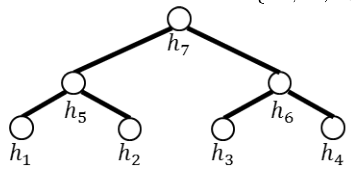
The intersection verification is secure if the D-Strong Bilinear Diffie-Hellman (D-SBDH) assumption holds.

Definition 19 (D-Strong Bilinear Diffie-Hellman Assumption). *Let $\mathbb{G}$ and $\mathbb{G}_T$ be cyclic groups of prime order p, g a generator of $\mathbb{G}$, and e a bilinear pairing. We say that the D-SBDH holds if, given $(g, g^{\alpha}, g^{\alpha^2}, ..., g^{\alpha^D}) \in \mathbb{G}^{D+1}$, for some randomly chosen $\alpha \in \mathbb{F}_p^*$, no PPT algorithm $\mathcal{A}$ can find a pair $(x, e(g,g)^{\frac{1}{\alpha+x}}) \in \mathbb{F}_p^* \times \mathbb{G}_T$ with a non-negligible advantage.*

[99] The Bézout identity states that if D is the GCD of two polynomials A and B then there exist two polynomials U and V such that $AU + BV = D$.

Figure 6.6: Merkle Tree Algorithms

/ We consider set $S = \{h_1, ..., h_n\}$, build a Merkle tree of S and authenticate an element $h \in S$. */*



```
▼ Algorithm: $\mathsf{T} \leftarrow \mathsf{BuildMT}(S, H)$
# Create Merkle tree $\mathsf{T}$ whose leaves are elements in set $S$
  1. $\mathsf{T} \leftarrow \{h_1, ..., h_n\}$
  2. For $1 \leq i < 2n - 1$, $(i + +2)$ do
        $h_{n+\lceil \frac{i}{2} \rceil} \leftarrow H(h_i \| h_{i+1})$;
        $\mathsf{T} \leftarrow \mathsf{T} \cup \{h_{n+\lceil \frac{i}{2} \rceil}\}$;
     End

▼ Algorithm: $\mathsf{path} \leftarrow \mathsf{GenerateMTProof}(\mathsf{T}, h_i)$
# Generate the authentication path for element $h_i$ in Merkle tree $\mathsf{T}$
  1. $\mathsf{path} \leftarrow \{\}$;
  2. While $|\mathsf{path}| < \log n$ do
        $k \leftarrow i + (-1)^{i+1}$;
        $\mathsf{path} \leftarrow \mathsf{path} \cup \{h_k\}$;
        $i \leftarrow n + \lceil \frac{i}{2} \rceil$;
     End

▼ Algorithm: $\{\mathsf{Accept}, \mathsf{Reject}\} \leftarrow \mathsf{VerifyMTProof}(h_i, \mathsf{path}, \sigma)$
# Verify the authentication path for $h_i$ with respect to root $\sigma$
  1. $\sigma' \leftarrow h_i$;
  2. For $1 \leq k \leq |\mathsf{path}|$ do
        If $i \bmod 2$
        Then $\sigma' \leftarrow H(\sigma' \| \mathsf{path}[k])$;
        Else $\sigma' \leftarrow H(\mathsf{path}[k] \| \sigma')$;
        $i \leftarrow n + \lceil \frac{i}{2} \rceil$;
     End
  3. If $\sigma' = \sigma$ then return Accept else return Reject;
```

6.4.3 Binary Merkle Trees

Merkle trees enable the authentication of elements in a set $S = \{h_1, ..., h_n\}$ without transferring the entire set. More precisely, Merkle trees allow any party having possession of the root to verify whether an element h is in set S. In our VCKS protocol, Merkle trees are used to authenticate the accumulators. Indeed, these accumulators will be stored at the cloud along with the outsourced files. Thus the Merkle trees will authenticate them so that a verifier can be sure that they really are the ones she expects.

In the following, we introduce the algorithms that build a binary Merkle tree for a set S and authenticate the elements in that set[100].

- $\mathsf{T} \leftarrow \mathsf{BuildMT}(S, H)$. On input of set S and a cryptographic hash function H, $\mathsf{BuildMT}$ builds a binary Merkle tree T. Each leaf L_i of the tree maps an element h_i in set S and each internal node stores the hash of the concatenation of the children of that node, i.e $H(\mathsf{left} \| \mathsf{right})$. Without loss of generality, we denote by σ the root of T.
- $\mathsf{path} \leftarrow \mathsf{GenerateMTProof}(\mathsf{T}, h_i)$. To authenticate the element h_i stored in leaf L_i of Merkle tree T, a prover runs $\mathsf{GenerateMTProof}$ which, on input of h_i and T, outputs the *authentication path* for leaf L_i corresponding to element h_i, that is, the set of the siblings of the nodes on the path from L_i to root σ. We denote path the authentication path output by $\mathsf{GenerateMTProof}$.
- $\{\mathsf{Accept}, \mathsf{Reject}\} \leftarrow \mathsf{VerifyMTProof}(h_i, \mathsf{path}, \sigma)$. On input of an element h_i stored in leaf

[100]In our protocol, set S represents a collection of accumulator values.

L_i, the corresponding authentication path and the correct root value σ of tree T, VerifyMTProof verifies that the value of the root computed from h and path equals the expected value σ.

Figure 6.6 shows the details of these Merkle tree algorithms. As a matter of illustration, we show and label a Merkle tree whose leaves correspond to set $\{h_1, h_2, h_3, h_4\}$.

6.5 Protocol Description

In our protocol for VCKS, data owner $\mathcal{O}$ outsources the storage of a set of files $\mathcal{F} = \{F_1, F_2, ..., F_n\}$ to a server $\mathcal{S}$. Once the data is uploaded, any third-party querier $\mathcal{Q}$ can search for some keywords in the set of files $\mathcal{F}$ and verify the correctness of the search results returned by $\mathcal{S}$. The collection of searchable keywords in $\mathcal{F}$ is sorted in the lexicographic order and is defined as $\mathbb{W} = \{\omega_1, \omega_2, ..., \omega_N\}$. As mentioned in Definition 13, the proposed protocol comprises three phases: *Setup*, *Search* and *Verification*. In the description of our scheme, we refer to the notations listed in Table 6.1.

Table 6.1: List of notations in our protocol for VCKS

Index	Description	Range
n	Number of files F_i in set $\mathcal{F}$	-
j	Index of a file	$[\![1, n]\!]$
N	Number of keywords contained in $\mathcal{F}$	-
k	Number of keywords contained in a search query $\mathcal{W}$	-
i	Index of keywords	$[\![1, N]\!]$ or $[\![1, k]\!]$
m	Number of buckets in index $\mathcal{I}$	
d	Number of keywords in each bucket in index $\mathcal{I}$	

6.5.1 Setup

Before outsourcing her files to the cloud, data owner $\mathcal{O}$ enters the *Setup* phase and invokes algorithm Setup to process the files and enable verifiable keyword search.

On input of security parameter 1^κ and set of files $\mathcal{F}$, algorithm Setup outputs the public parameters param, a public key $\mathsf{PK}_\mathcal{F}$ and a search key $\mathsf{LK}_\mathcal{F}$. As shown in Figure 6.7, Setup operates in four steps.

1. It first generates the public parameters needed for the protocol.
2. It builds index $\mathcal{I}$ for the set $\mathbb{W} = \{\omega_1, \omega_2, ..., \omega_N\}$ using Cuckoo hashing. Without loss of generality, we assume that $\mathbb{W}$ is composed of the list of distinct words in $\mathcal{F}$ sorted in a lexicographic order.
3. Setup authenticates index $\mathcal{I}$ with Merkle tree TW where each leaf is mapped to a bucket in $\mathcal{I}$. We denote σ_W the root of tree TW.
4. Setup builds Merkle tree TF, of root σ_F, to identify which files exactly contain the keywords.

At the end of this phase, data owner $\mathcal{O}$ publishes parameters param, publishes public key $\mathsf{PK}_\mathcal{F}$ and transmits the search key $\mathsf{LK}_\mathcal{F}$ to server $\mathcal{S}$. When the latter receives $\mathsf{LK}_\mathcal{F}$, it creates a hash table HT where each entry is mapped to a keyword ω_i and stores the pair $(i, \mathsf{pointer})$ such that: i is the position of keyword ω_i in set $\mathbb{W}$ and in tree TF; whereas pointer points to a linked list storing the identifiers of files $\mathcal{F}_{\omega_i}$ that contain keyword ω_i. As such, hash table HT enables server $\mathcal{S}$ to find the position of ω_i in TF and to identify the files containing ω_i easily. In the remainder of this chapter, we assume that server $\mathcal{S}$ does not store $\mathsf{LK}_\mathcal{F}$ as $(\mathcal{I}, \mathsf{TW}, \mathsf{TF}, \mathcal{F}, \mathbb{W}, \{\mathcal{F}_{\omega_i}\}_{1 \leq i \leq N})$, but rather as $\mathsf{LK}_\mathcal{F} = (\mathcal{I}, \mathsf{TW}, \mathsf{TF}, \mathcal{F}, \mathsf{HT})$.

Figure 6.7: Setup

```
▼ Algorithm: $(\mathsf{param}, \mathsf{PK}_{\mathcal{F}}, \mathsf{LK}_{\mathcal{F}}) \leftarrow \mathsf{Setup}(1^{\kappa}, \mathcal{F})$
# $\mathcal{F} = \{F_1, ..., F_n\}$: set of files
# $\mathbb{W} = \{\omega_1, .., \omega_N\}$: list of distinct words in $\mathcal{F}$ sorted in lexicographic order.
1. Parameter generation
    Pick $D, g, \mathbb{G}, \mathbb{G}_T, e, H : \{0,1\}^* \rightarrow \mathbb{F}_p$ as function of security parameter $1^{\kappa}$;
    Pick random $\alpha \in \mathbb{F}_p^*$ and compute public values $\{g, g^{\alpha}, ..., g^{\alpha^D}\}$;
2. Construction of the Index
    # Create an index $\mathcal{I}$ with $m$ buckets of size $d$ where $d < D$
    Identify $\mathbb{W}$ from $\mathcal{F}$;
    Pick random hash functions $\mathcal{H}_1, \mathcal{H}_2 : \{0,1\}^* \rightarrow [1, m]$;
    For $\omega_i \in \mathbb{W}$ do
        Compute $h_i = H(\omega_i)$;
        Run $\mathsf{CuckooInsert}(\mathcal{I}, \mathcal{H}_1, \mathcal{H}_2, h_i)$;
    End
3. Authentication of Index
    For $B_i \in \mathcal{I}$ do
        Compute $P_{B_i}(\alpha) = \prod_{h_j \in B_i} (\alpha - h_j)$;
        Compute $\mathsf{AW}_i = \mathcal{A}cc(B_i) = g^{P_{B_i}(\alpha)}$;
        Compute $\mathsf{HW}_i = H(\mathsf{AW}_i || i)$, where $i$ is the position of $B_i$ in $\mathcal{I}$;
    End
    $\mathsf{TW} = \mathsf{BuildMT}(\{\mathsf{HW}_i\}_{1 \leq i \leq m}, H)$;
4. Encoding of files
    # Identify which files contain the keywords
    For $F_j \in \mathcal{F}$ do
        Generate $\mathsf{fid_j}$;
    End
    For $\omega_i \in \mathbb{W}$ do
        Identify $\mathcal{F}_{\omega_i}$, the subset of files that contain $\omega_i$;
        Compute $P_i(\alpha) = \prod_{\mathsf{fid}_j \in \mathcal{F}_{\omega_i}} (\alpha - \mathsf{fid}_j)$;
        Compute $\mathsf{AF}_i = \mathcal{A}cc(\mathcal{F}_{\omega_i}) = g^{P_i(\alpha)}$;
        Compute $\mathsf{HF}_i = H(\mathsf{AF}_i || \omega_i)$;
    End
    $\mathsf{TF} = \mathsf{BuildMT}(\{\mathsf{HF}_i\}_{1 \leq i \leq N}, H)$.

 Return $\mathsf{param} = (g, \mathbb{G}, \mathbb{G}_T, e, H, \mathcal{H}_1, \mathcal{H}_2)$;

 Return $\mathsf{PK}_{\mathcal{F}} = (\{g^{\alpha^i}\}_{0 \leq i \leq D}, \sigma_W, \sigma_F)$;

 Return $\mathsf{LK}_{\mathcal{F}} = (\mathcal{I}, \mathsf{TW}, \mathsf{TF}, \mathcal{F}, \mathbb{W}, \{\mathcal{F}_{\omega_i}\}_{1 \leq i \leq N})$.
```

6.5.2 Search

In this phase, our protocol for VCKS uses the algorithms of verifiable test of membership and verifiable set intersection presented in Section 6.4. Indeed, the actual search realizes two operations. The first operation answers the question: *Do the queried keywords are present in the outsourced files?* A negative answer induces that at least one keyword in the query cannot be found. The verifiable test of membership is thus employed to prove that this particular keyword does not belong to the outsourced files. The second operation only occurs when the first question receives a positive answer (i.e all the keywords in the conjunctive search query are found). However this positive answer does not tell where the keywords can be found. Hence, the second operation aims at answering the question: *Which outsourced files contain all the keywords of the conjunctive search query?* The verifiable set intersection algorithm is thus applied to identify the files that contain the searched keywords.

We assume in what follows that a querier $\mathcal{Q}$ wants to identify the set of files $\mathcal{F}_{\mathcal{W}} \subset \mathcal{F}$ that contain all words in $\mathcal{W} = \{\omega_1, \omega_2, ..., \omega_k\}$. To that effect, as specified in our tailored definition of publicly verifiable computation in Definition 13, querier $\mathcal{Q}$ enters the *Search* phase and first runs algorithm QueryGen (cf. Figure 6.8) which returns the query $\mathcal{E}_Q = \mathcal{W}$ and the

Figure 6.8: Verifiable Conjunctive Keyword Search

```
▼ Algorithm: $\{\mathcal{E}_Q, \mathsf{VK}_Q\} \leftarrow \mathsf{QueryGen}(\mathcal{W}, \mathsf{PK}_\mathcal{F})$
   1. Assign $\mathcal{E}_Q = \mathcal{W}$ and $\mathsf{VK}_Q = (\mathsf{PK}_\mathcal{F}, \mathcal{W})$;
   2. Return $\{\mathcal{E}_Q, \mathsf{VK}_Q\}$;

▼ Algorithm: $\mathcal{E}_R \leftarrow \mathsf{Search}(\mathcal{E}_Q, \mathsf{LK}_\mathcal{F})$
   1. Parse $\mathcal{E}_Q = \mathcal{W}$ and $\mathsf{LK}_\mathcal{F} = (\mathcal{I}, \mathsf{TW}, \mathsf{TF}, \mathcal{F}, \mathsf{HT})$;
   2. For $\omega_i \in \mathcal{W}$ do
          Compute $h_i = H(\omega_i)$;
          If $\mathsf{CuckooLookup}(\mathcal{I}, \mathcal{H}_1, \mathcal{H}_2, h_i) = \mathsf{false}$ then
              # Keyword $\omega_i$ is not in $\mathcal{F}$
              Compute $i_1 = \mathcal{H}_1(h_i)$ and $i_2 = \mathcal{H}_2(h_i)$;
              Compute $\Pi_1 = \mathsf{GenerateWitness}(h_i, B_{i_1})$;
              Compute $\Pi_2 = \mathsf{GenerateWitness}(h_i, B_{i_2})$;
              Compute $\mathsf{AW}_{i_1} = \mathcal{A}cc(B_{i_1})$ and $\mathsf{HW}_{i_1} = H(\mathsf{AW}_{i_1}||i_1)$;
              Compute $\mathsf{AW}_{i_2} = \mathcal{A}cc(B_{i_2})$ and $\mathsf{HW}_{i_2} = H(\mathsf{AW}_{i_2}||i_2)$;
              Compute $\mathsf{path}_1 = \mathsf{GenerateMTProof}(\mathsf{TW}, \mathsf{HW}_{i_1})$;
              Compute $\mathsf{path}_2 = \mathsf{GenerateMTProof}(\mathsf{TW}, \mathsf{HW}_{i_2})$;
              Return $\mathcal{E}_R = (\emptyset, \omega, \mathsf{AW}_{i_1}, \mathsf{AW}_{i_2}, \Pi_1, \Pi_2, \mathsf{path}_1, \mathsf{path}_2)$;
          End
      End
   3. # All the keywords have been found
      For $\omega_i \in \mathcal{W}$ do
          Determine $\mathcal{F}_{\omega_i}$ using $\mathsf{HT}$;  # the set of files that contain $w_i$
          Compute $\mathsf{AF}_i = \mathcal{A}cc(\mathcal{F}_{\omega_i})$ and $\mathsf{HF}_i = H(\mathsf{AF}_i||\omega_i)$;
          Determine position $l$ of $w_i$ in $\mathsf{TF}$ using $\mathsf{HT}$;
          # $\mathsf{HF}_i$ is in the $l^{\text{th}}$ leaf of $\mathsf{TF}$
          Compute $\mathsf{path}_i = \mathsf{GenerateMTProof}(\mathsf{TF}, \mathsf{HF}_i)$;
      End
      # $\mathcal{F}_\mathcal{W} = \mathcal{F}_{\omega_1} \cap ... \cap \mathcal{F}_{\omega_k}$ is the set of files that contain all the words in $\mathcal{W}$
      Compute $(\mathcal{F}_\mathcal{W}, \Pi_\mathcal{W}) = \mathsf{ProveIntersection}(\mathcal{F}_{\omega_1}, ..., \mathcal{F}_{\omega_k})$;
      Return $\mathcal{E}_R = (\mathcal{F}_\mathcal{W}, \Pi_\mathcal{W}, \{\mathsf{AF}_i\}_{1\leq i\leq k}, \{\mathsf{path}_i\}_{1\leq i\leq k})$;
```

public verification key $\mathsf{VK}_Q = (\mathsf{PK}_\mathcal{F}, \mathcal{W})$. Querier $\mathcal{Q}$ then sends query $\mathcal{E}_Q$ to server $\mathcal{S}$.

On reception of query $\mathcal{E}_Q$, server $\mathcal{S}$ invokes algorithm Search (cf. Figure 6.8) which searches the index $\mathcal{I}$ for every individual keyword $\omega_i \in \mathcal{W}$. If all the keywords $\omega_i \in \mathcal{W}$ are found in the index, then Search identifies the subset of files $\mathcal{F}_{\omega_i}$ that contains ω_i and outputs the intersection of all these subsets $\mathcal{F}_\mathcal{W} = \mathcal{F}_{\omega_1} \cap ... \cap \mathcal{F}_{\omega_k}$. Moreover, to prove the correctness of the response (*i.e.* to prove that $\mathcal{F}_\mathcal{W}$ was computed correctly), Search (i) authenticates the accumulators of each set $\mathcal{F}_{\omega_i}$ using Merkle tree TF; and (ii) generates a proof of intersection for $\mathcal{F}_\mathcal{W} = \mathcal{F}_{\omega_1} \cap ... \cap \mathcal{F}_{\omega_k}$ using the verification algorithm described in Figure 6.5.

If at least one keyword ω_i is not found, then Search aborts for the remaining keywords in the query, returns ω_i and proves the correctness of its response by (i) authenticating the accumulators of buckets B_{i_1} and B_{i_2} associated with ω_i (if it was stored in index $\mathcal{I}$) using Merkle tree TW; and (ii) generating a proof of non-membership of keyword ω_i with respect to buckets B_{i_1} and B_{i_2} (cf. Figure 6.4).

6.5.3 Verification

On reception of the search result, verifier $\mathcal{V}$ checks the correctness of the server's response by calling algorithm Verify as shown in Figure 6.9. More precisely, if server $\mathcal{S}$ advertises that it has found all the keywords $\mathcal{W}$ in index $\mathcal{I}$, then algorithm Verify checks that the returned intersection $\mathcal{F}_\mathcal{W}$ is correct using the verification algorithm of Merkle tree and verifiable set intersection. Otherwise, $\mathcal{V}$ verifies that the returned keyword is actually not in $\mathcal{F}$ using the verification algorithm of Merkle tree and verifiable test of membership.

Figure 6.9: Search Verification

```
▼ Algorithm: $\mathsf{out} \leftarrow \mathsf{Verify}(\mathcal{E}_R, \mathsf{VK}_Q)$
  1. Parse $\mathsf{VK}_Q = (\mathsf{PK}_{\mathcal{F}}, \mathcal{W})$;
  2. If $\mathcal{W}$ found in $\mathcal{F}$ then
       Parse $\mathcal{E}_R = (\mathcal{F}_{\mathcal{W}}, \Pi_{\mathcal{W}}, \{\mathsf{AF}_i\}_{1\le i\le k}, \{\mathsf{path}_i\}_{1\le i\le k})$;
       For $\omega_i \in \mathcal{W}$ do
         If $\mathsf{VerifyMTProof}(H(\mathsf{AF}_i \| \omega_i), \mathsf{path}_i, \sigma_F) = \mathsf{Reject}$
         Then return $\mathsf{out} = \perp$;
       End
       Compute $\mathcal{Acc}(\mathcal{F}_{\mathcal{W}})$;
       If $\mathsf{VerifyIntersection}(\mathcal{F}_{\mathcal{W}}, \Pi_{\mathcal{W}}, \{\mathsf{AF}_i\}_{1\le i\le k}) = \mathsf{Accept}$;
       Then return $\mathsf{out} = \mathcal{F}_{\mathcal{W}}$ else return $\mathsf{out} = \perp$;
     End
  3. If at least one keyword $\omega_i$ is not found in $\mathcal{F}$ then
       Parse $\mathcal{E}_R = (\emptyset, \omega_i, \mathsf{AW}_{i_1}, \mathsf{AW}_{i_2}, \Pi_1, \Pi_2, \mathsf{path}_1, \mathsf{path}_2)$;
       Compute $h_i = H(\omega_i)$, $i_1 = \mathcal{H}_1(h_i)$ and $i_2 = \mathcal{H}_2(h_i)$;
       If $\mathsf{VerifyMTProof}(H(\mathsf{AW}_{i_1} \| i_1), \mathsf{path}_1, \sigma_W) = \mathsf{Reject}$
       Then return $\mathsf{out} = \perp$;
       If $\mathsf{VerifyMTProof}(H(\mathsf{AW}_{i_2} \| i_2), \mathsf{path}_2, \sigma_W) = \mathsf{Reject}$
       Then return $\mathsf{out} = \perp$;
       If $\mathsf{VerifyMembership}(h_i, \mathsf{AW}_{i_1}, \Pi_1) = \mathsf{Reject}$
       Then return $\mathsf{out} = \perp$;
       If $\mathsf{VerifyMembership}(h_i, \mathsf{AW}_{i_2}, \Pi_2) = \mathsf{Reject}$
       Then return $\mathsf{out} = \perp$;
       Return $\mathsf{out} = \emptyset$;
     End
```

Figure 6.10: Algorithm UpdateQuery

```
▼ Algorithm: $(\mathcal{U}_Q, \mathsf{VK}_Q) \leftarrow \mathsf{UpdateQuery}(F_j, \mathcal{W}_j, \mathsf{op})$
# Let denote $\mathcal{W}_{j,\mathsf{del}}$ the list of keywords deleted from updated file $F_j$.
# Let denote $\mathcal{W}_{j,\mathsf{add}}$ the list of keywords added to updated file $F_j$.
# Hence, $\mathcal{W}_j = (\mathcal{W}_{j,\mathsf{del}}, \mathcal{W}_{j,\mathsf{add}})$.
1. $\mathcal{U}_Q = (F_j, \mathcal{W}_j, \mathsf{op})$;
2. $\mathsf{VK}_Q = \mathsf{PK}_{\mathcal{F}}$;
3. Return $(\mathcal{U}_Q, \mathsf{VK}_Q)$;
```

6.5.4 Supporting Dynamic Data

Although we can use digital signatures instead of Merkle trees to authenticate the accumulators, they are not practical to support dynamic data. Thanks to Merkle trees, our solution enables data owner $\mathcal{O}$ to update its outsourced files and the set of searchable keywords efficiently. In particular, we are concerned by the fact that search operations should also be verifiable after updates (such as insertion, modification or deletion of a file in the set of outsourced files) without the data owner being required to download the whole database, perform the update and rebuild the entire system to enable verifiable conjunctive keyword search (namely to build from scratch the different data structures used in our protocol: the Cuckoo hash index, tables and Merkle trees). Figure 6.10 depicts the operations performed by UpdateQuery while Figure 6.11 and Figure 6.12 show the procedure of algorithms Update and VerifyUpdate respectively. We consider three possible update scenarios.

1. File update without updating the set of searchable keywords, $\mathbb{W}' = \mathbb{W}$: In this case, whatever the update operation is (file modification, deletion or insertion), the Cuckoo index $\mathcal{I}$ and tree TW remain unchanged since we assume the set of keywords is not affected by this update. On the other hand, server $\mathcal{S}$ is required to update Merkle tree TF and send a proof of correct update to the data owner. We will consider the different cases according to the nature of the update operation:

1. $\mathsf{op} = \mathsf{modify}$: The data owner executes $\mathsf{UpdateQuery}(F'_j, \mathcal{W}_j, \mathsf{modify})$. Then there exist

Figure 6.11: Algorithm Update

```
▼ Algorithm: $(\Pi_{\mathsf{upd}}, \mathsf{LK}'_{\mathcal{F}}) \leftarrow \mathsf{Update}(\mathsf{LK}_{\mathcal{F}}, \mathcal{U}_Q)$
1. Parse $\mathcal{U}_Q = (F'_j, \mathcal{W}_j, \mathsf{op})$;
2. Update $F'_j$ according to $\mathsf{op}$;
3. If $\mathcal{W}_j = \emptyset$ then
       $\Pi_{\mathsf{upd}} = \emptyset$;
       $\mathsf{LK}'_{\mathcal{F}} = \mathsf{LK}_{\mathcal{F}}$;
       Return $(\Pi_{\mathsf{upd}}, \mathsf{LK}'_{\mathcal{F}})$;
   End
4. If $\mathbb{W}' = \mathbb{W}$ then
     For $\omega_i \in \mathcal{W}_j$ do
         Compute $\mathsf{path}_i = \mathsf{GenerateMTProof}(\mathsf{HF}_i, \mathsf{TF})$;
         Compute $\mathsf{AF}'_i$ using FFT interpolation and $\mathsf{PK}_{\mathcal{F}}$;
         Compute $\mathsf{HF}'_i = H(\mathsf{AF}'_i || \omega_i)$;
     End
       Update $\mathsf{TF}$ with $\{\mathsf{HF}'_i\}_{\omega_i \in \mathcal{W}_j}$ to obtain $\mathsf{TF}'$ with root $\sigma'_F$;
     For $\omega_i \in \mathcal{W}_j$ do
         Compute $\mathsf{path}'_i = \mathsf{GenerateMTProof}(\mathsf{HF}'_i, \mathsf{TF}')$;
     End
       Assign $\Pi_{\mathsf{upd}} = (\{\mathsf{AF}_i, \mathsf{AF}'_i, \mathsf{path}_i, \mathsf{path}'_i\}_{\omega_i \in \mathcal{W}_j}, \sigma'_F)$;
       Assign $\mathsf{LK}'_{\mathcal{F}} = (\mathcal{I}, \mathsf{TW}, \mathsf{TF}', \mathcal{F}', \mathbb{W}, \{\mathcal{F}'_{\omega_i}\}_{1 \le i \le N})$;
       Return $(\Pi_{\mathsf{upd}}, \mathsf{LK}'_{\mathcal{F}})$;
   End
5. If $\mathbb{W}' = \mathbb{W} - \mathcal{W}_j$ then
       # This is the keyword deletion case
       # For simplicity we assume that $\mathcal{W}_j$ consists of a single keyword $\omega_i$
       Remove $\omega_i$ from assigned bucket $B_i$ in index $\mathcal{I}$ to obtain $\mathcal{I}'$;
       Compute $\mathsf{path}_{i,\mathsf{TW}} = \mathsf{GenerateMTProof}(\mathsf{HW}_i, \mathsf{TW})$;
       Compute $\mathsf{AW}'_i$ using FFT interpolation and $\mathsf{PK}_{\mathcal{F}}$;
       Compute $\mathsf{HW}'_i = H(\mathsf{AW}'_i || i)$;
       Update $\mathsf{TW}$ with $\mathsf{HW}'_i$ to obtain $\mathsf{TW}'$ with root $\sigma'_W$;
       Compute $\mathsf{HF}_i = H(\mathsf{AF}_i || \omega_i)$ and $\mathsf{path}_{i,\mathsf{TF}} = \mathsf{GenerateMTProof}(\mathsf{HF}_i, \mathsf{TF})$;
       Remove leaf associated with $\omega_i$ in $\mathsf{TF}$ to obtain $\mathsf{TF}'$ with root $\sigma'_F$;
       Assign $\Pi_{\mathsf{upd}} = (\mathsf{AW}_i, \mathsf{AW}'_i, \mathsf{path}_{i,\mathsf{TW}}, \sigma'_W, \mathsf{AF}_i, \mathsf{path}_{i,\mathsf{TF}}, \sigma'_F)$;
       Assign $\mathsf{LK}'_{\mathcal{F}} = (\mathcal{I}', \mathsf{TW}', \mathsf{TF}', \mathcal{F}', \mathbb{W}', \{\mathcal{F}'_{\omega_i}\}_{1 \le i \le N-1})$;
       Return $(\Pi_{\mathsf{upd}}, \mathsf{LK}'_{\mathcal{F}})$;
   End
6. If $\mathbb{W}' = \mathbb{W} \cup \mathcal{W}_j$ then
       # This is the keyword insertion case
       # For simplicity we assume that $\mathcal{W}_j$ consists of a single keyword $\omega_i$
       Run $\mathsf{CuckooInsert}$ in index $\mathcal{I}$ for $\omega_i$ and obtain $\mathcal{I}'$;
       # This possibly impacts several buckets. We denote $\mathsf{B}$ the set of impacted buckets
       # We denote, for each $B_k \in \mathsf{B}$, $\{\omega_i\}_{\mathsf{add},k}$ the set of keywords added to bucket $B_k$
       # We denote, for each $B_k \in \mathsf{B}$, $\{\omega_i\}_{\mathsf{del},k}$ the set of keywords removed from bucket $B_k$
       # Let $\mathsf{W} = \{(\{\omega_i\}_{\mathsf{add},k}, \{\omega_i\}_{\mathsf{del},k})\}_k$
     For $B_k \in \mathsf{B}$ do
         Compute $\mathsf{HW}_k = H(\mathsf{AW}_k || k)$ and $\mathsf{path}_{k,\mathsf{TW}} = \mathsf{GenerateMTProof}(\mathsf{TW}, \mathsf{HW}_k)$;
         Compute $\mathsf{AW}'_k$ using FFT interpolation and $\mathsf{PK}_{\mathcal{F}}$;
         Compute $\mathsf{HW}'_k = H(\mathsf{AW}'_k || k)$;
     End
       Update $\mathsf{TW}$ with $\{\mathsf{HW}'_k\}_{B_k \in \mathsf{B}}$ to obtain $\mathsf{TW}'$ with root $\sigma'_W$;
     For $B_k \in \mathsf{B}$ do
         Compute $\mathsf{path}'_{,\mathsf{TW}'} = \mathsf{GenerateMTProof}(\mathsf{TW}', \mathsf{HW}'_k)$;
     End
       Compute $\mathsf{path}_{i-1,\mathsf{TF}} = \mathsf{GenerateMTProof}(\mathsf{TF}, \mathsf{HF}_{i-1})$;
       Insert leaf associated with $\omega_i$ in $\mathsf{TF}$ to obtain $\mathsf{TF}'$ with root $\sigma'_F$;
       Assign $\Pi_{\mathsf{upd}} = (\mathsf{W}, \{\mathsf{AW}_k, \mathsf{AW}'_k, \mathsf{path}_{k,\mathsf{TW}}, \mathsf{path}'_{k,\mathsf{TW}'}\}_{B_k \in \mathsf{B}}, \sigma'_W, \mathsf{AF}_{i-1}, \mathsf{AF}_i, \mathsf{path}_{i-1,\mathsf{TF}}, \sigma'_F)$;
       Assign $\mathsf{LK}'_{\mathcal{F}} = (\mathcal{I}', \mathsf{TW}', \mathsf{TF}', \mathcal{F}', \mathbb{W}', \{\mathcal{F}'_{\omega_i}\}_{1 \le i \le N+1})$;
       Return $(\Pi_{\mathsf{upd}}, \mathsf{LK}'_{\mathcal{F}})$;
   End
```

Figure 6.12: Algorithm VerifyUpdate

```
▼ Algorithm: $\{\mathsf{Accept}, \mathsf{Reject}\} \leftarrow \mathsf{VerifyUpdate}(\mathsf{VK}_Q, \Pi_{\mathsf{upd}})$
1. If $\mathcal{W}_j = \emptyset$ Then return Accept;
2. If $\mathbb{W}' = \mathbb{W}$ then
     Parse $\Pi_{\mathsf{upd}} = (\{\mathsf{AF}_i, \mathsf{AF}'_i, \mathsf{path}_i, \mathsf{path}'_i\}_{\omega_i \in \mathcal{W}_j}, \sigma'_F)$;
     For $\omega_i \in \mathcal{W}_j$ do
         Compute $\mathsf{HF}_i = H(\mathsf{AF}_i || \omega_i)$;
         If $\mathsf{VerifyMTProof}(\mathsf{HF}_i, \mathsf{path}_i, \sigma_F) = \mathsf{Reject}$ Then return Reject;
         Compute $\mathsf{AF}'_i = \mathsf{AF}_i^{\frac{1}{\alpha - \mathsf{fid}_j}}$ if $\omega_i \in \mathcal{W}_{j,\mathsf{del}}$ or $\mathsf{AF}'_i = \mathsf{AF}_i^{\alpha - \mathsf{fid}_j}$ if $\omega_i \in \mathcal{W}_{j,\mathsf{add}}$ ;
         Compute $\mathsf{HF}'_i = H(\mathsf{AF}'_i || \omega_i)$;
         If $\mathsf{VerifyMTProof}(\mathsf{HF}'_i, \mathsf{path}'_i, \sigma'_F) = \mathsf{Reject}$ Then return Reject;
     End
     Return Accept;
   End
3. If $\mathbb{W}' = \mathbb{W} - \mathcal{W}_j$ then
     Parse $\Pi_{\mathsf{upd}} = (\mathsf{AW}_i, \mathsf{AW}'_i, \mathsf{path}_{i,\mathsf{TW}}, \sigma'_W, \mathsf{AF}_i, \mathsf{path}_{i,\mathsf{TF}}, \sigma'_F)$;
     Compute $\mathsf{HW}_i = H(\mathsf{AW}_i || i)$;
     If $\mathsf{VerifyMTProof}(\mathsf{HW}_i, \mathsf{path}_{i,\mathsf{TW}}, \sigma_W) = \mathsf{Reject}$ Then return Reject;
     Compute $\mathsf{HW}'_i = H(\mathsf{AW}'_i || i)$;
     If $\mathsf{VerifyMTProof}(\mathsf{HW}'_i, \mathsf{path}_{i,\mathsf{TW}}, \sigma'_W) = \mathsf{Reject}$ Then return Reject;
     Compute $\mathsf{HF}_i = H(\mathsf{AF}_i || \omega_i)$;
     If $\mathsf{VerifyMTProof}(\emptyset, \mathsf{path}_{i,\mathsf{TF}}, \sigma'_F) = \mathsf{Reject}$ Then return Reject;
     Return Accept;
   End
4. If $\mathbb{W}' = \mathbb{W} \cup \mathcal{W}_j$ then
     Parse $\Pi_{\mathsf{upd}} = (\mathsf{W}, \{\mathsf{AW}_k, \mathsf{AW}'_k, \mathsf{path}_{k,\mathsf{TW}}, \mathsf{path}'_{k,\mathsf{TW}'}\}_{B_k \in \mathsf{B}}, \sigma'_W, \mathsf{AF}_{i-1}, \mathsf{AF}_i, \mathsf{path}_{i-1,\mathsf{TF}}, \sigma'_F)$;
     For $B_k \in \mathsf{B}$ do
         Compute $\mathsf{HW}_k = H(\mathsf{AW}_k || k)$;
         If $\mathsf{VerifyMTProof}(\mathsf{HW}_k, \mathsf{path}_{k,\mathsf{TW}}, \sigma_W) = \mathsf{Reject}$ Then return Reject;
         Compute $P_k(\alpha) = \frac{\prod_{\omega_i \in \{\omega_i\}_{\mathsf{add},k}} (\alpha - \omega_i)}{\prod_{\omega_i \in \{\omega_i\}_{\mathsf{del},k}} (\alpha - \omega_i)}$;
         Compute $\mathsf{AW}'_k = \mathsf{AW}_k^{P_k(\alpha)}$;
         Compute $\mathsf{HW}'_k = H(\mathsf{AW}'_k || k)$;
         If $\mathsf{VerifyMTProof}(\mathsf{HW}'_k, \mathsf{path}'_{k,\mathsf{TW}'}, \sigma'_W) = \mathsf{Reject}$ Then return Reject;
     End
     Compute $\mathsf{HF}_i = H(\mathsf{AF}_i || \omega_i)$;
     Compute $\mathsf{HF}_{i-1} = H(\mathsf{AF}_{i-1} || \omega_{i-1})$;
     If $\mathsf{VerifyMTProof}(\mathsf{HF}_i, \mathsf{path}_{i,\mathsf{TF}} \cup \mathsf{HF}_{i-1}, \sigma'_F) = \mathsf{Reject}$ Then return Reject;
     Return Accept;
   End
```

three possible scenarios:

(a) **F'_j and F'_j have exactly the same keywords**, which means that $\mathcal{W}_j = \emptyset$. Then Update only consists in replacing F_j by F'_j in the database.

(b) **Some of the keywords in $\mathbb{W}$ that were in F_j are not in F'_j anymore**. Thus, we consider the set $\mathcal{W}_j = \{\omega_i \mid \omega_i \in F_j \wedge \omega_i \notin F'_j\}$. Note that for each $\omega_i \in \mathcal{W}_j$, if we denote $\mathcal{F}_{\omega_i}$ the set of files that contain ω_i and P_i its characteristic polynomial (before the update), then by denoting P'_i the updated polynomial, we have $P'_i(\alpha) = \frac{P_i(\alpha)}{\alpha - \mathsf{fid}_j}$, since fid_j is not part of $\mathcal{F}_{\omega_i}$ anymore. UpdateQuery returns $\mathcal{U}_Q = (F'_j, \mathcal{W}_j, \mathsf{modify})$ and $\mathsf{VK}_Q = \mathsf{PK}_{\mathcal{F}}$. In turn, the server executes algorithm Update which first replaces F_j by F'_j. Besides, for each $\omega_i \in \mathcal{W}_j$, algorithm Update computes the authentication path path_i for the old accumulator AF_i using algorithm GenerateMTProof (see Figure 6.6) and updates the new accumulator AF'_i (see step 4 in Figure 6.7). As server $\mathcal{S}$ does not have the knowledge of secret α, it computes AF'_i using $\mathsf{PK}_{\mathcal{F}}$ and the FFT interpolation technique. Afterwards, for each $\omega_i \in \mathcal{W}_j$, Update computes $\mathsf{HF}'_i = H(\mathsf{AF}'_i || \omega_i)$ and updates

Merkle tree TF', by updating the leaf corresponding to HF'_i and the nodes along the path to the new root σ'_F. It also generate the authentication path path'_i for the updated leaves as: $\mathsf{path}'_i = \mathsf{GenerateMTProof}(\mathsf{HF}'_i, \mathsf{TF}')$, for each $\omega_i \in \mathcal{W}_j$. Finally, algorithm Update returns $\Pi_{\mathsf{upd}} = (\{\mathsf{AF}_i, \mathsf{AF}'_i, \mathsf{path}_i, \mathsf{path}'_i\}_{\omega_i \in \mathcal{W}_j}, \sigma'_F)$ and $\mathsf{LK}'_{\mathcal{F}} = (\mathcal{I}, \mathsf{TW}, \mathsf{TF}', \mathcal{F}', \mathbb{W}, \{\mathcal{F}'_{\omega_i}\}_{1 \leq i \leq N})$. Upon reception of Π_{upd}, data owner $\mathcal{O}$ runs $\mathsf{VerifyUpdate}(\mathsf{VK}_Q, \Pi_{\mathsf{upd}})$. For each $\omega_i \in \mathcal{W}_j$, this algorithm first verifies, using $\mathsf{VerifyMTProof}(H(\mathsf{AF}_i || \omega_i), \mathsf{path}_i, \sigma_F)$, that AF_i is correct. Then, for each $\omega_i \in \mathcal{W}_j$, data owner $\mathcal{O}$ computes $\mathsf{AF}'_i = \mathsf{AF}_i^{\frac{1}{\alpha - \mathsf{fid}_j}}$ and verifies that σ'_F corresponds to the expected value by invoking $\mathsf{VerifyMTProof}(H(\mathsf{AF}'_i || \omega_i), \mathsf{path}'_i, \sigma'_F)$.

(c) **Some of the keywords in $\mathbb{W}$ that were not in F_j are now in F'_j.** In this case, we consider the set $\mathcal{W}_j = \{\omega_i \mid \omega_i \in F'_j \wedge \omega_i \notin F_j\}$. Then, for each $\omega_i \in \mathcal{W}_j$, $P'_i(\alpha) = P_i(\alpha)(\alpha - \mathsf{fid}_j)$, since $\mathsf{fid}_j \in \mathcal{F}_{\omega_i}$. As in the previous case, $\mathsf{UpdateQuery}$ returns $\mathcal{U}_Q = (F'_j, \mathcal{W}_j, \mathsf{modify})$ and $\mathsf{VK}_Q = \mathsf{PK}_{\mathcal{F}}$. Server $\mathcal{S}$ executes algorithm Update which first replaces F_j by F'_j. Similarly as before, for each $\omega_i \in \mathcal{W}_j$, algorithm Update computes path_i for the old accumulator AF_i, and updates the new accumulator AF'_i. Afterwards, for each $\omega_i \in \mathcal{W}_j$, Update accordingly modifies Merkle tree TF', by updating leaf HF'_i and the nodes along the path to the new root σ'_F. Thereafter, Update determines the authentication path path'_i for the updated leaves, for $\omega_i \in \mathcal{W}_j$. Finally, algorithm Update returns $\Pi_{\mathsf{upd}} = (\{\mathsf{AF}_i, \mathsf{AF}'_i, \mathsf{path}_i, \mathsf{path}'_i\}_{\omega_i \in \mathcal{W}_j}, \sigma'_F)$ and $\mathsf{LK}'_{\mathcal{F}} = (\mathcal{I}, \mathsf{TW}, \mathsf{TF}', \mathcal{F}', \mathbb{W}, \{\mathcal{F}'_{\omega_i}\}_{1 \leq i \leq N})$. Upon reception of Π_{upd} and $\mathsf{LK}'_{\mathcal{F}}$, for each $\omega_i \in \mathcal{W}_j$, data owner $\mathcal{O}$ runs $\mathsf{VerifyUpdate}$ which first verifies using $\mathsf{VerifyMTProof}(H(\mathsf{AF}_i || \omega_i), \mathsf{path}_i, \sigma_F)$ that AF_i is correct. Then, for each $\omega_i \in \mathcal{W}_j$, data owner $\mathcal{O}$ computes $\mathsf{AF}'_i = \mathsf{AF}_i^{\alpha - \mathsf{fid}_j}$ and verifies that it corresponds to the expected value by invoking $\mathsf{VerifyMTProof}(H(\mathsf{AF}'_i || \omega_i), \mathsf{path}_i, \sigma'_F)$.

