Implementational aspects of code-based cryptography

Bhaskar Biswas

ÉCOLE POLYTECHNIQUE

INRIA

PhD Thesis

Presented at

L'ÉCOLE POLYTECHNIQUE

To obtain the title

DOCTEUR EN SCIENCES

Subject **Information theory**

Defense 01 October, 2010 by

Bhaskar Biswas

Implementational aspects of code-based cryptography

Jury

Reporters

Philippe Gaborit	Université de Limoges, XLIM-DMI, France
Grigory Kabatiansky	Russian Academy of Sciences, Moscow, Russia

PhD supervisors

Nicolas Sendrier	INRIA Paris-Rocquencourt, équipe-projet SECRET, France
Luc Bouganim	INRIA Paris-Rocquencourt, équipe-projet SMIS, France

Examiners

Pierre Loidreau	DGA et Université de Rennes 1, France
Daniel Augot	École Polytechnique et INRIA, France

List of Figures

Acknowledgement

I am indebted to many people for the result that I am finishing my PhD thesis.

I would like to thank Prof. Ayanendranath Basu and Prof. Bimal Roy for introducing me to the cryptographic research. As for influence, my profound gratitude goes towards Nicolas Sendrier for being my supervisor during my PhD at Inria Paris-Rocquencourt. He helped with topics, been there whenever needed, encouraged me to extend my research interests, enriched my philosophical views towards life itself, which for me is very valuable. My PhD co-adviser of thesis Luc Bouganim, thank you for being patient. It was in him, I saw the hatred for mediocrity. It is an honour to know both my advisers let alone getting the opportunity to work with them.

A special thanks goes to all the people in our project "SECRET", I always felt home. It is a privilege to know them.

My friends at INRIA; Andre, Daria, Iskender, Roger, Vincent, Yasser and lately Sumanta, Suchana di and her delicious food and Sunandan - thank you for your support.

I must not forget to mention Christelle. She is the spirit of the project, the best secretary, and a very good friend of mine. Thank you.

A special thank for Mr. Satya Lokam, under whose guidance I spend three wonderful months at Microsoft Research, India. I shall always remember his continuous effort to achieve perfection and not be satisfied otherwise.

There are many people who visited or worked in our project "SECRET" and who contributed by discussion. I will mention them in alphabetic order and I hope that I do not forget anyone. I'm grateful to all of them.

Mamdouh Abbara, Nicolas Anciaux, Daniel Augot, Raghav Bhaskar, Céline Blondeau, Christina Boura, Anne Canteaut, Christophe Chabot, Pascale Charpin, Mathieu Cluzeau, Maxime Côte, Frédéric Didier, Cédric Faure, Matthieu Finiasz, Fabien Galand, Benoit Gérard, Christelle Guiziou, Stéphane Jacob, Deepak Kumar Dalai, Yann Laigle-Chapuy, Cédric Lauradoux, Stṕhne Manuel, Françoise Levy-dit-Vehel, Ayoub Otmani, Raphael Overbeck, Maria Naya Plasencia, Philippe Pucheral, Ludovic Perret, Andrea Roeck, Sushmita Ruj, Sumanta Sarkar, Christophe Salpawyck, Jean-Pierre Tillich, Marion Videau, and Alexander Zeh.

The reporters and juries of my thesis, Grigory Kabatiansky, Philippe Gaborit, Pierre Loidreu and Daniel Augot - my gratitude for your feed-back.

Once again I want to thank all the people who helped and supported me during my PhD. It is due to them that I can say now: *let's move on!*

Contents

Introduction

We start with a brief introduction of Public Key Cryptography (PKC). We then move to PKCs based on error correcting codes, emphasizing on the McEliece cryptosystem. We introduce an improved version of McEliece scheme, which we call Hybrid McEliece Encryption Scheme(HyMES), in which we aim to reduce the key size (by using the generator matrix in systematic form) and increase the information rate (by embedding information in the error pattern). We overview one of the key questions addressed for HyMES realization namely, root finding problem over finite fields and introduce our approach to obtain a faster method to solve the problem. Finally we summarize with the results of this thesis.

Public Key Cryptography

As the world increasingly turns to electronic business, electronic credentials that prove identity are becoming a critical necessity. Much like a passport proves identity in the offline world, public key cryptosystems deliver a way to prove identity in the online world. They ensure that people are who they say they are and also prove that documents haven't been tampered with, which is critical when conducting online transactions, such as placing orders or transferring money.