2. $\mathsf{op} = \mathsf{delete}$: The data owner executes $\mathsf{UpdateQuery}(F_j, \mathcal{W}_j = \{\omega_i\}_{\omega_i \in F_j}, \mathsf{delete})$, which outputs $\mathcal{U}_Q = (F_j, \mathcal{W}_j, \mathsf{delete})$ and $\mathsf{VK}_Q = \mathsf{PK}_{\mathcal{F}}$. Server $\mathcal{S}$ executes algorithm Update which first deletes F_j. The remainder of this update is similar to the case when $\mathsf{op} = \mathsf{modify}$. In particular, for each $\omega_i \in \mathcal{W}_j$, data owner $\mathcal{O}$ runs $\mathsf{VerifyUpdate}$ which computes $\mathsf{AF}'_i = \mathsf{AF}_i^{\frac{1}{\alpha - \mathsf{fid}_j}}$ and verifies that it corresponds to the expected value.

3. $\mathsf{op} = \mathsf{insert}$: The data owner executes $\mathsf{UpdateQuery}(F_j, \mathcal{W}_j = \{\omega_i\}_{\omega_i \in F_j}, \mathsf{insert})$, which returns $\mathcal{U}_Q = (F_j, \mathcal{W}_j, \mathsf{insert})$ and $\mathsf{VK}_Q = \mathsf{PK}_{\mathcal{F}}$. Server $\mathcal{S}$ executes algorithm Update which first insert F_j in the outsourced database. The remainder of this update is similar to the case when $\mathsf{op} = \mathsf{delete}$, with the difference that, for each $\omega_i \in \mathcal{W}_j$, data owner $\mathcal{O}$ computes $\mathsf{AF}'_i = \mathsf{AF}_i^{\alpha - \mathsf{fid}_j}$.

2. Keyword deletion, $\mathbb{W}' = \mathbb{W} - \mathcal{W}_j$: This event may probably occur if $\mathsf{op} = \mathsf{modify}$ or $\mathsf{op} = \mathsf{delete}$. Data owner $\mathcal{O}$ runs $\mathsf{UpdateQuery}(F_j, \mathcal{W}_j = \omega_i, \mathsf{op})$, which returns $\mathcal{U}_Q = (F_j, \mathcal{W}_j, \mathsf{op})$ and $\mathsf{VK}_Q = \mathsf{PK}_{\mathcal{F}}$, where ω_i is the keyword that will be deleted during the update operation indicated by op (for simplicity we assume that only a single keyword is deleted from the dictionary). For both types of operations (delete or modify), server $\mathcal{S}$ executes Update which is executed in two steps. First, Update deletes or modifies F_j and removes keyword ω_i from Cuckoo hash index $\mathcal{I}$, which yields the updated index $\mathcal{I}'$. Namely, if we denote B_i the bucket that stores ω_i, such that B_i is at position $\mathcal{H}_1(\omega_i)$ or $\mathcal{H}_2(\omega_i)$ (see Figure 6.2) then $B_i = B_i - \{\omega_i\}$. Secondly, Update computes the corresponding accumulator $\mathsf{AW}'_i = g^{P'_{B_i}(\alpha)}$. Note that $P'_{B_i}(\alpha) = \frac{P_{B_i}(\alpha)}{\alpha - H(\omega_i)}$. Afterwards, using algorithm $\mathsf{GenerateMTProof}$ (see Figure 6.6), Update calculates the authentication path, $\mathsf{path}_{i,\mathsf{TW}}$, of AW_i in Merkle tree TW (which is the same of AW'_i) and then updates the tree to obtain TW' with root σ'_W. The second step of the keyword deletion removes the leaf corresponding to the deleted keyword ω_i from tree TF, as depicted in Figure 6.13.

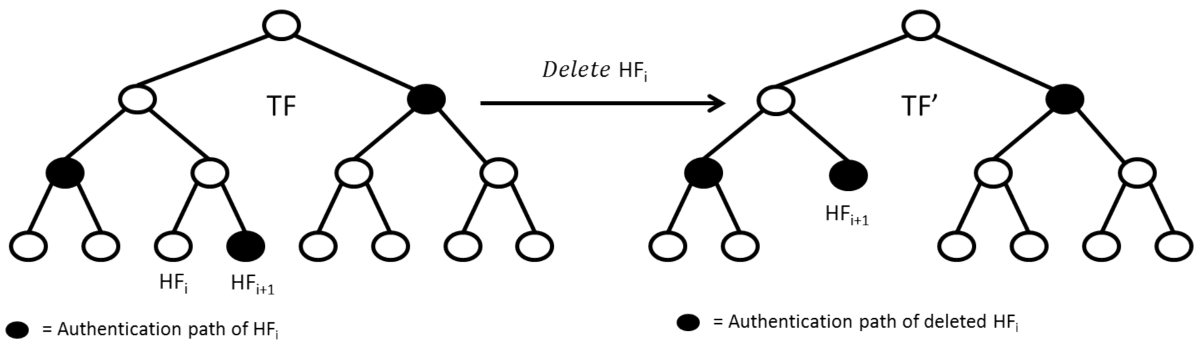

TF
Delete HF$_i$
TF′
HF$_i$
HF$_{i+1}$
HF$_{i+1}$
● = Authentication path of HF$_i$
● = Authentication path of deleted HF$_i$

Figure 6.13: Deletion in a Merkle tree

This operation produces a new tree TF' with root σ'_F. Besides, Update computes the authentication path, $\mathsf{path}_{i,\mathsf{TF}}$, of the old accumulator AF_i, using algorithm $\mathsf{GenerateMTProof}$. Finally, algorithm Update outputs $\Pi_{\mathsf{upd}} = (\mathsf{AW}_i, \mathsf{AW}'_i, \mathsf{path}_{i,\mathsf{TW}}, \sigma'_W, \mathsf{AF}_i, \mathsf{path}_{i,\mathsf{TF}}, \sigma'_F)$ and $\mathsf{LK}'_{\mathcal{F}} = (\mathcal{I}', \mathsf{TW}', \mathsf{TF}', \mathcal{F}', \mathbb{W}', \{\mathcal{F}'_{\omega_i}\}_{1\leq i\leq N})$. Consequently, data owner $\mathcal{O}$ checks that the update operation is correctly performed by invoking $\mathsf{VerifyUpdate}$. This algorithm first runs $\mathsf{VerifyMTProof}(H(\mathsf{AW}_i||\ i), \mathsf{path}_{i,\mathsf{TW}}, \sigma_W)$ to verify that AW_i returned by the server is correct. Secondly, $\mathsf{VerifyUpdate}$ computes $\mathsf{AW}'_i = \mathsf{AW}_i^{\frac{1}{\alpha - H(\omega_i)}}$ and checks that the new root σ'_W is correctly computed, using $\mathsf{VerifyMTProof}(H(\mathsf{AW}'_i||\ i), \mathsf{path}_{i,\mathsf{TW}}, \sigma'_W)$. On the other hand, $\mathsf{VerifyUpdate}$ checks that the leaf corresponding to deleted keyword ω_i is correctly deleted from Merkle tree TF' using $\mathsf{VerifyMTProof}(\emptyset, \mathsf{path}_{i,\mathsf{TF}}, \sigma'_F)$.

3. Keyword insertion, $\mathbb{W}' = \mathbb{W} \cup \mathcal{W}_j$: This event occurs if $\mathsf{op} = \mathsf{modify}$ or $\mathsf{op} = \mathsf{insert}$. Keyword insertion is the most expensive operation, as the insertion of new keyword can affect multiple buckets in index $\mathcal{I}$. Data owner $\mathcal{O}$ runs $\mathsf{UpdateQuery}(F_j, \mathcal{W}_j = \omega_i, \mathsf{op})$, which returns $\mathcal{U}_Q = (F_j, \mathcal{W}_j, \mathsf{op})$ and $\mathsf{VK}_Q = \mathsf{PK}_{\mathcal{F}}$, where ω_i is the keyword that will be inserted during the update operation indicated by op (for simplicity we consider that a single keyword is inserted). Server $\mathcal{S}$ executes Update which operates in two steps as in the previous case of keyword deletion. First, Update inserts or modifies F_j and then inserts keyword ω_i in the Cuckoo hash index $\mathcal{I}$, using $\mathsf{CuckooInsert}$. As we saw in Section 6.4.1, this insertion procedure may impact several buckets in index $\mathcal{I}$, since existing keywords might be kicked off from their current buckets. At the end of the keyword insertion procedure, we denote $\mathcal{I}'$ the updated index[101] and $\mathsf{B} = \{B_k\}_k$ the set of buckets affected by the insertion ω_i. We also denote $\mathsf{W} = \{(\{\omega_i\}_{\mathsf{add},k}, \{\omega_i\}_{\mathsf{del},k})\}_k$ the set of keywords affected by the insertion of ω_i, where $\{\omega_i\}_{\mathsf{add},k}$ is the set of keywords that are added to bucket $B_k \in \mathsf{B}$ and $\{\omega_i\}_{\mathsf{del},k}$ is the set of keywords that are removed from bucket $B_k \in \mathsf{B}$. Therefore, for all buckets $B_k \in \mathsf{B}$, Update (i) computes the authentication paths in tree TW of accumulator AW_k as $\mathsf{path}_{k,\mathsf{TW}} = \mathsf{GenerateMTProof}(\mathsf{TW}, H(\mathsf{AW}_k||\ k))$; and (ii) updates the new accumulators AW'_k using the FFT interpolation technique and $\mathsf{PK}_{\mathcal{F}}$. Then, Update accordingly modifies Merkle tree TW to obtain TW', with root σ'_W and for each bucket $B_k \in \mathsf{B}$, it computes the authentication paths in tree TW' of accumulator AW'_k as $\mathsf{path}'_{k,\mathsf{TW}'} = \mathsf{GenerateMTProof}(\mathsf{TW}', H(\mathsf{AW}'_k||\ k))$, to prove that the tree is correctly updated. In the second step of the keyword insertion, Update computes the authentication path $\mathsf{path}_{i-1,\mathsf{TF}}$ of the accumulator AF_{i-1} in the old tree TF, using algorithm $\mathsf{GenerateMTProof}$. Indeed, as the keywords are sorted in the lexicographic order, AF_i will be inserted after AF_{i-1} in tree TF, as shown in Figure 6.14. Then Update computes AF'_i and adds a new leaf corresponding to the inserted keyword ω_i in tree TF, which produces a new tree TF' with root σ'_F. The authentication path for AF_i is simply the authentication path of AF_{i-1} together with the leaf corresponding to AF_{i-1}, as shown in Figure 6.14. Finally, algorithm Update outputs $\Pi_{\mathsf{upd}} = (\mathsf{W}, \{\mathsf{AW}_k, \mathsf{AW}'_k, \mathsf{path}_{k,\mathsf{TW}}, \mathsf{path}'_{k,\mathsf{TW}'}\}_{B_k\in\mathsf{B}}, \sigma'_W, \mathsf{AF}_{i-1}, \mathsf{AF}_i, \mathsf{path}_{i-1,\mathsf{TF}}, \sigma'_F)$ and $\mathsf{LK}'_{\mathcal{F}} =$

[101]In principle, keyword insertion is possible only if index $\mathcal{I}$ did not reach its maximum capacity.

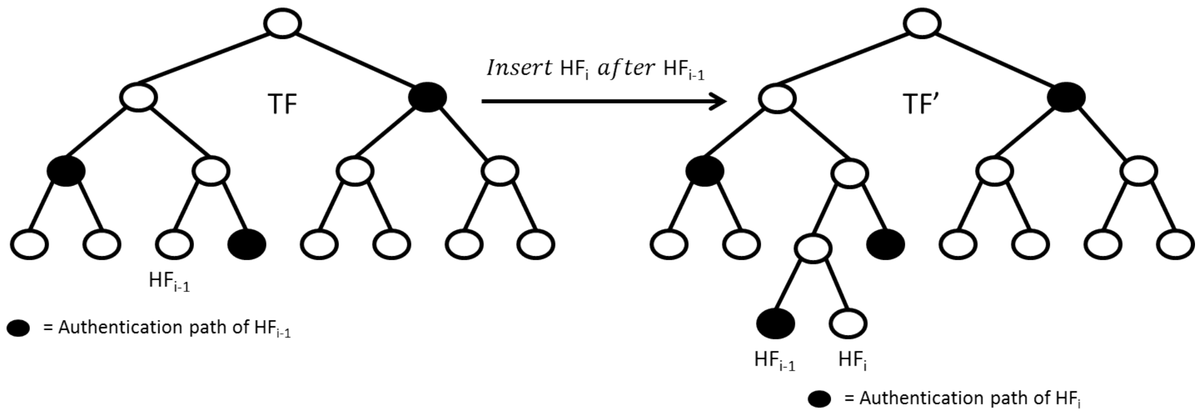

Insert HF$_i$ after HF$_{i-1}$
TF
TF′
HF$_{i-1}$
● = Authentication path of HF$_{i-1}$
HF$_{i-1}$
HF$_i$
● = Authentication path of HF$_i$

Figure 6.14: Insertion in a Merkle tree

$(\mathcal{I}', \mathsf{TW}', \mathsf{TF}', \mathcal{F}', \mathbb{W}', \{\mathcal{F}'_{\omega_i}\}_{1\leq i\leq N})$. Consequently, data owner $\mathcal{O}$ checks that the insert operation is correctly performed by invoking VerifyUpdate. This algorithm first checks that for each $B_k \in \mathsf{B}$, the old accumulators AW_k are correct, using $\mathsf{VerifyMTProof}(H(\mathsf{AW}_k||\ k), \mathsf{path}_{k,\mathsf{TW}}, \sigma_W)$. Then, VerifyUpdate computes the new accumulators of the buckets impacted by the keyword insertion. Namely, for each $B_k \in \mathsf{B}$, if we denote

$$P_k(\alpha) = \frac{\prod\limits_{\omega_i\in\{\omega_i\}_{\mathsf{add},k}} (\alpha - \omega_i)}{\prod\limits_{\omega_i\in\{\omega_i\}_{\mathsf{del},k}} (\alpha - \omega_i)},$$

then it calculates $\mathsf{AW}'_k = \mathsf{AW}_k^{P_k(\alpha)}$. VerifyUpdate thus checks that these accumulators correspond to the ones sent by the server and verifies that tree TW' was correctly updated thanks to $\mathsf{VerifyMTProof}(H(\mathsf{AW}'_k||\ k), \mathsf{path}'_{k,\mathsf{TW}'}, \sigma'_W)$, for all buckets $B_k \in \mathsf{B}$. On the other hand, VerifyUpdate checks that the leaf corresponding to keyword ω_i is correctly added to Merkle tree TF' using $\mathsf{VerifyMTProof}(H(\mathsf{AF}_i||\omega_i), \mathsf{path}_{i-1,\mathsf{TF}} \cup H(\mathsf{AF}_{i-1}||\omega_{i-1}), \sigma'_F)$.

6.6 Security Analysis

Our protocol satisfies the two security properties of correctness and soundness. In this section, we define and prove the two security theorems related with these properties.

6.6.1 Correctness

Theorem 7 (Correctness). *Our scheme is a correct verifiable conjunctive keyword search solution.*

Proof of Theorem 7. Suppose that a querier $\mathcal{Q}$ sends to server $\mathcal{S}$ the query $\mathcal{E}_Q = \mathcal{W} = \{\omega_1, ..., \omega_k\}$. $\mathcal{S}$ correctly runs algorithm Search and returns the search response $\mathcal{E}_R$. According to Figure 6.8, the content of $\mathcal{E}_R$ varies depending on whether:

All words in $\mathcal{W}$ are found in $\mathcal{F}$
Then $\mathcal{E}_R = (\mathcal{F}_\mathcal{W}, \Pi_\mathcal{W}, \{\mathsf{AF}_i\}_{1\leq i\leq k}, \{\mathsf{path}_i\}_{1\leq i\leq k})$ where:

- $\mathcal{F}_\mathcal{W} = \mathcal{F}_{\omega_1} \cap ... \cap \mathcal{F}_{\omega_k}$ such that $\mathcal{F}_{\omega_i}$ is the subset of files that contain keyword ω_i;

- $\Pi_{\mathcal{W}} = \{(\Delta_1, \Gamma_1), ..., (\Delta_k, \Gamma_k)\}$ is the proof of intersection;
- for all $1 \leq i \leq k$, $\mathsf{AF}_i = \mathcal{Acc}(\mathcal{F}_{\omega_i})$; if we denote P_i the characteristic polynomial of subset $\mathcal{F}_{\omega_i}$, then we can write $\mathsf{AF}_i = g^{P_i(\alpha)}$;
- for all $1 \leq i \leq k$, path_i is the authentication path of $H(\mathsf{AF}_i||\omega_i)$ in TF.

Firstly, if we assume that the Merkle tree authentication is correct, then verifier $\mathcal{V}$ accepts the accumulators AF_i computed by server $\mathcal{S}$. Secondly, since $\mathcal{S}$ computes the proof $\Pi_{\mathcal{W}}$ using algorithm ProveIntersection (cf. Figure 6.5), for all $1 \leq i \leq k$, we have the following:

- $\Delta_i = g^{U_i(\alpha)}$, where $U_i = \frac{P_i}{P}$ and $P = \gcd(P_1, P_2, ..., P_k)$ is the characteristic polynomial of $\mathcal{F}_{\mathcal{W}}$;
- $\Gamma_i = g^{V_i(\alpha)}$, such that $\sum_i U_i V_i = 1$.

It follows that for all $1 \leq i \leq k$:

$$\begin{aligned} e(\mathcal{Acc}(\mathcal{F}_{\mathcal{W}}), \Delta_i) &= e(g^{P(\alpha)}, g^{U_i(\alpha)}) = e(g,g)^{P(\alpha)\cdot U_i(\alpha)} \\ &= e(g,g)^{P_i(\alpha)} = e(\mathsf{AF}_i, g) \end{aligned}$$

This means that the first equality in algorithm VerifyIntersection (cf. Figure 6.5) holds. Furthermore, the second equality is also verified, indeed:

$$\begin{aligned} \prod_{\omega_i \in \mathcal{W}} e(\Delta_i, \Gamma_i) &= \prod_{\omega_i \in \mathcal{W}} e(g^{U_i(\alpha)}, g^{V_i(\alpha)}) = \prod_{\omega_i \in \mathcal{W}} e(g,g)^{U_i(\alpha)\cdot V_i(\alpha)} \\ &= e(g,g)^{\sum_{\omega_i \in \mathcal{W}} U_i(\alpha)\cdot V_i(\alpha)} = e(g,g) \end{aligned}$$

These computations thus prove the correctness of our solution in the case where the targeted keywords are all found.

There exists $\omega_i \in \mathcal{W}$ not found in $\mathcal{F}$
In this case, $\mathcal{E}_R = (\emptyset, \omega_i, \mathsf{AW}_{i_1}, \mathsf{AW}_{i_2}, \Pi_1, \Pi_2, \mathsf{path}_1, \mathsf{path}_2)$ such that:

- $\mathsf{AW}_{i_1} = \mathcal{Acc}(B_{i_1})$ and $\mathsf{AW}_{i_2} = \mathcal{Acc}(B_{i_2})$ are the accumulators of buckets B_{i_1} and B_{i_2} respectively, where i_1 and i_2 are the positions assigned to keyword ω_i in index $\mathcal{I}$;
- Π_1 and Π_2 are the proofs that ω_i is not a member of bucket B_{i_1} nor of bucket B_{i_2} respectively;
- path_1 and path_2 are the authentication paths of these two buckets in tree TW.

If we consider the Merkle tree to be correct, then verifier $\mathcal{V}$ will accept $\mathcal{Acc}(B_{i_1})$ and $\mathcal{Acc}(B_{i_2})$. Moreover, if we denote $P_{B_{i_1}}$ the characteristic polynomial of bucket B_{i_1}, then by definition $P_{B_{i_1}}(X) = \prod_{h_j \in B_{i_1}} (X - h_j)$ and $\mathcal{Acc}(B_{i_1}) = g^{P_{B_{i_1}}(\alpha)}$.

Recall now that the proof of non-membership Π_1 of keyword ω_i to bucket B_{i_1} is computed as: $\{P_{B_{i_1}}(h_i), \Omega_{B_{i_1}, h_i}\}$, such that $h_i = H(\omega_i)$, $\Omega_{B_{i_1}, h_i} = g^{Q_{B_{i_1}, h_i}(\alpha)}$ and $Q_{B_{i_1}, h_i}(X) = \frac{P_{B_{i_1}}(X) - P_{B_{i_1}}(h_i)}{X - h_i}$.

It follows that:

$$\begin{aligned} e(\Omega_{B_{i_1}}, g^{\alpha} \cdot g^{-h_i}) e(g^{P_{B_{i_1}}(h_i)}, g) &= e(g,g)^{Q_{B_{i_1}, h_i}(\alpha)\cdot(\alpha - h_i)} e(g,g)^{P_{B_{i_1}}(h_i)} \\ &= e(g,g)^{Q_{B_{i_1}, h_i}(\alpha)\cdot(\alpha - h_i) + P_{B_{i_1}}(h_i)} \\ &= e(g,g)^{P_{B_{i_1}}(\alpha)} \\ &= e(\mathcal{Acc}(B_{i_1}), g). \end{aligned}$$

This means that the first equality of algorithm $\mathsf{GenerateWitness}$ (cf. Figure 6.4) holds. Finally, since $\omega_i \notin B_{i_1}$, $P_{B_{i_1}}(h_i) \neq 0$. This implies that verifier $\mathcal{V}$ will accept the proof of non-membership for bucket B_{i_1} and conclude that ω_i is not in $\mathcal{F}$.

Similar computations can be performed for B_{i_2}, which proves the correctness of our solution in the case where a keyword $\omega_i \notin \mathcal{F}$.

Correctness of updates. The update procedures we presented in Section 6.5.4 mainly involves the Merkle tree operations. The proof Π_{upd} of correct updates includes the authentication paths of the updated accumulators. Therefore, since we consider H being a collision-resistant hash function, a honest verifier will always accept the proof of update generated by a honest server.

6.6.2 Soundness

Theorem 8 (Soundness). *Our solution for verifiable conjunctive keyword search is sound under the D-SDH and D-SBDH assumptions, provided that the hash function H used to build the Merkle trees is collision-resistant.*

Proof of Theorem 8. We observe that an adversary can break the soundness of our scheme through two types of forgery:

Type 1 forgery: On input of $\mathcal{W} = \{\omega_1, ..., \omega_k\}$ and search key $\mathsf{LF}_{\mathcal{F}}$, adversary $\mathcal{A}_1$ returns a search result that consists of a proof of non-membership of some keyword $\omega_i \in \mathcal{W}$ (meaning that ω_i is not in the set of files $\mathcal{F}$), although ω_i is in $\mathcal{F}$;

Type 2 forgery: On input of $\mathcal{W} = \{\omega_1, ..., \omega_k\}$ and search key $\mathsf{LF}_{\mathcal{F}}$, adversary $\mathcal{A}_2$ returns an incorrect $\widehat{\mathcal{F}_{\mathcal{W}}}$ and the corresponding proof. This means that adversary $\mathcal{A}_2$ claims that all keywords in $\mathcal{W}$ have been found in $\mathcal{F}$ and that $\widehat{\mathcal{F}_{\mathcal{W}}}$ is the subset of files that contain them, although $\widehat{\mathcal{F}_{\mathcal{W}}} \neq \mathsf{CKS}(\mathcal{F}, \mathcal{W})$. (We use the caret notation ($\widehat{\cdot}$) to distinguish the elements of the search response returned by adversary $\mathcal{A}_2$ from the ones that would be returned by a honest server.)

In the following, we demonstrate that if $\mathcal{A}_1$ and $\mathcal{A}_2$ runs Type 1 and Type 2 forgery respectively, then there exists another adversary $\mathcal{B}_1$ that breaks D-SDH and an adversary $\mathcal{B}_2$ that breaks D-SBDH respectively.

Lemma 3 (Type 1 forgery). *If $\mathcal{A}_1$ breaks the soundness of our protocol, then there exists adversary $\mathcal{B}_1$ that breaks the D-SDH assumption in $\mathbb{G}$.*

Let $\mathcal{O}_{\mathsf{D-SDH}}$ be an oracle which, when invoked, returns the D-SDH tuple $T(\alpha) = (g, g^{\alpha}, g^{\alpha^2}, ..., g^{\alpha^D}) \in \mathbb{G}^{D+1}$, for some randomly selected $\alpha \in \mathbb{F}_p^*$. Here we define an adversary $\mathcal{B}_1$ that breaks the D-SDH assumption:

1. $\mathcal{B}_1$ first calls $\mathcal{O}_{\mathsf{D-SDH}}$ which selects a random $\alpha \in \mathbb{F}_p^*$ and returns $T(\alpha)$.
2. $\mathcal{B}_1$ simulates the soundness game for adversary $\mathcal{A}_1$ (cf. Algorithm 5). Specifically, when $\mathcal{A}_1$ invokes $\mathcal{O}_{\mathsf{Setup}}$ with the sets of files $\mathcal{F}_k$ (for $1 \leq k \leq t$), $\mathcal{B}_1$ simulates $\mathcal{O}_{\mathsf{Setup}}$ and generates $(\mathsf{param}, \mathsf{PK}_{\mathcal{F}_k}, \mathsf{LK}_{\mathcal{F}_k})$, as follows:

(a) $\mathcal{B}_1$ selects the parameters $g, \mathbb{G}, \mathbb{G}_T, e$ and H;

(b) $\mathcal{B}_1$ computes the tuple $T_k(\alpha) = (g, g^{\alpha_k}, g^{\alpha_k{}^2}, ..., g^{\alpha_k{}^D})$ where $\alpha_k = \alpha \cdot \delta_k + \beta_k$ for some random $\delta_k, \beta_k \in \mathbb{F}_p^*$. Note that this tuple can be easily computed by $\mathcal{B}_1$, without having access to α, thanks to tuple $T(\alpha)$ and the Binomial Theorem: $\forall\ i \leq D,\ g^{\alpha_k{}^i} = g^{(\alpha \cdot \delta_k + \beta_k)^i} = \prod_{j=0}^{i} (g^{\alpha^j})^{\binom{i}{j} (\delta_k)^j \cdot \beta_k^{i-j}}$;

(c) The rest of the simulation of $\mathcal{O}_{\mathsf{Setup}}$ is operated as in Figure 6.7.

Afterwards, adversary $\mathcal{A}_1$ selects a collection of keywords $\mathcal{W}_k$ to search for in a set of files $\mathcal{F}_k$ and invokes $\mathcal{O}_{\mathsf{QueryGen}}$. Adversary $\mathcal{B}_1$ computes the response of $\mathcal{O}_{\mathsf{QueryGen}}$ and returns encoded query $\mathcal{E}_{Q,k} = \mathcal{W}_k$ and the verification key $\mathsf{VK}_{Q,k} = (\mathsf{PK}_{\mathcal{F}_k}, \mathcal{W}_k)$. Later, $\mathcal{A}_1$ returns an encoded response $\mathcal{E}_{R,k}$ and runs Verify.

3. In the challenge phase of the soundness game (cf. Algorithm 6), $\mathcal{A}_1$ first selects a public key $\mathsf{PK}^*_{\mathcal{F}}$ from the keys he has received earlier, and a collection of keywords $\mathcal{W}^*$ to search for in a set of files $\mathcal{F}^*$ associated $\mathsf{PK}^*_{\mathcal{F}}$ then $\mathcal{A}_1$ runs $\mathcal{O}_{\mathsf{QueryGen}}(\mathcal{W}^*, \mathsf{PK}^*_{\mathcal{F}})$ which is simulated by $\mathcal{B}_1$ and outputs the encoded query $\mathcal{E}^*_Q = \mathcal{W}^*$ and the verification key $\mathsf{VK}^*_Q = (\mathsf{PK}_{\mathcal{F}^*}, \mathcal{W}^*)$.

4. Then, $\mathcal{A}_1$ returns $\mathcal{E}^*_R = (\emptyset, \omega^*, \widehat{\mathsf{AF}}^*_1, \widehat{\mathsf{AF}}^*_2, \widehat{\Pi}^*_1, \widehat{\Pi}^*_2, \widehat{\mathsf{path}}^*_1, \widehat{\mathsf{path}}^*_2)$, with:

- The empty set, being the result of the search, meaning that the keyword $\omega^* \in \mathcal{W}^*$ was not found in $\mathcal{F}^*$, although ω^* is indeed in $\mathcal{F}^*$;
- The accumulators $\widehat{\mathsf{AF}}^*_1, \widehat{\mathsf{AF}}^*_2$ of the buckets at the positions associated to ω^* in index $\mathcal{I}^*$ of files $\mathcal{F}^*$;
- $\widehat{\Pi}^*_1, \widehat{\Pi}^*_2$, the proofs of non-membership of ω^* with respect to buckets $B^*_{i_1}, B^*_{i_2}$, where i_1 and i_2 are the positions assigned to keyword ω^* in index $\mathcal{I}^*$;
- $\widehat{\mathsf{path}}^*_1, \widehat{\mathsf{path}}^*_2$, the authentication paths in Merkle tree TW for the accumulators of buckets $\widehat{\mathsf{AF}}^*_1, \widehat{\mathsf{AF}}^*_2$.

5. Since we assume H is a collision-resistant hash function, the Merkle tree authentication proves that $\widehat{\mathsf{AF}}^*_1$ and $\widehat{\mathsf{AF}}^*_2$ are actually associated with leaves at positions i_1 and i_2 in TW. More precisely, it proves that $\widehat{\mathsf{path}}^*_1$ and $\widehat{\mathsf{path}}^*_2$ authenticate the values $H(\widehat{\mathsf{AF}}^*_1 || i_1)$ and $H(\widehat{\mathsf{AF}}^*_2 || i_2)$ and that $\widehat{\mathsf{AF}}^*_1$ and $\widehat{\mathsf{AF}}^*_2$ correspond to $\mathcal{A}cc(B^*_{i_1})$ and $\mathcal{A}cc(B^*_{i_2})$ that were computed in the setup phase by $\mathcal{B}_1$. Namely, $\widehat{\mathsf{AF}}^*_1 = g^{P_{B^*_{i_1}}(\alpha^*)}$ and $\widehat{\mathsf{AF}}^*_2 = g^{P_{B^*_{i_2}}(\alpha^*)}$.

6. We now show how $\mathcal{B}_1$ breaks the D-SDH assumption. Let us consider that $h^* = H(\omega^*)$ is indeed stored in the first bucket $B^*_{i_1}$ (similar consideration can be applied to $B^*_{i_2}$). As returned by $\mathcal{A}_1$, the forged proof of non-membership for ω^* consists of $\widehat{\Pi}^*_1 = \{\widehat{P}_{B_{i_1}}(h^*), \widehat{\Omega}_{B^*_{i_1}, h^*}\}$ (cf. Figure 6.4). Notice that $\widehat{P}_{B^*_{i_1}}(h^*) \neq 0$, as adversary $\mathcal{A}_1$ claims that ω^* is not in $\mathcal{F}^*$. If Verify accepts the proof of non-membership, then according to Figure 6.4, the following equality holds:

$$e(\widehat{\Omega}_{B^*_1, h^*}, g^{\alpha^* - h^*}) e(g^{\widehat{P}_{B_1}(h^*)}, g) = e(\widehat{\mathsf{AF}}^*_1, g) = e(g^{\widehat{P}_{B^*_{i_1}}(\alpha^*)}, g)$$

$$e(\widehat{\Omega}_{B^*_1, h^*}, g^{\alpha^* - h^*}) = e(g^{P_{B^*_1}(\alpha^*) - \widehat{P}_{B^*_1}(h^*)}, g) \tag{6.1}$$

On the other hand, by construction we have:

$$e(\Omega_{B^*_{i_1}, h^*}, g^{\alpha^* - h^*}) = e(g^{P_{B^*_{i_1}}(\alpha^*)}, g), \tag{6.2}$$

where $\Omega_{B^*_{i_1},h^*} = g^{Q_{B^*_{i_1},h^*}(\alpha^*)}$ such that $Q_{B^*_{i_1},h^*}(X) = \frac{P_{B^*_{i_1}}(X)}{X-h^*}$.
By dividing equation 6.1 with equation 6.2, we obtain:

$$e\left(\frac{\widehat{\Omega}_{B^*_{i_1},h^*}}{\Omega_{B^*_{i_1},h^*}}, g^{\alpha^*-h^*}\right) = e\left(\left(\frac{\widehat{\Omega}_{B^*_{i_1},h^*}}{\Omega_{B^*_{i_1},h^*}}\right)^{\alpha^*-h^*}, g\right) = e(g^{-\widehat{P}_{B^*_{i_1}}(h^*)}, g).$$

Therefore,

$$\left(\frac{\widehat{\Omega}_{B^*_{i_1},h^*}}{\Omega_{B^*_{i_1},h^*}}\right)^{\alpha^*-h^*} = g^{-\widehat{P}_{B^*_{i_1}}(h^*)}$$

$$\left(\frac{\widehat{\Omega}_{B^*_{i_1},h^*}}{\Omega_{B^*_{i_1},h^*}}\right)^{\frac{1}{-\widehat{P}_{B^*_{i_1}}(h^*)}} = \left(\frac{\Omega_{B^*_{i_1},h^*}}{\widehat{\Omega}_{B^*_{i_1},h^*}}\right)^{\frac{1}{\widehat{P}_{B^*_{i_1}}(h^*)}} = g^{\frac{1}{\alpha^*-h^*}}.$$

We have $\alpha^* = \alpha\cdot\delta^* + \beta^*$, where (δ^*, β^*) are randomly selected from set $\{\delta_k, \beta_k\}_{1\leq k\leq t}$ generated earlier. Accordingly,

$$\left(\frac{\Omega_{B^*_{i_1},h^*}}{\widehat{\Omega}_{B^*_{i_1},h^*}}\right)^{\frac{1}{\widehat{P}_{B^*_{i_1}}(h^*)}} = g^{\frac{1}{\alpha\cdot\delta^*+\beta^*-h^*}} = g^{\frac{1}{\delta^*(\alpha+\frac{\beta^*-h^*}{\delta^*})}}$$

$$\left(\frac{\Omega_{B^*_{i_1},h^*}}{\widehat{\Omega}_{B^*_{i_1},h^*}}\right)^{\frac{\delta^*}{\widehat{P}_{B^*_{i_1}}(h^*)}} = g^{\frac{1}{\alpha+\frac{\beta^*-h^*}{\delta^*}}}$$

Since $\beta^* \neq h^*$ with an overwhelming probability ($\Pr(\beta^* = h^*) = \frac{1}{p}$), then adversary $\mathcal{B}_1$ breaks D-SDH by outputting the pair $\left(\frac{\beta^*-h^*}{\delta^*}, \left(\frac{\Omega_{B^*_{i_1},h^*}}{\widehat{\Omega}_{B^*_{i_1},h^*}}\right)^{\frac{\delta^*}{\widehat{P}_{B^*_{i_1}}(h^*)}}\right)$ with a non-negligible advantage $\varepsilon_B \geq \varepsilon_A \cdot (1-\frac{1}{p})$ where ε_A is the advantage of adversary $\mathcal{A}_1$ in breaking the soundness of our scheme.

Lemma 4 (Type 2 forgery). *If $\mathcal{A}_2$ breaks the soundness of our protocol, then there exists an adversary $\mathcal{B}_2$ that breaks the D-SBDH assumption in $\mathbb{G}$.*

Let $\mathcal{O}_{\mathsf{D-SBDH}}$ be an oracle that returns for any random $\alpha \in \mathbb{F}_p^*$, the tuple $T(\alpha) = (g, g^\alpha, g^{\alpha^2}, ..., g^{\alpha^D}) \in \mathbb{G}^{D+1}$. In the following lines, we describe an adversary $\mathcal{B}_2$ that breaks the D-SBDH assumption:

1. To break D-SBDH, $\mathcal{B}_2$ calls $\mathcal{O}_{\mathsf{D-SBDH}}$: this oracle picks a random α and returns the corresponding tuple $T(\alpha)$.
2. $\mathcal{A}_2$ enters the soundness game as described in Algorithm 5 and when $\mathcal{A}_2$ invokes $\mathcal{O}_{\mathsf{Setup}}$ with the sets of files $\mathcal{F}_k$ (for $1 \leq k \leq t$), $\mathcal{B}_2$ simulates $\mathcal{O}_{\mathsf{Setup}}$ and generates $(\mathsf{param}, \mathsf{PK}_{\mathcal{F}_k}, \mathsf{LK}_{\mathcal{F}_k})$, as follows:
   (a) $\mathcal{B}_2$ selects the parameters $g, \mathbb{G}, \mathbb{G}_T, e$ and H;
   (b) $\mathcal{B}_2$ computes the tuple $T_k(\alpha) = (g, g^{\alpha_k}, g^{\alpha_k^2}, ..., g^{\alpha_k^D})$ where $\alpha_k = \alpha\cdot\delta_k + \beta_k$ for some random $\delta_k, \beta_k \in \mathbb{F}_p^*$. Similar to Type 1 forgery, $T_k(\alpha)$ can be computed by $\mathcal{B}_2$ using the Binomial Theorem;

(c) The rest of the simulation is operated as in Figure 6.7.

Thereupon, adversary $\mathcal{A}_2$ selects a collection of keywords $\mathcal{W}_k$ to search for in a set of files $\mathcal{F}_k$ and invokes $\mathcal{O}_{\mathsf{QueryGen}}$. Adversary $\mathcal{B}_2$ computes the response of $\mathcal{O}_{\mathsf{QueryGen}}$ and returns encoded query $\mathcal{E}_{Q,k} = \mathcal{W}_k$ and the verification key $\mathsf{VK}_{Q,k} = (\mathsf{PK}_{\mathcal{F}_k}, \mathcal{W}_k)$. Later, $\mathcal{A}_2$ returns an encoded response $\mathcal{E}_{R,k}$ and runs Verify.

3. Afterwards, adversary $\mathcal{A}_2$ enters the challenge phase of the soundness experiment, specified in Algorithm 6. On input of search key $\mathsf{LK}_{\mathcal{F}^*}$ and a query on a collection of keywords $\mathcal{W}^* = \{\omega_1^*, .., \omega_k^*\}$ to be searched for in the set of files $\mathcal{F}^*$ associated with a public key $\mathsf{PK}_{\mathcal{F}}^*$ obtained earlier, $\mathcal{A}_2$ runs $\mathcal{O}\mathsf{QueryGen}$. $\mathcal{B}_1$ simulates the output of this oracle and returns the encoded query $\mathcal{E}_Q^* = \mathcal{W}^*$ and the verification key $\mathsf{VK}_Q^* = (\mathsf{PK}_{\mathcal{F}^*}, \mathcal{W}^*)$. $\mathcal{A}_2$ then produces $\mathcal{E}_R^* = (\widehat{\mathcal{F}}_{\mathcal{W}^*}, \widehat{\Pi}_{\mathcal{W}^*}, \{\widehat{\mathsf{AF}}_i\}_{1\leq i\leq k}, \{\widehat{\mathsf{path}}_i^*\}_{1\leq i\leq k})$, where:

- $\widehat{\mathcal{F}}_{\mathcal{W}^*}$ is the returned search response, that is the set of files containing $\mathcal{W}^*$;
- $\widehat{\Pi}_{\mathcal{W}^*} = \{(\widehat{\Delta}_1, \widehat{\Gamma}_1), ..., (\widehat{\Delta}_k, \widehat{\Gamma}_k)\}$, the proof of this intersection;
- $\{\widehat{\mathsf{AF}}_i\}_{1\leq i\leq k}$, the accumulation values of sets $\mathcal{F}_{\omega_i^*}$ containing keywords ω_i^*;
- $\{\widehat{\mathsf{path}}_i^*\}_{1\leq i\leq k}$, the authentication paths in Merkle tree TF for the accumulators $\{\widehat{\mathsf{AF}}_i\}$.

Here, the returned response $\widehat{\mathcal{F}}_{\mathcal{W}^*}$ is different from the expected search result $\mathcal{F}_{\mathcal{W}^*} = \mathsf{CKS}(\mathcal{F}^*, \mathcal{W}^*)$. Therefore, this is possible according to two cases:

Case (a). $\widehat{\mathcal{F}}_{\mathcal{W}^*}$ contains a file with file identifier fid^* that is not in $\mathcal{F}_{\mathcal{W}^*}$. Put in another way, $\widehat{\mathcal{F}}_{\mathcal{W}^*}$ breaks the **subset** rule of set intersection (see Figure 6.5).

Case (b). There is a file with file identifier fid^* that is in $\mathcal{F}_{\mathcal{W}^*}$ but missing from $\widehat{\mathcal{F}}_{\mathcal{W}^*}$. In other words, $\widehat{\mathcal{F}}_{\mathcal{W}^*}$ does not satisfy the **complement disjointness** property of set intersection (c.f Figure 6.5).