In conventional cryptography, encryption and decryption are symmetric operations. Anyone who has access to the key could both encrypt and decrypt. Such cryptosystems that use a single key for both encryption and decryption are called symmetric cryptosystems. While these type of systems are and remain widely useful in practice, some applications, as the ones stated above cannot be satisfactorily addressed with symmetric key cryptography.

The concept of "asymmetric cryptography" was first developed in 1976 by Diffie and Hellman [20], to solve the problem of exchanging a key over a network. Since everyone connected to the network has the receiver's public key, anyone can send him/her a message by encrypting it with that public key. Only the receiver can read the message by decrypting it with his/her private key. In this way, there is no need to exchange a sensitive or secret key, reducing the risk of exposing the message.

A public key encryption scheme consists of three components,

1. *A Key generation function.* Generates a random pair of keys $(K_{public}, K_{private})$, called the *public key* and the *private key.*

2. *An encryption function.* The encryption function $\mathcal{E}(K_{public}, m)$, takes the plain text message m and the public key to compute the corresponding cipher text message c, *i.e.*

$$\mathcal{E}(K_{public}, m) \rightarrow c.$$

3. *A decryption function.* The decryption function $\mathcal{D}(K_{private}, c)$, takes the private key and the cipher text c and regenerate the corresponding plain text message m, *i.e.*

$$\mathcal{D}(K_{private}, c) = \mathcal{D}(K_{private}, \mathcal{E}(K_{public}, m)) \rightarrow m.$$

Given a public key K_{public} , a cipher text c, and a description of the system, it should be computationally infeasible to compute the whole message m or a big part of it.

Code Based Cryptography

Since 1970's, there are only a handful of supposedly secure public key crypto-systems. Some well known systems proposed are (not exhaustive),
RSA [47] - based on the difficulty of factoring large integers.
Merkle-Hellman Knapsack [41] - based on the difficulty of the subset sum problem (N-P complete problem).
ElGamal citeELG85 - based on the difficulty of the discrete logarithmic problem for finite fields.
Chor- Rivest [18] - Also a knapsack type problem.
Elliptic curve [32] - basically modification of other systems with increased efficiency. This type of El Gamal works with smaller keys.
McEliece [40] - based on the algebraic coding theory, on the problem of decoding linear code.

The code based systems play a distinct role in public key cryptosystems. It has a firm, well developed mathematical back ground. The systems supposed to resist the threat, that will be posed by quantum computers[53]. While this threat remains a theoretical one for the time being, it is unlikely that this will remain the case indefinitely.

Evolution of Code Based Cryptography

The idea of code based cryptosystems is almost as old as public key cryptography itself; the first such system was proposed by Robert J. McEliece in 1978 [40]. During the thirty years that has elapsed since, it's security as an one way trapdoor encryption scheme has never been seriously threatened (though the original parameters proposed by McEliece, have been shown to be too small [15]).

Most of the previous works have been devoted to cryptanalysis and to semantic security but fewer attempts have been made to examine implementation issues. Implementing a

(public key) cryptosystem is a trade-off between security and efficiency *i.e.* the larger parameters give us more security but make the system slower. For that reason, cryptanalysis and implementation have to be considered in unison.

There has been many attempts to modify McEliece construction. Niederreiter proposed a variant in 1986 [43]. Sidelnikov proposed [54], Janwa and Moreno suggested another variant [30].

The attacks against McEliece system are mainly decoding attacks. Adams and Meijer [1], Lee and Brickel [33], Leon [34], Stern [56], Canteaut and Chabaud [14] addressed this problem.

The structural analysis includes Sidelnikov and Shestakov's polynomial time attack against the proposal of Niederreiter's Reed-Solomon variant [55], Nicolas Sendrier's attack against concatenated code [48] and Loidreau and Sendrier's attack against weak keys [38].

Though the public key size is rather large, the McEliece encryption scheme possesses some strong features. It has a good security reduction and low complexity algorithms for encryption and decryption. As a consequence, it is conceivable, compared with number-theory based cryptosystems, to gain an order of magnitude in performance.

Results Presented in this Thesis

After we recall the theoretical background and establish the notions and notations at the beginning of the thesis we shall pursue each of *implementation aspects of McEliece system*, in particular, *how to find roots of polynomials over finite fields efficiently* in details.

Let us give a brief overview of each topic presented in this thesis.

Implementation of HyMES

In Chapter 2, we present the implementation details of Hybrid McEliece Encryption Scheme (HyMES), a improved version of the original McEliece scheme developed with Nicolas Sendrier[1].