4. Since H is a collision-resistant hash function, the Merkle tree authentication proves that $\{\widehat{\mathsf{AF}}_i\}_{1\leq i\leq k}$ are actually associated with leaves at position i in tree TF. More precisely, it proves that each path in $\{\widehat{\mathsf{path}}_i^*\}_{1\leq i\leq k}$ authenticates the respective values $H(\widehat{\mathsf{AF}}_i^*||\omega_i^*)$ and that for $1 \leq i \leq k$, $\widehat{\mathsf{AF}}_i^*$ corresponds to $\mathcal{A}cc(\mathcal{F}_{\omega_i^*})$ that was computed in the setup phase by $\mathcal{B}_2$. Specifically, $\widehat{\mathsf{AF}}_i^* = g^{P_i^*(\alpha^*)}$ with $P_i^*(X) = \prod_{\mathsf{fid}_j \in \mathcal{F}_{\omega_i^*}}(X - \mathsf{fid}_j)$.

5. Given accumulators $\widehat{\mathsf{AF}}_i$, we show how $\mathcal{B}_2$ breaks the D-SBDH assumption in the two cases **(a)** and **(b)**. Note that these cases can occur at the same time, but for the sake of simplicity, we treat them independently:

Case (a). In this case, there exists a keyword $\omega^* \in \mathcal{W}^*$ such that $\mathsf{fid}^* \notin \mathcal{F}_{\omega^*}$. Therefore, if we denote P^* the characteristic polynomial of $\mathcal{F}_{\omega^*}$, $(X - \mathsf{fid}^*)$ does not divide $P^*(X)$. However, since $\mathsf{fid}^* \in \widehat{\mathcal{F}}_{\mathcal{W}^*}$, then $(X - \mathsf{fid}^*)$ divides $\widehat{P}(X)$ where $\widehat{P}$ is the characteristic polynomial of $\widehat{\mathcal{F}}_{\mathcal{W}^*}$. Using polynomial division, we find that there exist polynomial Z_1, Z_2 and $R \in \mathbb{F}_p$ such that $P^*(X) = (X - \mathsf{fid}^*) \cdot Z_1(X) + R$ and $\widehat{P}(X) = (X - \mathsf{fid}^*) \cdot Z_2(X)$. Hence, when $\mathcal{B}_2$ verifies the first equality of $\mathsf{VerifyIntersection}$ (cf. Figure 6.5), she gets for $1 \leq i \leq k$:

$$e(\mathcal{A}cc(\widehat{\mathcal{F}}_{\mathcal{W}^*}), \widehat{\Delta}_i) = e(\mathcal{A}cc(\mathcal{F}_{\omega^*}), g)$$

$$e(g, \Delta_i)^{\widehat{P}(\alpha^*)} = e(g, g)^{P^*(\alpha^*)}$$

$$\begin{aligned}
e(g,\widehat{\Delta}_i)^{(\alpha^*-\mathsf{fid}^*)\cdot Z_2(\alpha^*)} &= e(g,g)^{(\alpha^*-\mathsf{fid}^*)\cdot Z_1(\alpha^*)+R}\\
e(g,\widehat{\Delta}_i)^{Z_2(\alpha^*)} &= e(g,g)^{Z_1(\alpha^*)}\cdot e(g,g)^{\frac{R}{\alpha^*-\mathsf{fid}^*}}\\
(e(g,\widehat{\Delta}_i)^{Z_2(\alpha^*)}\cdot e(g,g)^{-Z_1(\alpha^*)})^{\frac{1}{R}} &= e(g,g)^{\frac{1}{\alpha^*-\mathsf{fid}^*}}.
\end{aligned}$$

Assuming that we have $\alpha^* = \alpha\cdot\delta^* + \beta^*$, where (δ^*,β^*) are randomly selected from set $\{\delta_k,\beta_k\}_{1\le k\le t}$ generated earlier, we can write:

$$\begin{aligned}
\Big(e(g^{Z_2(\alpha^*)},\widehat{\Delta}_i)\cdot e(g^{-Z_1(\alpha^*)},g)\Big)^{\frac{1}{R}} &= e(g,g)^{\frac{1}{\delta^*(\alpha+\frac{\beta^*-\mathsf{fid}^*}{\delta^*})}}\\
\Big(e(g^{Z_2(\alpha^*)},\widehat{\Delta}_i)\cdot e(g^{-Z_1(\alpha^*)},g)\Big)^{\frac{\delta^*}{R}} &= e(g,g)^{\frac{1}{\alpha+\frac{\beta^*-\mathsf{fid}^*}{\delta^*}}}
\end{aligned}$$

In other words, we construct an adversary $\mathcal{B}_2$ that breaks the D-SBDH assumption by outputting the pair $\left(\frac{\beta^*-\mathsf{fid}^*}{\delta^*},\Big(e(g^{Z_2(\alpha^*)},\widehat{\Delta}_i)\cdot e(g^{-Z_1(\alpha^*)},g)\Big)^{\frac{\delta^*}{R}}\right)$. Notice that β^* is randomly generated in $\mathbb{F}_p^*$, and therefore $\Pr(\beta^* = \mathsf{fid}^*) = \frac{1}{p}$. this means that if $\mathcal{A}_2$ has a non-negligible advantage ε_A to break the soundness of our scheme, then there is an adversary $\mathcal{B}_2$ that breaks D-SBDH with a non-negligible advantage $\varepsilon_B \ge \varepsilon_A\cdot(1-\frac{1}{p})$.

Case (b). In this case, fid^* is in $\mathcal{F}_{\mathcal{W}^*}$ but not in $\widehat{\mathcal{F}}_{\mathcal{W}^*}$. Since, we exclude Case **(a)** here, it means that $\widehat{\mathcal{F}}_{\mathcal{W}^*}\subset\mathcal{F}_{\mathcal{W}^*}$. Besides, fid^* can be found in all sets $(\mathcal{F}_{\omega_i^*}\backslash\widehat{\mathcal{F}}_{\mathcal{W}^*})$, for all $1\le i\le k$. We denote R_i the characteristic polynomials of $(\mathcal{F}_{\omega_i^*}\backslash\widehat{\mathcal{F}}_{\mathcal{W}^*})$.

We also have $P_i(X) = R_i(X)\cdot\widehat{P}(X)$ where P_i denote the characteristic polynomial of $\mathcal{F}_{\omega_i^*}$ and $\widehat{P}$ is the characteristic polynomial of $\widehat{\mathcal{F}}_{\mathcal{W}^*}$. If algorithm Verify accepts $\mathcal{A}_2$'s proof then it means that $e(\mathcal{A}cc(\widehat{\mathcal{F}}_{\mathcal{W}^*}),\widehat{\Delta}_i) = e(\mathcal{A}cc(\mathcal{F}_{\omega_i^*}),g)$, which can be written as $e(g,\widehat{\Delta}_i)^{\widehat{P}(\alpha^*)} = e(g,g)^{P_i(\alpha^*)}$. It follows that $\Delta_i = g^{R_i(\alpha)}$. In addition, $(X-\mathsf{fid}^*)$ divides $R_i(X)$ and we can write $R_i(X) = (X-\mathsf{fid}^*)\cdot Z_i(X)$. When $\mathcal{B}_2$ verifies the second equality of $\mathsf{VerifyIntersection}$, he gets:

$$\begin{aligned}
\prod_{i=1}^{k} e(\widehat{\Delta}_i,\widehat{\Gamma}_i) = \prod_{i=1}^{k} e(g,\widehat{\Gamma}_i)^{R_i(\alpha^*)} &= e(g,g)\\
\prod_{i=1}^{k} e(g,\widehat{\Gamma}_i)^{(\alpha^*-\mathsf{fid}^*)\cdot Z_i(\alpha^*)} &= e(g,g)\\
(\prod_{i=1}^{k} e(g,\widehat{\Gamma}_i)^{Z_i(\alpha^*)})^{(\alpha^*-\mathsf{fid}^*)} &= e(g,g)\\
\prod_{i=1}^{k} e(g,\widehat{\Gamma}_i)^{Z_i(\alpha^*)} &= e(g,g)^{\frac{1}{\alpha^*-\mathsf{fid}^*}}
\end{aligned}$$

Since we have $\alpha^* = \alpha\delta^* + \beta^*$, with (δ^*,β^*) randomly selected from set $\{\delta_k,\beta_k\}_{1\le k\le t}$ generated earlier, it follows that:

$$\begin{aligned}
\prod_{i=1}^{k} e(g^{Z_i(\alpha^*)},\widehat{\Gamma}_i) &= e(g,g)^{\frac{1}{\delta^*(\alpha+\frac{\beta^*-\mathsf{fid}^*}{\delta^*})}}\\
(\prod_{i=1}^{k} e(g^{Z_i(\alpha^*)},\widehat{\Gamma}_i))^{\delta^*} &= e(g,g)^{\frac{1}{\alpha+\frac{\beta^*-\mathsf{fid}^*}{\delta^*}}}
\end{aligned}$$

Therefore, if $\beta^* \neq \mathsf{fid}^*$, then adversary $\mathcal{B}_2$ breaks the D-SBDH assumption with the pair $\left(\frac{\beta^*-\mathsf{fid}^*}{\delta^*}, (\prod_{i=1}^{k} e(g^{Z_i(\alpha^*)}, \widehat{\Gamma}_i))^{\delta^*}\right)$. Since $\beta^* \neq \mathsf{fid}^*$ with probability $\frac{1}{p}$, we can safely conclude that if there is an adversary $\mathcal{A}_2$ that breaks the soundness of our scheme with a non-negligible advantage ε_A, then there is an adversary $\mathcal{B}_2$ that breaks D-SBDH with a non-negligible advantage $\varepsilon_B \geq \varepsilon_A \cdot (1-\frac{1}{p})$.

Security of updates. All the critical update operations we presented in Section 6.5.4 are performed on the server side, which mainly updates the Merkle trees TW and TF. Besides, the server generates the proof Π_{upd} which ascertains of the correctness of the tree update. It consists of the updated accumulators and their authentication paths. Therefore, since we consider H being a collision-resistant hash function, the server cannot make the verifier accept incorrect updates.

6.7 Performance Evaluation

This section discusses the efficiency of our VCKS protocol. We also present some experimental results based on the implementation of a prototype of our solution.

6.7.1 Discussion on Efficiency

Hereafter, we evaluate the costs of our scheme with respect to storage, computation and communication complexities. Table 6.2 gives a summary of these various costs.

6.7.1.1 Storage

With respect to storage complexity, the data owner stores and publishes public key $\mathsf{PK}_{\mathcal{F}} = (\{g^{\alpha^i}\}_{0\leq i\leq D}, \sigma_W, \sigma_F)$, of size $\mathcal{O}(D)$. On the other hand, cloud server $\mathcal{S}$ keeps search key $\mathsf{LK}_{\mathcal{F}} = (\mathcal{I}, \mathsf{TW}, \mathsf{TF}, \mathcal{F}, \mathsf{HT})$, resulting in a storage complexity in $\mathcal{O}(N+n)$, where N is the total number of keywords and n is the number of files. Indeed, the size of index $\mathcal{I}$, hash table HT and Merkle trees TW and TF is linear in N.

6.7.1.2 Computation

In light of the performances of the several building blocks (Cuckoo hashing, polynomial-based accumulators and Merkle trees), we analyze in the following the computational costs of our solution.

1. **Setup:** The *Setup* phase of our protocol is a one-time pre-processing operation that is **amortized** over an unlimited number of fast verifications. The computational cost of this phase is dominated by:

    - The public parameter generation which amounts to D exponentiations in $\mathbb{G}$;
    - N calls to $\mathsf{CuckooInsert}$ where, as shown in [74], each insertion is expected to terminate in $(1/\varepsilon)^{\mathcal{O}(\log d)}$ time ($\varepsilon > 0$);
    - The computation of m accumulators AW which requires m exponentiations in $\mathbb{G}$ and md multiplications in $\mathbb{F}_p$ (in the worst case);
    - The computation of N accumulators AF which involves N exponentiations in $\mathbb{G}$ and Nn multiplications in $\mathbb{F}_p$ (in the worst case);

- The generation of Merkle tree TW (respectively TF) which consists of $2m$ hashes (respectively $2N$).

2. **QueryGen:** This algorithm does not require any computation. It only constructs the query for the k keywords together with the corresponding VK_Q.

3. **Search:** Although this algorithm seems expensive, we highlight the fact that it is executed by the cloud server, who has more computational resources than the data owner and the users. Search runs k CuckooLookup which consists in $2k$ hashes and $2kd$ comparisons to search for all the k queried keywords (in the worst case). Following this operation, the complexity of this algorithm depends on whether all the keywords have been found:

   - $\mathsf{out} = \mathcal{F}_{\mathcal{W}}$: The complexity of Search is governed by:
     - The computation of k file accumulators AF. Without the knowledge of trapdoor α, and using FFT interpolation as specified in [52], this operation performs $kn \log n$ multiplications in $\mathbb{F}_p$ and k exponentiations in $\mathbb{G}$;
     - The generation of the authentication paths in tree TF for these accumulators, which amounts to $k \log N$ hashes;
     - The generation of the proof of intersection that takes $\mathcal{O}((kn)\log^2(kn)\log\log(kn))$ multiplications[102] in $\mathbb{F}_p$ to compute the gcd of the characteristic polynomials of the sets involved in the query result.
   - $\mathsf{out} = \emptyset$: The computational costs of this phase consist in:
     - The generation of the proof of membership for the missing keyword by calling twice GenerateWitness. This operation requires $2(d + d\log d)$ multiplications in $\mathbb{F}_p$ and $2d$ exponentiations in $\mathbb{G}$;
     - The computation of 2 bucket accumulators AW, which amounts to $2d\log d$ multiplications in $\mathbb{F}_p$ and $2d$ exponentiations in $\mathbb{G}$;
     - The generation of 2 authentication paths for these 2 buckets by running GenerateMTProof on tree TW, which performs $2\log m$ hashes.

4. **Verify:** We also analyze the complexity of this algorithm according to whether all the keywords have been found:

   - $\mathsf{out} = \mathcal{F}_{\mathcal{W}}$: Verify runs k instances of VerifyMTProof on tree TF, which requires $k \log N$ hashes. It computes the accumulator of the intersection, which, in the worst case, amounts to n multiplications in $\mathbb{F}_p$ and 1 exponentiation in $\mathbb{G}$. Then, it executes VerifyIntersection which computes $3k$ pairings and k multiplications in $\mathsf{G_T}$.

   - $\mathsf{out} = \emptyset$: Verify runs twice VerifyMTProof on tree TW that computes $2\log m$ hashes and it invokes twice VerifyMembership that evaluates $3 \times 2 = 6$ pairings.

In summary, to verify the search results, a verifier $\mathcal{V}$ performs very light computations compared to the computations undertaken by the server when answering keyword search queries and generating the corresponding proofs. Besides, the verification cost depends on k only in the case where all the keywords have been found and is independent otherwise. Furthermore, we believe that for large values of k, the probability that the search returns a set of files containing all the k keywords is low. Hence, the verification cost will be constant and small (6 pairings and $2\log m$ hashes). On the other hand, for smaller values of k, the verification cost remains efficient.

[102]More details on this complexity computation can be found in [140, 52].

Table 6.2: Complexities of our protocol for verifiable keyword search

Notations
D: parameter of our system, $n \leq D$: number of files, N: number of keywords
m: the number of buckets in the index, d: size of a bucket
k: number of keywords in a query.

Storage	*$\|G\|$ refers to the size (in bits) of elements in set G.*	
Data owner	$\mathcal{O}(D)$	$D \cdot \|\mathbb{G}\| + 2 \times 256$ bits
Server	$\mathcal{O}(N+n)$	$md \cdot \|\mathbb{F}_p\| + (2m-1) \times 256 + (2n-1) \times 256$ bits
	We assume H is SHA-256.	
Communication		
Outbound	$\mathcal{O}(k)$	$k \cdot \|\{0,1\}^*\|$ bits
Inbound		
out $= \mathcal{F}_{\mathcal{W}}$	$\mathcal{O}(n+k)$	$n \cdot \|\{0,1\}^*\| + 3k \cdot \|\mathbb{G}\| + k \log n \times 256$ bits
out $= \emptyset$	$\mathcal{O}(1)$	$2 \cdot \|\mathbb{G}\| + 2 \cdot (\|G\| + \|\mathbb{F}_p\|) + 2 \log m \times 256$ bits

Operations	Setup	QueryGen	Search		Verify	
			out $= \mathcal{F}_{\mathcal{W}}$	out $= \emptyset$	out $= \mathcal{F}_{\mathcal{W}}$	out $= \emptyset$
Hashes in $\{0,1\}^*$	$2(m+N+1)$	-	$k \cdot (3 + \log N)$	$2 \log m + 3k$	$k \log N$	$2 \log m$
Multiplications in $\mathbb{F}_p$	$md + Nn$	-	$kn \cdot \log n$	$2d + 4d \log d$	n	-
Multiplications in $\mathbb{G}_T$	-	-	-	-	k	-
Exponentiations in $\mathbb{G}$	$D + m + N$	-	k	$4d$	1	-
Pairings	-	-	-	-	$3k$	6
CuckooInsert	N	-	-	-	-	-
ProveIntersection	-	-	1	-	-	-
Light Computation	-	k	-	-	-	-

6.7.1.3 Communication

In terms of communication complexity, our protocol for VCKS is relatively lightweight. In particular, sending a search query to server $\mathcal{S}$ amounts to $\mathcal{O}(k)$ space, where k is the number of keywords in that query.

The size of server's answer to the query depends on the outcome of Search algorithm, as shown in Figure 6.8:

If all the keywords are found: Then server's result is of size $\mathcal{O}(n+k)$, where n corresponds to the worst case scenario where all the outsourced files contain the conjunction of keywords, and k results from the underlying proof of intersection of k sets of files $\mathcal{F}_{\omega_i}$.

If one keyword is not found: In this case, server $\mathcal{S}$ has to send a search result of size $\mathcal{O}(1)$ that contain the keyword that is not found and its corresponding proof of non-membership.

6.7.1.4 Impact of D on the performance.

This performance analysis assumes $n \leq D$, where n is the number of files. The value of D solely depends on security parameter κ, and as such, defines an upper-bound to the size of sets for which we can compute a polynomial-based accumulator. It follows that in our protocol, the number of files that a data owner can outsource at once is bounded by D. However, it is still possible to accommodate files' sets that exceed the bound D. The idea is to divide the set of size n into $n' = \lceil \frac{n}{D} \rceil$ smaller sets of size D. By using the same public parameters, Setup accordingly creates for each set of D files an index and the corresponding Merkle trees. This increases the complexity of the Setup by a factor of n'. Namely, the data owner is required to build n' Cuckoo indexes and $2n'$ Merkle trees. However, since D is now constant, the computation of each file accumulator $\mathcal{A}cc(\mathcal{F}_\omega)$ will take $\mathcal{O}(D \log D) = \mathcal{O}(1)$ time. Besides, the server has to run n' ProveIntersection which runs in $\mathcal{O}((kD)\log^2(kD)\log\log(kD)) = \mathcal{O}(k\log^2(k)\log\log(k))$ time.

6.7.2 Experimental Analysis

This section evaluates the performance of our proposed PVCKS scheme with two datasets:

- For the purpose of experimental analysis, we created a collection of files which gathers the complete plays of Molière[103]. We will refer to this dataset as "Molière". This dataset consists of $n = 25\,293$ files and is of size 1.34 MB. It is populated by $N = 16\,720$ distinct keywords.
- The Enron email dataset[104] is used to test our protocol in a real-world scenario. This collection of data includes emails generated by 150 former employees of Enron, an American energy company that faced a bankruptcy in the beginning of years 2000s. The dataset was acquired for a matter of investigation after Enron's collapse and made publicly available for study and research purposes. The investigation of Enron email dataset applies to a scenario where our PVCKS would be an interesting tool. Indeed, some verifiers from financial auditing companies or legal authorities might be required to search in the collection of emails for some keywords related to their investigations. This search might be delegated to the cloud which is then compelled to return the search results with a proof of correct search. The Enron email dataset contains 2.54 GB of $n = 517\,401$ files and $N = 2\,141\,711$ unique keywords.

Experimental Setup. Our prototype is written in C and uses the Pairing-Based Cryptography (PBC) library[105]. We run the experiments on two different machines. The Molière dataset was tested on a laptop with the following characteristics: processor Intel Core i5-4258U CPU@2.4 GHz; RAM of 4 GB; system on 64-bit.

We tested the Enron email dataset on a desktop machine with the following characteristics: processor Intel Core i5-2500 CPU@3.80GHz; RAM of 16 GB; system on 64-bit.

6.7.2.1 Benchmarks on the Molière dataset

Evaluation of the Setup phase. To evaluate the performance of the Setup phase, we run our prototype on three different scenarios based on the Molière dataset. We arrange the collection of files so that $n = 1$, $n = 253$ or $n = 25293$ files; each of these settings involves $N = 16720$ unique keywords (the same set of N keywords for each file collection). The various benchmark times mentioned here correspond to the average of times of 5 trials of our prototype against the data set. Table 6.3 shows the experimental results of the different steps of the Setup phase in the case it is run on the collection of $n = 25293$ files. In the different cases, the prototype builds a Cuckoo index that can store up to $m \times d = 65536$ keywords. Hence, 26% of the index is populated.

As we can see in Table 6.3, the most computationally expensive operation is the computation of the accumulators AF. This is in accordance with our performance evaluation: algorithm Setup must compute N accumulators AF, one for each keyword, and the underlying polynomial of each of these accumulators is of degree at most n. Nevertheless, notice that this computational burden is allowed in the amortized model our protocol adopts. Setup is executed once for multiple search queries and verifications. We will discuss about the amortization later on.

Figure 6.15 compares the average total time for the Setup phase for different values of n and m (thus different values for d since N is fixed and m, d and N are linked with the relation $m = \frac{1+\varepsilon}{d}N$). In particular, according to our performance analysis conducted in Table 6.2,

[103]"Molière œuvres complètes", http://tiny.cc/17hu8x [Accessed: February 4, 2016].

[104]The Enron Email Dataset, http://tiny.cc/d9hu8x [Accessed: February 4, 2016].

[105]The Pairing-Based Cryptography Library: https://crypto.stanford.edu/pbc/ [Accessed: February 4, 2016].

Table 6.3: Time of Setup phase $n = 25293$ files

m	d	Index $\mathcal{I}$ (s)	AW (s)	$\{\mathcal{F}_{\omega_i}\}$ (s)	AF (s)	TW (s)	TF (s)	Total (s)
8192	8	3.489	7.252	11.484	26.591	0.147	2.284	51.248
4096	16	1.819	4.240	10.568	28.843	0.039	2.341	47.851
2048	32	2.037	2.038	10.505	29.760	0.012	2.357	46.708
1024	64	1.940	1.074	10.998	27.889	0.003	2.270	44.175

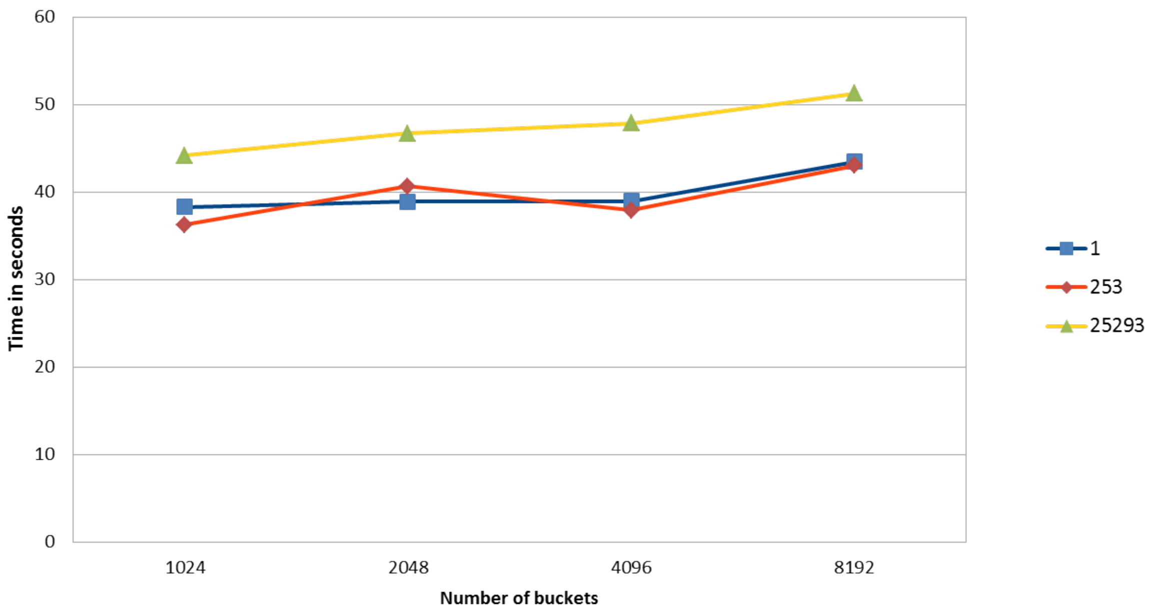


Figure 6.15: Average total time of the Setup phase

for N and m fixed, the Setup operations is linearly dependent on n (in the worst case) only because of the computation of accumulators AF. However, in our setting, we do not reach this worst case (namely, all the keywords are not in all the files). This can be explained by the fact that one particular keyword is not present in all the n files of the tested data collection. Empirical analysis shows that on average, in the case of $n = 253$ files, a keyword is present in 6 files while in the case of $n = 25293$ files, a keyword is present in 13 files. We also highlight the fact that the overall efficiency of the Setup phase is acceptable for practical scenarios.

Evaluation of the Search phase. To appreciate the performance of algorithms Search and Verify, we use the same data set as before. We want to evaluate the dependencies of these algorithms with n (the number of files) and k (the number of searched keywords), while N is fixed ($N = 16720$ unique keywords in the entire data set). We also look at the two cases where the search outputs an empty or a non-empty result. All the reported times in this paragraph are computed as the average of 25 executions of our prototype.

We first run our program for a fixed number of searched keywords: $k = 2$. In this setting, we evaluate the impact of n in the efficiency of the algorithms when all the keywords of the search query are found. Hence, our prototype launches the search operations for several values of n: $n = 10, 100, 1000, 10000$ files (selected from our test data set). Figure 6.16 and Table 6.4 present the results of this benchmark. As we can see, the increase in n slows the search operation. This is due to the computation of each accumulator AF which, in the worst case, involves n files. ProveIntersection is also a computationally demanding operation, since it has to compute the gcd of k sets, where each of the sets may include, in the worst case, the identifiers of n files. Therefore, we consider ProveIntersection as the most expensive operation performed in our solution. Besides, as we saw in Section 6.7.1, the time required by Verify does not depend on the number of searched keywords. Figure 6.16 and Table 6.4 validate this property.

Table 6.4: Time of Search phase (fixed number of searched keywords)

n	ProveIntersection (s)	Search (s)	Verify (s)
10	0.002	0.005	0.009
100	0.005	0.008	0.009
1000	0.075	0.114	0.009
10000	0.703	1.063	0.011

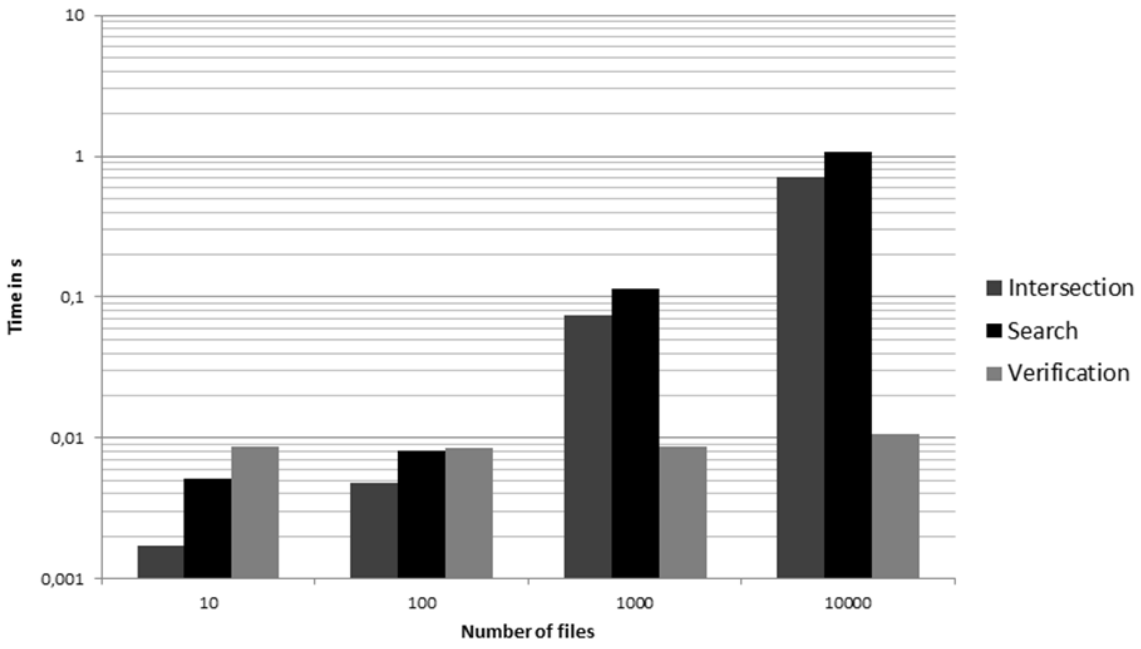


Figure 6.16: Total time of the Search phase (fixed number of searched keywords)

Thereafter, we conduct a second experiment where n is fixed to $n = 25293$ files and the number of searched keywords k varies according to the following serie $k = 2, 5, 10$. This scenario assesses the effect of k in the efficiency of Search and Verify. Table 6.5 and Figure 6.17 display respectively the times to compute the proof of intersection, the search and the verification in the case when all the keywords of the search query are found.

Table 6.5: Time of Search phase - all keywords are found

k	ProveIntersection (s)	Search (s)	Verify (s)
2	13.121	21.369	0.009
5	120.867	172.121	0.017
10	305.062	368.500	0.036

First, let us consider the time required by algorithm Verify. As expected by the theoretical performance evaluation, it linearly increases with k the number of keywords to be searched for and is negligible compared to the running time of algorithm Search. In addition, Figure 6.17 shows that the step that generates the intersection of sets and its proof determines the time

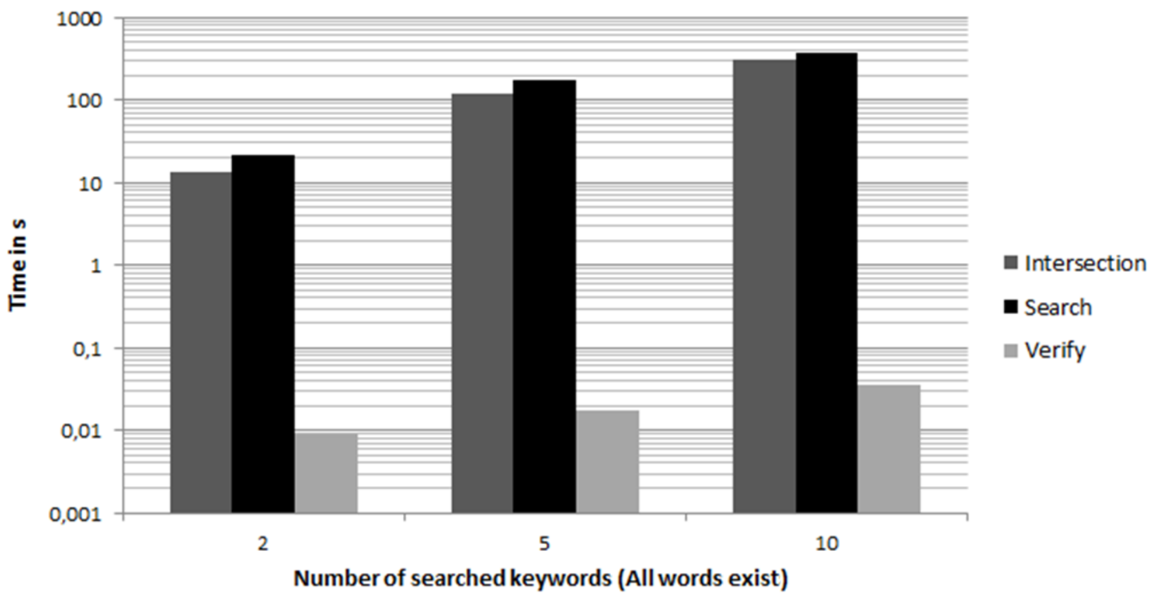


Figure 6.17: Total time of the Search phase - all keywords are found

of Search. Besides, while N and n are fixed in this experiment, we observe that the times for ProveIntersection and Search are not strictly linear in k as expected by the performance evaluation conducted in Table 6.2. We explain this divergence by the fact that our theoretical evaluation does not take into account the different sizes of the sets accumulated in AF. Indeed, Table 6.2 presents the worst case complexities according to which each of the N keywords exist in all the n files. Practically, in the case where $n = 25293$ files, a keyword is present in only 13 files. Hence, between two distinct values of k, the computation of accumulators AF and the proof of intersection of their respective sets impacts differently the time of operating the search.

Finally, we study the performance of our protocol when the search has not found a keyword from the query. In the following, we evaluate the evolution of the time required by algorithms Search and Verify in function of k the number of searched keywords in a query. These results are presented in Table 6.6 and depicted in Figure 6.18. First, we can notice that in accordance with the theoretical performance evaluation presented in Table 6.2, the running time of Verify is independent from k. Secondly, we highlight the fact that in this experiment, the search takes much less time than in the case where all the keywords in the search query were found. Furthermore, while we were expecting the time of Search to be linear in k, Figure 6.18 shows a different evolution. We can explain this discrepancy by the fact that our theoretical performance was studied in the worst case, where each of the bucket in Cuckoo index $\mathcal{I}$ was populated by d keywords (the maximum capacity of the buckets). In reality, all the buckets in the index might not be full. Hence, the computation of the accumulators AW for the missing keyword may take less time than in the worst-case scenario. Therefore, in our experiment, we believe that in the case where $k = 5$, the buckets corresponding to the missing keyword were empty, while in the case where $k = 2$, these buckets were probably filled by at most d keywords (in this experiment $d = 8$ keywords).

Table 6.6: Time of Search phase - one keyword is not found

k	Search (s)	Verify (s)
2	0.039	0.008
5	0.008	0.008
10	0.049	0.008

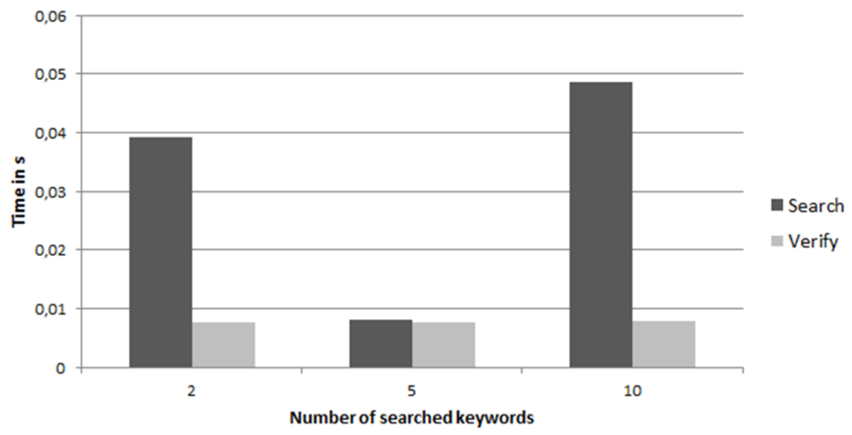


Figure 6.18: Total time of the Search phase - one keyword is not found

Discussion on Amortization and Outsourceability. We discuss the amortization and the outsourceability of the conjunctive keyword search operation. In particular, we will define in terms of computation, storage and bandwidth consumption the criteria that make the use our verifiable keyword search protocol a more interesting strategy for a data owner than performing the search locally.

In terms of computational performance, we compare our protocol executed over the Molière dataset with a local search performed at the client side (without resorting to cloud technology). This local search requires a pre-processing step which includes the construction of the Cuckoo index $\mathcal{I}$ as well as the indexing $\{\mathcal{F}_{\omega_i}\}$ of files according to the keywords that the files in Molière dataset contain. Then, the data owner can perform the search for a conjunction of keywords by executing algorithm $\mathsf{CuckooLookup}$ over index $\mathcal{I}$ and computing the intersection of sets $\{\mathcal{F}_{\omega_i}\}$ for all the keywords targeted by the search. Table 6.7 depicts the time to run the pre-processing step and the search for 2 keywords for different sizes of the Molière data set.

Table 6.7: Local search computational performance

n	Pre-Processing (s)	Search (s)
10	0.002	2.524×10^{-5}
100	0.017	2.968×10^{-4}
1000	0.203	3.326×10^{-3}
10000	2.618	0.021

First, we can assess from Table 6.7 that while, for a large number of files, our protocol satisfies the efficiency requirement we gave in Section 3.4.2 (namely, the verification time showed in Table 6.5 is less than the local search time, for $n = 10000$), it is not outsourceable in the computational definition of the amortized model. To better understand this concept we introduce the following definition for a verifiable keyword search solution.

Definition 20 (Outsourceability - Computation). *The criterion x of outsourceability in computation for a verifiable keyword search scheme is determined by a parameter $x \geq 0$ such that:*

$$t_{\mathsf{Setup}} + x \cdot (t_{\mathsf{ProbGen}} + t_{\mathsf{Verify}}) \leq t_{\mathsf{Preprocessing}_{\mathsf{Local}}} + x \cdot t_{\mathsf{Search}_{\mathsf{Local}}}$$

where t_{algo} is the time required to execute algorithm algo. Hence, x is defined by the relation:

$$x = \left\lceil \frac{t_{\mathsf{Setup}} - t_{\mathsf{Preprocessing}_{\mathsf{Local}}}}{t_{\mathsf{Search}_{\mathsf{Local}}} - (t_{\mathsf{ProbGen}} + t_{\mathsf{Verify}})} \right\rceil$$

Regarding the tests we performed on the Molière dataset, we cannot find an $x \geq 0$ that meets the outsourceability definition. Nonetheless, our protocol is still arguably outsourceable. Indeed, the above definition for outsourceability only takes into account the computational performances of a verifiable keyword search solution. However, in real-world scenarios, storage and bandwidth consumption are also to be considered in the outsourceability criteria:

Storage: Our protocol targets large databases that cannot be locally stored at the data owner side due to its heavy consumption of memory. If we consider the case where users may use laptops, smartphones or other devices with low memory resources, outsourcing the storage and thus the search operation to the cloud will be necessary. Therefore, the outsourceability definition must take into account this storage criterion: data owners with limited memory resources choose to outsource their large databases to the cloud while delegating the search operation to the cloud server as the best strategy.

Bandwidth: In the context of a publicly delegatable and verifiable keyword search solution, bandwidth consumption is a criterion of paramount importance in the notion of outsourceability. Indeed, without our protocol, if a data owner delegates to third-party queriers the capability to search their large datasets, she has to transmit for each querier the search indexes, which consumes a non-negligible amount of bandwidth. Hence, in this context, outsourceability can be defined as the amount of search queries needed to amortize the expensive setup phase. Namely, it should be less consuming to delegate the capability of requesting a search operation and verifying its results than to transmit x times the search indexes.

Definition 21 (Outsourceability - Bandwidth). *The criterion x of outsourceability in bandwidth for a publicly delegatable and verifiable keyword search scheme is determined by a parameter $x \geq 0$ that defines the number of search queries (issued by different users) such that:*

$$x \cdot \mathsf{BW}(\mathcal{O} \rightarrow \mathcal{Q}) \geq \mathsf{BW}(\mathcal{O} \rightarrow \mathcal{S}) + x \cdot \mathsf{BW}(\mathcal{S} \leftrightarrows \mathcal{Q})$$

where:

- $\mathsf{BW}()$ *is the bandwidth consumed to exchange data between two parties,*
- *"$\mathcal{O} \rightarrow \mathcal{Q}$" translates the transmission of search indices by data owner $\mathcal{O}$ to a third-party querier $\mathcal{Q}$,*
- *"$\mathcal{O} \leftarrow \mathcal{S}$" corresponds to the upload of the database and the corresponding search indices to server $\mathcal{S}$, and*
- *"$\mathcal{S} \leftrightarrows \mathcal{Q}$" is the challenge-response between $\mathcal{S}$ and $\mathcal{Q}$ in which $\mathcal{Q}$ sends a search query and $\mathcal{S}$ returns the search result.*

6.7.2.2 Benchmarks on the Enron email dataset

In this section, we evaluate the performance of our publicly verifiable conjunctive keyword search scheme with a real-world dataset in order to demonstrate its practical efficiency. We remind the reader that the Enron dataset includes of $n = 517\,401$ files which amounts to 2.54 GB in memory size. The entire set contains $N = 2\,141\,711$ unique keywords. The different parameters and running times of our prototype against the Enron email dataset are listed in Table 6.8.

Table 6.8: Parameters and experimental results on Enron email dataset

	Values
n	517 401
N	2 141 711
m	16 384
d	512
Setup time	7.46 h
Search time	
• for 3 frequent keywords	1.30 h
• for 3 infrequent keywords	4.13 min
• for missing keyword	1.05 s
Verify time	
• for 3 frequent keywords	2.50 s
• for 3 infrequent keywords	0.09 s
• for missing keyword	0.01 s
Storage overhead at the server	564.3 MB

Description of the experiment. We run the preprocessing phase, that is algorithm Setup, in the entire collection of emails without any filtering or pruning of the dataset. Setup creates the Cuckoo index $\mathcal{I}$ with the parameters $d = 512$ and $m = 16384$. Hence, 26% of the index is full. As described in Section 6.5, the algorithm also determines the file index, that is the set of files $\{\mathcal{F}_{\omega_i}\}$, and computes the two types of accumulators and the two Merkle trees. To test the efficiency of algorithms Search and Verify, we launch three types of search queries:

1. The first query requests 3 common keywords in the Enron email dataset (such as *report*, *meeting*, *Monday*). They are present together in 997 files;
2. The second query targets 3 keywords that exist in the Enron email dataset (*Robertson*, *Stephens*, *finances*) but which are present together in 10 files only;
3. The third query searches for keywords that are not present in the Enron email dataset.

Preprocessing at the data owner. As we already explained for the Molière dataset, algorithm Setup is a (one-time) expensive phase. To process the entire Enron email dataset, our implementation takes approximately 7.46 hours, due to the computation of accumulators. While expensive, we will see that it allows for very fast search result verifications.

Storage overhead at the server. Naturally, the server stores the dataset which amounts to 2.54 GB of memory. Additionally, it is required to store $\mathcal{I}$ and the sets $\{\mathcal{F}_{\omega_i}\}$, which consists of 564.3 MB of memory in total, or 26.7% of the entire dataset. As regards of the two Merkle trees that are built by algorithm Setup, the server is not compelled to store them. Indeed, it can build them on the fly whenever it received a search request. As a matter of fact, the generation of tree TW (respectively tree TF) only takes 0.017 seconds (respectively 4.007 seconds).

Search efficiency. In this paragraph, we analyze and comment the times it takes to perform the three test queries:

Query 1: Responding to this type of query is relatively expensive for the server compared to the two other types. Indeed, as the searched keywords are present in many files (here in 997 files), the server has to compute 3 polynomials of degree 997, to compute accumulators and the proof of set intersection. It took nearly 1.30 hours to return the search results. We argue that in a real scenario this type of query will not induce valuable information for any investigator that would like to mine the email dataset. Indeed, such verifiers would search for more *sensitive* keywords for which the search results return highly instructive information. Besides, this search is run by the cloud server. Thus in a real cloud environment, the time to search would be less than what we measured in a desktop machine.

Query 2: The second query targets 3 less common keywords. For example, by searching the set {*Robertson*, *Stephens*, *finances*}, an investigator that analyzes the Enron email dataset is interested in emails exchanged between individuals named Robertson and Stephens and which refer to some finances issues. Our test query returned only 10 files in approximately 4 minutes. In a real context, the amount of time taken to return the search result is acceptable and does not impede from resorting to cloud technologies.

Query 3: A search that returns an empty result is, as expected, the fastest type of search query. It takes only a bit more than 1 second for the server to perform this search.

The analysis of the search efficiency at the server side shows that the time to execute the search operation highly depends on the size of the intersection, that is, on the *popularity* of searched keywords. The more the searched keywords are popular, the longer it will take to perform the search and to generate the proof of correct results. In the case where a keyword is not present in the file collection, the search operation is very fast even with large datasets.

Verification efficiency. We saw, in relation with the Molière dataset, that the search result verification is a very fast and lightweight operation performed by the verifier. This result is also confirmed by our experiment in the Enron email dataset. As depicted in Table 6.8, for the case where 3 keywords are found in 997 files, it takes only 2.50 seconds to verify the search results, while in the case of less common keywords, verification is performed in 0.088 seconds only. Furthermore, in the case where the search returns an empty result, the verification is even faster (0.011 seconds). Therefore, our solution allows fast and practical search verifications even for a large number of files and popular keywords.

Amortization. Our last analysis perspective discusses the amortization of our protocol for the Enron email dataset. To appreciate the computational outsourceability as defined in Definition 20, we list in Table 6.9 the times required to perform a conjunctive keyword search in two cases: in the case where the search is performed by the data owner and the case where the search operation is outsourced to a cloud server.

Table 6.9: The Enron email dataset and amortization

	Search at the data owner			Search outsourced at the cloud	
	Cuckoo Index (s)	File index (s)	Local Search (s)	Setup (s)	Verify (s)
Query 1	2266.956	21390.553	5.197	26870.847	2.500
Query 2	2266.956	21390.553	0.026	26870.847	0.088
Query 3	2266.956	21390.553	1.25×10^{-4}	26870.847	0.011

According to the criterion of outsourceability in computation we defined in Definition 20, we learn that in the case of a query of type 1, the expensive operations performed by algorithm Setup are amortized after 1277 queries. However, in the case of queries of type 2 and 3, computation performances are not amortized. Hence, we resort to two another parameters, that we already mentioned in the case of the Molière dataset, and which support the viability of our solution:

Storage: As shown in Table 6.8, the search over the Enron dataset incurs a storage overhead of 564.3 MB, due to the generation of search indices. This additional storage should be considered even if the data owner does not outsource the storage and the search operation over this file collection to the cloud. Hence, for users with limited storage resources, the burden of keeping the dataset in addition with the indices is substantially costly. The best strategy for this kind of users is therefore to outsource the dataset to the cloud and thus, the search operation over this data set.

Bandwidth: We recall that our solution is publicly delegatable and verifiable. In this context, third-party queriers can search for a conjunction of keywords in the Enron dataset, without any interaction with the data owner. If the data owner decides to outsource the search operation to the cloud, she will transmit the dataset of 2.54 GB and the search indices of 564.3 MB. This bandwidth is consumed only once in the Setup phase of our prototype. The Search and Verification phase are less bandwidth consuming compared to the Setup phase. We estimate that one search query and search response amount to 5290 bytes[106] of consumed bandwidth, in the case of Query 1. On the other hand, if the data owner does not outsource the large dataset, to enable anyone to search for keywords, she will need to transmit the search indices for each querier. Based on the formula given in Definition 21, we compute that 6 queries is the threshold above which outsourcing the Enron dataset to the cloud is the best strategy to optimize the bandwidth consumption.

Concluding thoughts. The above experiments of our verifiable keyword search solution over the Enron dataset show that our proposal enables relatively fast conjunctive keyword search, even with large file collection. Since our tests were performed on a desktop machine, we can safely envision that on a real cloud server the performances for the search operation would yield better running times. Besides, in spite of an expensive Setup pre-processing phase, our protocol is a viable and practical solution by the fact that search results verifications are efficient, regardless of the file collection size.

6.8 Conclusion to Verifiable Keyword Search

We described in this chapter our publicly delegatable and publicly verifiable conjunctive keyword search solution. It is based on a combination of well-established cryptographic primitives: polynomial-based accumulators, Cuckoo hashing and Merkle trees. Besides, our scheme handles updates of files while still enabling verifiable search without the need for the data owner to download the database to perform the update operations. We proved that our proposal is correct and sound, according to the definitions we gave in Section 6.2.3. In addition, our verifiable keyword search solution satisfies the efficiency requirement for any VC scheme and adopts the amortized model approach.

Compared to relevant existing work on verifiable keyword search, we discuss the practicality of our proposal and its outsourceability. Besides, our scheme ensures public delegatability and verifiability while, in contrast, most existing work [129, 113, 59, 198, 62, 196] do not provide these two requirements simultaneously.

[106]We compute this value based on the communication complexity analyzed in Table 6.2. We assume a keyword in encoded on 4 bytes, the file identifiers on 3 bytes and $|G| = 160$ bits.

Part III

Accountability and Verifiability

Chapter 7

An Accountability Policy Language

7.1 Introduction

The two previous parts illustrated how cryptography supports *verifiability* in the cloud. We looked at cryptographic proofs of retrievability to verify that a cloud server stores outsourced data *as expected*. We also focused on three cryptographic protocols that generate proofs stating that the cloud evaluates the outsourced operation *as expected*. In this part, we turn our attention to how to express the statement "*as expected*" in more formal terms. In particular, we will extend our study on verifiability to the broader notion of *accountability*.

We regard **accountability** for the cloud as the delineation of data governance in which organizations that are entrusted with personal and confidential data are held responsible and liable for storing, processing and sharing data according to contractual and legal constraints, called **obligations**, from the time the data is stored until when the data is destroyed. The accountable organization must implement appropriate actions and handle remediation procedures in case of failure to act properly [146]. Obligations must respond to a collection of many regulations, including the European Union (EU) General Data Protection Regulation [80] and its reform[107]. In addition, cloud services are delivered in form of contracts and agreements. Such agreements may not explicitly address in which way obligations are handled, as they are often formulated by cloud providers and not cloud customers [45]. These customers lack control over the way their data is managed by the cloud, therefore, they grant a high level of trust on the cloud compared to the guarantees they finally obtain. The set of obligations corresponds to the intuitive concept of "*as expected*" we mentioned above. Obligations also help to clarify the accountability relationships in the cloud, *i.e.* who is responsible to whom and for what. Hence, defining appropriate policies mapping the accountability obligations is a fundamental requirement for control mechanisms, in the sense that policies mitigate risks, provided that their enforcement and the verification of their compliance are applicable.

In this setting, cryptographic techniques, such as the ones we develop in Part I and Part II, alone do not fully capture the essence of accountability. Indeed, as we will see, accountability encompasses multiple aspects such as notifications of security breaches, that cryptography does not enable to satisfy. Therefore, we look at another tool, namely a policy language, to express in terms of *policies* the accountability obligations that the cloud has to comply with during the correct execution of the services it delivers. The objective of our work in this part consists in analyzing how and to what extent we can convey accountability obligations via *expressive* and *declarative* policies in such a way that these policies are easy to write, manage, enforce and validate. We are interested in **machine-readable** representations of policies expressing accountability obligations. In order to express such policies, we design in this work a new policy language, Accountability-PPL (A-PPL), that is both *expressive* and

[107]At the time of writing this thesis, the Reform of the data protection legal framework in the EU is still in progress. "Reform of EU Data Protection Rules", European Commission, http://tiny.cc/neiu8x [Accessed: February 4, 2016].

declarative. The term *expressive* means that the language is easy to understand for both a human reader and a machine reader. In the case of a policy language, the machine reader is called a policy engine and enforces the policies written in that language. Our work also designs a new policy engine to enforce A-PPL statements: the A-PPL Engine (A-PPLE). The term *declarative* encompasses the concept that the language should describe *what to do*. We highlight the fact that at the time of writing this thesis our effort in designing a machine-readable policy language for accountability obligations is unprecedented in the state of the art.

This part of this thesis offers the following contributions:

- We present and analyze a collection of accountability obligations from which we derive the requirements for an accountability policy language.
- We detail the design of a new language called A-PPL[108] (shorthand for Accountability-PPL, where PPL [175] is an existing policy language devoted to privacy). We also describe A-PPLE[109], the policy engine that takes care of the enforcement of A-PPL rules.
- We finally illustrate the expressive and declarative properties of our accountability policy language by modeling a use case. We show how to translate into A-PPL policies the obligations extracted from this use case.

This work on a policy language for accountability has been undertaken within the European FP7 research project, the A4Cloud project[110], whose goal consists in increasing trust in cloud computing by developing methods and tools that enable to make cloud actors accountable for the security and privacy of the data held in the cloud and to give cloud users more control and transparency on the way the data is stored and processed by the cloud.

7.2 The Concepts of Accountability

As we mentioned in the introduction to this chapter, the concept of accountability for the cloud embraces the notion of verifiability in the cloud, which is the main topic of this thesis. To be more specific, accountability is a complex concept that can be modeled under a set of various elements. First, accountability includes a collection of principles, that we call *accountability attributes*. Based on the identification of these attributes by the A4Cloud project [58], we regroup them into a number of four: **responsibility**, **transparency**, **verifiability** and **remediability**. Moreover, from these attributes, the concept of accountability implies a certain number of *practices* or behaviors that an accountable cloud should engage. There are four such practices: delineating data governance, implementing appropriate actions such that the cloud is accountable, demonstrating compliance with policies and regulations and remedying any failure in the correct execution of the actions. Finally, the concept of accountability involves *mechanisms* that support accountability practices. Three types of mechanisms apply: operational processes (auditing), non-technical mechanisms (contracts, policies) and technical tools (privacy-enhancing technologies, proofs of retrievability, etc.).

In this section, we give more details on the above accountability model (Section 7.2.1). We also describe the different actors and their respective roles in a cloud computing environment (Section 7.2.2). We finally highlight the importance of policies in an accountable setting (Section 7.2.3).

7.2.1 Accountability Model

Modeling accountability requires a hybrid approach that combines attributes, practices and mechanisms. These three aspects are naturally interconnected to each other: while attributes

[108] A-PPL is pronounced “a people”.
[109] A-PPLE is pronounced “apple”.
[110] The Cloud Accountability project (A4Cloud), http://www.a4cloud.eu/ [Accessed: February 4, 2016].

characterize accountability at a conceptual level, accountability practices interpret at the operational level the attributes' essence whereas the accountability mechanisms implement those practices.

7.2.1.1 Accountability Attributes

Accountability attributes [82] support or are strongly related to the concept of accountability. We identified four key attributes that are interconnected to each other [82]:

Responsibility: This characterizes the state of undertaking some assigned actions to comply with a set of policies or rules.

Transparency: This principle requires a transparent system to provide visibility on the way this system delivers the services it provides. Transparency involves the provision of enough and easy-to-access information about the behavior of the system.

Verifiability: Verifiability, as we saw in Part I and Part II, refers to the provision of proof or evidence showing whether the system delivers the service it offers correctly. In other terms, verifiability suggests that it is possible to verify, to audit or to be convinced that the considered system complies (or not) with obligations, that are translated into rules of policies.

Remediability: This attribute characterizes an accountable system that is able to take corrective procedures as a result of failure to comply with obligations.

In the description of these accountability attributes, we pointed out three fundamental notions that capture the essence of what an accountable system is accountable for:

Obligations: They define the rules for data storing, processing and sharing and can be expressed in terms of policies that derive from law or contracts.

Behavior: This notion characterizes the operations (actions and reactions) handled by the system to fulfill the obligations.

Compliance: This concept involves the comparison of the system's behavior with the obligations in order to show the conformity of the behavior with the obligations.

Having defined these three notions, we can summarize the outcome of the characterization of accountability attributes and their interconnection for a cloud system: Regulations, contracts between cloud providers and customers or ethical norms impose *obligations* that the accountable system is *responsible* for *complying* with. The accountable system therefore takes some actions such that its *behavioral* operations are *transparent* and *verifiable.* If the *verification* of this *behavior* detects *non-compliance* with obligations, the accountable system is *responsible* to take *transparent* and *verifiable remedies* for any failure to act properly.

7.2.1.2 Accountability Practices

Accountability practices designate the operational behavior that should be adopted by an accountable system in order to apply the accountability attributes:

- In relation to the *responsibility* and *transparency* attributes: The cloud should define and inform how data is managed so as to comply with obligations that it commits to.
- In relation to the *responsibility*, *transparency* and *verifiability* attributes: The cloud should ensure the implementation of the appropriate operations (the accountability mechanisms) to comply with obligations and to demonstrate their compliance.
- In relation to the *responsibility*, *transparency* and *remediability* attributes: The cloud should respond to any failure to act as specified by the obligations. In particular, it should take the appropriate corrective measures to redress from these failures.

7.2.1.3 Accountability Mechanisms

Accountability mechanisms relate to the tools and techniques that implement and support the accountability practices and attributes we listed above. These mechanisms range from privacy and security mechanisms to risk assessment and audits, including notification techniques and policy definition. As we discussed in Part I and Part II, our proposals for proofs of retrievability and verifiable computation can fall into the mechanisms that support accountability: Obviously, our cryptographic solutions establish verifiability in a cloud system since they meet the definition we gave for verifiability. Nonetheless, as we specified in Section 7.1, cryptographic techniques alone do not suffice to support verifiability and thus accountability. Logs and notifications are two examples of tools that are related to the transparency attribute since they provide a certain amount of information that offers visibility of the way the cloud processes the data it handles. Moreover, auditing logs is another mechanism that enhances verifiability in an accountable system. Indeed, audits ensure a detective control on whether the cloud comply with advertised obligations. Besides, an incidence response tool may help to inform cloud users about a security breach that may occur during a normal execution of cloud procedures. All these examples show that there exist a wide range of techniques that create accountability for cloud services.

This thesis investigates another type of mechanism to support accountability: machine-readable policies. As we will see in Section 7.2.3, policies are at the core of our accountability model as they play a pivotal role between the three notions we highlighted above, namely obligations, behavior and compliance.

7.2.2 Accountability Actors and their Roles

Before delving into policies and their relevance in an accountable system, we delineate here the different actors that come into play in an accountable cloud *ecosystem*. This term refers to the complex system of components that interact with each other to deliver, consume, enjoy or monitor cloud services. Identifying the actors and their roles in an accountable ecosystem is essential to provide accountability. Besides, policies must reflect these actors and their interdependencies. Our analysis is based upon two different sources: the NIST cloud reference architecture [121] and the data protection regulation taxonomy [80].

The NIST reference architecture defines the following three main cloud actors:

Cloud Provider: This is an organization that delivers cloud computing services or technologies to interested parties. We typically called this party the *cloud server* in Part I and Part II.

Cloud Customer: This is a person or an organization that contracts a business relationship with a Cloud Provider. It consumes the services delivered by this Cloud Provider. This party may correspond to the *data owner* we introduced in Part I and Part II. This definition can also apply to the *querier* user mentioned in Part II since this entity uses the cloud resources to submit a computation request (Chapter 4, Chapter 5) or a search query (Chapter 6).

Cloud Auditor: This entity operates independent assessment of cloud services, their operations, their performance and their security.

Each of these actors endorses some roles defined by the EU regulation on data protection, roles that help characterize the actors in an accountable cloud ecosystem:

Data Subject: It refers to the individual from whom personal data is collected. Data Subjects are often the end users of a cloud service.

Data Controller: This term identifies "the natural or legal person, public authority, agency or any other body which alone or jointly with others determines the purposes and means of the processing of personal data" [80]. Organizations that purchase cloud services, namely Cloud Customers, are often Data Controllers. Besides, multinational Cloud Providers that offer cloud services (e.g. Amazon[111], Facebook[112] and Google[113]) can also become Data Controllers.

Data Processor: This expression includes "the natural or legal person, public authority, agency or any other body which processes personal data on behalf of the Data Controller. Cloud Providers will become Data Processors when their customers use their services to process personal data" [80].

Data Protection Authorities: They represent national supervisory authorities, such as the Information Commissioner's Office[114] (UK), the French CNIL[115], the German BFDI[116], etc.

Table 7.1 depicts the correspondence and the possible combinations between the cloud actors and the roles they endorse in an accountable cloud ecosystem. For the rest of this part, we may interchange cloud actors or their corresponding roles.

NIST cloud actors	**Data protection roles**
Cloud Customer	Data Subject Data Controller Data Processor
Cloud Provider	Data Controller Data Processor
Cloud Auditor	Data Protection Authorities

Table 7.1: Correspondence between Cloud Actors and Data Protection Roles

7.2.3 Accountability and Policies

The term *policy* refers to "a set of rules related to a particular purpose. A rule can be expressed as an obligation, an authorization, a permission or a prohibition" [105]. As we specified before, policies play a central role in our accountability model: Accountability policies are particularly useful for specifying *obligations* in cases where Cloud Customers (data owners) outsource the processing of personal data to Cloud Providers. Besides, Cloud Providers operate the data according to the *behavior* defined by the stated policies. Finally, *compliance* checking against the obligations can be done via comparing the policies with the behavior traces and evidences. More concretely, accountability policies convey the different accountability attributes of responsibility, transparency, verifiability and remediability that the Cloud Providers (Data Controllers and Data Processors) should satisfy. Therefore, defining and enforcing those policies is one of the accountability practices we mentioned above. A machine-readable policy language can implement these accountability policies. The goal of

[111]Amazon Web Services: https://aws.amazon.com [Accessed: February 4, 2016].
[112]Facebook: https://www.facebook.com/ [Accessed: February 4, 2016].
[113]Google: https://google.com/ [Accessed: February 4, 2016].
[114]Information Commissioner's Office: https://ico.org.uk/ [Accessed: February 4, 2016].
[115]Commission Nationale de l'Informatique et des Libertés: www.cnil.fr/ [Accessed: February 4, 2016].
[116]Bundesbeauftragte Für den Datenschutz und die Informationsfreiheit www.bfdi.bund.de [Accessed: February 4, 2016].

the present work is to design an accountability policy language that eases and automates the enforcement of accountability policies.

7.3 Motivating Scenario

This section describes a use case that illustrates accountability concerns in the cloud ecosystem. This scenario has been developed within the A4Cloud project [32, 34] in order to demonstrate how accountability mechanisms devised or used in the project can be applied to a real-case scenario which involves storing and processing data by different actors in a cloud ecosystem. The use case highlights the intrinsic obligations for accountability that these actors have to fulfill. Related to our work on the design of a policy language for accountability, the scenario enables us to determine the requirements that such a language should satisfy in order to convey the accountability obligations into the form of policies. After outlining the A4Cloud use case in this section[117], Section 7.4 investigates the obligations related to this scenario and derive from them the requirements for a policy language for accountability.

The use case defined by the A4Cloud project [34] develops a healthcare system, the "M" platform, depicted in Figure 7.1, used to support elderly people. The hospital, that considers these elderly people as its patients, adopts the "M" platform to collect medical data from sensors and analyzes them for diagnosis purposes via the service offered by this platform. The (wireless) sensors monitor patients' vital signs such as blood pressure, heart pulse rate or body temperature. This collected data is stored in a cloud-based storage service where it will also be processed. In addition, the sensor data is likely to be exchanged between the elderly people, their relatives, the hospital caregivers and healthcare personnel.

The "M" Platform is used by an hospital who subscribed to a service delivered by service provider M. This service provider made the choice to outsource the data collected by the sensors to a cloud-based storage service (Cloud X in Figure 7.1) which also takes care of some initial processing. Besides, a second cloud provider (Cloud Y in Figure 7.1) is responsible for backing up the sensor data. Cloud Z, which is provided by M's own infrastructure, is in charge of some processing and visualization of data. It communicates with Cloud X and Cloud Y. Figure 7.1 shows that the "M" platform interacts with different actors involved in the scenario through Graphical User Interfaces (GUIs).

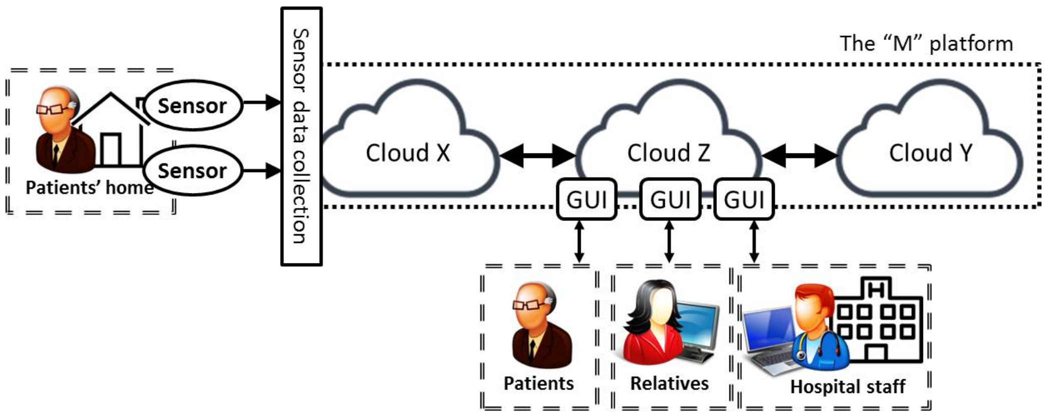


Figure 7.1: Healthcare Scenario [34]

We map to this scenario these actors with the roles we identified in Table 7.1 [33]:

• The patients are the Data Subjects. Indeed, the "M" platform collects the sensor data from them.

• The hospital is the Data Controller of the patients' data. It has selected the "M" platform delivered by service provider M to process this data.

[117]A comprehensive description and an in-depth analysis of the use case we outline here are presented in the A4Cloud's "Consolidated Use Case Report" [34].

- Service provider M, together with Cloud Z belonging to its own infrastructure, are Data Processors of the patients' data.
- Cloud Providers X and Y selected by service provider M to store and process patients' data are also Data Processors.
- Relatives, friends and hospital staff can be considered as Data Subjects, and under certain circumstances also Data Controllers (with respect to the patients' personal data)

7.4 Requirements for an Accountability Policy Language

From the motivating scenario depicted in Section 7.3, we identify the accountability obligations that the described system should endorse (Section 7.4.2). These results will generalize for any accountable cloud ecosystem. Then, from this analysis, we derive the design requirements for a policy language conveying, in a machine-readable manner, the accountability obligations (Section 7.4.4).

7.4.1 Source of Accountability Obligations

Accountability obligations generally derive from three perspectives: regulatory, contractual and ethical. First, regulations and in particular those related to data protection, such as the EU regulation for data protection [80], are a primary source of obligations. In particular, the EU regulation controls the processing of personal data and puts the responsibility for compliance with the rules it defines to the Data Controller. These rules give priority to transparency: the Data Subject has the right to know when, how and where her personal data is to be processed. Put in another way, it means that the Data Controller must inform the Data Subject on who processes the data, where the data is being stored, the purpose for which the data is collected and processed, with whom the data will be shared and any other information making the data processing transparent to the Data Subject.

The second source of obligations consists of contracts. In our setting, the term "contracts" includes SLAs, privacy policies and Terms of Use (ToU). An SLA is a contract, in natural language, binding a service provider (in the context of cloud computing, a Cloud Provider) and its customers. The agreement states the quality level of the provided services: it contains clauses on services' performance, availability, but also security; it defines the responsibilities of the service provider and the appropriate remedies applicable when some of the clauses are not satisfied. Privacy policies are a document in natural language generated by the service provider and stating how the service handles personal data from their customers. Many websites define and advertise privacy policies to their visitors. Such policies declare which personal data is gathered, explain how it is processed and stored and for which purpose, and disclose if personal data may be shared or sold to third parties. There is no explicit consent from the Data Subject concerning privacy policies: if a visitor accesses a web page with some privacy policies, she tacitly accepts them. Note that it is not clear whether privacy policies legally bind the service provider with their customers, unlike ToU. ToU are prescribed by the service provider and define what it can do with the data it collects. They also detail what the provider intends to do with respect to the security and privacy of data.

Finally, the ethical perspective completes the list of source of obligations. It gives the opportunity to the Data Controller to 'do the right thing' [146] with respect to data protection and accountability. In a nutshell, ethics define a set of guidelines and rules in order to fulfill the expectations of Data Subjects and Cloud Customers (such as consent, transparency, security, privacy, etc.)

7.4.2 Accountability Relationships and Obligations

Having identified the source of accountability obligations, we determine the accountability relationships that exist between the different actors of our motivating healthcare scenario [18] and derive the obligations, stemming from regulations, contracts and ethics, that have to be satisfied. Note that the results we outline here can be generalized to any cloud ecosystem.

Data Controller is accountable to Data Subject. We focus on the obligations that the hospital, which is, in our scenario, the Data Controller, has to fulfill with respect to the Data Subjects, namely the elderly patients. Hence, the hospital is accountable to the patients for:

The right to information: Data Subjects have the right to know that their personal data is processed and for which purpose.

Data quality: This term means that personal data must be, processed fairly and lawfully, collected for specified, explicit and legitimate purposes, and kept in a form which permits identification of the patients for no longer than is necessary for the purposes for which the data was collected or further processed [80].

Confidentiality: The hospital shall implement appropriate technical and organizational measures to ensure an appropriate level of security in relation to the risks represented by the processing and the nature of the personal data to be protected.

Data Processor is accountable to Data Controller. Service provider M, Cloud X and Cloud Y endorse obligations, such that they are made accountable to the hospital for:

Contractual obligations: This means that they are required to provide the service as specified in the contracts.

Confidentiality/Security control obligations/Data integrity: These notions relate to regulatory obligations of data security, breach, data loss and confidentiality.

Notification: This refers to the obligation for the Data Processors to collect and retain certain information that might be relevant to security breaches, but also to conduct a reasonable investigation of the security breach.

Data Location: The objects stored in a region must never leave the region unless the hospital transfers them out.

Data Controller is accountable to Data Protection Authorities. Here, we are interested in the accountability obligations that the hospital should meet to be accountable for the Data Protection Authorities[118]. The hospital is accountable to Data Protection Authorities for:

Notification on processing operations of personal data: The hospital must explain the context of the personal data processing and justify the purposes of the processing.

International data transfers (change of data location): Some international legal mechanisms frame personal data transfers across countries, for instance, Binding Corporate Rules[119]. The hospital is hence accountable for obtaining authorization from the Data Protection Authorities for international transfers.

The assignment of processing operations to data processors: The hospital is accountable to Data Protection Authorities for choosing those Data Processors (here, Cloud X and Cloud Y) that provide sufficient safeguards concerning the technical security and the organizational measures required in relation to the processing to be carried out on their behalf.

[118]These Data Protection Authorities are not represented in Figure 7.1, depicting our scenario.

[119]More details on Binding Corporate Rules are given in http://tiny.cc/vpiu8x [Accessed: February 4, 2016].

Besides, the above identified relationships highlight the needs for collecting evidence on the cloud service operations and implemented security controls. For instance, audits from Data Protection Authorities may require the collection of logs generated by the Data Processors (Cloud X and Cloud Y) and which record the actions performed by the Data Processors.

To further analyze the accountability relationships and obligations of the instantiated use case, we draw in the following a classification of the mechanisms into three categories capturing the level of controls these mechanisms suggest: preventive, detective and corrective controls:

Preventive. This type of control relate to the obligations that mitigate the risks of accountability breaches. Preventive controls include security and privacy obligations, access control obligations or other authorization controls with respect to data location, data retention periods or purpose of processing.

Detective. Detective controls are used to determine whether or not the system complies with the obligations. They are used to identify any security breach. Logging and monitoring techniques, audit mechanisms, generation of cryptographic proofs such as the one studied in Part I and Part II are part of detective controls.

Corrective. This category of control is used to fix and inform any accountability breach or any failure in satisfying the obligations. Corrective controls include in particular notifications and reports.

7.4.3 Accountability Obligations

From the source of accountability obligations and after describing the accountability relationships, we provide the following list of accountability obligations. Although we refer to our motivating scenario, we highlight the fact that these obligations remain general and relevant for any other business use cases. A more exhaustive list of obligations was presented in [34] within the A4Cloud project. Here we only give the prominent obligations that help us determine the requirements for the design of an accountability policy language.

Obligation 1: The right to access, correct and delete personal data. According to the General Data Protection Regulation [80], the hospital must ensure that the patients have read and write access to their personal data that have been collected and stored in the cloud. There must be also means to enforce the deletion of such data. Obligation 1 implies *preventive* controls that must be translated into privacy, access control and usage control policies.

Obligation 2: Duration and Purpose of processing. The hospital must make sure that the patients' personal data is only processed for specific, explicit and legitimate purposes. Here processing also includes the notion of storage. In this case, the purposes of processing are to help diagnosing and curing the patients. In addition, the hospital must guarantee that the patients' data is kept and processed for the specified purposes only for the duration imposed by regulations. Obligation 2 entails *preventive* mechanisms that authorize the collection, storage and processing of the specified purposes for the appropriate duration.

Obligation 3: Breach notification. In case of security or personal data breaches, cloud providers X and Y must notify M, which in turn must notify the hospital and the hospital must notify the patients. This notification should not exceed an appropriate period of time (for example, the notification should be sent within 24 hours after the detection of the breach[120]). Obligation 3 involves a kind of *corrective* control. As a matter of fact, it consists in a *reactive* control after a security breach is observed.

[120]Opinion 03/2014 on Personal Data Breach Notification: http://ec.europa.eu/justice/data-protection/article-29/documentation/opinion-recommendation/files/2014/wp213_en.pdf.

Obligation 4: Evidence of the correct and timely deletion of personal data. Cloud providers X and Y must be able to provide evidence to the platform provider M, and M must be able to provide evidence to the hospital on the correct and timely deletion of personal data. These evidences serve for auditing purposes. Indeed, a Data Protection Authority may audit Data Controllers on the way the data is hold and the manner in which they are processed. Obligation 4 covers *detective* mechanisms to observe security breaches as well as compliance with other obligations.

Obligation 5: Location of processing. Cloud providers X and Y, as well as service provider M have contractual obligations towards their respective customers on the location of the data processing. Obligation 5 implies *preventive* control on geographical location of this processing.

7.4.4 Policy Language Requirements

Machine-readable accountability policies are particularly useful for conveying the above obligations. A policy language allows to represent such policies in a machine-readable format. Our goal is to design an accountability policy language that eases and automates the enforcement of these policies. Therefore we derive from the analysis of the accountability obligations several design requirements for our proposed policy language. We classify our requirements into two categories: requirements for data handling policies and specific requirements for accountability. The former refers to the need to express privacy constraints, access and usage control rules. The latter corresponds to the requirements that are specific to accountability and which are often not addressed by existing policy languages such as audits, logging and notifications. Table 7.2 summarizes our analysis of accountability obligations concerning personal data processing.

Requirement	**Category**
(R1) Capturing Privacy Policies	Data Handling
(R2) Access Control Rules	Data Handling
(R3) Usage Control Rules	Data Handling
(R4) Data Retention Period	Data Handling
(R5) Reporting and Notification	Accountability
(R6) Controlling Data Location	Accountability
(R7) Auditability	Accountability
(R8) Logging	Accountability

Table 7.2: Accountability policy language requirements.

- **Data handling requirements:**

(R1) Capturing privacy policies: An accountability policy language must allow the expression of privacy policies about the usage of personal data. This requirement is related to Obligations 1,2 and 3.

(R2) Access Control Rules: In relation with Obligation 1, the accountability policy language must enable the specification of access control policies to personal data. The access requester should in particular be defined by a set of attributes such as its name, its role, or the group it belongs to.

(R3) Usage Control Rules: Obligations 1 and 2 suggest the definition of appropriate usage control rules. The accountability policy language must allow the expression of such rules. In particular, it should express the conditions under which an action on the data is permitted or prohibited (such as sharing the data with third parties, usage for a particular purpose). It should also define the operations on the data that has to be performed after its collection (such as deletion, anonymization, etc.).

(R4) Data Retention Period: In accordance with Obligation 2, the accountability language must be able to express rules about data retention such as retention time.

- **Accountability requirements:**

(R5) Reporting and Notification: The policy language we design should enable the sending of notifications to Data Subjects and third parties. This requirement addresses Obligation 3.

(R6) Controlling Data Location: As controlling data location is required by Obligation 5, the language must enable the expression of rules about data location in a policy.

(R7) Auditability: Obligation 4 states that accountable services may be audited to verify compliance with obligations. Therefore, the accountability policy language must make possible the auditing of operations performed in the cloud (such as deletion, transfer, modification, access, etc.). The language must also specify what information is targeted by an audit, and which evidence should be collected to perform the audit.

(R8) Logging: Evidence collection is ruled by Obligation 4. Logs can be a particular type of evidence. Therefore, the policy language must specify which events have to be logged and what information related to the logged event have to be added in the log.

One may argue that these requirements can be expressed and enforced using multiple existing languages. We advocate that centralizing these concerns in a single policy expressed in a unique policy language will increase the accountability of the actors processing personal data in the cloud. In the next section, we use these requirements to review and analyze existing policy languages so as to design a suitable accountability policy language.

7.5 State of the Art on Policy Languages

7.5.1 Methodology

A number of policy languages have been proposed in recent years for policy representation. We reviewed several existing policy languages, defined either as standards (XACML [134], WS-* standards [135, 136] and P3P [123]) or as academic/industrial proposals (PPL [8], USDL [25], SLAng [117], SecPal4P [26], Ponder [133] and ConSpec [4]). We studied their suitability to accountability representation according to the accountability requirements identified in Section 7.4.4. In this review, the main question was to determine the ability of the existing policy languages to represent accountability obligations defined in Section 7.4.2. Rather than imposing a brand new language, we consider to select and extend one or several existing languages with accountability-aware features in order to map accountability requirements as much as possible.

7.5.2 Survey of Existing Languages against the Language Requirements

We present here the outcome of this survey. The reader can find the detailed description of this analysis of exiting languages in the A4Cloud report "Policy Representation Framework" [63].

Although most of the reviewed languages fail at meeting all the accountability requirements at first glance, some of them can be extended to support accountability policies. As a result, we elect PrimeLife Policy Language (PPL) as the policy language that captures the best the accountability requirements. We therefore build our proposed accountability policy language upon PPL. This choice is motivated by the following four reasons:

1. PPL is the language that covers the more language requirements;
2. PPL presents many extension points making the language easy to extend for accountability;
3. PPL is based on eXtensible Access Control Markup Language (XACML) which is a standard policy language for access control;
4. PPL is well-documented as in the work done by Ardagna *et al.* [8].

Before presenting our proposal for an accountability policy language, the next section provides the reader with more details on PPL. Besides, since PPL is based on XACML, we also outline the main characteristics of this language.

7.5.2.1 XACML

XACML [134] stands for eXtensible Access Control Markup Language and consists of an OASIS standard that defines both a declarative language for expressing access control policies and a request-response message exchange protocol to obtain access control decisions. The XACML request-response message exchange is used to express queries to a decision engine about whether an action should be allowed (request) and describes the respective answers (response). The language is expressed in eXtensible Markup Language (XML) [47] that is both human and machine-readable.

A set of rules is encapsulated in a `Policy`. The main components of a `Rule` are:

- A Target that defines the requests to which the rule is intended to apply. It is expressed in terms of subjects, resources and actions.
- An Effect that indicates the rule-writer's intended consequence if the evaluation of the rule outputs true. The possible effects are either `Permit` or `Deny`.
- A Condition that refines the applicability of the rule by putting restrictions on `Target`'s attributes (subjects, resources or actions).

The structure of an XACML request comprises one or more attributes that specify: (i) the entity making the access request (a subject), (ii) the resource to which the subject(s) has requested access, identified by its URI, and (iii) the action that the subject(s) wishes to take on the resource (read, write, etc.).

General usage scenario. XACML provides a standard reference architecture to achieve the enforcement of XACML policies. Figure 7.2 depicts a simplified version of the XACML policy engine, responsible for this enforcement. The Policy Enforcement Point (PEP) forms a request based on the attributes of the requester, the resource and the action on that resource that the requester wants to perform. The PEP then sends this request to the Policy Decision Point (PDP). The PDP evaluates the requests and finds policies that apply to that request, from the Policy Administration Point (PAP). As a result of evaluating the policy, the PDP sends a response context that specifies the access decision taken `Permit` or `Deny`. PAP maintains and stores the policies.

XACML provides a standard to express access control policies. An advantage of this language is that it presents many extensible points that can serve to express accountability requirements. However, the definition of XACML obligations lacks support for privacy and usage control obligations.

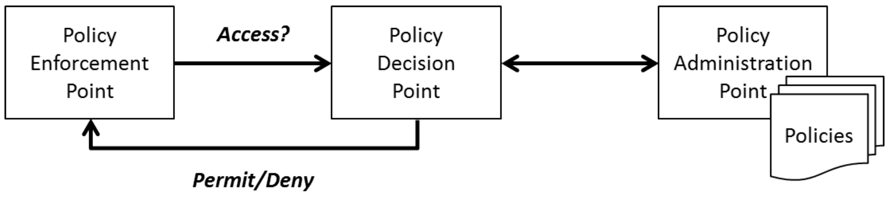


Figure 7.2: XACML reference architecture (simplified) [169]

7.5.2.2 The PrimeLife Policy Language

The PrimeLife Policy Language (PPL) [8] is an XML-based policy language proposed by the PrimeLife project [152] that extends XACML. The language combines access and data handling policies. The goal of PPL is to make service customers aware of the conditions under which their data are handled. Therefore PPL gives service providers automatic means of defining and managing privacy policies while applications are enabled to compare these service privacy policies with user privacy preferences.

Terminology. The language is symmetric: a similar syntax is used to express privacy policies and preferences. A PPL policy can be defined by a service provider to specify its privacy policies. In particular, the service provider defines in the policy how the collected data will be handled by the Data Controller and the entities the data is shared with. PPL makes it possible to automatically match these privacy policies with Data Subjects' privacy preferences. The outcome of the matching procedure generates a sticky policy or an annotated sticky policy that points out the difference between a user's preference and a controller's policy. This sticky policy is bound to the data and travels with the data. The sticky policy specifies statements on:

- **access control**, which is inherited from XACML that PPL extends with privacy-friendly credential-based features. Conditions on access control describe how access to which resource under which condition can be granted.
- **authorizations**, that detail actions that the Data Controller is allowed to perform with respect to the purpose of usage of collected data. In addition, authorizations enable to define the conditions of what in PPL specification [8] is called *downstream usage*[121]. This kind of authorizations are applicable for other Data Controllers, the downstream usage becomes the sticky policy for the data as it goes downstream.
- **obligations**, that the PPL specification [8] defines as "a promise made by a Data Controller to a Data Subject in relation to the handling of his/her personal data. The Data Controller is expected to fulfill the promise by executing and/or preventing a specific action after a particular event, e.g. time, and optionally under certain conditions". In PPL, an obligation is expressed using the pair Trigger-Action. Triggers are events related to an obligation and filtered by conditions. For example, PPL defines the trigger `TriggerPersonalDataDeleted` that occurs whenever the personal data related to the obligation is deleted. **Triggers** fire **Actions** that are performed by the Data Controller. For instance, PPL provides the action `ActionNotifyDataSubject`. The complete list of available PPL Triggers and Actions can be found in Table 7.3.

PPL extends XACML with privacy-enhancing, credential-based and usage control features. The structure of XACML is preserved and PPL introduces new elements in XACML in order to enable the description of privacy policies. Such elements are the `DataHandlingPolicy` element that enables the Data Controller to express how the data received from the Data

[121]Downstream usage specifies under which conditions data can be shared with other Data Controllers.

Name	Description
Triggers	
TriggerAtTime	Occurs based on a particular defined time
TriggerPeriodic	Occurs repeatedly according to a well-specified period
TriggerPersonalData-AccessedForPurpose	Occurs each time the personal data bound to the obligation is accessed of one of the defined purposes
TriggerPersonalDataDeleted	Occurs when the personal data associated with the obligation is deleted
TriggerPersonalDataSent	Occurs when the personal data akin to the obligation is forwarded to a third-party
TriggerDataSubjectAccess	Occurs when the data subject requests access to ts own personal data collected by the data controller
Actions	
ActionDeletePersonalData	Deletes a piece of personal data (data retention)
ActionAnonymizePersonalData	Anonymizes a particular piece of data
ActionNotifyDataSubject	Notifies the data subject when triggered, that is, send the information concerning the event that triggers the obligation to the data subject
ActionLog	Logs an event, that is, writes in a log file the information concerning the event that triggers the obligation
ActionSecureLog	Logs an event and ensures integrity and authentication of origin of the event

Table 7.3: List of Triggers and Actions in PPL

Subject will be treated, the `DataHandlingPreferences` element that enables the Data Subject to specify how its data has to be treated by the Data Controller, `Obligation` that specifies which specific actions to execute when given events occur (triggers) and `Authorization` that specifies the actions that are allowed to be performed. Figure 7.3 presents the structure of a PPL policy.

PPL presents a generic and user-extensible structure for authorizations and obligations which are more specific than what is defined in XACML. Indeed, the users are enabled to specify and add their own authorization and obligation vocabularies. Furthermore, the Obligation Enforcement Engine [5] that makes sure that committed obligations as part of a sticky policy are indeed enforced may be extended with audit features to check compliance with policies.
Although the adoption of PPL is still limited today, since it has not been developed until recently (2011), PPL presents features that capture most of the identified requirements (see Section 7.4). Besides, PPL can be extended to provide further components that can address even more specifically those requirements.