We present a modified version of the original scheme (which we call *hybrid*). It has two modifications, the first increases the information rate by putting some data in the error pattern. The second reduces the public key size by making use of a generator matrix in systematic form. We will show that the same security reduction as for the original system holds. We then describe the key generation, the encryption and the decryption algorithms and their implementation. Finally we will give some computation time for various parameters, compare them with the best known attacks, and discuss the best trade-offs.

The idea of McEliece scheme is to hide the structure of the code by means of a transformation of the generator matrix. The transformed generator matrix becomes the public key and the trapdoor information is the structure of the Goppa code together with the

[1]freely available at `http://www-roc.inria.fr/secret/CBCrypto/index.php?pg=hymes`

transformation parameters. The security relies on the fact that the decoding problem for general linear code is NP-complete.

While the RSA public-key cryptosystem has become most widely used, McEliece cryptosystem has not been quite as successful. Partly because of the large public key, which impose less problem with the advance in hardware today. Our aim has been to implement a fairly fast and concise software implementation that may be used as a reference benchmark.

We present the algorithmic details of our implementation as well. That is to specify the algorithms we use and the way we use them.

Root Finding of Polynomials

Root finding of polynomials over finite fields is a classical algebraic algorithmic problem. It is considered as one of the most time-consuming subprocess of the decoding process of Reed-Solomon, BCH and Goppa codes. There are some well known approaches for finding roots of the so-called error locator polynomial. The most widely known root finding algorithm is Chien search method [17], which is an evaluation of the polynomial at all elements of the field, so it has very high time complexity for large fields and polynomials of high degree. Berlekamp Trace Algorithm (BTA) [6] is another well known method. It is a recursive method based on the trace function properties.

McEliece decryption process employs an algebraic decoding algorithm which is often broken up in three parts. Namely, syndrome computation, finding the solution of the key equation and the root finding of error-locator polynomial. We noticed that, this last step consumes around three fourth of the total decryption time.

We present a hybrid method involving BTA and a method proposed by Zinoviev [60]. Zinoviev proposed direct root finding procedures for polynomials with degree at most 10. Our idea is to compute directly the roots with Zinoviev procedures up to some degree and to use recursively BTA for greater degrees. Moreover, we improve Zinoviev procedures for polynomials of degree 2 and 3 with time-memory trade-offs. We analyze both the theoretical complexities and the experimental complexities of our proposal. For typical parameters, we obtain a theoretical gain of 93% against Chien method and 46% against BTA. Experimental results confirm theory up to degree 4 at least. For instance with $m = 11$, $t = 32$ and $d_{max} = 4$, our method takes 60% of total decryption time with respect to 72% for BTA and 87% for Chien.

Part I

Background

Chapter 1

Background

We shall try to encompass the theoretical background in this chapter. We start with defining *finite fields*, we move to extension of finite fields, the different representations of them and we describe the algorithmic choices we made for implementation. We briefly visit error correcting codes, their decoding methods and specifically see some details regarding binary Goppa codes, which are of our main interest. We mention some classical algorithms used in the context.

1.1 Finite fields

We are going to confine ourselves to the fields of *characteristic* 2 as it is of primary interest for us. A field with *finite* number of elements is called a finite field.

Definition 1.1 (Field) *A* **field** $(\mathbb{F}, +, \cdot)$ *is a set of elements* $\mathbb{F}$ *associated with* two binary operations *"$+$" and "$\cdot$", such that the following axioms are satisfied for all* $a, b, c \in \mathbb{F}$.

1. **Closure under $+$ and $\cdot$**
   For all $a, b \in \mathbb{F}$, *both* $a + b$ *and* $a \cdot b$ *are in* $\mathbb{F}$.

2. **Associativity of the operations "$+$" and "$\cdot$"**
   For all $a, b, c \in \mathbb{F}$, *the following equalities hold:* $a + (b + c) = (a + b) + c$ *and* $a \cdot (b \cdot c) = (a \cdot b) \cdot c$.

3. **Commutativity of both the operations**
   For all $a, b \in \mathbb{F}$, *the following equalities hold:* $a + b = b + a$ *and* $a \cdot b = b \cdot a$.