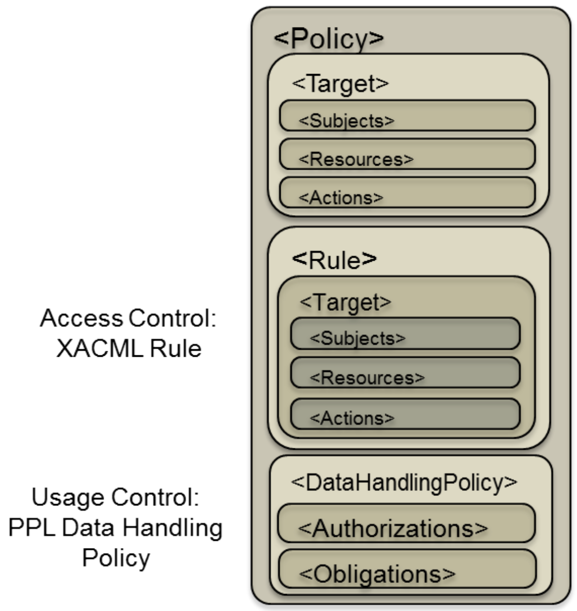


Figure 7.3: General structure of a PPL Policy

Limitations of PPL. PPL provides elements to declare some of the accountability specific obligations such as notifications (R5) and logging (R8). However, these elements need more specification and they may be unpractical when directly used within an accountability policy. For example, in the current version of PPL, the Notify element only allows the Data Controller to notify the Data Subject. In accountability scenarios, notifications are not exclusive to the Data Subject. Instead, notifications may be sent all along the accountability chain to notify the actors within the chain of an occurred event. As far as logging is concerned, the current specification of PPL allows to declare the action to log, but we cannot specify what information has to be put in the log. Moreover, there is no way in PPL to specify the location of the data (R6). Besides, auditing (R7) is not part of the PPL language since the focus of the PrimeLife project was on privacy and not accountability. In the language specification, the way policy violations and security breaches are detected and handled remains unclear. Our policy language aims at providing a way to declare the conditions under which an audit is required, which are not provided in PPL. An audit may require the provision of evidence to enable the verification of compliance with policies, data subjects' preferences, contracts or regulations. Evidence appears then to be an accountability object of paramount importance. As audits are not part of PPL, this language fails to capture the concept of an auditor that intrinsically plays a relevant role in an accountable cloud environment. This auditor is responsible of specific tasks, such as requesting evidence or notifying actors in the accountability chain for policy violation.

In Section 7.6, we present an enhanced version of PPL with extensions that address the accountability requirements that we identified in Section 7.4.4. We call this accountability policy language **A-PPL**, for Accountability-PPL.

7.5.2.3 Related Work on PPL extension

Recent related work [48, 103, 72] support our choice to extend PPL with some of the accountability features we identified in Section 7.4.4. However, these works do not propose a complete accountability policy language. They rather propose some extensions to the PPL language to address the requirements of logging, location and duration of storage that the authors of [48, 103, 72] have identified as being keys for accountability.

Concomitantly to our work, Butin et al. [48] leverage PPL to design logs for accountability.

They develop two case studies (a private bank account and a hospital handling personal data) and they show the limitations of PPL with respect to logs. In particular, similarly to our PPL analysis, they identify the lack of expressiveness of PPL logging action. Indeed, the `ActionLog` element does not provide sufficient information in the logs. Besides, they discuss the fact that the PPL element `ActionNotifyDataSubject` does not allow to specify the content of the notification. In addition, the authors of [48] suggest that the PPL obligations should express the causal relationship between Triggers and Actions, *i.e.* the obligations should make explicit which Triggers fired which Actions. Therefore, the authors propose to add a `TriggerId` attribute in the Trigger elements and a `TriggerReference` attribute in the Action elements. Finally, the authors recommend to develop a more fine-grained downstream usage specification, by allowing to define a list of authorized or unauthorized third parties to whom the personal data can be forwarded.

Similarly, Henze et al. [103] focus on the goal to make cloud computing aware of data handling requirements. They identify location and duration of storage as the two main challenges in cloud data handling scenarios. They propose to create *data annotations* that contain the data handling obligations (e.g "delete after 30 days"). These annotations are transmitted to the cloud service before the annotated data in order to match the corresponding obligation against the cloud service's data handling policies. If the annotations match the data handling policies, the cloud service signs the annotation and sends it back to the user, who now has a proof that the cloud will process the data as stated in the annotation. The authors in [103] suggest to leverage the PPL language to formalize the data annotation. Without giving more details, they propose to address the obligation of duration of storage by introducing maximum and minimum duration of storage attributes. They also define an extension to PPL with an element that restricts the location of stored data. On the other hand, sending annotations can impact data privacy, since the annotations leak potentially privacy-sensitive information to the cloud service.

A recent research work by Di Cerbo *et al.* [72] extends PPL for imposing restrictions to cloud providers on the location of storage. The authors propose a new PPL element, `ActionStorageLocation`, to make explicit the obligation to store outsourced data only in data centers that are located in countries specified in the policy. This element allows to define a set of attributes corresponding to the countries where storage is permitted.

7.6 A-PPL: a Policy Language for Accountability

The newly proposed A-PPL maintains the overall policy structure of PPL. Figure 7.4 shows the structure of an A-PPL policy, that derives from the XACML structure and highlights the new extensions provided by A-PPL. To support the accountability features that A-PPL has to provide, we introduce new elements and we extend the PPL engine.

7.6.1 Roles

PPL implicitly identifies the Data Subject and Data Controller roles. To make the identification of roles more explicit in an accountable cloud, we include in a policy a reference to the role of the different entities involved in the policy. These roles are those identified in Section 7.2.2. Thus, we create a role attribute identifier `subject:role` to be included as an attribute of the standard XACML element `<Subject>`. In addition, as the auditor plays an important role in accountability in order to conduct independent assessments of cloud services, this role has to be interpreted in terms of policies. We propose to define the role of the auditor in A-PPL. This new role is useful for accountability specific obligations such as reporting and notification (R5) or auditability (R7).

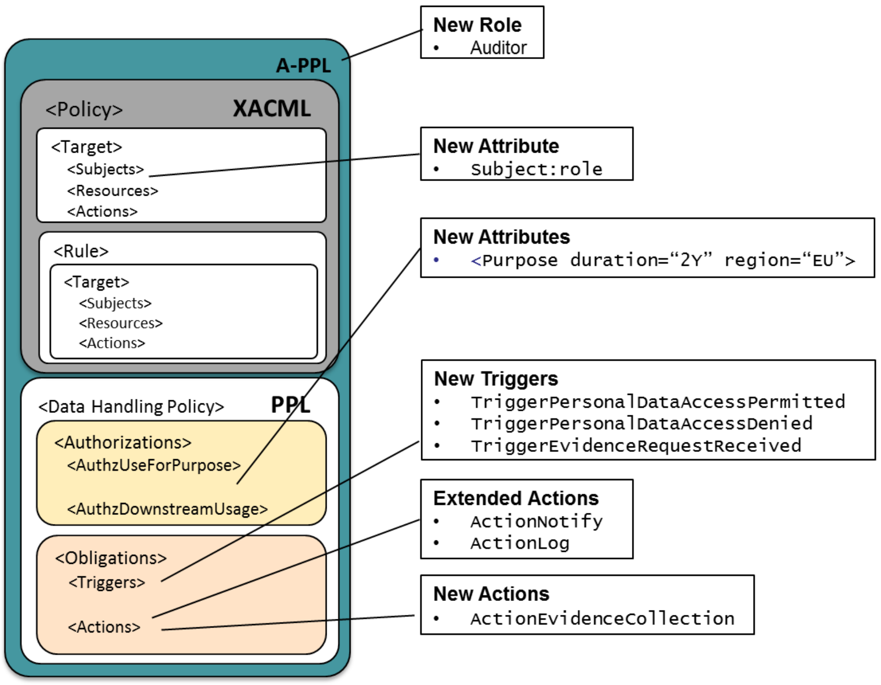


Figure 7.4: Structure of an A-PPL Policy

7.6.2 Capturing Privacy Policies (R1)

As the purpose of PPL was to define privacy policies, A-PPL inherits the privacy-related language elements from PPL.

7.6.3 Access Control Rules (R2)

We introduce two new triggers which condition the execution of an obligation based on the result of an access decision. In other words, we propose `TriggerPersonalDataAccess-Permitted` and `TriggerPersonalDataAccessDenied` that occur when the evaluation of the access control on the targeted data results in “Permit”, respectively “Deny”.

7.6.4 Usage Control Rules (R3)

PPL already defines a set of Triggers and Actions for the purpose of usage control. Therefore, A-PPL is granted with the same elements.

7.6.5 Data Retention (R4)

PPL provides an element `Purpose` that allows to specify for which purpose a piece of data can be collected or accessed. In A-PPL, we define the `duration` attribute for `Purpose` that allows to specify for how long the data can be processed for a particular purpose. For instance, a particular piece of data is used for research purposes for 2 years but has to be kept for legal purposes for 5 years. In addition, this attribute implies that when all durations for each purpose have expired, the data has to be deleted, since the data cannot be used for any purpose anymore.

7.6.6 Reporting and Notification (R5)

PPL’s `ActionNotifyDataSubject` element enables the data controller to send notifications to the data subject only. Our language provides a more fine-grained notification action in order to satisfy the requirement (R5). We modify the existing PPL `ActionNotifyData-Subject` element and call the newly created notify action `ActionNotify`. Notifications are not limited to notifications to the data subject only. Instead, we provide an attribute `recipient` that allows to indicate the recipient of the notification. The `ActionNotify` element presents an attribute `type` that specifies the type of notification to be sent to the recipient (policy violation report, audit reports, etc.). Table 7.4 describes the `ActionNotify` element.

Table 7.4: ActionNotify element.

Name	**ActionNotify**	
Description	This action notifies a cloud actor when triggered	
Parameters	Media	The media used to notify the user (e-mail, SMS, etc.)
	Address	The corresponding address (e-mail address, phone number, etc.)
	Recipient	The identity of the recipient of the notification
	Type	The type of notification(policy violation, evidence, redress, etc.)

7.6.7 Controlling Data Location (R6)

We propose in A-PPL a standard identifier `region` to express the location of collected data. This attribute should be used as an attribute of the A-PPL `Purpose` element that is nested inside a `<AuthzUseForPurpose>` environment. Thus we will limit the region among which the data can be transferred without violating the policy access control rules. This is directly responding to our requirement on controlling data location in the policy language (R6).

7.6.8 Auditability (R7)

Auditing plays an important role in accountability and evidence is key in the auditing processes. We identify several types of evidence such as logs or cryptographic proofs such as the ones we presented in Part I and Part II for verifiability in storage and computation. An audit protocol may involve two parties: an auditor (such as a Data Protection Authority) and an auditee (e.g a Data Controller, a Data Processor, etc.). The auditee is responsible for collecting evidence about his processing practices. We propose two extensions that relate to audits and collection of evidence. Based on the evidence request that the auditee receives from the auditor, the auditee collects the requested evidence. This evidence collection is governed by a new A-PPL trigger `TriggerOnEvidenceRequestReceived`, and a new A-PPL action `ActionEvidenceCollection`. The combination of the two A-PPL elements initiates the evidence collection by the Data Controller. Table 7.5 describes the `ActionEvidenceCollection` element.

Table 7.5: ActionEvidenceCollection element.

Name	**ActionEvidenceCollection**	
Description	This action collects the requested evidence	
Parameters	Evidence	The type of evidence to generate (logs, crypto proofs, etc)
	Resource	The ID of the resource the evidence is based on
	Subject	The ID of the data subject the evidence is based on
	Recipient	The ID of the recipient of the evidence (the auditor)

7.6.9 Logging (R8)

PPL's `ActionLog` action element fails to capture the concept of logging detailed in Section 7.4.2. The logging action should provide a way to specify not only the details of the event that has to be logged but also the security properties of the logs (integrity or confidentiality of the logs for instance). Therefore, we extend the `ActionLog` element in A-PPL.

In particular, we introduce several parameters to make explicit which information about an event needs to be logged. A timestamp is required to log the time of the event. The policy must indicate to log the action that is performed on the data (e.g. `SEND`), the identity of the subject who performed the action (e.g. `Cloud x`) and the purpose of the action (e.g. `marketing`). To trace events based on data, the policy must require the identifier of the data. Other details must also be written in the logs such as some security flags that may state whether the log entry is encrypted or not. Table 7.6 describes the `ActionLog` element.

Table 7.6: ActionLog element.

Name	**ActionLog**	
Description	This action logs an event based on the details in the policy	
Parameters	Timestamp	The time of occurrence of the logged event
	Action	The action that is logged
	Purpose	The purpose of the action that is logged
	Subject ID	The identity of the subject that performed the action
	Resource ID	The identifier of the resource the action was made on
	Resource Location	The location of the resource
	Security Flag	1 if the log is confidential, 2 for integrity check, 3 for both

Table 7.7 compiles the new Triggers and new or extended Actions we propose in A-PPL.

7.7 A-PPLE: a policy engine for A-PPL

Policy enforcement is the task of a policy engine. In the case of policies written in A-PPL, A-PPLE handles their interpretation and automatic enforcement.

7.7.1 Description of A-PPLE

A privacy policy engine supporting PPL policies was originally designed in the PrimeLife project [175]. We adapt its architecture to implement the new obligations of accountability, creating the architecture depicted in Figure 7.5.

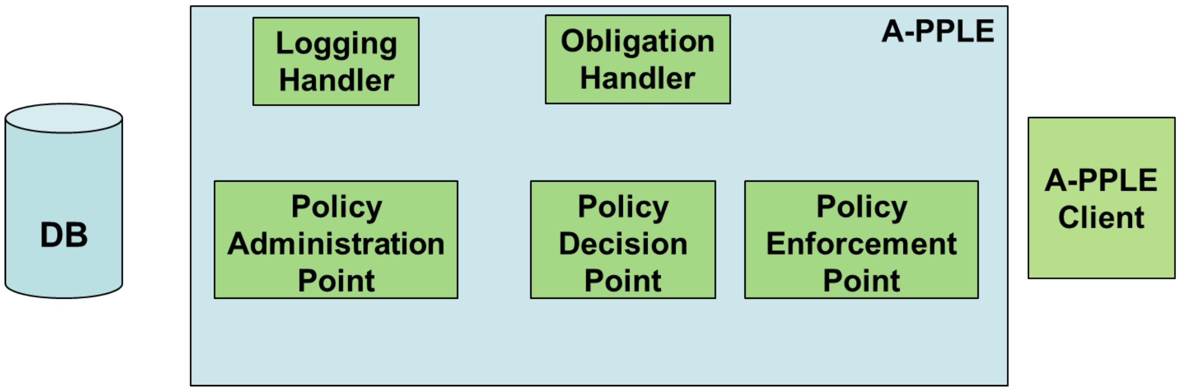


Figure 7.5: A-PPLE architecture

The core elements of the policy engine are:

PEP: It acts as an orchestrator of the enforcement process and it constitutes the entry point of A-PPLE at the Data Controller side. The PEP coordinates the Obligation Handler module that we present below.

Name	Description
Triggers	
TriggerPersonalDataAccessPermitted	Occurs when access to a queried data is permitted
TriggerPersonalDataAccessDenied	Occurs when access to a queried data is denied
TriggerEvidenceRequestReceived	Occurs when the Data Controller receives a request for evidence generation
Actions	
ActionNotify	Notifies the recipient with the information concerning the event that triggers this action
ActionLog	Logs an event, that is, writes in a log file the detailed information concerning the event that triggers the obligation
ActionEvidenceCollection	Initiates the collection (or generation) of requested evidence

Table 7.7: List of extended or new Triggers and Actions in A-PPL.

PDP: This is the component where the access control decision is taken. PDP relies on the access control engine implementation based on HERAS-AF[122] for the evaluation of XACML part of an A-PPL policy. Apart from the standard attribute-based access control, the other information evaluated by the PDP is usage authorization. The usage authorization basically consists of the comparison of the list of purposes specified in the access control request with the one specified by the Data Controller in the policy.

PAP: The Policy Administration Point is responsible for storing and deleting data and policies from the database.

Obligation Handler: It analyzes the Triggers and Actions which an A-PPL policy consists of. The obligations represented by this policy are related either to timely scheduled events or to events associated with the data life cycle, e.g. retrieval or deletion. This component keeps track of these events to generate Triggers that are part of the obligation statements in the A-PPL policy. In turn, Triggers fire the Actions associated with them. The Obligation Handler is invoked by the PEP and interfaces with the database.

Logging Handler: This module is responsible for creating and storing logs in the database. The Logging Handler is called to execute an `ActionLog` that is defined in the policy. In this case it includes extra information in the message such as the subject who requested a data, the action and purpose of the access, the location of data and the expiration date of the log. It interfaces with the Obligation Handler.

A-PPLE Client: This module represents an A-PPLE client in order to interact with A-PPLE.

Database: This component provides storage for A-PPL policies and personal data.

[122] Holistic Enterprise-Ready Application Security Architecture Framework (HERAS-AF) is an open source implementation of XACML: http://www.herasaf.org/.

7.7.2 Operations of A-PPLE

To illustrate the operational behavior of A-PPLE we consider the case where a Cloud Customer requests the Data Controller to delete her data. The Cloud Customer runs an A-PPLE client module whereas the Data Controller hosts on its side the entire engine. The sequence of operations of this delete request is showed in Figure 7.6.

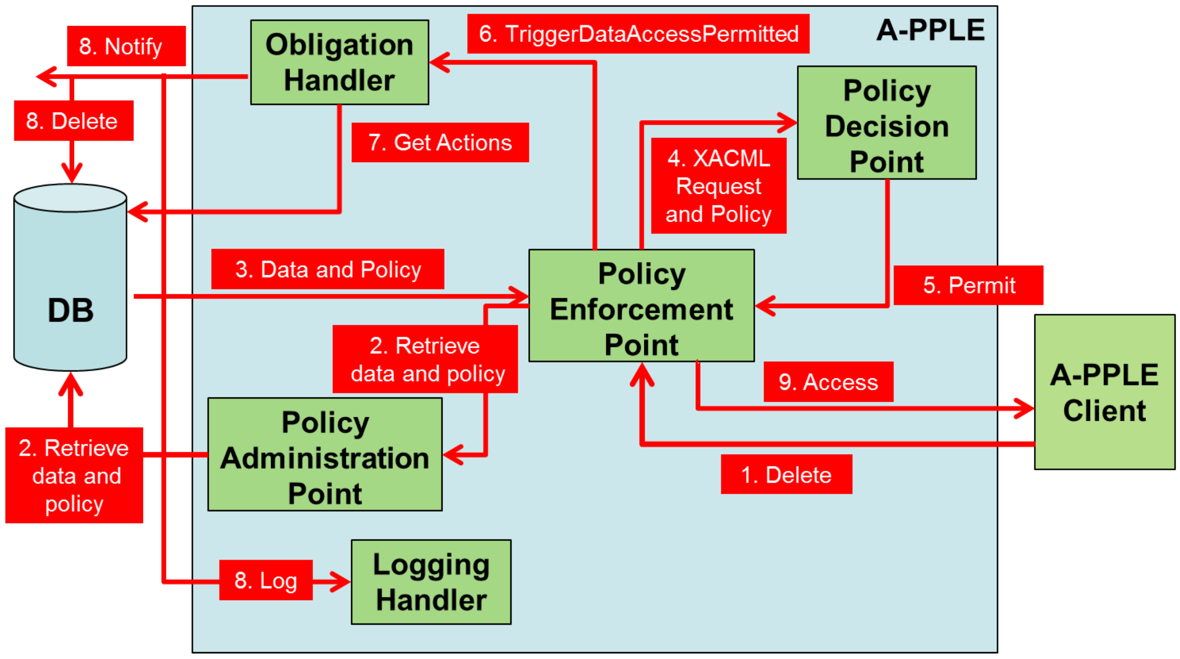


Figure 7.6: Sequence of operations for a Delete request.

1. The Cloud Customer, through the A-PPLE client interface, generates a Delete request for some of her data. This request is transmitted to the Data Controller.

2. At the Data Controller side, the PEP receives the Delete request and asks the PAP to retrieve the targeted data along with its policies. In turn, the PAP generates a database query for this data and policy.

3. The database replies to this query and sends back the data and the policy to the PEP.

4. The PEP forms an XACML request[123] that is forwarded to the PDP along with the policy related to the targeted data.

5. This XACML request is evaluated by the PDP to determine whether the Cloud Customer is authorized to access and delete the targeted data.

6. In this case, the PDP acknowledges that the Cloud Customer is authorized to access and delete the data. Therefore, the PDP returns to the PEP the authorization message “Permit”.

7. Then the PEP generates a `TriggerDataAccessPermitted` that is consumed by the Obligation Handler.

[123]We recall the reader that A-PPL is based on PPL which in turn is based on XACML. Therefore, the XACML request-response messages exchange protocol are part of the A-PPL specification. The XACML request is an XML-based message that contains the several attributes (subject, resource, action) related to the considered access request. The XACML request is compared to the policy by the PDP to grant or deny access to the targeted resource.

8. The Obligation Handler queries the database for the Actions related to that `Trigger`. Here, we suppose that `TriggerDataAccessPermitted` fires three types of Actions: Log, Notify and, naturally, Delete, which was the action requested by the Cloud Customer.

9. Once, the Obligation Handler receives the Actions from the Database, it executes them. Namely, it deletes the targeted data, notifies the Cloud Customer of this deletion and requests the Logging Handler to log information on the deletion and the notification.

10. Finally, the PEP informs the Cloud Customer on the fact that her delete access request is granted.

7.7.3 Integration of our StealthGuard Prototype in A-PPLE

In relation with the work we presented in Part I on Proofs of Retrievability (POR), we propose in this section to integrate the prototype for $\mathcal{S}$tealthGuard in the implementation of the policy engine. Indeed, the provision of POR is regarded as a particular obligation with respect to data storage: the Cloud Provider must provide evidence that it correctly stores the data of the Data Owner (this obligation can be related with Obligation 4 in Section 7.4.3).

Chapter 2 presented an implementation of $\mathcal{S}$tealthGuard. In the perspective of the present chapter, $\mathcal{S}$tealthGuard is considered as an automated mechanism to provide the evidence of correct storage and thus to enforce the obligation on evidence provision. We can therefore leverage the two A-PPL language elements we introduced in Section 7.6, namely `TriggerEvidenceRequestReceived` and `ActionEvidenceCollection`.

Practically, the A-PPLE Client module shown in Figure 7.5 embeds the operations of a $\mathcal{S}$tealthGuard's data owner and verifier, that is, this module is enhanced with algorithms Keygen, Encode, $\mathsf{Challenge}$ and Verify of $\mathcal{S}$tealthGuard (see Section 2.2.4). On the other hand, the Obligation Handler module is in charge of executing the watchdog search involved in algorithm $\mathsf{ProofGen}$. In Figure 7.7, we depict the sequence diagram of an execution of $\mathcal{S}$tealthGuard within A-PPLE.

1. When a Data Owner wishes to upload a piece of file, she runs the A-PPLE Client module which pre-processes the file according to the underlying algorithm Encode of $\mathcal{S}$tealthGuard, that we described in Figure 2.3. Namely, it applies an ECC to the file, encrypts it and inserts the watchdogs in random positions in the data.

2. When this operation is complete, the A-PPLE Client module uploads the retrievable version of the file to the Cloud Provider who runs an instance of A-PPLE.

3. The PEP receives the file, forwards the upload request to the PAP who stores it within the database.

4. Later on, a $\mathcal{S}$tealthGuard's verifier runs the A-PPLE Client module to audit the Cloud Provider on whether it correctly stores the outsourced file. This module generates a POR query that is sent to the Cloud Provider. We recall that this POR query consists in a privacy-preserving search query for some watchdogs inserted in the data.

5. At the cloud side, the PEP receives the POR query and initiates a `TriggerEvidence-RequestReceived` that is transmitted to the Obligation Handler module.

6. The Obligation Handler receives the trigger and fires the A-PPL action `Action-EvidenceCollection`. In a nutshell, this action executes algorithm $\mathsf{ProofGen}$ (see Figure 2.5) which performs the privacy-preserving search for the targeted watchdogs in the POR query.

7. The output of the search is returned as a POR response to the PEP which forwards it to the A-PPLE Client module of the verifier.

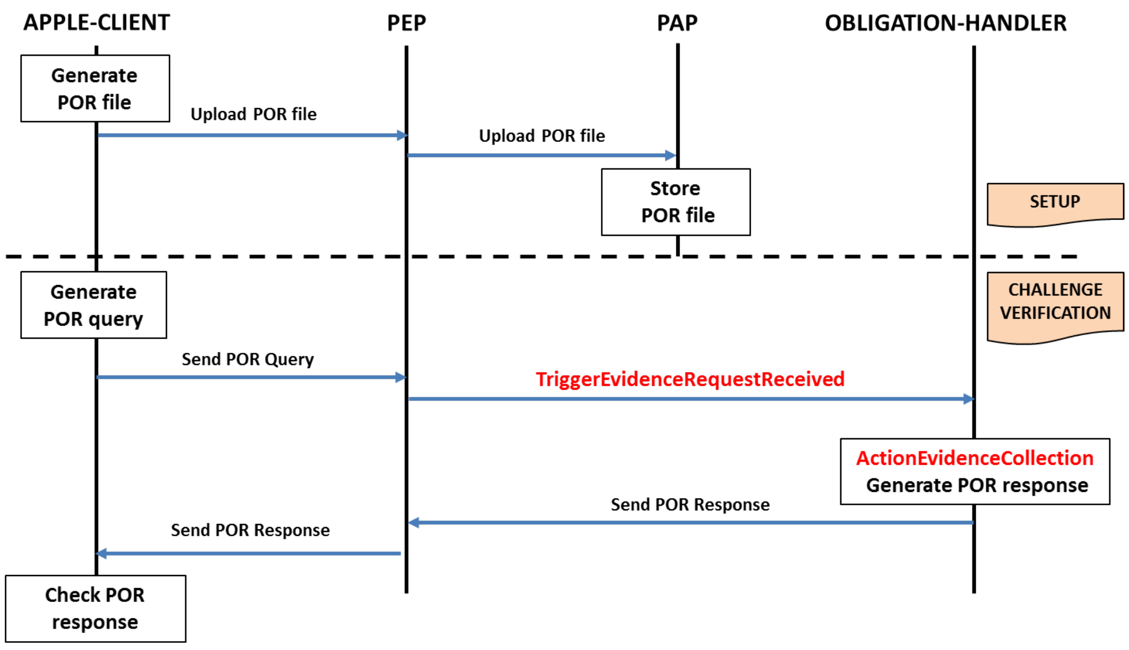


Figure 7.7: Sequence diagram of StealthGuard in A-PPLE

8. Finally, the A-PPLE Client module runs algorithm Verify (see Figure 2.6) and makes a decision about the retrievability of the targeted file.

7.8 Example of A-PPL Statements with respect to our Healthcare Scenario

This section gives some examples of use of the A-PPL elements, based on the healthcare scenario presented in Section 7.3 and the obligations listed in Section 7.4.3. We briefly recall these obligations and we define for each of them an A-PPL rule.

Obligation 1: The right to access, correct and delete personal data. The hospital must ensure that the patients have read and write access to their personal data that have been collected and stored in the cloud. The right to access is expressed in XACML rules that A-PPL is built upon. The data controller grants both read and write access to the data subject. In addition, the deletion of the personal data can be ruled by an A-PPL data handling policy whereby the obligation to delete the data can be expressed using the A-PPL `ActionDeletePersonalData` in conjunction with the trigger `TriggerAtTime`.

Obligation 2: Duration and Purpose of processing. The hospital must make sure that the patients' personal data is only processed for specific, explicit and legitimate purposes. A-PPL uses authorizations to express such purposes using `AuthzUseForPurpose` that allows to specify the purposes for which the processors are authorized to use the collected data. In addition, with the duration attribute for purposes, one can specify different durations for different purposes. Figure 7.8 shows an example of such authorization definitions.

Obligation 3: Breach notification. In case of security or personal data breaches, cloud X and Y must notify M, which in turn must notify the hospital and the hospital must notify the patients. A-PPL provides a way to notify those actors using the `ActionNotify` element. Figure 7.9 shows an example of policy that makes the data controller responsible for notification in case of a policy violation or a loss of data.

```
<a-ppl:AuthzUseForPurpose>
<!-- Authorization for following purposes-->
  <a-ppl:Purpose duration=2Y>diagnosis</a-ppl:Purpose>
  <a-ppl:Purpose duration=5Y>research</a-ppl:Purpose>
</a-ppl:AuthzUseForPurpose>
```

Figure 7.8: Authorization for the specified list of purposes

```
<Obligation>
 <!-- Notify the data subject when triggered -->
 <TriggersSet>
   <TriggerOnPolicyViolation/>
   <TriggerOnDataLost/>
 </TriggersSet>
 <ActionNotify>
   <Media>e-mail</Media>
   <Address>data-subject@example.com</Address>
   <Recipients>Patient:Data subject</Recipients>
   <Type>Policy Violation</Type>
 </ActionNotify>
</Obligation>
```

Figure 7.9: Notification in case of a breach

Obligation 4: Evidence of the correct and timely deletion of personal data. Cloud providers X and Y must be able to provide evidence to the platform provider M, and M must be able to provide evidence to the hospital on the correct and timely deletion of personal data. Therefore, we can use, for example, the A-PPL `ActionLog` element to tell the Data Processors (that is, Cloud X and Y) to track the collection, processing and deletion of personal data. Combined with the A-PPL trigger, `TriggerPersonalDataDeleted`, the logged event will constitute the requested evidence. Besides, we use the action `ActionEvidence-Collection` combined with the trigger `TriggerOnEvidenceRequestReceived` to require the data processor to collect logs for the deletion as evidence of its correctness. Figure 7.10 and Figure 7.11 show a piece of policy that expresses the obligations to log the deletion of the personal data and to collect the requested evidence.

```
<Obligation>
 <TriggerOnPersonalDataDeleted/>
 <ActionLog>
  <Timestamp>
   <EnvironmentAttributeSelector AttributeId="current-time"/>
  </Timestamp>
  <Action>
   <AttributeValue DataType=string>
           Personal Data deleted
   </AttributeValue >
  </Action>
  <Subject>
   <AttributeValue DataType=string>
           Cloud X
   </AttributeValue >
  </Subject>
  <Resource>
   <ResourceAttributeDesignator  DataType="string"
        AttributeId="resource:resource-id"/>
  </Resource>
 </ActionLog>
</Obligation>
```

Figure 7.10: Logging of the deletion of personal data

```
<Obligation>
 <TriggerOnEvidenceRequestReceived/>
 <ActionEvidenceCollection>
  <Evidence>
   <Attribute AttributeId="evidence-type" DataType="string">
    <AttributeValue>Logs</AttributeValue>
   </Attribute>
  </Evidence>
  <Resource>
   <Attribute AttributeId="resource-id" DataType="string">
    <AttributeValue>Personal Data</AttributeValue>
   </Attribute>
  </Resource>
  <Subject>
   <Attribute AttributeId="subject-id"  DataType="string">
   <AttributeValue>Patient</AttributeValue>
   </Attribute>
  </Subject>
  <Recipient>
   <Attribute AttributeId="recipient-id" DataType="string">
    <AttributeValue>Data Controller</AttributeValue>
   </Attribute>
  </Recipient>
 </ActionEvidenceCollection>
</Obligation>
```

Figure 7.11: Collection of the deletion log

```
<AuthzUseForPurpose>
 <Purpose duration=2Y region=Europe>diagnosis</Purpose>
 <Purpose duration=2Y region=Europe>research</Purpose>
</AuthzUseForPurpose>
```

Figure 7.12: Control of the location of data in Europe

Obligation 5: Location of processing. Cloud providers X and Y, as well as the M Platform provider have contractual obligations towards their respective customers on the location of the data processing. In order to be sure that the personal data is not shipped towards location that are not authorized, A-PPL extends XACML with the `region` attribute to be placed in the `AuthzUseForPurpose` environment. For example, we specify in Figure 7.12 that only utilization of collected data in Europe are authorized.

7.9 Conclusion

Part III attempted to answer the following problem: How and to what extent can we convey accountability obligations via policies in such a way that policies are easy to write and enforce and such that Data Controllers and Data Processors can be held accountable for these obligations, thus increasing trust between Cloud Customers and Cloud Providers?

1. We demonstrate that machine-readable policies are suitable means to mitigate the accountability risks by expressing and enforcing accountability obligations related to cloud computing services.

2. From an analysis of these obligations, we identify some design requirements for an accountability policy language.

3. We propose A-PPL, as an extension of the XACML standard and the PPL language. A-PPL enables the specification of access control rules, usage control rules and accountability specific requirements (consent, audits, notification, logging).

4. We describe an architecture for the enforcement of A-PPL policies: the A-PPL engine.

This work on a policy language for accountability is unprecedented. Before A-PPL, there existed no other language that enables to express several accountability obligations within the same framework

Limitations There exists a trade-off between accountability obligations which can be stated in a policy and the technical burden of its enforcement. Automating all accountability processes is not always possible. An example concerns data location and data transfer obligations. Regulations and contracts impose restrictions on sharing data with third-party data processors that are located in a geographical area that does not provide an adequate level of data protection. This kind of restriction would be translatable into an A-PPL policy. However, enforcing this policy, which is *controlling*, seems to be an open issue. Nonetheless, A-PPL and A-PPLE can be leveraged to *monitor* data transfer, that is verifying thanks to audit and logs that data transfers is compliant with obligations.

General Conclusions

In this dissertation, we tackled the problem of loss of control and lack of trust in cloud computing. Since cloud users relinquish to untrusted cloud servers the control of data and computation, these users should be empowered with the ability to verify that the servers are correctly handling these assets. **Verifiability** is hence perceived as a problem of paramount importance in cloud computing. Besides, we looked at the broader notion of **accountability**. The cloud servers should be held accountable for the way they deliver the services they provide. Namely, they must comply with a set of accountability obligations that are conveyed via machine-readable policies. In particular, users should be able to audit them and receive notifications if any security breach is detected.

We stated three problems that this thesis aimed to answer. We provide here an overview of the findings that this thesis reveals.

Problem 1: Verifiable Storage. The goal was to give to cloud users some control over the data they outsourced to an untrusted cloud, by verifying that this cloud possesses the data in its entirety. Well-established cryptographic solutions based on digital signatures computed over the data to check integrity would suffice to answer this problem. However, in the cloud computing context, they are unpractical and cannot scale with big data, since they may incur large communication cost per verification. Therefore, we aimed at designing a protocol in which the cloud generates efficient and probabilistic cryptographic proofs of storage that convince the data owners that their data is correctly stored. We focused on proofs of retrievability, a particular type of proofs of storage that, in addition to the integrity guarantee, ensure the data owner that the data can be recovered even if corruption affects it. We design $\mathcal{S}$tealthGuard, a new POR protocol that inserts in the outsourced data special blocks as watchdogs that witness whether or not the data remains untouched in the remote storage server. We showed its practicality in terms of storage, computation and communication performance. In addition, we proved the security of our scheme. Indeed, a malicious cloud cannot deceitfully pretend to correctly store the outsourced data without being detected.

Problem 2: Verifiable Computation. Cloud users outsource to the cloud computationally demanding operations, especially in the case when these users only can afford limited computation resources. However, the lack of trust in cloud servers imposes users to verify the outcome of the operations they outsourced. Hence, we designed three protocols that generate cryptographic proofs enabling cloud users to check the integrity of computation results returned by the untrusted cloud. We considered the two security requirements that these proofs must satisfy: correctness and soundness. Namely, an untrusted cloud cannot make a verifier accept incorrect results. We proved that our protocols are secure in the standard model and rely on falsifiable assumptions. We also took care about the efficiency requirement of verifiable computation protocol. In addition, our protocols adopt the amortized model approach in which the user who outsources the computation is required to perform a one-time expensive pre-processing of this computation in order to enable verifiability. The cost of this pre-processing step is then amortized by the costs of a possibly unbounded number of verifications. In addition, we looked at publicly delegatable and verifiable solutions which may apply in real-world scenarios, as the space agency scenario we presented in this part.

Problem 3: Accountability Finally, we investigated the possibility of expressing accountability obligations in terms of policies using a machine-readable policy language. To a certain extent, these obligations rule the verifiability requirement of the protocols presented in Part I and Part II, which refers to the ability to prove whether the cloud deviates from the correct provision of the service it delivers (correct storage and correct computation). Hence, we defined the design requirements for an accountability policy language. We then presented A-PPL, which handles accountability specific requirements such as notification, logging and evidence collection. We also described an architecture for A-PPLE, the policy engine that enforces A-PPL policies. We expressed several obligations from a healthcare scenario and defined the corresponding A-PPL rules. In relation to the previous parts of this thesis, we integrated in the A-PPLE framework our $\mathcal{S}$tealthGuard protocol which, by means of appropriate A-PPL policies, requires the cloud to provide evidence (PORs) that it correctly stores outsourced data.

In conclusion, this thesis showed that giving some control to cloud users on the way the cloud servers operate their assets is possible and does not cancel out the advantages of outsourcing storage and computation. Deploying cryptography-based protocols for verifiable storage and computation not only detects cloud servers' misconduct but also deters servers from deviating from a correct operational behavior. The techniques of cryptographic proof verifications tend to foster adoption of cloud technologies. Indeed, proofs of retrievability enable cloud users to audit cloud servers' promise to continuously store their data intact, while proofs of correct computation allow the detection of counterfeit results. In addition, by adopting A-PPL, our policy language for accountability, cloud services can be held responsible for any failure in honoring accountability obligations. The combination of accountability policies with unforgeable proofs for verifiability contributes to the commitment of cloud services to store data and handling outsourced operations. Therefore, ensuring a proper use and provision of cloud services is not the prerogative of a single aspect of computer security such as cryptography, but it is rather the interaction of several mechanisms (policy language, notification, cryptographic protocols, etc.). The work presented in this thesis supports this concluding remark: we built a language for accountability that eases the enforcement of the cryptographic protocols we designed.

Directions for future research work

A future research direction is to investigate verifiability of outsourced data encryption, data location and data deletion. Although they have received much less attention than the problem of POR, we argue that these concerns belong to a broader view of verifiable storage in the sense that they provide verifiability for the different steps of the lifecycle of data *at rest* (that is, data that is not processed or transferred to third parties). We believe these topics should be carefully considered in order to provide a comprehensive cloud storage solution. There exists limited prior art on these aspects. Nevertheless some researchers propose initial frameworks for proofs of encryption, proofs of location and proofs of deletion. These researchers adopt a similar approach to the one followed by POR protocols. Therefore, solutions to these problems can be seen as applications of POR schemes.

To the best of our knowledge, only one existing solution addresses the problem of proofs of encryption. Van Dijk *et al.* [178] introduce the concept of *hourglass schemes* which provides an encryption framework to securely store data at rest and that enables data owners to remotely verify, using a POR-like challenge-response protocol, that their data is indeed encrypted by the storage server. *Hourglass* functions, employed by such schemes, deter *economically rational* storage servers (thus, not fully malicious) from storing the data unencrypted by imposing significant operating costs. While the work by Van Dijk *et al.* [178] offers an initial solution, the problem of proofs of encryption has not been investigated by other research work yet. Proofs of encryption are of interest since data protection rules entail storage service to store data of their users in an encrypted form. Hence, such proofs may become a hot topic in the coming years.

On the other hand, data location becomes a critical issue in cloud computing, as mentioned for example by Peterson *et al.* [148] and in Section 7.4.3. The concern here is to prove that outsourced data is stored in a given geographical location. Watson *et al.* [187] formalize the concept of Proofs of Location (POL) that gives a solution for verifying the location of data in distributed storage systems such as the cloud. The solution proposed in [187] involves a challenge-response protocol between users and servers that combines (i) an Internet Geolocation System such as the ones proposed in [188, 109], that allows to check geographical location of a server using trusted *landmarks* and network latencies, with (ii) a POR scheme, that enables to verify that a server actually stores the data it claims to store. Compared to a POR protocol, users of a POL solution must detect that (i) the server does not actually store the data, or (ii) the server does store the data but forges its location. As in the case for proofs of encryption, and despite their importance in verifiable storage and accountable systems, POL have not yet been thoroughly investigated in the cloud computing security research community.

Along with proofs of encryption and proofs of location, proofs of deletion should convince a data owner that her data is faithfully deleted. Proofs of deletion have first spurred the interest in the context of local secure storage whereby a user wants to verify that the deletion of her data is actually performed by a software-based erasure program [102, 147]. Yet, little attention was paid to outsourced storage deletion in the context of cloud computing [145]. Therefore, future work should include investigations on this topic.

A possible direction of research in the area of verifiable computation would integrate privacy-preserving mechanisms in the context of proofs of correct computation. Indeed, the schemes we presented in Part II do not consider encrypted data. We believe that in the context of cloud computing, performing computation on outsourced encrypted data and verifying the integrity of the results are relevant problems that we should care about. The concerns in this setting would be to allow the cloud to perform operations on encrypted data, but to prevent it from learning anything from (i) the outsourced data; (ii) the computation results and; (iii) the acceptance or the rejection of the results by the users who requested the computation. Hence, we would like to devise a solution that can address these concerns while not sacrificing efficiency.

The main direction of research in the policy language work is to implement enforcement tools. For our policy language framework to be a success, a critical challenge is to develop the components that will enforce the different obligations written in A-PPL within A-PPLE. Integrating $\mathcal{S}$tealthGuard to A-PPLE was a first step. Indeed, collecting and checking proofs of retrievability enable to verify that a cloud server complies with the obligation of storing data. Further components to be integrated in the policy engine would be mechanisms that allow secure logging, that is, that generate non-repudiable logs while protecting their confidentiality and integrity.