4. **Identity elements**
   There exists an element of $\mathbb{F}$, *called the* additive identity element (associated with the operation "$+$") *and denoted by* 0, *such that for all* $a \in \mathbb{F}$, $a + 0 = a$. *Likewise, there is an element, called the* multiplicative identity element (for "$\cdot$") *and denoted by* 1, *such that for all* $a \in \mathbb{F}$, $a \cdot 1 = a$. *Note the additive identity and the multiplicative identity are required to be distinct.*

5. **Inverse elements**
   *For every $a \in \mathbb{F}$, there exists an element $-a \in \mathbb{F}$, such that $a + (-a) = 0$ (*additive inverse*). Similarly, for any $a \in \mathbb{F}$ other than 0, there exists an element $a^{-1} \in F$, such that $a \cdot a^{-1} = 1$. (The elements $a + (-b)$ and $a \cdot b^{-1}$ are also denoted $a - b$ and a/b, respectively.) In other words, subtraction and division operations exist.*

6. **Distributivity of "$\cdot$" over "$+$"** *For all $a, b, c \in \mathbb{F}$, the following equality holds: $a \cdot (b + c) = (a \cdot b) + (a \cdot c)$.*

or simplicity we shall write ab instead of $a \cdot b$ for field multiplication for the rest of the document.

Definition 1.2 (Prime fields) *For every prime p, the set $\{0, 1, \cdots, p-1\}$ forms a field (denoted by $\mathbb{F}_p$) under* mod p *addition and multiplication.*

Definition 1.3 (Order of a finite field) *The order of a finite field is the number of elements in the field.*

Definition 1.4 (Subfield) *If a subset S of the elements of a field $\mathbb{F}$ satisfies the field axioms with the same operations of $\mathbb{F}$, then S is called a subfield of $\mathbb{F}$.*

Proper subfield is a subfield which is strictly smaller than the field in which it is contained.

1.1.1 Extension fields

The extensive applicability of extension fields leads us to detail it's characteristics.

Definition 1.5 (Extension field) *Let $\mathbb{F}_q$ be a field. A subset of $\mathbb{F}_q$ that is a field by itself is called a subfield and $\mathbb{F}_q$ is called the* extension field.

Note that for finite fields containing no proper sub-field are *prime fields*.

Definition 1.6 (Characteristic of a field) *The characteristic of a field $\mathbb{F}$ is the least positive integer m such that, $\sum_1^m 1 = 0$, where $1 \in \mathbb{F}$ is the* multiplicative identity *of the field. If no such m exists, the characteristic is* 0.

If the characteristic p of a field is non-zero, then p is a prime number.

The simplest example of a finite field is the $\mathbb{F}_2$ field, the operations in this field are addition and multiplication modulo 2.

Theorem 1.7 *If $\mathbb{F}$ is a finite field of cardinality $q \geq 2$, then*

- *$q = p^m$, where p is prime and m is an integer.*
- *$\mathbb{F}$ is unique up to isomorphism.*

Theorem 1.8 *The multiplicative group of $\mathbb{F}_q$, noted as $\mathbb{F}_q^*$ is cyclic and generators of this group are called* primitive elements.

Definition 1.9 (Trace) *For $\alpha \in \mathbb{F}_{p^m}$, the trace $Tr_{\mathbb{F}_{p^m}/\mathbb{F}_p}(\alpha)$ of α over $\mathbb{F}_p$ is defined by $Tr_{\mathbb{F}_{p^m}/\mathbb{F}_p}(\alpha) = \alpha + \alpha^p + \cdots + \alpha^{p^{m-1}}$*

1.1.2 Polynomials

Let R be an arbitrary ring. The expression,

$$f(x) = \sum_{i=0}^{n} a_i x^i = a_0 + a_1 x + \ldots + a_n x^n$$

where $n \geq 0$, the *coefficients* $a_i \in R$ and x is *indeterminate* over R is called a *polynomial* over R. When $n \neq 0$, n is called the *degree* of the polynomial.

All polynomials over a *ring* forms a *ring* among themselves with the associated operations.

Addition can be defined on the elements of $\mathbb{F}_p[x]$, where $f(x) = \Sigma_{i=0}^{n} a_i x^i$ and $g(x) = \Sigma_{i=0}^{n} b_i x^i$ and $deg(f)$ denotes highest degree of the polynomial as,

$$f(x) + g(x) = \Sigma_{i=0}^{max(deg(f),deg(g))}(a_i + b_i)x^i$$

It is important to emphasize that, since a_i and b_i are both elements of $\mathbb{F}_p$, their addition is performed $\mod P$. *Multiplication* in the field is defined in a similar way, for any integer k,

$$f(x)g(x) = \Sigma_{k=0}^{deg(f)+deg(g)}(\Sigma_{i+j=k} f_i g_i)x^k \mod P$$

Definition 1.10 (Polynomial ring) *The ring formed by the polynomials over R with the associated operations (namely,* addition *and* multiplication*), is called the polynomial ring over R and denoted by $R[x]$.*

It can be easily seen that,

1. $R[x]$ is *commutative* iff R is *commutative.*
2. $R[x]$ is a ring with *identity* iff R has an *identity.*
3. $R[x]$ is an *integral domain* iff R is an *integral domain.*

This in fact one condition short for $R[x]$ (existence of multiplicative inverse) of being a field by itself. This in fact brings in the notion of *irreducible polynomial* in the field.

Definition 1.11 (Irreducible polynomial) *A polynomial P is said to be* irreducible *over a field $\mathbb{F}_p$ (or irreducible in $\mathbb{F}_p[x]$), if P has positive degree and cannot be represented as the product of two non-constant polynomials from $\mathbb{F}_p[x]$.*

Theorem 1.12 *For $P \in \mathbb{F}_p[x]$, the residue class ring $\mathbb{F}_p[x]/P$ is a field iff P is irreducible over $\mathbb{F}_p$.*

Definition 1.13 (Polynomial over an extension field) *A polynomial, whose coefficients are elements of $\mathbb{F}_{p^m}$, is said to be a polynomial over $\mathbb{F}_{p^m}$.*

1.1.3 Vector spaces

Definition 1.14 (Vector space) *Let V be a set of elements (n-tuple of elements of $\mathbb{F}$) called* vectors *and let $\mathbb{F}$ be a field of elements called* scalars. *An addition operation "$+$" is defined between vectors and a scalar multiplication operation "$\cdot$" is defined such that for a scalar $a \in \mathbb{F}$ and a vector $v \in V$, $a \cdot v \in V$. Then V is a* vector space *over $\mathbb{F}$ if "$+$" and "$\cdot$" satisfies the following,*

1. *V forms a* commutative *group under "$+$".*
2. *The operations "$+$" and "$\cdot$" distribute,* i.e. *for $a, b \in \mathbb{F}$ and $u, v \in V$, $(a + b) \cdot v = a \cdot v + b \cdot v$ and $a \cdot (u + v) = a \cdot u + a \cdot v$.*
3. *The operation "$\cdot$" is associative,* i.e. *for $a, b \in \mathbb{F}$ and $v \in V$, $(a \cdot b) \cdot v = a \cdot (b \cdot v)$.*

$\mathbb{F}$ is called the scalar field of the vector space V.

The vector $c_1x_1 + c_2x_2 + \ldots + c_mx_m$ with arbitrary scalar coefficients $c_1, c_2, \ldots, c_m$ is called a **linear combination** of the vectors $x_1, x_2, \ldots, x_m$.

A set of vectors $x_1, x_2, \ldots, x_m$, is said to be **linearly dependent** if at least one of the vectors in the set can be expressed as a *linear combination* of one or more of the other vectors in the set. Otherwise the set is called **linearly independent**.

Consider a matrix formed from m n-length vectors with each vector corresponding to a row in the matrix. If the rank of the matrix is m the set of vectors is linearly independent. If the rank is less than m the set of vectors is linearly dependent. If the rank r is less than m then there are exactly r vectors in the set which are linearly independent and the remaining vectors can be expressed as a linear combination of these r independent vectors. Thus *linear dependence or independence* of a set of vectors is determined from the rank of a matrix formed from them.

A **basis** of a vector space is any set of linearly independent vectors that spans the space. Each vector of the space is then a unique linear combination of the vectors of this basis.

The **dimension** of a vector space is the number of elements of a basis.

Note that the extension field $\mathbb{F}_{q^m}$ is in an m-dimensional vector over $\mathbb{F}_q$.

1.1.4 Finite field representations

The representation of the field elements determines the behavior of the finite field arithmetic. The most common representations are [28],

- **the powers representation:** Let α be a primitive element of $\mathbb{F}_{p^m}$. In the powers representation, the set of elements of the field can then be represented as,

$$\{0, 1, \alpha, \alpha^2, \cdots, \alpha^{p^m-2}\}$$

- **normal basis:** For the field $\mathbb{F}_{p^m}$, where p is the characteristic of the field, $\exists\beta$ such that, the m elements

$$\{\beta, \beta^p, \beta^{p^2}, \cdots, \beta^{p^{m-1}}\}$$

are linearly independent and hence form a basis. Such a basis is called normal.