Bibliography

[1] 104th Congress. Health Insurance Portability and Accountability Act of 1996. *Public Law*, 104:191, 1996. 9, 204

[2] Shweta Agrawal and Dan Boneh. Homomorphic MACs: MAC-based Integrity for Network Coding. In *Applied Cryptography and Network Security*, pages 292–305. Springer, 2009. 76

[3] Rudolf Ahlswede, Ning Cai, Shuo-Yen Robert Li, and Raymond W Yeung. Network Information Flow. *IEEE Transactions on Information Theory*, 46(4):1204–1216, 2000. 76

[4] Irem Aktug and Katsiaryna Naliuka. ConSpec – a formal language for policy specification. In *Electronic Notes in Theoretical Computer Science*, volume 197, pages 45–58. Elsevier, 2008. 167

[5] Muhammad Ali, Laurent Bussard, and Ulrich Pinsdorf. Obligation Language and Framework to Enable Privacy-Aware SOA. In Joaquin Garcia-Alfaro, Guillermo Navarro-Arribas, Nora Cuppens-Boulahia, and Yves Roudier, editors, *Data Privacy Management and Autonomous Spontaneous Security*, volume 5939 of *Lecture Notes in Computer Science*, pages 18–32. Springer Berlin Heidelberg, 2010. 170

[6] David P Anderson. Volunteer Computing: The Ultimate Cloud. *ACM Crossroads*, 16 (3):7–10, 2010. 71

[7] David P Anderson, Jeff Cobb, Eric Korpela, Matt Lebofsky, and Dan Werthimer. SETI@Home: An Experiment in Public-Resource Computing. *Communications of the ACM*, 45(11):56–61, 2002. 71

[8] Claudio A. Ardagna, Laurent Bussard, Sabrina De Capitani Di Vimercati, Gregory Neven, Stefano Paraboschi, Eros Pedrini, Stefan Preiss, Dave Raggett, Pierangela Samarati, Slim Trabelsi, and Mario Verdicchio. Primelife policy language. http://www.w3.org/2009/policy-ws/papers/Trabelisi.pdf, 2009. 167, 168, 169

[9] Frederik Armknecht, Jens-Matthias Bohli, Ghassan O Karame, Zongren Liu, and Christian A Reuter. Outsourced proofs of retrievability. In *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security*, pages 831–843. ACM, 2014. 29, 30

[10] Sanjeev Arora and Shmuel Safra. Probabilistic Checking of Proofs: A New Characterization of NP. *Journal of the ACM (JACM)*, 45(1):70–122, 1998. 72, 74, 84, 214

[11] Sanjeev Arora, Carsten Lund, Rajeev Motwani, Madhu Sudan, and Mario Szegedy. Proof verification and the hardness of approximation problems. *Journal of the ACM (JACM)*, 45(3):501–555, 1998. 72

[12] Mikhail J Atallah and Keith B Frikken. Securely Outsourcing Linear Algebra Computations. In *Proceedings of the 5th ACM Symposium on Information, Computer and Communications Security*, pages 48–59. ACM, 2010. 79

[13] Giuseppe Ateniese and Breno de Medeiros. Identity-Based Chameleon Hash and Applications. In *Financial Cryptography*, pages 164–180. Springer, 2004. 58

[14] Giuseppe Ateniese, Randal Burns, Reza Curtmola, Joseph Herring, Lea Kissner, Zachary Peterson, and Dawn Song. Provable Data Possession at Untrusted Stores. In *Proceedings of the 14th ACM Conference on Computer and Communications Security*, pages 598–609. ACM, 2007. 18, 21, 22, 23, 24, 25, 27, 30, 76, 209

[15] Giuseppe Ateniese, Roberto Di Pietro, Luigi V. Mancini, and Gene Tsudik. Scalable and Efficient Provable Data Possession. In *Proceedings of the 4th International Conference on Security and Privacy in Communication Networks*, SecureComm'08, pages 9:1–9:10, New York, NY, USA, 2008. ACM. 24, 30

[16] Giuseppe Ateniese, Seny Kamara, and Jonathan Katz. Proofs of Storage from Homomorphic Identification Protocols. In *ASIACRYPT*, pages 319–333, 2009. 23

[17] Giuseppe Ateniese, Randal C. Burns, Reza Curtmola, Joseph Herring, Osama Khan, Lea Kissner, Zachary N. J. Peterson, and Dawn Song. Remote Data Checking Using Provable Data Possession. *ACM Transactions on Information and System Security (TISSEC)*, 14(1):12, 2011. 27, 28, 30, 52, 53, 54

[18] Azraoui, Monir and Elkhiyaoui, Kaoutar and Önen, Melek and Bernsmed, Karin and De Oliveira, Anderson Santana and Sendor, Jakub. A-PPL: An Accountability Policy Language. In *Data Privacy Management, Autonomous Spontaneous Security, and Security Assurance*, pages 319–326. Springer, 2015. 163

[19] László Babai. Trading Group Theory for Randomness. In *Proceedings of the Seventeenth Annual ACM Symposium on Theory of Computing*, STOC '85, pages 421–429. ACM, 1985. 72, 84

[20] Michael Backes, Dario Fiore, and Raphael M Reischuk. Verifiable Delegation of Computation on Outsourced Data. In *Proceedings of the 2013 ACM SIGSAC conference on Computer and Communications Security*, pages 863–874. ACM, 2013. 77, 84

[21] Michael Backes, Manuel Barbosa, Dario Fiore, and Raphael M. Reischuk. ADSNARK: Nearly Practical and Privacy-Preserving Proofs on Authenticated Data. In *IEEE Symposium on Security and Privacy (Oakland)*, May 2015. 77, 84

[22] Ricardo Baeza-Yates and Berthier Ribeiro-Neto. *Modern Information Retrieval.* ACM Press, 1999. 80

[23] Boaz Barak, Oded Goldreich, Rusell Impagliazzo, Steven Rudich, Amit Sahai, Salil Vadhan, and Ke Yang. On the (im) Possibility of Obfuscating Programs. In *Advances in cryptology—CRYPTO 2001*, pages 1–18. Springer, 2001. 28

[24] Niko Baric and Birgit Pfitzmann. Collision-free Accumulators and Fail-stop Signature Schemes Without Trees. In *Proceedings of the 16th Annual International Conference on Theory and Application of Cryptographic Techniques*, EUROCRYPT'97, pages 480–494, Berlin, Heidelberg, 1997. Springer-Verlag. 122

[25] Barros, Alistair and Oberle, Daniel. *Handbook of Service Description: USDL and Its Methods.* Springer Publishing Company, Incorporated, 2012. 167

[26] Moritz Y Becker, Alexander Malkis, and Laurent Bussard. S4P: A generic language for specifying privacy preferences and policies. *Microsoft Research*, 2010. 167

[27] Mihir Bellare, Ran Canetti, and Hugo Krawczyk. Keying Hash Functions for Message Authentication. In *Proceedings of the 16th Annual International Cryptology conference on Advances in Cryptology, CRYPTO'96*, pages 1–15. LNCS, August 1996. 27

[28] Eli Ben-Sasson, Oded Goldreich, Prahladh Harsha, Madhu Sudan, and Salil Vadhan. Short PCPs Verifiable in Polylogarithmic Time. In *Proceedings. Twentieth Annual IEEE Conference on Computational Complexity, 2005.*, pages 120–134. IEEE, 2005. 72

[29] Eli Ben-Sasson, Oded Goldreich, Prahladh Harsha, Madhu Sudan, and Salil Vadhan. Robust PCPs of proximity, shorter PCPs, and applications to coding. *SIAM Journal on Computing*, 36(4):889–974, 2006. 72

[30] Siavosh Benabbas, Rosario Gennaro, and Yevgeniy Vahlis. Verifiable Delegation of Computation over Large Datasets. In Phillip Rogaway, editor, *Advances in Cryptology – CRYPTO 2011*, volume 6841 of *Lecture Notes in Computer Science*, pages 111–131. Springer Berlin Heidelberg, 2011. 24, 27, 78, 80, 84, 214

[31] Josh Benaloh and Michael De Mare. One-Way Accumulators: A Decentralized Alternative to Digital Signatures. In *Advances in Cryptology – EUROCRYPT'93*, pages 274–285. Springer, 1994. 80, 121, 122

[32] Karin Bernsmed, Massimo Felici, Anderson Santana De Oliveira, Jakub Sendor, Nils Brede Moe, Thomas Rübsamen, Vasilis Tountopoulos, and Bushra Hasnain. Use Case Descriptions. Deliverable, Cloud Accountability (A4Cloud) Project, 2013. 162

[33] Karin Bernsmed, Hon Kuan, and Christopher Millard. Deploying Medical Sensor Networks in the Cloud - Accountability Obligations from a European Perspective. *Submitted for publication*, 2014. 162

[34] Karin Bernsmed, Vasilis Tountopoulos, Paul Brigden, Thomas Rübsamen, Massimo Felici, Nick Wainwright, Anderson Santana De Oliveira, Jakub Sendor, Mohammed Sellami, Jean-Claude Royer, Melek Önen, and Mario Südholt. Consolidated Use Case Report. Deliverable, Cloud Accountability (A4Cloud) Project, 2014. xiii, 162, 165

[35] Etienne Bézout. *Théorie générale des équations algébriques*. Imprimerie Ph.-D. Pierres, 1779. 125

[36] Nir Bitansky, Ran Canetti, Alessandro Chiesa, and Eran Tromer. From Extractable Collision Resistance to Succinct Non-Interactive Arguments of Knowledge, and Back Again. In *Proceedings of the 3rd Innovations in Theoretical Computer Science Conference*, pages 326–349. ACM, 2012. 75, 76, 82, 84

[37] Erik-Oliver Blass, Roberto Di Pietro, Refik Molva, and Melek Önen. PRISM - Privacy-Preserving Search in MapReduce. In *Proceedings of the 12th Privacy Enhancing Technologies Symposium (PETS 2012)*. LNCS, July 2012. 31, 37, 41, 211

[38] Matt Blaze, Gerrit Bleumer, and Martin Strauss. Divertible Protocols and Atomic Proxy Cryptography. In *Advances in Cryptology—EUROCRYPT'98*, pages 127–144. Springer, 1998. 26

[39] Dan Boneh and David Mandell Freeman. Homomorphic Signatures for Polynomial Functions. In *Advances in Cryptology–EUROCRYPT 2011*, pages 149–168. Springer, 2011. 76, 77, 84

[40] Dan Boneh, Craig Gentry, Ben Lynn, and Hovav Shacham. Aggregate and Verifiably Encrypted Signatures From Bilinear Maps. In *Advances in cryptology—EUROCRYPT 2003*, pages 416–432. Springer, 2003. 26

[41] Dan Boneh, Ben Lynn, and Hovav Shacham. Short Signatures from the Weil Pairing. *J. Cryptology*, 17(4):297–319, September 2004. 23, 25, 26, 27

[42] Sara Bouchenak, Gregory Chockler, Hana Chockler, Gabriela Gheorghe, Nuno Santos, and Alexander Shraer. Verifying Cloud Services: Present and Future. *ACM SIGOPS Operating Systems Review*, 47(2):6–19, 2013. 11

[43] Kevin D. Bowers, Ari Juels, and Alina Oprea. Proofs of Retrievability: Theory and Implementation. In *Proceedings of the 2009 ACM Workshop on Cloud Computing Security*, CCSW '09, pages 43–54, New York, NY, USA, 2009. ACM. 27, 30

[44] Kevin D Bowers, Ari Juels, and Alina Oprea. HAIL: a High-Availability and Integrity Layer for Cloud Storage. In *Proceedings of the 16th ACM Conference on Computer and Communications Security*, pages 187–198. ACM, 2009. 27, 30

[45] Simon Bradshaw, Christopher Millard, and Ian Walden. Contracts for Clouds : Comparison and Analysis of the Terms and Conditions of Cloud Computing Services. Technical Report Legal Studies Research Paper 63/2010, Queen Mary, University of London, 2010. 157

[46] Benjamin Braun, Ariel J Feldman, Zuocheng Ren, Srinath Setty, Andrew J Blumberg, and Michael Walfish. Verifying Computations with State. In *Proceedings of the Twenty-Fourth ACM Symposium on Operating Systems Principles*, pages 341–357. ACM, 2013. 74

[47] Bray, Tim and Paoli, Jean and Sperberg-McQueen, C Michael and Maler, Eve and Yergeau, François. Extensible markup language (XML). *World Wide Web Journal*, 2 (4):27–66, 1997. 168

[48] Denis Butin, Marcos Chicote, and Daniel Le Métayer. Log Design For Accountability. In *Security and Privacy Workshops (SPW), 2013 IEEE*, pages 1–7. IEEE, 2013. 171, 172

[49] Jan Camenisch and Anna Lysyanskaya. Dynamic Accumulators and Application to Efficient Revocation of Anonymous Credentials. In *Advances in Cryptology – CRYPTO 2002*, pages 61–76. Springer, 2002. 122, 123

[50] L Jean Camp and M Eric Johnson. *The Economics of Financial and Medical Identity Theft*. Springer Science & Business Media, 2012. 4

[51] Ran Canetti, Ben Riva, and Guy N Rothblum. Practical Delegation of Computation using Multiple Servers. In *Proceedings of the 18th ACM Conference on Computer and Communications Security*, pages 445–454. ACM, 2011. 71

[52] Ran Canetti, Omer Paneth, Dimitrios Papadopoulos, and Nikos Triandopoulos. Verifiable Set Operations over Outsourced Databases. In *Public-Key Cryptography–PKC 2014*, pages 113–130. Springer, 2014. 78, 81, 82, 123, 124, 125, 143, 214

[53] David Cash, Alptekin Küpçü, and Daniel Wichs. Dynamic Proofs Of Retrievability via Oblivious RAM. In *EUROCRYPT*, pages 279–295, 2013. 28, 30, 57

[54] Kenneth R. Castleman. *Digital Image Processing*. Prentice Hall Press, Upper Saddle River, NJ, USA, 1996. 99

[55] Dario Catalano and Dario Fiore. Practical Homomorphic MACs for Arithmetic Circuits. In *EUROCRYPT*, pages 336–352. Springer, 2013. 76, 77

[56] Dario Catalano, Dario Fiore, Rosario Gennaro, and Luca Nizzardo. Generalizing Homomorphic MACs for Arithmetic Circuits. In *Public-Key Cryptography–PKC 2014*, pages 538–555. Springer, 2014. 77

[57] Dario Catalano, Dario Fiore, and Bogdan Warinschi. Homomorphic Signatures with Efficient Verification for Polynomial Functions. In *Advances in Cryptology–CRYPTO 2014*, pages 371–389. Springer, 2014. 76, 77, 84

[58] Daniele Catteddu, Massimo Felici, Giles Hogben, Amy Holcroft, Eleni Kosta, Ronald Leenes, Christopher Millard, Maartje Niezen, David Nuñez, Nick Papanikolaou, *et al.* Towards a Model of Accountability for Cloud Computing Services. In *Proceedings of the DIMACS/BIC/A4Cloud/CSA International Workshop on Trustworthiness, Accountability and Forensics in the Cloud (TAFC)(May 2013)*, 2013. 158

[59] Qi Chai and Guang Gong. Verifiable Symmetric Searchable Encryption for semi-Honest-but-Curious Cloud Servers. In *Communications (ICC), 2012 IEEE International Conference on*, pages 917–922. IEEE, 2012. 81, 82, 84, 153

[60] Bo Chen and Reza Curtmola. Robust dynamic provable data possession. In *ICDCS Workshops*, pages 515–525, 2012. 28, 30

[61] Lanxiang Chen. Using Algebraic Signatures to Check Data Possession in Cloud Storage. *Future Generation Computer Systems*, 29(7):1709–1715, 2013. 23, 30

[62] Rong Cheng, Jingbo Yan, Chaowen Guan, Fangguo Zhang, and Kui Ren. Verifiable Searchable Symmetric Encryption from Indistinguishability Obfuscation. In *Proceedings of the 10th ACM Symposium on Information, Computer and Communications Security*, pages 621–626. ACM, 2015. 82, 153

[63] Roman-Alexandre Cherrueau, Rémi Douence, Hervé Grall, Jean-Claude Royer, Mohamed Sellami, M Südholt, Monir Azraoui, Kaouthar Elkhiyaoui, Refik Molva, Melek Önen, Alexandr Garaga, Anderson Santana de Oliveira, Jakub Sendor, and Karin Bernsmed. Policy Representation Framework. Deliverable, Cloud Accountability (A4Cloud) Project, 2013. 167

[64] Kai-Min Chung, Yael Kalai, and Salil Vadhan. Improved Delegation of Computation using Fully Homomorphic Encryption. In *Advances in Cryptology–CRYPTO 2010*, pages 483–501. Springer, 2010. 76, 78, 82

[65] Kai-Min Chung, Yael Tauman Kalai, Feng-Hao Liu, and Ran Raz. Memory Delegation. In *Advances in Cryptology–CRYPTO 2011*, pages 151–168. Springer, 2011. 76

[66] Graham Cormode, Michael Mitzenmacher, and Justin Thaler. Practical Verified Computation with Streaming Interactive Proofs. In *Proceedings of the 3rd Innovations in Theoretical Computer Science Conference*, ITCS '12, pages 90–112, New York, NY, USA, 2012. ACM. 72

[67] Reza Curtmola, Osama Khan, Randal Burns, and Giuseppe Ateniese. MR-PDP: Multiple-Replica Provable Data Possession. In *The 28th International Conference on Distributed Computing Systems, 2008. ICDCS'08.* , pages 411–420. IEEE, 2008. 23, 30

[68] Joan Daemen and Vincent Rijmen. AES proposal: Rijndael, 1999. 36

[69] Ivan Damgård and Nikos Triandopoulos. Supporting Non-membership Proofs with Bilinear-map Accumulators. *IACR Cryptology ePrint Archive*, 2008:538, 2008. 119, 123, 124

[70] Jeffrey Dean and Sanjay Ghemawat. MapReduce: Simplified Data Processing on Large Clusters. *Communications of the ACM*, 51(1):107–113, 2008. 74

[71] Yves Deswarte and Jean-Jacques Quisquater. Remote Integrity Checking. In *Sixth Working Conference on Integrity and Internal Control in Information Systems (IICIS)*, pages 1–11. Kluwer Academic Publishers, 1 2004. 21, 30

[72] Francesco Di Cerbo, Doliere Francis Some, Laurent Gomez, and Slim Trabelsi. PPL V2.0: Uniform Data Access and Usage Control on Cloud and Mobile. In *Proceedings of the First International Workshop on TEchnical and LEgal Aspects of Data pRIvacy*, TELERISE '15. IEEE Press, 2015. 171, 172

[73] Giovanni Di Crescenzo and Helger Lipmaa. Succinct NP Proofs from an Extractability Assumption. In *Logic and Theory of Algorithms*, pages 175–185. Springer, 2008. 75

[74] Martin Dietzfelbinger and Christoph Weidling. Balanced Allocation and Dictionaries with Tightly Packed Constant Size Bins. *Theoretical Computer Science*, 380(1):47–68, 2007. 119, 121, 142

[75] Yevgeniy Dodis, Salil Vadhan, and Daniel Wichs. Proofs of retrievability via hardness amplification. In *Theory of Cryptography*, pages 109–127. Springer, 2009. 27, 30

[76] Taher El Gamal. A Public Key Cryptosystem and a Signature Scheme Based on Discrete Logarithms. In *CRYPTO 84 on Advances in Cryptology*, pages 10–18. Springer New York, Inc., 1985. 73, 79

[77] Chris Erway, Alptekin Küpçü, Charalampos Papamanthou, and Roberto Tamassia. Dynamic provable data possession. In *Proceedings of the 16th ACM Conference on Computer and Communications Security*, CCS '09, pages 213–222, New York, NY, USA, 2009. ACM. 24, 25, 30

[78] Ertem Esiner, Adilet Kachkeev, Samuel Braunfeld, Alptekin Küpçü, and Öznur Özkasap. FlexDPDP: FlexList-based Optimized Dynamic Provable Data Possession. *IACR Cryptology ePrint Archive*, 2013:645, 2013. 24, 25, 30

[79] European Parliament and the Council of the European Union. Directive 2008/30/EC of the European Parliament and of the Council of 11 March 2008 Amending Directive 2006/43/EC on Statutory Audits of Annual Accounts and Consolidated Accounts, 2008. 9, 204

[80] European Parliament and the Council of the European Union. Regulation (EU) 2016/679 of the European Parliament and of the Council of 27 April 2016 on the Protection of Natural Persons with regard to the Processing of Personal Data and on the Free Movement of such Data, 2016. 9, 157, 160, 161, 163, 164, 165, 204, 218, 219

[81] Li Fan, Pei Cao, Jurassa Almeida, and Andrei Z. Broder. Summary Cache: a Scalable Wide-Area Web Cache Sharing Protocol. *IEEE/ACM Trans. Netw.*, 8(3):281–293, June 2000. ISSN 1063-6692. 80

[82] Massimo Felici and Siani Pearson. Accountability for Data Governance in the Cloud. In *Accountability and Security in the Cloud*, pages 3–42. Springer, 2015. 159

[83] Amos Fiat and Adi Shamir. How to Prove Yourself: Practical Solutions to Identification and Signature Problems. In *Advances in Cryptology—CRYPTO'86*, pages 186–194. Springer, 1987. 74, 214

[84] Décio Luiz Gazzoni Filho and Paulo Sérgio Licciardi Messeder Barreto. Demonstrating Data Possession and Uncheatable Data Transfer. *IACR Cryptology ePrint Archive*, 2006:150, 2006. 21, 30

[85] Dario Fiore and Rosario Gennaro. Publicly Verifiable Delegation of Large Polynomials and Matrix Computations, with Applications. In *Proceedings of the 2012 ACM Conference on Computer and Communications Security*, CCS '12, pages 501–512. ACM, 2012. 78, 79, 80, 84, 95, 96, 100, 109, 110, 214, 215

[86] Freedman, Michael J and Nissim, Kobbi and Pinkas, Benny. Efficient Private Matching and Set Intersection. In *Advances in Cryptology-EUROCRYPT 2004*, pages 1–19. Springer, 2004. 123

[87] Sanjam Garg, Craig Gentry, and Shai Halevi. Candidate Multilinear Maps from Ideal Lattices. In *Advances in Cryptology–EUROCRYPT 2013*, pages 1–17. Springer, 2013. 77

[88] Shelly Garg, Craig Gentry, Shai Halevi, Mariana Raykova, Anant Sahai, and Brent Waters. Candidate Indistinguishability Obfuscation and Functional Encryption for All Circuits. In *IEEE 54th Annual Symposium on Foundations of Computer Science (FOCS), 2013*, pages 40–49. IEEE, 2013. 28

[89] Rosario Gennaro and Daniel Wichs. Fully Homomorphic Message Authenticators. In *Advances in Cryptology-ASIACRYPT 2013*, pages 301–320. Springer, 2013. 76, 77, 84

[90] Rosario Gennaro, Craig Gentry, and Bryan Parno. Non-Interactive Verifiable Computation: Outsourcing Computation To Untrusted Workers. In *In Proceedings of CRYPTO*. Citeseer, 2010. 5, 24, 27, 61, 63, 71, 75, 78, 82, 84, 94, 212, 214

[91] Rosario Gennaro, Craig Gentry, Bryan Parno, and Mariana Raykova. Quadratic Span Programs and Succinct NIZKs without PCPs. In *EUROCRYPT*, volume 7881, pages 626–645. Springer, 2013. 74, 75, 78, 81, 84

[92] Craig Gentry. Fully Homomorphic Encryption Using Ideal Lattices. In *STOC*, volume 9, pages 169–178, 2009. 71, 75, 76

[93] Craig Gentry and Daniel Wichs. Separating Succinct Non-Interactive Arguments From All Falsifiable Assumptions. In *Proceedings of the Forty-Third Annual ACM Symposium on Theory of Computing*, pages 99–108. ACM, 2011. 75

[94] Oded Goldreich and Rafail Ostrovsky. Software Protection and Simulation on Oblivious RAMs. *Journal of the ACM (JACM)*, 43(3):431–473, 1996. 28

[95] Oded Goldreich, Shafi Goldwasser, and Silvio Micali. How to Construct Random Functions. *Journal of the ACM (JACM)*, 33(4):792–807, 1986. 36

[96] Shafi Goldwasser and Silvio Micali. Probabilistic Encryption & How to Play Mental Poker Keeping Secret All Partial Information. In *Proceedings of the Fourteenth Annual ACM Symposium on Theory of Computing*, STOC '82, pages 365–377, New York, NY, USA, 1982. ACM. 37

[97] Shafi Goldwasser, Silvio Micali, and Charles Rackoff. The Knowledge Complexity of Interactive Proof Systems. *SIAM Journal on Computing*, 18:186–208, 1989. 72, 84

[98] Shafi Goldwasser, Yael Tauman Kalai, and Guy N Rothblum. Delegating Computation: Interactive Proofs for Muggles. In *Proceedings of the 40th annual ACM Symposium on Theory of Computing*, pages 113–122. ACM, 2008. 72, 75, 78, 84, 214

[99] Vipul Goyal, Omkant Pandey, Amit Sahai, and Brent Waters. Attribute-based Encryption for Fine-grained Access Control of Encrypted Data. In *Proceedings of the 13th ACM conference on Computer and Communications Security*, pages 89–98. ACM, 2006. 76

[100] Chaowen Guan, Kui Ren, Fangguo Zhang, Florian Kerschbaum, and Jia Yu. Symmetric-Key Based Proofs of Retrievability Supporting Public Verification. In *Computer Security–ESORICS 2015*, pages 203–223. Springer, 2015. 28, 29, 30

[101] Christian Hanser and Daniel Slamanig. Efficient Simultaneous Privately and Publicly Verifiable Robust Provable Data Possession from Elliptic Curves. In *SECRYPT 2013 - Proceedings of the 10th International Conference on Security and Cryptography, Reykjavík, Iceland, 29-31 July, 2013*, pages 15–26, 2013. 23, 30

[102] Feng Hao, Dylan Clarke, and Avelino Francisco Zorzo. Deleting Secret Data with Public Verifiability. Cryptology ePrint Archive, Report 2014/364, 2014. 185

[103] Martin Henze, Marcel Großfengels, Maik Koprowski, and Klaus Wehrle. Towards Data Handling Requirements-aware Cloud Computing. In *2013 IEEE International Conference on Cloud Computing Technology and Science (CloudCom)*, 2013. 171, 172

[104] Yuval Ishai, Eyal Kushilevitz, and Rafail Ostrovsky. Efficient arguments without short PCPs. In *Twenty-Second Annual IEEE Conference on Computational Complexity, 2007. CCC'07.*, pages 278–291. IEEE, 2007. 73, 84

[105] ISO/IEC. ISO/IEC 15414:2015. Information technology – Open distributed processing – Reference model – Enterprise language. Technical report, International Organization for Standardization (ISO), 2015. 161

[106] Robert Johnson, David Molnar, Dawn Song, and David Wagner. Homomorphic Signature Schemes. In *Topics in Cryptology–CT-RSA 2002*, pages 244–262. Springer, 2002. 76

[107] Ari Juels and Burton Stephen Kaliski. PORs: Proofs Of Retrievability For Large Files. In Peng Ning, Sabrina De Capitani di Vimercati, and Paul F. Syverson, editors, *ACM Conference on Computer and Communications Security*, pages 584–597. ACM, 2007. 18, 21, 22, 26, 27, 29, 30, 32, 34, 35, 52, 53, 54, 209, 210

[108] Aniket Kate, Gregory M. Zaverucha, and Ian Goldberg. Constant-Size Commitments to Polynomials and Their Applications. In *Advances in Cryptology-ASIACRYPT 2010*, pages 177–194. Springer, 2010. 27, 79, 95

[109] Ethan Katz-Bassett, John P John, Arvind Krishnamurthy, David Wetherall, Thomas Anderson, and Yatin Chawathe. Towards IP Geolocation Using Delay and Topology Measurements. In *Proceedings of the 6th ACM SIGCOMM Conference on Internet Measurement*, pages 71–84. ACM, 2006. 185

[110] Joe Kilian. A Note on Efficient Zero-Knowledge Proofs and Arguments. In *Proceedings of the Twenty-Fourth Annual ACM Symposium on Theory of Computing*, pages 723–732. ACM, 1992. 70, 72, 73, 74, 84, 214

[111] Joe Kilian. Improved Efficient Arguments. In *Advances in Cryptology–CRYPTO'95*, pages 311–324. Springer, 1995. 72, 73, 74, 214

[112] Zachary A Kissel and Jie Wang. Verifiable Phrase Search over Encrypted Data Secure against a Semi-Honest-but-Curious Adversary. In *Distributed Computing Systems Workshops (ICDCSW), 2013 IEEE 33rd International Conference on*, pages 126–131. IEEE, 2013. 81, 82

[113] Ahmed E Kosba, Dimitrios Papadopoulos, Charalampos Papamanthou, Mahmoud F Sayed, Elaine Shi, and Nikos Triandopoulos. TRUESET: Faster verifiable set computations. In *USENIX Security*, 2014. 81, 84, 153

[114] Hugo Krawczyk and Tal Rabin. Chameleon Signatures. In *Proceedings of the Network and Distributed System Security Symposium, NDSS 2000, San Diego, California, USA*, 2000. 58

[115] Łukasz Krzywiecki and Mirosław Kutyłowski. Proof of Possession for Cloud Storage via Lagrangian Interpolation Techniques. In *Network and System Security*, pages 305–319. Springer, 2012. 23, 30

[116] Alptekin Küpçü. Official Arbitration with Secure Cloud Storage Application. *The Computer Journal*, 58(4):831–852, 2015. 58

[117] D. Davide Lamanna, James Skene, and Wolfgang Emmerich. SLAng: A Language for Defining Service Level Agreements. In *Proceedings of the The Ninth IEEE Workshop on Future Trends of Distributed Computing Systems*, Washington, DC, USA, 2003. IEEE Computer Society. 167

[118] Adeline Langlois, Damien Stehlé, and Ron Steinfeld. GGHLite: More Efficient Multilinear Maps from Ideal Lattices. In *Advances in Cryptology–EUROCRYPT 2014*, pages 239–256. Springer, 2014. 77

[119] Tancrède Lepoint and Mehdi Tibouchi. Cryptanalysis of a (Somewhat) Additively Homomorphic Encryption Scheme Used in PIR. In Michael Brenner, Nicolas Christin, Benjamin Johnson, and Kurt Rohloff, editors, *Financial Cryptography and Data Security*, volume 8976 of *Lecture Notes in Computer Science*, pages 184–193. Springer Berlin Heidelberg, 2015. 37

[120] Jiangtao Li, Ninghui Li, and Rui Xue. Universal Accumulators with Efficient Non-Membership Proofs. In *Applied Cryptography and Network Security*, pages 253–269. Springer, 2007. 122

[121] Fang Liu, Jin Tong, Jian Mao, Robert Bohn, John Messina, Lee Badger, and Dawn Leaf. NIST Cloud Computing Reference Architecture. *NIST special publication*, 500: 292, 2011. 160

[122] Pengliang Liu, Jianfeng Wang, Hua Ma, and Haixin Nie. Efficient Verifiable Public Key Encryption with Keyword Search Based on KP-ABE. In *Ninth International Conference on Broadband and Wireless Computing, Communication and Applications, BWCCA 2014, Guangdong, China, November 8-10, 2014*, pages 584–589, 2014. 81

[123] Massimo Marchiori. The Platform for Privacy Preferences 1.0 (P3P1.0) Specification. W3C recommendation, W3C, 2002. 167

[124] Peter Mell and Tim Grance. The NIST Definition of Cloud Computing, 2011. 7

[125] Ralph C Merkle. A Digital Signature Based on a Conventional Encryption Function. In *Advances in Cryptology–CRYPTO'87*, pages 369–378. Springer, 1988. 24, 25, 73, 119, 217

[126] Silvio Micali. Computationally Sound Proofs. *SIAM Journal on Computing*, 30(4): 1253–1298, 2000. 74, 75, 76, 82, 84, 214

[127] Payman Mohassel. Efficient and Secure Delegation of Linear Algebra. *IACR Cryptology ePrint Archive*, 2011:605, 2011. 79, 84

[128] David Molnar. The SETI@home problem. *ACM Crossroads*, 7:55, 2000. URL http://tiny.cc/f0c38x. 4, 5

[129] Ruggero Morselli, Bobby Bhattacharjee, Justin Katz, and Pete Keleher. Trust-Preserving Set Operations. In *INFOCOM 2004. Twenty-third Annual Joint Conference of the IEEE Computer and Communications Societies*, volume 4, pages 2231–2241. IEEE, 2004. 80, 84, 153

[130] Satoshi Nakamoto. Bitcoin: A Peer-to-Peer Electronic Cash System. *Consulted*, 1 (2012):28, 2008. 29

[131] National Institute of Standards and Technology. FIPS 180-2, Secure Hash Standard, Federal Information Processing Standard (FIPS), Publication 180-2. Technical report, Department of Commerce, August 2002. 39

[132] Lan Nguyen. Accumulators From Bilinear Pairings and Applications. In *Topics in Cryptology–CT-RSA 2005*, pages 275–292. Springer, 2005. 80, 119, 123, 216

[133] Nicodemos Damianou and Naranker Dulay and Emil Lupu and Morris Sloman. The Ponder Policy Specification Language. In *POLICY*, pages 18–38, 2001. 167

[134] OASIS Standard. eXtensible Access Control Markup Language (XACML) Version 3.0. 22 January 2013. http://docs.oasis-open.org/xacml/3.0/xacml-3.0-core-spec-os-en.html, 2013. 167, 168

[135] OASIS Web Service Security (WSS) TC. Web Services Security: SOAP Message Security 1.1, 2006. 167

[136] OASIS Web Services Secure Exchange (WS-SX) TC. WS-Trust 1.4, 2012. 167

[137] Rasmus Pagh and Flemming Friche Rodler. Cuckoo Hashing. *Journal of Algorithms*, 51(2):122–144, 2004. 121, 217

[138] Pascal Paillier. Public-Key Cryptosystems Based on Composite Degree Residuosity Classes. In Jacques Stern, editor, *Advances in Cryptology at EUROCRYPT'99*, volume 1592 of *Lecture Notes in Computer Science*, pages 223–238. Springer Berlin Heidelberg, 1999. 73, 79

[139] Stavros Papadopoulos, Dimitris Papadias, Weiwei Cheng, and Kian-Lee Tan. Separating Authentication From Query Execution in Outsourced Databases. In *IEEE 25th International Conference on Data Engineering, 2009. ICDE'09.*, pages 1148–1151. IEEE, 2009. 82

[140] Charalampos Papamanthou, Roberto Tamassia, and Nikos Triandopoulos. Optimal Verification of Operations on Dynamic Sets. In *Advances in Cryptology–CRYPTO 2011*, pages 91–110. Springer, 2011. 78, 80, 81, 82, 84, 123, 143, 214

[141] Charalampos Papamanthou, Elaine Shi, and Roberto Tamassia. Signatures of Correct Computation. In *Theory of Cryptography*, pages 222–242. Springer, 2013. 75, 78, 79, 84, 95, 96, 214

[142] Bryan Parno, Jonathan M McCune, and Adrian Perrig. *Bootstrapping Trust in Modern Computers*, volume 10. Springer Science & Business Media, 2011. 71

[143] Bryan Parno, Mariana Raykova, and Vinod Vaikuntanathan. How to Delegate and Verify in Public: Verifiable Computation from Attribute-Based Encryption. In Ronald Cramer, editor, *Theory of Cryptography*, volume 7194 of *Lecture Notes in Computer Science*, pages 422–439. Springer Berlin Heidelberg, 2012. 64, 76, 78, 82, 84

[144] Bryan Parno, Jon Howell, Craig Gentry, and Mariana Raykova. Pinocchio: Nearly Practical Verifiable Computation. In *Security and Privacy (SP), 2013 IEEE Symposium on*, pages 238–252. IEEE, 2013. 75, 77, 78, 84, 214

[145] Mithun Paul and Ashutosh Saxena. Proof Of Erasability for Ensuring Comprehensive Data Deletion in Cloud Computing. In Natarajan Meghanathan, Selma Boumerdassi, Nabendu Chaki, and Dhinaharan Nagamalai, editors, *Recent Trends in Network Security and Applications*, volume 89 of *Communications in Computer and Information Science*, pages 340–348. Springer Berlin Heidelberg, 2010. 185

[146] S. Pearson, V. Tountopoulos, D. Catteddu, M. Sudholt, R. Molva, C. Reich, S. Fischer-Hubner, C. Millard, V. Lotz, M.G. Jaatun, R. Leenes, Chunming Rong, and J. Lopez. Accountability for Cloud and Other Future Internet Services. In *2012 IEEE 4th International Conference on Cloud Computing Technology and Science (CloudCom)*, pages 629–632, 2012. 157, 163, 218

[147] Daniele Perito and Gene Tsudik. Secure Code Update for Embedded Devices via Proofs of Secure Erasure. In *ESORICS*, volume 6345, pages 643–662. Springer, 2010. 185

[148] Zachary NJ Peterson, Mark Gondree, and Robert Beverly. A Position Paper on Data Sovereignty: The Importance of Geolocating Data in the Cloud. In *Proceedings of the 8th USENIX Conference on Networked Systems Design and Implementation*, 2011. 185

[149] James S Plank and Lihao Xu. Optimizing Cauchy Reed-Solomon codes for fault-tolerant network storage applications. In *Fifth IEEE International Symposium on Network Computing and Applications, 2006. NCA 2006.*, pages 173–180. IEEE, 2006. 28

[150] Piotr Porwik and Agnieszka Lisowska. The Haar-Wavelet Transform in Digital Image Processing: Its Status and Achievements. *Machine graphics and vision*, 13:79–98, 2004. 63

[151] Mark W Powell, Ryan Rossi, Khawaja Shams, *et al.* A Scalable Image Processing Framework for Gigapixel Mars and Other celestial Body Images. In *Aerospace Conference, 2010 IEEE*, pages 1–11. IEEE, 2010. 112

[152] PrimeLife Consortium. PrimeLife, 2011. 169

[153] William Pugh. Skip Lists: a Probabilistic Alternative to Balanced Trees. *Communications of the ACM*, 33(6):668–676, 1990. 24

[154] Irving S. Reed and Gustave Solomon. Polynomial Codes Over Certain Finite Fields. *Journal of the Society of Industrial and Applied Mathematics*, 8(2):300–304, 06/1960 1960. 26, 54

[155] Ronald L Rivest, Adi Shamir, and Yael Tauman. How To Leak A Secret. In *Advances in Cryptology—ASIACRYPT 2001*, pages 552–565. Springer, 2001. 26

[156] David K Ruch and Patrick J Van Fleet. *Wavelet theory: An Elementary Approach with Applications*. John Wiley & Sons, 2011. 99

[157] Ahmad-Reza Sadeghi, Thomas Schneider, and Marcel Winandy. Token-Based Cloud Computing. In *Trust and Trustworthy Computing*, pages 417–429. Springer, 2010. 71

[158] Joshua Schiffman, Thomas Moyer, Hayawardh Vijayakumar, Trent Jaeger, and Patrick McDaniel. Seeding Clouds with Trust Anchors. In *Proceedings of the 2010 ACM Workshop on Cloud Computing Security Workshop*, pages 43–46. ACM, 2010. 71

[159] Thomas SJ Schwarz and Ethan L Miller. Store, Forget, and Check: Using Algebraic Signatures to Check Remotely Administered Storage. In *26th IEEE International Conference on Distributed Computing Systems, 2006. ICDCS 2006*, pages 12–12. IEEE, 2006. 21, 22, 23, 30

[160] Francesc Sebé, Josep Domingo-Ferrer, Antoni Martínez-Balleste, Yves Deswarte, and Jean-Jacques Quisquater. Efficient Remote Data possession Checking in Critical Information Infrastructures. *IEEE Trans. Knowl. Data Eng.*, 20(8):1034–1038, 2008. 21, 30

[161] Srinath Setty, Andrew J Blumberg, and Michael Walfish. Toward Practical and Unconditional Verification of Remote Computations. In *Proceedings of the 13th USENIX Conference on Hot Topics in Operating Systems, HotOS*, volume 13, pages 29–29, 2011. 73, 74, 75, 84

[162] Srinath Setty, Benjamin Braun, Victor Vu, Andrew J Blumberg, Bryan Parno, and Michael Walfish. Resolving the Conflict between Generality and Plausibility in Verified Computation. In *Proceedings of the 8th ACM European Conference on Computer Systems*, pages 71–84. ACM, 2013. 74, 75, 84

[163] Srinath T. V. Setty, Richard McPherson, Andrew J. Blumberg, and Michael Walfish. Making Argument Systems for Outsourced Computation Practical (sometimes). In *19th Annual Network and Distributed System Security Symposium, NDSS 2012, San Diego, California, USA, February 5-8, 2012*, 2012. 73, 74, 75, 84

[164] Srinath TV Setty, Victor Vu, Nikhil Panpalia, Benjamin Braun, Andrew J Blumberg, and Michael Walfish. Taking Proof-Based Verified Computation a Few Steps Closer to Practicality. In *USENIX Security Symposium*, pages 253–268, 2012. 74, 75, 84

[165] Hovav Shacham and Brent Waters. Compact Proofs Of Retrievability. In *Proceedings of the 14th International Conference on the Theory and Application of Cryptology and Information Security: Advances in Cryptology*, ASIACRYPT '08, pages 90–107, Berlin, Heidelberg, 2008. Springer-Verlag. 18, 27, 28, 29, 30, 32, 52, 53, 54, 76

[166] Shiuan-Tzuo Shen and Wen-Guey Tzeng. Delegable Provable Data Possession for Remote Data in the Clouds. In *Information and Communications Security*, pages 93–111. Springer, 2011. 26, 30

[167] Elaine Shi, Emil Stefanov, and Charalampos Papamanthou. Practical Dynamic Proofs of Retrievability. In *ACM Conference on Computer and Communications Security*, pages 325–336, 2013. 28, 30, 57

[168] Sarvjeet Singh and Sunil Prabhakar. Ensuring Correctness Over Untrusted Private Database. In *Proceedings of the 11th International Conference on Extending Database Technology*, pages 476–486. ACM, 2008. 82

[169] Steel, Christopher and Lai, Ray and Naggapan, Ramesh. Core Security Patterns: Identity Management Standards and Technologies . http://www.informit.com/articles/article.aspx?p=1398625&seqNum=12, 2009. xiii, 169

[170] Emil Stefanov, Marten van Dijk, Ari Juels, and Alina Oprea. Iris: a scalable cloud file system with efficient integrity checks. In *ACSAC*, pages 229–238, 2012. 28, 30, 32, 53, 57

[171] Wenhai Sun, Bing Wang, Ning Cao, Ming Li, Wenjing Lou, Y. Thomas Hou, and Hui Li. Verifiable Privacy-Preserving Multi-Keyword Text Search in the Cloud Supporting Similarity-Based Ranking. *IEEE Transactions on Parallel and Distributed Systems*, 25 (11):3025–3035, 2014. 81

[172] Wenhai Sun, Xuefeng Liu, Wenjing Lou, Y Thomas Hou, and Hui Li. Catch You If You Lie To Me: Efficient Verifiable Conjunctive Keyword Search Over Large Dynamic Encrypted Cloud Data. In *IEEE Conference on Computer Communications (INFOCOM), 2015* , pages 2110–2118. IEEE, 2015. 81

[173] Justin Thaler. Time-Optimal Interactive Proofs for Circuit Evaluation. In Ran Canetti and JuanA. Garay, editors, *Advances in Cryptology – CRYPTO 2013*, volume 8043 of *Lecture Notes in Computer Science*, pages 71–89. Springer Berlin Heidelberg, 2013. 72, 80

[174] Top Threats Working Group and others. The Notorious Nine: Cloud Computing Top Threats in 2013. *Cloud Security Alliance*, 2013. 9, 10, 17

[175] Slim Trabelsi, Gregory Neven, Dave Raggett, Claudio Ardagna, Carine Bournez, Laurent Bussard, Michele Bezzi, Jan Camenisch, Sabrina de Capitani di VIMERCATI, Fatih Gey, Aleksandra Kuczerawy, Sebastian Meissner, Gregory Neven, Akram Njeh, Stefano Paraboschi, Eros Pedrini, Sara Foresti, Ulrich Pinsdorf, Franz-Stefan Preiss, Jakub Sendor, Christina Tziviskou, Dave Raggett, Thomas Roessler, Pierangela Samarati, Jan Schallaboeck, Stuart Short, Dieter Sommer, Mario Verdicchio, and Rigo Wenning. D5.3.4 - Report on design and implementation of the PrimeLife Policy Language and Engine. Deliverable, Primelife Project, 2011. 158, 175

[176] Jonathan Trostle and Andy Parrish. Efficient Computationally Private Information Retrieval from Anonymity or Trapdoor Groups. In *Proceedings of Conference on Information Security*, pages 114–128, Boca Raton, USA, 2010. xiii, 37, 38, 50, 51

[177] Joseph D Twicken, Bruce D Clarke, Stephen T Bryson, Peter Tenenbaum, Hayley Wu, Jon M Jenkins, Forrest Girouard, and Todd C Klaus. Photometric Analysis in the Kepler Science Operations Center pipeline. In *SPIE Astronomical Telescopes+ Instrumentation*, pages 774023–774023. International Society for Optics and Photonics, 2010. 63, 85

[178] Marten Van Dijk, Ari Juels, Alina Oprea, Ronald L Rivest, Emil Stefanov, and Nikos Triandopoulos. Hourglass Schemes: How to Prove that Cloud Files Are Encrypted. In *Proceedings of the 2012 ACM Conference on Computer and Communications Security*, pages 265–280. ACM, 2012. 184, 185

[179] Victor Vu, Srujay Setty, Andrew J Blumberg, and Michael Walfish. A Hybrid Architecture for Interactive Verifiable Computation. In *IEEE Symposium on Security and Privacy (SP), 2013*, pages 223–237. IEEE, 2013. 74

[180] Riad S Wahby, Srinath Setty, Zuocheng Ren, Andrew J Blumberg, and Michael Walfish. Efficient RAM and Control Flow in Verifiable Outsourced Computation. In *Proceedings of the ISOC NDSS*, 2015. 74

[181] Michael Walfish and Andrew J. Blumberg. Verifying Computations Without Reexecuting Them. *Communications of the ACM*, 58(2):74–84, January 2015. 70

[182] Boyang Wang, Baochun Li, and Hui Li. Knox: Privacy-Preserving Auditing for Shared Data with Large Groups in the Cloud. In *Applied Cryptography and Network Security*, pages 507–525. Springer, 2012. 26

[183] Boyang Wang, Baochun Li, and Hui Li. Oruta: Privacy-Preserving Public Auditing for Shared Data in the Cloud. In *IEEE 5th International Conference on Cloud Computing (CLOUD)*, pages 295–302. IEEE, 2012. 26, 30

[184] Boyang Wang, Baochun Li, and Hui Li. Panda: Public Auditing for Shared Data with Efficient User Revocation in the Cloud. *IEEE Transactions on Services Computing*, 8 (1):92–106, 2015. 26

[185] Cong Wang, Qian Wang, Kui Ren, and Wenjing Lou. Privacy-preserving public auditing for data storage security in cloud computing. In *INFOCOM*, pages 525–533, 2010. 25, 30

[186] Qian Wang, Cong Wang, Jin Li, Kui Ren, and Wenjing Lou. Enabling public verifiability and data dynamics for storage security in cloud computing. In *Proceedings of the 14th European conference on Research in computer security*, ESORICS'09, pages 355–370, Berlin, Heidelberg, 2009. Springer-Verlag. 25, 26, 30

[187] Gaven J Watson, Reihaneh Safavi-Naini, Mohsen Alimomeni, Michael E Locasto, and Shivaramakrishnan Narayan. LoSt: Location Based Storage. In *Proceedings of the 2012 ACM Workshop on Cloud Computing Security Workshop*, pages 59–70. ACM, 2012. 185

[188] Bernard Wong, Ivan Stoyanov, and Emin Gün Sirer. Octant: A Comprehensive Framework for the Geolocalization of Internet Hosts. In *Proceedings of the 4th USENIX Conference on Networked Systems Design & Implementation*, pages 23–23. USENIX Association, 2007. 185

[189] Jia Xu and Ee-Chien Chang. Towards efficient proofs of retrievability. In *ASIACCS*, pages 79–80, 2012. 27, 30, 52, 53, 54

[190] Yin Yang, Dimitris Papadias, Stavros Papadopoulos, and Panos Kalnis. Authenticated Join Processing in Outsourced Databases. In *Proceedings of the 2009 ACM SIGMOD International Conference on Management of Data*, pages 5–18. ACM, 2009. 82

[191] Andrew C Yao. Protocols for Secure Computations. In *IEEE Annual Symposium on Foundations of Computer Science*. IEEE, 1982. 71, 75, 76

[192] Liang Feng Zhang and Reihaneh Safavi-Naini. Generalized Homomorphic MACs with Efficient Verification. In *Proceedings of the 2nd ACM Workshop on ASIA Public-key Cryptography*, ASIAPKC '14, pages 3–12, New York, NY, USA, 2014. ACM. 77

[193] Liang Feng Zhang and Reihaneh Safavi-Naini. Verifiable Delegation of Computations with Storage-Verification Trade-off. In Mirosław Kutyłowski and Jaideep Vaidya, editors, *Computer Security - ESORICS 2014*, volume 8712 of *Lecture Notes in Computer Science*, pages 112–129. Springer International Publishing, 2014. 78, 79, 84, 95, 109, 214

[194] Yihua Zhang and Marina Blanton. Efficient Dynamic Provable Possession of Remote Data via Balanced Update Trees. In *Proceedings of the 8th ACM SIGSAC Symposium on Information, Computer and Communications Security*, ASIACCS'13, pages 183–194, New York, NY, USA, 2013. ACM. 25, 30

[195] Yihua Zhang and Marina Blanton. Efficient Secure and Verifiable Outsourcing of Matrix Multiplications. Cryptology ePrint Archive, Report 2014/133, 2014. 78, 79, 84, 110, 214

[196] Yupeng Zhang, Jonathan Katz, and Charalampos Papamanthou. IntegriDB: Verifiable SQL for Outsourced Databases. In *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security*, pages 1480–1491. ACM, 2015. 82, 153

[197] Qingji Zheng and Shouhuai Xu. Fair and dynamic proofs of retrievability. In *CODASPY*, pages 237–248, 2011. 29, 30, 58

[198] Qingji Zheng, Shouhuai Xu, and Giuseppe Ateniese. VABKS: Verifiable Attribute-Based Keyword Search over Outsourced Encrypted Data. In *INFOCOM, 2014 Proceedings IEEE*, pages 522–530. IEEE, 2014. 78, 81, 84, 153, 214

Résumé Français

L'informatique nuagique (le *cloud computing*) est considéré comme le “saint graal” pour faire face à la gestion de quantités énormes de données recueillies chaque jour à travers les réseaux sociaux, les appareils mobiles, les réseaux de capteurs, etc. Par conséquent, l'externalisation du stockage et du traitement de ces données vers le *cloud* élimine la nécessité d'investir dans du matériel informatique et logiciel coûteux. Cependant, de nombreuses entreprises sont réticentes à l'idée de recourir à des technologies nuagiques. En effet, l'inévitable transfert de contrôle sur le stockage et le calcul vers des serveurs *cloud* non-fiables soulève divers problèmes de sécurité. À cet égard, la vérification des opérations effectuées par le *cloud* et la possibilité de lui imputer la responsabilité de ses actions peuvent aider les utilisateurs à avoir plus de contrôle sur leurs ressources et peuvent réduire l'impact de la méfiance envers le *cloud*. L'état de l'art décrit des méthodes pour externaliser de façon vérifiable le stockage de données et le calcul sur ces données. Cependant, la plupart des procédés existants impliquent des techniques cryptographiques lourdes qui rendent les solutions inefficaces. En outre, très peu de solutions techniques ont été envisagées pour l'imputabilité des actions effectuées par le *cloud*.

Cette thèse propose de nouveaux protocoles cryptographiques, plus efficaces que l'existant, et qui permettent aux utilisateurs du nuage informatique de vérifier (i) la bonne conservation des données externalisées et (ii) l'exécution correcte de calculs externalisés. Nous décrivons d'abord un protocole cryptographique qui génère des preuves de récupérabilité, qui permettent aux propriétaires de données de vérifier que le *cloud* stocke leurs données correctement. Nous détaillons ensuite trois schémas cryptographiques pour vérifier l'exactitude des calculs externalisés en se focalisant sur trois opérations fréquentes dans les procédures de traitement de données, à savoir l'évaluation de polynômes, la multiplication de matrices et la recherche de conjonction de mots-clés. La sécurité de nos solutions est analysée dans le cadre de la sécurité prouvable et nous démontrons également leur efficacité grâce à des prototypes. Nous présentons également A-PPL, un langage de la politique pour l'imputabilité qui permet l'expression des obligations de responsabilité et de traçabilité dans un format compréhensible par la machine. Nous espérons que nos contributions pourront encourager l'adoption du *cloud* par les entreprises encore réticentes à l'idée d'utiliser ce paradigme prometteur.

Introduction

Le terme “informatique nuagique” (en anglais *cloud computing* ou juste *cloud*) est sans doute l'un des concepts les plus populaires dans le monde des technologies de l'information et de la communication (TICs) en ce début du XXI$^{\text{e}}$ siècle. L'Institut américain des Normes et de la Technologie (NIST) a fait paraître en 2011 sa définition du *cloud*, souvent citée dans les articles scientifiques du domaine des TICs:

> Le *cloud computing* est un modèle permettant l'accès à la demande, *via* le réseau, à un ensemble de ressources informatiques (réseaux, serveurs, stockage, applications et services), configurables et partagées, et qui peuvent être rapidement mises à disposition avec un effort ou une intervention du fournisseur de service minimum.

En d'autres termes, l'informatique en nuage possède les caractéristiques suivantes :

Service à la demande : les utilisateurs peuvent accéder aux services du *cloud* à la demande;

Accessibilité : le *cloud* est accessible sur l'ensemble du réseau, quelque soit l'appareil utilisé pour l'accès;

Mutualisation des ressources : les services du *cloud* utilisent des ressources mutualisées qui sont mises à la disposition de plusiers utilisateurs;

Élasticité : ces services doivent être élastiques, c'est-à-dire qu'ils s'adaptent rapidement aux variations des besoins des utilisateurs.

Les bénéfices du *cloud*

Utiliser les ressources du *cloud* présente de nombreux avantages aux utilisateurs (que ce soient des organisations ou des individus) :

La réduction des coûts : grâce à la mutualisation des ressources, les entreprises n'ont plus besoin d'investir dans des insfrastructures pour le stockage et la puissance de calcul;

Des accès rapides et des usages facilités : les services *cloud* sont accessibles n'importe quand, n'importe où, sur tout support via l'Internet;

La disponibilité du service : le *cloud* permet d'assurer à ses utilisateurs des services hautement réactifs avec peu de latence;

La flexibilité : grâce à la propriété d'élasticité du *cloud*, les utilisateurs peuvent adapter leurs demandes en ressources *cloud* en temps réel et ne payer que les ressources qu'ils consomment (*pay-as-you-go*).

Les problèmes de sécurité et de vie privée liés au *cloud*

Cependant, malgré tous ces avantages prometteurs, nombreux sont les organisations et les utilisateurs qui sont encore réticents à l'idée de migrer leurs données vers le *cloud*. La plupart des obstacles à une adoption généralisée du *cloud* découle d'une combinaison de deux facteurs interconnectés, à savoir la perte de contrôle et la méfiance envers le *cloud*:

- Les utilisateurs du *cloud* perdent le contrôle sur leurs données et leur traitement en les transférant aux fournisseurs de services. En effet, en externalisant leurs données et leurs opérations au *cloud*, les utilisateurs ne les possèdent plus et doivent compter sur les fournisseurs du *cloud* pour implémenter les contrôles requis sur de nombreux aspects comme le stockage, l'accès, l'usage, la confidentialité, l'intégrité ou la disponibilité des données. Cette perte de contrôle est critique pour des sociétés ou des organismes devant respecter des obligations réglementaires comme le Règlement général sur la protection des données [80], la loi américaine *Health Insurance Portability and Accountability Act* [1] ou la directive européene EuroSox [79]. Donc, la perte de contrôle sur les données dans le *cloud* implique un défi de conformité en rapport avec ces réglementations.

- Les freins à l'adoption généralisée du *cloud* sont également liés à la méfiance à l'égard des fournisseurs de *cloud*. Cette méfiance s'exprime selon deux aspects. D'abord, des défaillances involontaires dans les services du *cloud* peuvent causer des pertes de données ou une indisponibilité du service qui peuvent être critiques pour les organismes qui dépendent de ces services. Ensuite, les fournisseurs de *cloud* sont eux-mêmes considérés comme malveillants et peuvent adopter deux types de comportements malintentionnés. (i) Un *cloud* curieux met en péril la confidentialité des données et des calculs, par exemple en effectuant des operations d'extraction de données pour acquérir de précieuses informations. Ce scénario est envisageable pour le cas des nombreuses données sensibles et personnelles générées ou collectées par les réseaux sociaux du genre Facebook, Twitter ou LinkedIn. (ii) Un *cloud* malveillant fait volontairement mauvais usage de ses ressources pour compromettre la confidentialité, l'intégrité ou la disponibilité des données et des opérations dont il a la charge. Par ailleurs, le manque de confiance envers les fournisseurs de *cloud* est également lié à l'absence de transparence sur le traitement des données et des calculs, ainsi que sur leur localisation et leur but. Des fournisseurs de *cloud* pourraient par exemple être tentés de cacher à leurs utilisateurs une défaillance du sytème ou une fuite de données pour préserver leur réputation.

En résumé, le transfert de contrôle vers un fournisseur de *cloud* non digne de confiance pose de sérieux problèmes liés à la sécurité et à la préservation de la vie privée des données et des opérations des utilisateurs. Ces problèmes sont corrélés avec des problèmes de transparence, de responsabilité et de conformité.

Malheureusement, les caractéristiques singulières du *cloud* compromettent l'utilisation directe des mécanismes traditionnels de sécurité comme le chiffrement pour garantir la confidentialité des données, les techniques cryptographiques pour garantir l'intégrité des données et des calculs, l'isolation de code pour des opérations sécurisées, etc. Cela est dû à plusieurs enjeux spécifiques au *cloud*:

1. La prolifération de données due au *big data* et le nombre croissant d'appareils portatifs incitent les particuliers et les sociétés à utiliser les services de *cloud* pour leur confier leurs données et leurs applications.
2. Cela implique que les utilisateurs de *cloud* perdent le contrôle sur leurs données et transfèrent ce contrôle au fournisseur de service. Donc les utilisateurs doivent compter sur le fournisseur du *cloud* pour implémenter les mesures de sécurité adéquates.
3. Les exigences de conformité réglementaire et contractuelle rendent difficile l'application de mesures de sécurité dans le *cloud*, particulièrement les règles et les lois qui contrôlent le stockage et l'usage des données. Cela suppose que la manière dont le *cloud* stocke et traite les données externalisées doit être vérifiée afin de déterminer que le *cloud* suit les exigences de conformité. Dans une optique plus large, la conformité implique la possibilité d'attribuer au fournisseur de service *cloud* la responsabilité de l'ensemble de ses actions. Ce dernier concept sous-entend l'imputabilité du *cloud*.

En outre, les mesures de sécurité dans le *cloud* doivent être conçues de telle sorte que l'externalisation des données et des applications reste une solution intéressante pour les utilisateurs.

Problématique

Cette thèse tente de répondre à ces trois problèmes :

Problème 1: Stockage vérifiable. La *vérifiabilité* est un concept d'importance dans le *cloud*. Le but est de concevoir des mécanismes utilisés par les utilisateurs afin de contrôler et de vérifier que le *cloud* fournit correctement les services de stockage et de calcul. Il est légitime de croire que les utilisateurs qui consomment et paient ces services s'attendent à ce que leurs données soient correctement stockées et que leurs opérations soient correctement effectuées. Le concept de vérifiabilité appuie celui de *transparence* : vérifier les actions du *cloud* permet de connaître les contrôles que le *cloud* met en place pour gérer les données et les opérations externalisées.

Cette thèse se focalise en particulier sur deux aspects de la vérifiabilité dans le contexte du nuage informatique, à savoir le stockage et le calcul. Nous abordons en premier lieu le problème lié au stokage : un utilisateur confie ses données au *cloud* et s'attend à ce que celui-ci les stocke correctement. Par conséquent, l'utilisateur veut vérifier l'intégrité de ses données. En d'autres termes, l'utilisateur doit être convaincu que les données ne sont pas supprimées, ni modifiées. Pour vérifier cette proporiété, l'utilisateur doit pouvoir vérifier que le *cloud* stocke correctement ses données. Autrement dit, le *cloud* doit produire des **preuves de stockage** qui permettent d'affirmer que celui-ci stocke réellement une version intacte des données. Par ailleurs, la vérification de ces preuves par l'utilisateur doit être efficace. Cela veut dire qu'elle ne doit pas générer des coûts prohibitifs pour l'utilisateur.

Problème 2: Calcul vérifiable. Conjointement à l'externalisation de données, l'infonuagique permet l'externalisation de calcul : les utilisateurs peuvent déléguer au *cloud* l'exécution d'opérations coûteuses. Dans ce cas, les utilisateurs doivent être convaincus que le *cloud* retournera toujours des résultats corrects, c'est-à-dire que les résultats fournis par le *cloud* sont ceux que les utilisateurs auraient obtenu s'ils avaient eux-mêmes effectué le calcul externalisé. En d'autres termes, les utilisateurs doivent être capables de vérifier que le *cloud* renvoie le bon résultat. Dans cette optique, le *cloud* doit pouvoir produire des **preuves de calcul** garantissant aux utilisateurs que les valeurs retournées par le *cloud* correspondent à une exécution correcte de l'opération externalisée. Par ailleurs, la vérification des ces preuves doit être nettement moins exigeante en termes de calcul que l'opération externalisée elle-même. Autrement, confier l'opération au *cloud* n'apporte aucun bénéfice.

Problème 3: Imputabilité. Outre la vérifiabilité, nous étudions le concept plus large d'imputabilité pour l'informatique nuagique, qui est correlé aux notions de tranparence, de responsabilité et de conformité. Les fournisseurs de *cloud* doivent se conformer à des législations et des contrats. De plus, ils doivent rendre des comptes et être tenus responsables en ce qui concerne leurs façons de gérer et de traiter les données des utilisateurs. En d'autres termes, un ensemble d'obligations lie les utilisateurs et les fournisseurs de *cloud* de sorte que le *cloud* fonctionne de façon transparente. Nous nous intéressons particulièrent aux politiques pour exprimer ces obligations d'imputabilité. Il n'existe actuellement aucun cadre permettant aux utilisateurs d'appréhender la façon avec laquelle le *cloud* honore ses obligations d'imputabilité. Nous pensons que les politiques d'imputabilité fournissent un moyen d'exprimer les obligations. C'est pourquoi nous étudions la possibilité de concevoir un langage de politique interprétable par la machine de telle sorte que les politiques écrites avec ce langage soient facilement appliquées de manière automatique.

Contributions

La thèse propose les contributions suivantes en rapport avec les trois problèmes énoncés plus haut.

Preuves de Stockage. Sous l'hypothèse d'un fournisseur de *cloud* malveillant, nous concevons un protocole qui produit des preuves cryptographiques prouvant que les données externalisées sont correctement stockées. Le protocole naïf, dans lequel le propriétaire de la donnée stocke cette donnée avec une signature électronique chez le *cloud* et, pour vérifier que celui-ci stocke la donnée de façon conforme, la télécharge et vérifie la signature, ne serait pas très efficace dans le contexte de l'infonuagique et du *big data*. Donc les solutions de preuves de stockage doivent être plus efficaces que ce simple mécanisme. Pour cette raison, nous proposons $\mathcal{S}$tealthGuard, notre protocole de preuve de stockage, basé sur l'idée d'insérer dans la donnée des blocs spéciaux appelés "chiens de garde".

Preuves de Calcul. Sous l'hypothèse d'un fournissuer de *cloud* non digne de confiance, il est possible de déléguer et de vérifier les résultats d'une opération couteûse. Le serveur *cloud* doit envoyer les résultats avec une preuve cryptographique garantissant que l'opération a été effectuée correctement. Nous concevons trois protocoles dans lesquels la vérification de la preuve opère de manière efficace, c'est-à-dire de telle sorte que la vérification de la preuve prend nettement moins de temps que l'exécution de l'opération. Ces trois protocoles concernent trois types d'opérations, fréquemment utilisées dans des programmes d'exploration de données, à savoir l'évaluation de polynomes, la multiplication matricielle et la recherche de conjonction de mots-clés. Ces trois solutions sont basées sur de simples propriétés mathématiques et d'outils cryptographiques bien connues, rendant nos protocoles plus efficaces que l'état de l'art. Par ailleurs, nos solutions se singularisent par le fait qu'elles permettents deux propriétés intéressantes : la délégation publique (n'importe qui, et pas seulement la personne qui a externalisé la fonction peut solliciter le *cloud* d'effectuer un calcul) et la vérifiabilité publique (n'importe qui, et pas seulement l'utilisateur qui a soumis la requête, peut vérifier les résultats retournés par le *cloud*).

Langage de Politique pour l'Imputabilité. Nous concevons A-PPL, un langage de politique qui permet d'exprimer des obligations d'imputabilité qui conditionnent les opérations par le *cloud* sur les données externalisées. Ce langage de politique est compréhensible et interprétable par la machine afin de faciliter l'automatisation de la mise en oeuvre de ces politiques. Nous élaborons également A-PPLE, le moteur qui permet de mettre en pratique les politiques d'imputabilité exprimées avec A-PPL.

1 Preuves de Stockage

La perte de données est une des plus grandes menaces dans le *cloud*. Le terme de *perte de données* inclut non seulement la suppression non autorisée de données, mais aussi la modification irréversible de toute ou partie des données. En d'autres termes, la perte de données compromet l'intégrité et la disponobilité de ces données.

Les propriétaires des données confiées au *cloud* devraient pouvoir vérifier que le fournisseur de *cloud* les stocke correctement, c'est-à-dire, vérifier que les données sont intactes et disponibles tout au long de la période de stockage. Cette problématique est abordée dans le domaine de la recherche en **preuves de stockage**. Ces preuves permettent au propriétaire de données de les confier au *cloud* tout en ayant la capacité de vérifier que le *cloud* les stocke correctement. Les preuves de stockage sont des preuves cryptographiques qui sont générées et vérifiées dans le contexte d'un protocole entre le propriétaire de la donnée et le *cloud*.

1.1 Définition d'un Protocole de Preuves de Stockage

Ce type de protocole fait participer trois acteurs :

Le Propriétaire des données $\mathcal{O}$: Il souhaite confier le stockage d'une liste de documents $\mathcal{F}$ à un serveur de *cloud* $\mathcal{S}$ et souhaite obtenir de $\mathcal{S}$ l'assurance de l'intégrité de ses documents.

Le serveur de *cloud* $\mathcal{S}$: Considéré comme potentiellement malveillant, le serveur est censé stocker chaque fichier $F \in \mathcal{F}$ dans son intégralité. En pratique, $\mathcal{S}$ stocke une version vérifiable $\hat{F}$ du fichier F de sorte que $\mathcal{S}$ puisse produire des preuves montrant que F est correctement stocké.

Le Vérificateur $\mathcal{V}$: Pour le compte du propriétaire $\mathcal{O}$ de la donnée externalisé, le vérificateur $\mathcal{V}$ interagit avec $\mathcal{S}$ pour vérifier si $\mathcal{S}$ stocke le fichier $F \in \mathcal{F}$. Ce rôle peut être joué par $\mathcal{O}$ lui-même ou par n'importe quelle entité habilitée.

Sans perte de généralité, nous supposons que chaque fichier $F \in \mathcal{F}$ est composé de n sections $\{S_1, S_2, ..., S_n\}$ de tailles égales (L bits). Chacune de ces sections S_i se compose de

m blocs $\{b_{i1}, b_{i2}, ..., b_{im}\}$ de l bits. Le protocole de Preuves de Stockage est défini par cinq algorithmes répartis en trois phases:

Definition 22 (Protocole de Preuves de Stockage). *Nous donnons ici la définition d'un tel protocole :*

► **Configuration.** *Cette phase implique le propriétaire des données $\mathcal{O}$. Il exécute l'algorithme* KeyGen *qui génère les clés requises pour le déroulement du protocole puis $\mathcal{O}$ invoque l'algorithme* Encode *qui prépare une version vérifiable $\hat{F}$ d'un fichier $F \in \mathcal{F}$:*

▷KeyGen$(1^\kappa) \to K$: *Cet algorithme probabiliste de génération de clés prend en entrée le paramètre de sécurité 1^κ et renvoie en sortie une clé secrète $K \in \{0,1\}^*$.*

▷Encode$(K, F) \to (\mathsf{fid}, \hat{F})$: *Cet algorithme prend pour paramètres la clé K et le fichier $F = \{S_1, S_2, ..., S_n\}$ et retourne le fichier vérifiable $\hat{F} = \{\hat{S}_1, \hat{S}_2, ..., \hat{S}_n\}$ et l'unique identifiant* fid *de F.*

À la fin de la phase de configuration, le serveur $\mathcal{S}$ est censé stocker le fichier $\hat{F}$ avec son identifiant fid*, tandis que le propriétaire de la donnée $\mathcal{O}$ supprime F de son stockage local et ne conserve que la clé générée par* KeyGen*.*

► **Challenge.** *La phase Challenge consiste en plusieurs interactions entre le vérificateur $\mathcal{V}$ et le server $\mathcal{S}$. En substance, $\mathcal{V}$ exécute l'algorithme* Challenge *qui génère des requêtes envoyées à $\mathcal{S}$ pour des preuves de stockage, afin de vérifier l'intégrité du fichier F. En retour, le serveur invoque l'algorithme* ProofGen *qui répond aux requêtes envoyées par $\mathcal{V}$ en produisant les preuves demandées.*

▷Challenge$(K, \mathsf{fid}) \to \mathsf{chal}$: *Cet algorithme probabiliste génère une requête* chal *pour le fichier F dont l'identifiant correspond à* fid*. L'algorithme prend en entrée la clé secrète K et l'identifiant* fid*, et retourne la requête* chal*.*

▷ProofGen$(\mathsf{fid}, \mathsf{chal}) \to \mathcal{P}$: *Cet algorithme est invoqué par le serveur $\mathcal{S}$ pour générer la preuve de stockage $\mathcal{P}$ pour le fichier ciblé $\hat{F}$ dont l'identifiant est* fid*.*

La preuve produite $\mathcal{P}$ est ensuite envoyée au verificateur $\mathcal{V}$ pour vérification.

► **Vérification.** *Après avoir reçu les preuves de stockage pour le fichier F, $\mathcal{V}$ exécute l'algorithme* Verify *pour vérifier leur validité.*

▷Verify$(K, \mathsf{fid}, \mathsf{chal}, \mathcal{P}) \to b \in \{0,1\}$: *Cet algorithme décide si $\mathcal{P}$ est une réponse valide à la requête* chal*. Il prend en paramètre la clé K, l'identifiant* fid*, la requête* chal *et la preuve $\mathcal{P}$. L'algorithme retourne le bit $b = 1$ si la preuve $\mathcal{P}$ est valide, $b = 0$ autrement.*

Les protocoles de preuves de stockage doivent satisfaire les exigences suivantes :

La sécurité : Le protocole doit être robuste, même en cas d'un *cloud* malveillant, qui clamerait à tort qu'il stocke la donnée intacte.

Un nombre illimité de vérifications : Les données doivent être vérifiables autant de fois qu'elles sont confiées au *cloud*.

L'efficacité : Quatre indicateurs permettent de mesurer l'efficacité du protocole : (i) la consomnation de bande passante, (ii) la complexité de calcul de l'algorithme Verify, (iii) la complexité de calcul de l'algorithme ProofGen, et (iv) la quantité de stockage requis chez le propriétaire de la donnée.

En complément de ces exigences, un protocole de preuves de stockage peut présenter certaines propriétés intéressantes :

Extractabilité : Certains protocoles permettent de récupérer la donnée en plus de vérifier son intégrité. Ce sont les protocoles de **preuves de récupérabilité**.

1.2 État de l'art

De nombreux protocoles de preuves de stockage ont été identifiés et analysés dans l'état de l'art. Il existe deux grands types protocoles de preuves de stockage : les protocoles de possession de données prouvables (Provable Data Possession (PDP)) et les protocoles de preuves de récupérabilité (Proofs of Retrievability (POR)). Les PDP ne satisfont pas la propriété d'extractabilité : les propriétaires de données n'obtiennent l'assurance que seulement une partie de leurs données est intacte chez le *cloud*. Ces protocoles ont été conçus pour la première fois par Ateniese *et al.* [14]. Les protocoles de POR en revanche permettent d'assurer que la donnée peut être récupérée dans son intégralité. En particulier, les protocoles de POR emploient des codes correcteurs d'erreurs (ECC) qui permettent de corriger des erreurs dans la donnée.

En nous basant sur l'analyse de l'état de l'art, nous proposons $\mathcal{S}$tealthGuard, un nouveau protocole de preuves de récupérabilité.

2 Preuves de Récupérabilité : StealthGuard

Dans ce chapitre nous présentons $\mathcal{S}$tealthGuard, un nouveau protocole de preuves de récupérabilité qui combine l'utilisation d'un algorithme de recherche de mots préservant la vie privée (PPWS) et l'insertion dans les données à externaliser de courtes séquences de bits générées aléatoirements et appelées *chiens de garde*. Notre protocole poursuit une idée déjà proposée par Juels and Kaliski [107] reposant sur l'insertion à des positions aléatoires dans la donnée à externaliser de blocs aléatoires particuliers appelés chiens de garde. Les preuves de récupérabilité consistent alors à vérifier que certains de ces blocs sont toujours intacts dans les données externalisées. Nous considérons la notion de *récupérabilité* comme étant une combinaison des concepts d'intégrité et de disponibilité des données. Les preuves de récupérabilité (POR) sont un cas spécial des preuves de stockages et donc héritent des propriétés précedemment énoncées.

2.1 Modèle de sécurité

Un protocole de POR doit être complet et robuste. L'exigence de complétude signifie que le protocole ne produit aucun faux négatif, c'est-à-dire qu'un vérificateur accepte toujours une preuve construite par un serveur honnête. La robustesse caractérise le fait qu'il est impossible pour un serveur malveillant de faire accepter par un vérificateur des preuves de récupérabilité contrefaites.

2.2 Aperçu du protocole

En substance, lors de la phase de configuration, pour préparer une version vérifiable $\hat{F}$ d'un fichier F, le propriétaire de ce fichier $\mathcal{O}$ exécute l'algorithme Encode qui chiffre le fichier et y insère les chiens de garde générés pseudo-aléatoirement. Le chiffrement garantit que les blocs de données et les blocs de chiens de garde sont indifférentiables. Par ailleurs, Encode applique un code correcteur d'erreur sur le fichier pour rendre possible la récupération du fichier corrompu par de "petites" erreurs. Une fois que la donnée est confiée au serveur $\mathcal{S}$, le vérificateur $\mathcal{V}$ qui souhaite vérifier la récupérabilité du fichier F interagit avec $\mathcal{S}$ dans la phase de challenge. $\mathcal{V}$ exécute l'algoritme $\mathsf{Challenge}$ qui génère des requêtes pour plusieurs chiens de garde, afin de vérifier qu'ils sont toujours intacts dans le fichier F stocké chez le serveur $\mathcal{S}$. En réponse, celui-ci invoque l'algorithme ProofGen qui traite ces requêtes et produit les preuves pour les chiens de garde ciblés. Si une partie du fichier est altérée, alors ces altérations affecteront aussi les chiens de garde avec une grande probabilité.

Notre solution diffère de celle proposée par Juels and Kaliski [107] dans le génération de la preuve avec l'algorithme ProofGen. Dans l'article [107], le vérificateur $\mathcal{V}$ choisit un ensemble de chiens de garde et envoie leurs positions supposées au serveur $\mathcal{S}$ qui retourne les blocs correspondant à ces positions. Ensuite, $\mathcal{V}$ vérifie que les blocs reçus sont bien les chiens de garde demandés. Cette solution présentée dans [107] ne permet cependant pas un nombre illimité de vérifications car en révélant la position des chiens de garde au serveur, ces chiens de garde ne peuvent plus être utilisés pour des vérifications futures. Donc le serveur ne pourrait garder que les blocs correspondant aux chiens de garde, supprimer les blocs de données et produire des preuves de récupérabilité correctes alors même que le serveur ne stocke pas la donnée dans son intégralité.

Pour faire face à ce problème, notre protocole $\mathcal{S}$tealthGuard utilise un algorithme de recherche de mot préservant la vie pricée en combinaison des chiens de garde. Ce type d'algorithme permet de garantir que le serveur ne découvrira pas quels chiens de garde sont ciblés par des requêtes de recherche. Par conséquent, $\mathcal{V}$ peut envoyer un nombre illimité de requêtes, même pour le même chien de garde, sans le besoin de mettre à jour les chiens de garde déjà consommés. De plus, les résultats de la recherche sont illisibles par le serveur qui ne pourra donc pas savoir si le chien de garde a bien été récupéré ou non. En conséquence, le seul moyen pour le serveur de convaincre $\mathcal{V}$ de la récupérabilité du fichier F est de retourner des résultats de recherche valides, c'est-à-dire en stockant F dans son intégralité et en exécutant la recherche correctement.

2.3 Détails du protocole

Comme tout protocole de preuves de stockage, $\mathcal{S}$tealthGuard se divise en trois phases : configuration, challenge et vérification. Nous donnons dans les lignes qui suivent les détails des opérations effectuées dans chacune des phases.

2.3.1 Configuration

Cette phase prépare une version vérifiable $\hat{F}$ du fichier F à confier au *cloud*. Nous invitons le lecteur à se reporter à la Figure 7.13.

KeyGen : Le propriétaire du fichier $\mathcal{O}$ exécute cet algorithme pour générer plusieurs clés : un clé principale K et $n+3$ clés dérivant de K, à savoir K_{enc}, K_{wdog}, $K_{\mathsf{perm}F}$ et n clés $K_{\mathsf{perm}S_i}$.

Encode : Le proprétaire du fichier $\mathcal{O}$ exécute cet algorithme pour générer d'abord un unique identifiant fid pour le fichier F. Puis, Encode segmente F en n sections $\{S_1, S_2, ..., S_n\}$, où chaque section est divisée en m blocs $\{b_{i1}, b_{i2}, ..., b_{im}\}$. Ensuite, l'algorithme Encode traite le fichier F en plusieurs opérations : (i) l'application d'un code correcteur d'erreurs sur chaque section; (ii) une permutation appliquée à l'ensemble des blocs du fichier, blocs de parité inclus; (iii) le chiffrement de chaque bloc de fichier en utilisant un chiffrement sûr sémantiquement afin de rendre les blocs de données et les chiens de garde indifférentiables; (iv) la création des chiens de garde de façon pseudo-aléatoire; (v) l'insertion de ces chiens de garde fraîchement créés dans des positions choisies pseudo-aléatoirement dans le fichier F.

2.3.2 Challenge

Une fois que le fichier vérifiable $\hat{F}$ est confié au *cloud*, $\mathcal{V}$ souhaite vérifier sa récupérabilité. Pour ce faire, $\mathcal{V}$ crée des requêtes de recherche pour un certain nombre de chiens de garde

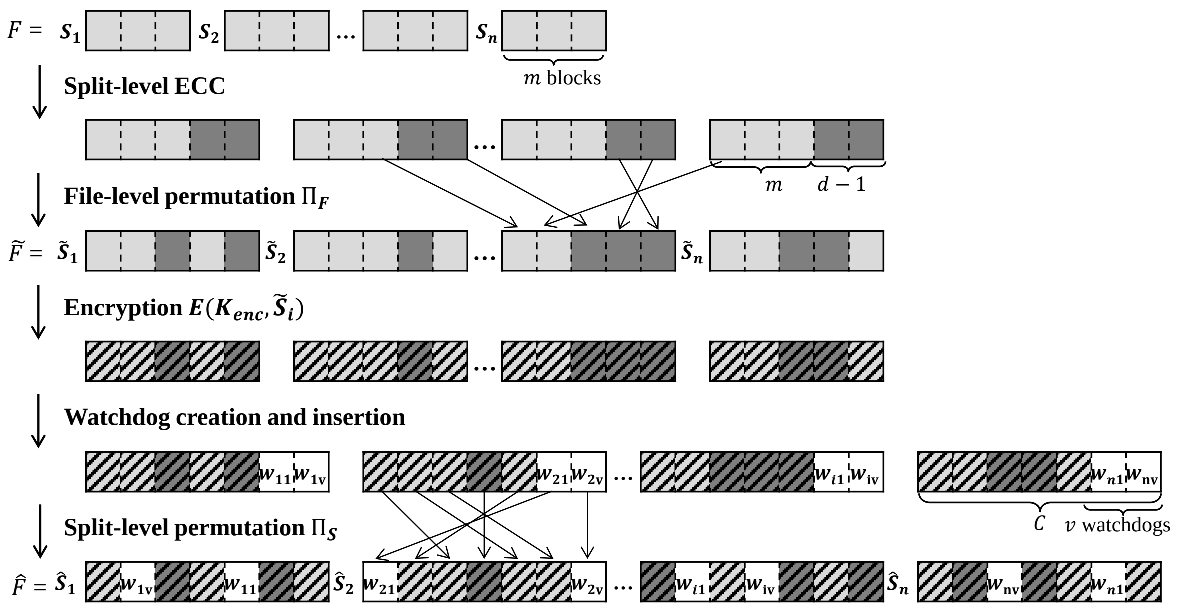

$F = S_1$ S_2 S_n
m blocks
Split-level ECC
m $d-1$
File-level permutation Π_F
$\tilde{F} = \tilde{S}_1$ $\tilde{S}_2$ $\tilde{S}_n$
Encryption $E(K_{enc}, \tilde{S}_i)$
Watchdog creation and insertion
w_{11} w_{1v} w_{21} w_{2v} w_{i1} w_{iv} w_{n1} w_{nv}
C v watchdogs
Split-level permutation Π_S
$\hat{F} = \hat{S}_1$ $\hat{S}_2$ $\hat{S}_n$

Figure 7.13: Configuration de $\mathcal{S}$tealthGuard

sélectionnés aléatoirement. Grâce à un algorithme de recherche privée, le serveur $\mathcal{S}$ traite ces requêtes sans savoir quels chiens de garde il doit rechercher ni où ceux-ci sont localisés dans le fichier. Notre algorithme de recherche, WDSearch, s'inspire d'un algorithme existant appelé Prism [37], lui-même basé sur un mécanisme cryptographique d'extraction privée (PIR).

Avant de décrire comment $\mathcal{V}$ crée une requête de recherche pour un chien de garde, voyons d'abord comment $\mathcal{S}$ traite cette requête. $\mathcal{S}$ exécute l'algorithme ProofGen. Cet algorithme crée pour chaque requête reçue un nombre q d'index de recherche correspondant à q matrices dont le nombre d'éléments est égal au nombre de blocs contenus dans une section S_i du fichier $\hat{F}$. Chacune de ces matrices de recherche est remplie avec des bits témoingant de la présence des blocs dans la section ciblée par la requête : une position (donc un bit) dans chaque matrice correspond à un bloc dans la section. Ce bit est calculé à l'aide d'une fonction de hachage prenant en entrée le bloc correspond à la positoin courante et un *nonce* cryptographique. Le nombre q correspond au nombre q de bits requis pour la sécurité du protocole.

En se basant sur la requête de recherche générée par l'algorithme Challenge exécuté par $\mathcal{V}$, ProofGen récupére de maniére privée la valeur des bits à la position correspondant au chien de garde recherché. Cette opération s'appuie sur l'algorithme d'extraction privée sous-jacent notre algorithme de recherche privée. En conséquence, l'algorithme Challenge crée une requête de recherche privée qui traduit, de façon chiffrée, le fait que le serveur doit récupérer les bits correspondant à la position du chien de garde ciblé par la requête. Finalement, la preuve de récupérabilité comprend les q bits-témoins correspondant au chien de garde.

2.3.3 Vérification

Une fois que $\mathcal{V}$ reçoit de la part de $\mathcal{S}$ la preuve de récupérabilité, la phase de vérification permet vérifier que ladite preuve est valide. Cette opération consiste à vérifier que les bits retournés par $\mathcal{S}$ sont ceux attendus par $\mathcal{V}$.

2.4 Analyse de sécurité

Nous prouvons dans l'analyse de sécurité que notre protocole $\mathcal{S}$tealthGuard est complet et robuste. En particulier, dans le cas de la robustesse de notre solution, nous montrons que le vérificateur doit créer un nombre γ de requêtes de recherche de chiens de garde afin de pouvoir décider de la récupérabilité du fichier confié au *cloud* avec une probabilité proche de

1. Ce nombre γ dépend de plusieurs facteurs parmi lesquels : le paramètre de sécurité de notre schéma, le taux de correction du code correcteur d'erreur ainsi que le nombre de blocs contenus dans chaque section du fichier externalisé.

3 Calcul Vérifiable

L'avènement de l'informatique nuagique offre aux particuliers et aux entreprises un paradigme non seulement pour externaliser le stockage de leurs données potentiellement considérables mais aussi l'exécution d'opérations très demandeuses en puissance de calcul.

Cependant, externaliser ces opérations peut compromettre leur confidentialité et intégrité, ce qui peut dissuader finalement l'adoption des technologies du *cloud*. En effet, comme mentionné plus hat, le *cloud* n'est pas digne de confiance. Un des problémes rencontrés est l'intégrité des calculs externalisés. En particulier, nous consdérons le scénario suivant : un utilisateur souhaite déléguer au *cloud* l'exécution d'une operation $\mathfrak{f}$ de telle sorte que cet utilisateur peut soumettre des valeurs d'entrée x et recevoir du *cloud* le résultat $y = \mathfrak{f}(x)$. Le problème soulevé dans ce scénario est le suivant : comment l'utilisateur peut-il être sûr que y correspond de façon légitime à l'exécution de $\mathfrak{f}$ avec l'entrée x ? En d'autres termes, le *cloud* doit non seulement exécuter correctement la fonction demandée mais aussi convaincre lútilisateur que le résultat est correct. Le défi pour répondre à cette problématique réside dans le fait que lútilisateur cède la fonction $\mathfrak{f}$ au *cloud*. Donc la solution hypothétique selon laquelle l'utilisateur recalcule $y^* = \mathfrak{f}(x)$ et ensuite compare $y^* = y$ ne peut pas être considérée. Par ailleurs, cette solution triviale ne serait pas efficace car recalculer y^* serait aussi coûteux que calculer y, ce qui annule l'intêret d'externaliser la fonction $\mathfrak{f}$ au départ. Pour faire face à ces défis, nous allons concevoir des protocoles pour le calcul vérifiable dans lesquels le *cloud* peut convaincre un utilisateur de l'exactitude des calculs de telle sorte qu'il est toujours plus profitable pour l'utilisateur de déléguer la fonction au *cloud* plutôt que de la calculer chez lui. Gennaro *et al.* [90] a formalisé le concept de calcul vérifiable, où l'utilisateur délègue l'exécution d'une fonction au *cloud* et reçoit le résultat accompagné d'une preuve cryptographique assurant l'exécution correcte de l'opération demandée.

Un protocole de calcul vérifiable doit répondre à plusieurs exigences :

Efficacité : Pour ne pas annuler le bénéfice d'externaliser une fonction au *cloud*, le coût pour l'utilisateur de soumettre une valeur d'entrée et de vérifier la preuve de calcul doit être nettement moindre que celui d'exécuter la fonction localement. De plus, les protocoles de calcul vérifiable adoptent un modèle d'amortissement qui autorise l'utilisateur à exécuter une phase de configuration, coûteuse mais unique, préparant la fonction avant son externalisation. Cette phase de configuration est ensuite amortie avec un nombre illimité de vérifications de résultats rapides.

Sécurité : Les preuves de calcul doivent satisfaire deux propriétés classiques en cryptographie : l'exactitude (un serveur honnête ne peut pas être accusé d'avoir mal exécuté la fonction) et la robustesse (si le serveur dévie d'une correcte exécution de la fonction externalisée alors il ne pourra pas créer des preuves factices qu'un vérificateur acceptera).

Autres propriétés : Dans certaines applications où des données publiques sont impliquées, un protocol de calcul vérifiable peut présenter les propriétés de *délégation publique* (n'importe qui peut soumettre une valeur d'entrée qu *cloud* pour la fonction externalisée) et de *vérification publique* (n'importe qui peut vérifier un résultat retourné par le *cloud*).

3.1 Définition d'un protocole de calcul vérifiable public

Quatre acteurs sont impliqués dans ce type de protocole :

Le propriétaire $\mathcal{O}$: $\mathcal{O}$ externalise l'exécution d'une fonction $\mathfrak{f}$ appartenant à une famille de fonctions $\mathcal{F}$ à un serveur *cloud* nommé $\mathcal{S}$. $\mathcal{O}$ produit une clé d'évaluation $\mathsf{EK}_{\mathfrak{f}}$ utilisée par $\mathcal{S}$ pour réponde à n'importe quelle requête de calcul sur $\mathfrak{f}$. De plus, $\mathcal{O}$ peut déléguer à n'importe qui la capacité de soumettre une valeur d'entrée et de vérifier le calcul : il publie une clé publique $\mathsf{PK}_{\mathfrak{f}}$ qui permet de créer des requêtes de calcul pour la fonction $\mathfrak{f}$.

Le serveur $\mathcal{S}$: $\mathcal{S}$ est censé exécuter la fonction $\mathfrak{f}$ sur une valeur d'entrée x et produire une preuve que la valeur de sortie $\mathfrak{f}(x)$ est correcte.

Le requêteur $\mathcal{Q}$: Ayant accès la clé publique $\mathsf{PK}_{\mathfrak{f}}$, l'utilisateur $\mathcal{Q}$ sollicite $\mathcal{S}$ pour exécuter la fonction $\mathfrak{f}$ sur l'entrée x du domaine $\mathcal{D}_{\mathfrak{f}}$ de $\mathfrak{f}$. $\mathcal{Q}$ souhaite obtenir de $\mathcal{S}$ l'assurance que le résultat retouné est correct. Par conséquent, $\mathcal{Q}$ génère une clé publique de vérification VK_x associée à l'entrée x.

Le vérificateur $\mathcal{V}$: Par le truchement de la clé publique de vérification VK_x, $\mathcal{V}$ vérifie que le résultat $\mathfrak{f}(x)$ retourné par $\mathcal{S}$ est correct.

Étant données les parties impliquées dans un protocole de calcul vérifiable public, nous donnons ici la définition formelle d'un tel protocole.

Definition 23 (Protocole de calcul vérifiable public). *Ce protocole comprend quatre algorithmes en temps polynomial distribués en trois phases :*

▶ **Configuration.** *Cette phase inclut $\mathcal{O}$ uniquement. Il exécute l'algorithme* Setup *pour produire les clés nécessaires au protocole et préparer la fonction $\mathfrak{f}$ avant son externalisation :*

> ▷$\mathsf{Setup}(1^\kappa, \mathfrak{f}) \to (\mathsf{param}, \mathsf{PK}_{\mathfrak{f}}, \mathsf{EK}_{\mathfrak{f}})$: *Il s'agit d'un algorithme aléatoire exécuté par $\mathcal{O}$. Il prend en entrée le paramètre de sécurité 1^κ et une description de la fonction $\mathfrak{f}$ à externaliser, et renvoie en sortie une liste de paramètres publics* param*, une clé publique $\mathsf{PK}_{\mathfrak{f}}$, et une clé d'exécution $\mathsf{EK}_{\mathfrak{f}}$.*

▶ **Calcul.** *La phase de calcul comprend deux étapes. $\mathcal{Q}$ exécute l'algorithme* ProbGen *qui prépare la valeur d'entrée x soumise à $\mathcal{S}$. En retour, le serveur invoque l'algorithme* Compute *qui exécute le fonction $\mathfrak{f}$ sur l'entrée x et génère la preuve de calcul.*

> ▷$\mathsf{ProbGen}(x, \mathsf{PK}_{\mathfrak{f}}) \to (\sigma_x, \mathsf{VK}_x)$: *Étant donné x du domaine $\mathcal{D}_{\mathfrak{f}}$ de la fonction $\mathfrak{f}$ et la clé publique $\mathsf{PK}_{\mathfrak{f}}$, $\mathcal{Q}$ appelle cet algorithme pour produire un codage σ_x de l'entrée x qui sera transmis à $\mathcal{S}$, et une clé publique de vérification VK_x qui sera ensuite utilisée par le vérificateur $\mathcal{V}$ pour vérifier l'exactitude du résultat.*
>
> ▷$\mathsf{Compute}(\sigma_x, \mathsf{EK}_{\mathfrak{f}}) \to \sigma_y$: *Ayant reçu σ_x et étant donné la clé d'exécution $\mathsf{EK}_{\mathfrak{f}}$, le serveur $\mathcal{S}$ invoque cet algorithme pour calculer un codage σ_y du résultat $y = \mathfrak{f}(x)$.*

▶ **Vérification.** *Après avoir reçu le résultat et la preuve de calcul de la part de $\mathcal{S}$, $\mathcal{V}$ exécute l'algorithme* Verify *pour vérifier leur validité.*

> ▷$\mathsf{Verify}(\sigma_y, \mathsf{VK}_x) \to \mathsf{out}_y$: *$\mathcal{V}$ utilise cet algorithme pour vérifier l'exactitude du résultat σ_y fourni par $\mathcal{S}$. Plus précisément, cet algorithme décode σ_y, ce qui permet d'obtenir la valeur y, et ensuite il utilise la clé publique de vérification VK_x associée à σ_x pour décider si y est égal au résultat attendu $\mathfrak{f}(x)$. Si c'est le cas,* Verify *retourne $\mathsf{out}_y = y$ signifiant que $\mathfrak{f}(x) = y$; autrement l'algorithme retourne une erreur $\mathsf{out}_y = \perp$.*

Pour qu'un tel protocole soit viable, il doit être efficace, c'est-à-dire que, étant donné $\mathsf{PK}_\mathfrak{f}$ et pour n'importe quelle entrée x et n'importe quel σ_x, le temps mis pour exécuter $\mathsf{ProbGen}(x, \mathsf{PK}_\mathfrak{f})$ plus le temps mis pour exécuter $\mathsf{Verify}(\sigma_y, \mathsf{VK}_x)$ (où VK_x est généré par $\mathsf{ProbGen}$) est $o(T)$, où T est le temps requis pout calculer $\mathfrak{f}(x)$.

3.2 État de l'art

Nous avons analysé l'état de l'art en matière de vérification de calcul externalisé. Il existe un multitude de solutions traitant de ce problème, que nous avons catégorisées ainsi :

Les solutions pour des fonctions arbitraires : Ces solutions sont elles-mêmes classées selon si le protocole sous-jacent est interactif ou non. Elles ne font pas ou peu d'hypothèses sur la fonction à externaliser.

Protocoles interactifs : En plus d'être interactives, ces solutions sont probabilistes du fait que le vérificateur n'est convaincu de l'exactitude du calcul qu'avec une certaine probabilité. Parmi ces solutions, nous pouvons citer les preuves *Muggle* [98], où la fonction est traduite en circuit booléen avant son envoi au *cloud*. Le vérificateur et le serveur entre en interaction de plusieurs tours pour vérifier l'exactitude du calcul. Nous pouvons également citer les *preuves vérifiables en probabilité* PCP [10]. La preuve est codée de telle manière que le vérificateur peut la vérifier en accédant qu'un nombre constant de bits de la preuve. Les *arguments efficaces* proposés par Kilian [110, 111] combinent les PCP avec des moyens cryptographiques afin d'en assurer la robustesse.

Protocoles non-interactifs : Contrairement aux protocoles interactifs, ces solutions ne nécessitent pas plusieurs tours d'interaction entre le vérificateur et le serveur. Parmi ces solutions, nous pouvons mentionner les preuves *robustes en calcul* CS [126] qui combinent l'idée derrière les PCP avec l'heuristique de Fiat et Shamir [83], ainsi que les SNARKs qui poursuivent l'idée des preuves CS. Le protocole baptisé Pinocchio [144] combine un outil cryptographique appelé QAP avec un circuit arithmétique représentant la fonction à externaliser. Gennaro *et al.* [90] utilise des circuits Booléens, combinés à un chiffrement totalement homomorphique (FHE).

Les solutions pour des fonctions spécifiques : Plusieurs solutions de l'état de l'art ne considèrent qu'un certain type de fonctions. En effet, ces protocoles exploitent les propriétés particulières de ces fonctions pour permettre une délégation et une vérification efficace. Ces fonctions sont : l'évaluation de polynômes [30, 85, 193, 141], la multiplication matricielle [85, 195, 193], les opérations sur les ensembles [52, 140] ainsi que la recherche de mots-clé [30, 198]. L'attention portée à ces types de fonctions est due au fait qu'elles sont souvent utilisées comme opérations fondamentales pour des fonctions plus complexes comme l'exploration de données (*data mining*) ou le traitement d'image.

De cette analyse de l'état de l'art, nous avons porté notre attention sur les solutions pour des fonctions spécifiques et nous proposons trois protocoles pour les opérations suivantes : l'évaluation de polynomes, la multiplication matricielle et la recherche de mots-clés. Nos solutions sont publiquement délégables et vérifiables.

4 Évaluation de polynômes vérifiable publiquement

Dans le cas de l'évaluation de polynôme, le modèle de protocole de calcul vérifiable public se traduit ainsi : le propriétaire $\mathcal{O}$ souhaite déléguer un polynôme A de degré d pouvant être très grand et contacte le serveur $\mathcal{S}$ pour calculer $y = A(x)$.

La solution que nous proposons s'appuie sur les propriétés de la division euclidienne des polynômes : pour n'importe quelle paire de polynômes A et $B \neq 0$ de degré respectif d et 2, il existe une unique paire de polynômes Q et R telle que $A = BQ + R$ et le degré du polynôme quotient est $d-2$ alors que le polynôme reste est de degré inférieur à 1.

Par conséquent, $\mathcal{O}$ qui souhaite externaliser l'évaluation du polynôme A de degré d exécute d'abord l'algorithme Setup qui définit un polynôme $B(X) = X^2 + b_0$ pour un b_0 choisi aléatoirement, puis divise A par B pour obtenir le polynôme quotient $Q(X) = \sum_{i=0}^{d-2} q_i X^i$ et le polynôme reste $R(X) = r_1 X + r_0$. Ensuite, $\mathcal{O}$ confie les polynômes A et Q à $\mathcal{S}$ et publie la clé publique $\mathsf{PK}_A = (g^{b_0}, g^{r_1}, g^{r_0})$.

Plus tard, quand un utilisateur $\mathcal{Q}$ souhaite évaluer le polynôme A à un certain point x, celui-ci exécute l'algorithme $\mathsf{ProbGen}$ qui calcule et publie la clé publique de vérification $\mathsf{VK}_x = (\mathsf{VK}_{x,B}, \mathsf{VK}_{x,R}) = (g^{B(x)}, g^{R(x)})$, puis transmet $\sigma_x = x$ à $\mathcal{S}$. Ce dernier invoque l'algorithme $\mathsf{Compute}$ qui calcule $y = A(x)$ et génère la preuve $\pi = Q(x)$. Après réception de la réponse $\sigma_y = (y, \pi)$ de la part de $\mathcal{S}$, un vérificateur $\mathcal{V}$ exécute l'algorithme Verify qui vérifie si $g^y = (g^{B(x)})^\pi g^{R(x)}$.

L'efficacité de la vérification découle du fait que B et R ont un degré petit. En effet, pour vérifier l'exactitude d'un résultat σ_y, $\mathcal{V}$ effectue un nombre faible et constant d'opérations, contrairement aux $\mathcal{O}(d)$ operations pour évaluer le polynôme A.

La robustesse de ce protocole s'appuie sur la confidentialtié des polynômes B et R. Cependant, puisque B est de degré 2, la confidentialité de ces deux polynômes peut être facilement menacée en divulguant le polynôme quotient Q. Pour remédier à cet inconvénient, l'algorithme Setup code le polynôme Q en utilisant un codage homomorphe pour l'addition à sens unique. Plus précisément, chaque coefficient q_i du polynôme Q est codé selon h^{q_i}. Par conséquent, EK_A comprend tous ces h^{q_i}. De cette manière, l'algorithme $\mathsf{Compute}$ génère la preuve $\pi = h^{Q(x)}$ tout en préservant la confidentialité des polynômes B et R.

Pour finir, nous nous servons des opérateurs de couplage bilinéaires pour permettre à $\mathcal{V}$ de vérifier l'exactitude du résultat retourné par $\mathcal{S}$. Autrement dit, l'algorithme Verify vérifie que $e(g, h^y) = e(\mathsf{VK}_{x,B}, \pi) e(g, \mathsf{VK}_{x,R})$. En conséquence, notre protocole est robuste sous l'hypothèse $\lfloor d/2 \rfloor$*-forte de Diffie-Hellman* ($\lfloor d/2 \rfloor$-SDH) précisée ci-dessous.

Hypothèse D-SDH

Soient $\mathbb{G}_1$, $\mathbb{G}_2$ et $\mathbb{G}_T$ trois groupes cycliques du même ordre premier p tels qu'il existe un opérateur de couplage bilinéaire $e : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$.

L'hypothèse D**-forte de Diffie-Hellman** (D-SDH) est valable, si étant donné le tuple $(g, g^\alpha, h, h^\alpha, ..., h^{\alpha^D}) \in \mathbb{G}_1^2 \times \mathbb{G}_2^{D+1}$ pour un $\alpha \in \mathbb{F}_p^*$ choisi aléatoirement, la probabilité de générer une paire $(\beta, h^{1/(\beta+\alpha)}) \in \mathbb{F}_p \backslash \{-\alpha\} \times \mathbb{G}_2$ est negligeable.

5 Multiplication matricielle vérifiable publiquement

Dans le cas de la multiplication matricielle, le modèle de protocole de calcul vérifiable public se traduit ainsi : le propriétaire $\mathcal{O}$ souhaite déléguer une matrice M de taille (n, m) (pouvant être très grande) et contacte le serveur $\mathcal{S}$ pour calculer $\vec{y} = M\vec{x}$.

Comme dans l'article publié par Fiore and Gennaro [85], notre protocole de multiplication matricielle vérifiable publiquement poursuit l'idée suivante. Dans le but de vérifier que le serveur $\mathcal{S}$ multiplie correctement une matrice M de taille (n, m) à éléments M_{ij} avec un vecteur colonne $\vec{x} = (x_1, x_2, ..., x_m)^\intercal$, $\mathcal{O}$ invoque l'algorithme Setup qui choisit aléatoirement une matrice secrète R de taille (n, m) à éléments R_{ij} et fournit à $\mathcal{S}$ la matrice M et une matrice auxiliaire $\mathcal{N}$ de taille (n, m) telle que $\mathcal{N}_{ij} = \tilde{g}^{M_{ij}} g^{R_{ij}}$ (où $\tilde{g} = g^\delta$ pour un δ généré aléatoirement).

Par conséquent, quand un utilisateur $\mathcal{Q}$ fait appel à l'algorithme $\mathsf{ProbGen}$ pour solliciter $\mathcal{S}$ à multiplier la matrice M avec un vecteur $\vec{x}$ de son choix, celui-ci exécute l'algorithme $\mathsf{Compute}$ qui retourne le vecteur $\vec{y} = (y_1, y_2, ..., y_n)^\intercal$ et la preuve de calcul $\vec{\pi} = (\pi_1, \pi_2, ..., \pi_n)^\intercal$, de telle sorte que $\pi_i = \tilde{g}^{y_i} g^{\sum_{j=1}^{m} R_{ij} x_j}$, si $\mathcal{S}$ est honnête. Si on pose $\pi_i = g^{\gamma_i}$ et $\vec{\gamma} = (\gamma_1, \gamma_2, ..., \gamma_n)^\intercal$, alors le processus de vérification consiste à vérifier si $\vec{\gamma} = \delta\vec{y} + R\vec{x}$.

Pour transformer cette intuition en une solution pratique, on doit assurer que le processus de vérification est beaucoup moins coûteux que la multiplication $M\vec{x}$ pour n'importe quel vectuer $\vec{x}$. Pour ce faire, nous observons que pour n'importe quel vecteur $\vec{\lambda} = (\lambda_1, \lambda_2, ..., \lambda_n)$, vérifier que $\vec{\lambda}\vec{\gamma} = \delta\vec{\lambda}\vec{y} + \vec{\lambda}(R\vec{x})$ (c'est-à-dire que la projection de l'équation de vérification sur un vecteur $\vec{\lambda}$ aléatoire) ne requiert que $\mathcal{O}(n)$ de temps si le vecteur $\vec{\lambda}R$ est calculé préalablement. Donc, nous définissons la clé publique comme un codage à l'exposant de $\vec{\lambda}R$ et la clé publique de vérification associée au vecteur $\vec{x}$ comme un codage à l'exposant de $(\vec{\lambda}R)\vec{x}$.

Plus conrètement, l'algorithme Setup génère les éléments de la matrice auxiliaire $\mathcal{N}$ selon $\mathcal{N}_{ij} = \tilde{g}_i^{M_{ij}} g_i^{R_{ij}}$ avec $g_i = g^{\lambda_i}$, et la clé publique PK_M est un vecteur à m composantes $\mathsf{PK}_j = e(\prod_{i=1}^{n} g_i^{R_{ij}}, h)$. L'algorithme $\mathsf{ProbGen}$ calcule la clé de vérification pour le vecteur $\vec{x}$ selon $\mathsf{VK}_x = \prod_{j=1}^{m} \mathsf{PK}_j^{x_j}$. L'algorithme $\mathsf{Compute}$ génère la preuve de calcul en calculant $\pi = \prod_{i=1}^{n} \prod_{j=1}^{m} \mathcal{N}_{ij}^{x_j}$ et l'algorithme Verify vérifie que $e(\pi, h) = e(\prod_{i=1}^{n} g_i^{y_i}, \tilde{h})\mathsf{VK}_x$. Par conséquent, l'algorithme $\mathsf{ProbGen}$ combiné à l'algorithme Verify ne requiert que $\mathcal{O}(n+m)$ opérations, contrairement aux $\mathcal{O}(nm)$ opérations nécessaires pour calculer la multiplication $M\vec{x}$.

La solution que nous proposons dans cette thèse est vérifiable et déléguable publiquement. Elle est aussi robuste sous l'hypothèse de Diffie-Hellman co-calculatoire (co-CDH).

Hypothèse co-CDH

Soient $\mathbb{G}_1$, $\mathbb{G}_2$ et $\mathbb{G}_T$ trois groupes cycliques du même ordre premier p tels qu'il existe un opérateur de couplage bilinéaire $e : \mathbb{G}_1 \times \mathbb{G}_2 \rightarrow \mathbb{G}_T$.

L'hypothèse de **Diffie-Hellman co-calculatoire** (co-CDH) est valable sur $\mathbb{G}_1$, si étant donné $g, g^\alpha \in \mathbb{G}_1$ et $h, h^\beta \in \mathbb{G}_2$ pour $\alpha, \beta \in \mathbb{F}_p^*$, la probabilité de calculer $g^{\alpha\beta}$ est négligeable.

6 Recherche de conjonction de mots-clés vérifiable publiquement

Dans le cas de la recherche de conjonction de mots-clés, le modèle de protocole de calcul vérifiable public se traduit ainsi : le propriétaire $\mathcal{O}$ souhaite déléguer un ensemble de fichiers $\mathcal{F}$ et contacte le serveur $\mathcal{S}$ pour chercher dans $\mathcal{F}$ une conjonction de mots-clés $\mathcal{W} = \{\omega_1, \omega_2, ..., \omega_k\}$. La figure 7.14 donne un aperçu de notre protocole.

Notre protocole s'appuie sur l'outil cryptographique appelé accumulateur à base de polynômes [132]. Par définition, cet outil permet de représenter un ensemble sous la forme d'un polynôme unique de telle sorte que les racines du polynôme sont exactement les éléments de cet ensemble. Plus formellement, soit un ensemble $S = \{h_1, ..., h_n\}$ d'éléments dans $\mathbb{F}_p$. Cet ensemble peut être codé sous la forme d'un unique polynôme $P_S(X) = \prod_{h_i \in S}(X - h_i)$. Soit g un générateur quelconque d'un groupe bilinéaire $\mathbb{G}$ d'ordre premier p. Étant donné le tuple $(g, g^\alpha, g^{\alpha^2}, ..., g^{\alpha^D})$, où α est choisi aléatoirement dans $\mathbb{F}_p^*$ et $D \geq n$, Nguyen [132] définit l'accumulateur public des éléments dans S : $\mathcal{Acc}(S) = g^{P_S(\alpha)} \in \mathbb{G}$. Les accumulateurs à base de polynômes rendent par ailleurs possible un test vérifiable d'appartenance, qui peut être adapté au problème de recherche vérifiable de mots-clés.

Une approche naïve pour adapter les accumulateurs au problème de recherche serait de représenter les mots-clés de chaque fichier contenu dans $\mathcal{F}$ avec un unique accumulateur pour chaque fichier. Pour vérifier qu'un mot est présent dans un fichier de $\mathcal{F}$, l'utilisateur $\mathcal{Q}$ envoie d'abord une requête de recherche au serveur $\mathcal{S}$, à partir de laquelle ce dernier génère une preuve d'appartenance si ce mot-clé existe dans le fichier en question, ou une preuve de non-appartenance sinon. Cependant, cette solution n'est pas efficace. Étant données les propriétés des accumulateurs à base de polynômes, la complexité de la recherche dans un

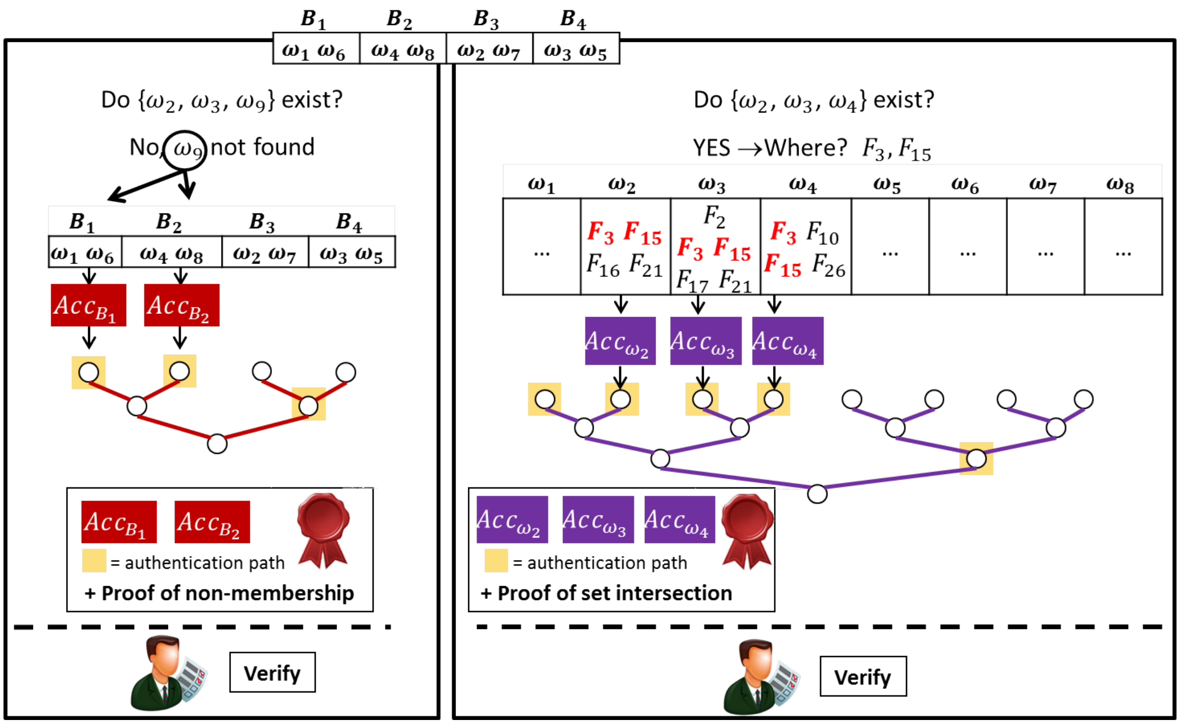

B_1 B_2 B_3 B_4
$\omega_1\ \omega_6$ $\omega_4\ \omega_8$ $\omega_2\ \omega_7$ $\omega_3\ \omega_5$
Do $\{\omega_2, \omega_3, \omega_9\}$ exist?
No, ω_9 not found
Acc_{B_1} Acc_{B_2}
= authentication path
+ Proof of non-membership
Verify
Do $\{\omega_2, \omega_3, \omega_4\}$ exist?
YES → Where? F_3, F_{15}
ω_1 ω_2 ω_3 ω_4 ω_5 ω_6 ω_7 ω_8
$F_3\ F_{15}\ F_{16}\ F_{21}$
$F_2\ F_3\ F_{15}\ F_{17}\ F_{21}$
$F_3\ F_{10}\ F_{15}\ F_{26}$
Acc_{ω_2} Acc_{ω_3} Acc_{ω_4}
= authentication path
+ Proof of set intersection
Verify

Figure 7.14: Aperçu de notre protocole pour la recherche de conjonction de mots-clés vérifiable

seul fichier serait linéaire par rapport au nombre de mots-clés présents dans ce fichier. Par ailleurs, pour identifier quels fichiers contiennent le mot-clé recherché, $\mathcal{Q}$ doit chercher chaque fichier de $\mathcal{F}$ un par un.

Pour éviter cet inconvénient, notre solution combine les accumulateurs avec un arbre de Merkle [125] pour construire un index authentifié des mots-clés contenus dans l'ensemble $\mathcal{F}$, de telle sorte que la recherche côté serveur s'exécute en temps logarithmique. Plus spécifiquement, le propriétaire des fichiers $\mathcal{O}$ invoque l'algorithme Setup qui organise les mots-clés en un index de recherche $\mathcal{I}$ (comme une table de hachage), où chaque entrée correspond à une boîte contenant au plus d mots-clés. Pour construire un index $\mathcal{I}$ de façon efficace, notre protocole utilise le hashage *Coucou* introduit par Pagh and Rodler [137], et qui garantit une recherche en temps constant et un minimum de capacité de stockage. Ensuite, $\mathcal{O}$ authentifie l'index $\mathcal{I}$ comme suit : (i) pour chaque boîte de l'index, il calcule un accumulateur des mots-clés qui sont référencés dans cette boîte; (ii) et il construit un arbre de Merkle TW qui authentifie ces accumulateurs. Les fichiers de $\mathcal{F}$ ainsi que l'index $\mathcal{I}$ et l'arbre TW sont ensuite confiés au serveur $\mathcal{S}$.

Quand $\mathcal{S}$ reçoit un requête de recherche pour un mot-clé ω, il exécute l'algorithme Search (*i.e.* l'algorithme Compute pour la recherche), il localise la boîte correspondant à ω dans l'index de recherche $\mathcal{I}$, calcule l'accumulateur correspondant, génère une preuve d'appartenance (ou non-appartenance) et authentifie l'accumulateur en utilisant l'arbre de Merkle TW. Par ailleurs, n'importe qui possédant la racine de l'arbre TW peut exécuter l'algorithme Verify permettant de vérifier le résultat de la recherche retournée par $\mathcal{S}$.

Cependant, la solution esquissée ci-dessus ne permet toujours pas d'identifier les fichiers qui contiennent le mot-clé ω et ni de répondre au problème des requêtes avec des conjonctions de mots-clés. Donc l'algorithme Setup construit un autre arbre de Merkle TF où chaque feuille représente un seul mot-clé et est associée à l'accumulateur du sous-ensemble de fichiers qui contiennent ce mot. $\mathcal{O}$ confie alors les fichiers $\mathcal{F}$, l'index $\mathcal{I}$ et les deux arbres TW et TF au serveur $\mathcal{S}$. Étant donnée la racine de l'arbre TF, $\mathcal{V}$ va pouvoir identifier quels sous-ensembles de fichiers contiennent le mot recherché. De plus, puisque les accumulateurs à base de polynômes permettent de calculer l'intersection d'ensemble de manière vérifiable et efficace, l'utilisateur $\mathcal{Q}$ va pouvoir envoyer des requêtes sur une conjonction de mots-clés sur les fichiers $\mathcal{F}$.

Notre solution est robuste sous deux hypotèses : l'hypothèse D-forte de Diffie-Hellman déjà énoncée dans le cas des polynômes et l'hypothèse D-forte de Diffie-Hellman bilinéaire (D-SBDH) qui se formule ainsi :

Hypothèse D-SDH

Soient $\mathbb{G}$ et $\mathbb{G}_T$ deux groupes cycliques du même ordre premier p tels qu'il existe un opérateur de couplage bilinéaire $e : \mathbb{G}_1 \times \mathbb{G}_1 \rightarrow \mathbb{G}_T$. Soit g un générateur de $\mathbb{G}$.

L'hypothèse **D-forte de Diffie-Hellman bilinéaire** (D-SBDH) est valable, si étant donné le tuple $(g, g^{\alpha}, ..., g^{\alpha^D}) \in \mathbb{G}^{D+1}$ pour un $\alpha \in \mathbb{F}_p^*$ choisi aléatoirement, la probabilité de générer une paire $(x, e(g,g)^{1/(x+\alpha)}) \in \mathbb{F}_p \backslash \{-\alpha\} \times \mathbb{G}$ est negligeable.

7 Langage de Politiques d'Imputabilié

Alors que les protocoles présentés précédemment traitaient d'outils cryptographiques destinés à vérifier qu'un serveur de *cloud* effectue les opérations demandées, notre travail sur le langage de politiques permet d'étendre le concept de vérifiabilité à celui d'imputabilité. Nous condirérons l'imputabilité comme étant une notion qui permet de définir la gouvernance des données au sein de laquelle les entreprises, à qui sont confiées des données personnelles et sensibles, sont tenues responsables pour stocker, traiter et partager les données selon des obligations contractuelles et règlementaires. L'organisme, sous cette définition de l'imputabilité, doit implémenter des actions adéquates et prendre en compte des mesures de restauration dans les cas où cet organisme échoue à agir convenablement [146]. Les obligations proviennent de plusieurs sources, notamment le Règlement Général sur la Protection des données [80]. Elles permettent de clarifier les relations liées à l'imputabilité dans le *cloud*, c'est-à-dire qui est responsable de quoi et envers qui. Donc définir des politiques appropriées représentant les obligations liées à l'imputabilité est une exigence fondamentale pour des mecanismes de contrôle dans le sens où les politiques réduisents les risques, à condition que leurs mises en application et la vérification de leur conformité soient rendues possibles.

Nous étudions alors la conception d'un langage de politiques qui permet d'exprimer les obligations d'imputabilité. L'objectif de notre travail est d'analyser comment et dans quelle mesure nous pouvons transmettre des obligations via un langage expressif et déclaratif, de telle sorte que ces politiques soient faciles à écrire, à gérer, à mettre en œuvre et à valider. Nous nous intéressons alors à des politiques interpretables par la machine. Nous proposons donc A-PPL, un nouveau langage de politique basé sur un langage existant PPL, lui-même basé sur un langage devenu standard, à savoir XACML. A-PPL est à la fois expressif et déclaratif.

Pour concevoir ce langage, nous avons d'abord analysé les concepts derrière la notion d'imputabiité. Nous avons aussi étudié les obligations afin d'identifier les exigences pour un langage de politiques d'imputabilité. Nous avons passé en revue l'état de l'art afin de choisir un langage, dans le but de l'améliorer afin de répondre aux exigences liées à l'imputabilité. Finalement, nous proposons nos extensions ainsi que le moteur, l'A-PPL *engine*, qui permet de mettre en œuvre une politique d'imputabilité écrite avec A-PPL. Ce travail s'inscrit dans le projet européen A4Cloud.

7.1 Cadre conceptuel

Le concept d'imputabilité dans le *cloud* inclut plusieurs notions : des attributs, des pratiques et des mécanismes.

Les attributs de l'imputabilité sont la responsabilité, la transparence, la vérifiabilité et la rémédiation. À ces notions s'ajoutent celles des obligations, des comportements et de la conformité.

Les pratiques désignent le comportement opérationnel qui doit être adopté par un système imputable afin de mettre en application les attributs identifiés plus hauts. Par exemple, le *cloud* doit définir et informer les utilisateurs sur la manière dont les données sont traitées (en relation avec la transparence). Autre exemple : le *cloud* doit garantir l'implémentation des mecanismes afin d'être conforme avec les obligations et de démontrer cette conformité.

Les mécanismes d'imputabilité correspondent aux outils et techniques qui mettent en œuvre les pratiques et les attributs de l'imputabilité. Ceux-là incluent les techniques de vérifiabilité présentées dans cette thèse, les techniques de consignations des actions (*logging*) et les techniques d'audit.

Chaque acteur du *cloud*, à savoir le fournisseur de *cloud*, l'utilisateur et l'auditeur endossent des rôles bien spécifiques et définis dans le Règlement européen [80] : le sujet de la donnée (*data subject* - DS), le controleur de la donnée (*data controller* - DC), le processeur de la donnée (*data processor* - DP) et les autorités de protection de la donnée (*data protection authorities* - DPA).

7.2 Obligations liées à l'imputabilité

Comme évoqué plus haut, les obligations d'imputabilité proviennent de plusieurs sources : réglementaire, contractuel et éthique. Ces sources permettent en particulier de définir les relations entre DS, DC, DP et DPA, et d'identifier plusieurs types de contrôle, à savoir un contrôle préventif, détectif ou correctif.

À partir d'un scénario bien défini lié à un hôpital désireux de stocker des données de patients dans un *cloud*, le projet A4Cloud a déterminé les huit obligations suivantes : (1) le DS a le droit d'accéder, de corriger et de supprimer les données confiées au *cloud*; (2) les données sont traitées pour une durée et un objectif bien définis; (3) les violations de sécurité doivent être notifiées; (4) le *cloud* doit pouvoir fournir des preuves de la suppression correcte et ponctuelle de données personnelles; et (5) les données doivent être stockées et traitées en un lieu géographique bien défini et autorisé.

Un langage de politique d'imputbilité doit pouvoir exprimer ces obligations. Par conséquent, nous avons étudié les exigences auxquelles doit satisfaire ce langage. Celles-ci sont représentées dans le tableau 7.8. Elles sont classifiées selon si elles traduisent des critères liés à la gestion des données ou à l'imputabilité seule.

Critères	**Categorie**
(R1) Exprimer des politiques de respect de la vie privée	Gestion de données
(R2) Règles de contrôle d'accès	Gestion de données
(R3) Règles de contrôle d'usage	Gestion de données
(R4) Période de conservation de données	Gestion de données
(R5) Notification et signalement	Imputabilité
(R6) Localisation des données	Imputabilité
(R7) Auditabilité	Imputabilité
(R8) Consignation des actions	Imputabilité

Table 7.8: Critères d'un langage de politique d'imputabilité.

7.3 A-PPL

L'approche adoptée par le projet A4Cloud est de développer un langage de politique existant avec de nouvelles extensions afin de pouvoir exprimer des règles liées à l'imputabilité. Nous

Nom	Description
Triggers	
TriggerPersonalDataAccessPermitted	Déclenché quand l'accès à une donnée est permis
TriggerPersonalDataAccessDenied	Déclenché quand l'accès à une donnée est refusé
TriggerEvidenceRequestReceived	Déclenché quand le DC reçoit une requête pour générer des preuves
Actions	
ActionNotify	Notifie le destinataire avec l'information relative à l'événement qui déclenche cette action
ActionLog	Consigne un événement, c'est-à-dire, écrit dans un fichier journal les informations détaillées relatives à l'événement qui déclenche cette action
ActionEvidenceCollection	Initie la collecte (ou la production) des preuves demandées

Table 7.9: Extensions d'A-PPL.

avons donc étudié et analysé plusieurs langages de politiques et avons choisi celui qui satisfait le plus les critères énoncés plus hauts et celui qui peut être facilement extensible. Le meilleur candidat se trouve être le langage PPL développé, par le projet PrimeLife. PPL est à l'origine un langage de politique permettant d'exprimer des politiques de protection de la vie privée. Basé sur XACML, PPL présente de nombreux points d'extension. Il permet de définir des règles de contrôle d'accès, des autorisations ainsi que des obligations. Une obligation en langage PPL définit une promesse faire par le DC au DS en relation avec le traitement de ses données personnelles. Le DC doit tenir sa promesse en exécutant une action particulière après un événement spécifique, et facultativement, sous certaines conditions. En pratique, une obligation en PPL s'exprime en termes de Triggers (déclencheur) et d'Actions. Un Trigger est un événement conditionné qui déclencle une action que le *cloud* doit réaliser dans le cadre de l'obligation en question.

Les extensions d'A-PPL. Nous avons d'abord rendu plus explicite la définition des rôles de chacun des acteurs du *cloud* à l'aide d'un nouvel attribut `subject:role`.

Ensuite, le tableau 7.9 compile les nouveaux Triggers et les nouvelles (ou améliorées) Actions que nous proposons dans notre nouveau langage A-PPL.

7.4 A-PPLE

La mise en œuvre de la potique écrite en A-PPL est l'affaire du moteur appelé A-PPLE. A-PPLE étend le moteur utilisé par les langages sur lesquels A-PPL est basé, à savoir XACML et PPL.

En particulier, nous avons développé le composant qui s'occupe de mettre en œuvre les obligations d'imputabilité comme les notifications ou la consignation des actions. Par ailleurs, nous avons intégrer nos travaux sur les preuves de récupérabilité avec $\mathcal{S}$tealthGuard dans le moteur A-PPLE afin de permettre des audits du *cloud* quant à l'intégité des données.

List of Publications

- **StealthGuard: Proofs of Retrievability with Hidden Watchdogs**, Monir Azraoui, Kaoutar Elkhiyaoui, Refik Molva and Melek Önen. In *Computer Security-ESORICS 2014*, pages 239–256. Springer International Publishing, 2014.
- **Efficient Techniques for Publicly Verifiable Delegation of Computation**, Kaoutar Elkhiyaoui, Melek Önen, Monir Azraoui and Refik Molva. In *Proceedings of the 11th ACM on Asia Conference on Computer and Communications Security, AsiaCCS 2016*, pages 119–128. ACM, 2016.
- **Publicly Verifiable Conjunctive Keyword Search in Outsourced Databases**, Monir Azraoui, Kaoutar Elkhiyaoui, Melek Önen and Refik Molva. SPC 2015, 1st IEEE Workshop on Security and Privacy in the Cloud. In *Proceedings of the 2015 IEEE Conference on Communications and Network Security (CNS)*, pages 619-627. IEEE, 2015.
- **A-PPL: An Accountability Policy Language**, Monir Azraoui, Kaoutar Elkhiyaoui, Melek Önen, Karin Bernsmed, Anderson Santana De Oliveira, Jakub Sendor. In *Data Privacy Management, Autonomous Spontaneous Security, and Security Assurance*, pages 319–326. Springer, 2015.
- **A Cloud Accountability Policy Representation Framework**, Walid Benghabrit, Hervé Grall, Jean-Claude Royer, Mohamed Sellami, Monir Azraoui, Kaoutar Elkhiyaoui, Melek Önen, Anderson Santana De Oliveira, Karin Bernsmed. In *CLOSER-4th International Conference on Cloud Computing and Services Science*, pages 489–498. 2014.