The first normal basis multiplication algorithm was reported by Massey and Omura[44] and its first implementation was reported by Wang *et al.*[59].

- **standard basis:** The standard basis is a natural representation of finite field elements as polynomials over a ground field, which is also known as polynomial basis. It is defined as follows,

Let $\alpha \in \mathbb{F}_{p^m}$ be the root of an irreducible polynomial of degree m over $\mathbb{F}_p$. The standard or polynomial basis of $\mathbb{F}_{p^m}$ is then

$$\{1, \alpha, \cdots, \alpha^{m-1}\}$$

Thus, in this representation each element of $\mathbb{F}_{p^m}$ is expressed as a polynomial

$$c_0 + c_1\alpha + c_2\alpha^2 + \cdots + c_{m-1}\alpha^{m-1}$$

over $\mathbb{F}_{p^m}$.

Because of its simplicity, the standard basis representation has been widely used.

- **dual basis:** The dual basis is not a concrete basis like the polynomial basis or the normal basis; it rather provides a way of using a second basis for computations. Using a dual basis can provide a way to easily communicate between devices that use different bases, rather than having to explicitly convert between bases using the change of bases formulas. The original dual basis representation for finite field multiplication is due to Berlekamp[10].

Let $\alpha_0, \alpha_1, \cdots, \alpha_{m-1}$ be a basis and h be a linear function $h : \mathbb{F}_{p^m} \mapsto \mathbb{F}_p$, *i.e.*$\forall a, b \in \mathbb{F}_{p^m}$ and $c \in \mathbb{F}_p$, $h(a+b) = h(a)+h(b)$ and $h(ca) = c \cdot h(a)$. The dual basis of the basis $\alpha_0, \alpha_1, \cdots, \alpha^{m-1}$ with respect to h is $\beta_0, \beta_1, \cdots, \beta_{m-1}$, such that for $0 \leq i, j \leq m-1$, $h(\alpha_i\beta_j) = 1$ if $i = j$, 0 otherwise. Usually h is the trace function.

As an example of a finite field extension, consider the field $\mathbb{F}_8$. We can use three alternate and equivalent representations to represent each element in the field.

Let α be root of the primitive polynomial $x^3 + x + 1 \in \mathbb{F}_2[x]$. So we can represent $\mathbb{F}_8$ as,

$$\mathbb{F}_8 \cong \mathbb{F}_2[x]/(x^3 + x + 1)$$

as the ring of all polynomials over $\mathbb{F}_2$ modulo the third-degree irreducible polynomial $x^3 + x + 1$. Then the elements of $\mathbb{F}_8$ are $0 = 0, \alpha^0 = 1, \alpha^1 = \alpha, \alpha^2 = \alpha^2, \alpha^3 = \alpha + 1, \alpha^4 = \alpha^2 + \alpha, \alpha^5 = \alpha^2 + \alpha + 1, \alpha^6 = \alpha^2 + 1$.

- In the powers representation all non-zero elements in $\mathbb{F}_8$ may be represented as powers of a primitive field element α (see details in [36]), then each non-zero element is of the form α^n for $n = 0, 1, \cdots, 6$.

- In the polynomial representation each element in the field $\mathbb{F}_8 = \mathbb{F}_{2^3}$ is represented as polynomials with degree less than 3 whose coefficients belong to $\mathbb{F}_2$. The polynomials are defined according to the irreducible polynomial that generates the field.

- In the m-tuple representation each element in the field $\mathbb{F}_8 = \mathbb{F}_{2^3}$ can be represented as an 3-dimensional binary vector, *i.e.*, a binary 3-tuple. Each vector is determined by the coefficients of the respective polynomial representation.

We can take advantage of the powers representation in a mathematical framework while the m-tuple representation is convenient to deal with digital hardware.

Among the above mentioned, two types of representations have been of fundamental interest. The powers (or, logarithmic as it is sometimes called) representation and the basis expansion representation. The logarithmic representation yields simple multiplication, division, exponentiation and inversion operations. By contrast, the addition is much more complicated and elements to be added are usually converted to a basis expansion representation and the sum is reconverted to a logarithmic representation. The basis expansion representation is based on an expansion of elements of the extension field with respect to a field basis. In this representation, addition is easily implemented but all of multiplication, division, exponentiation, inversion are more complicated than with the logarithmic representation. Fortunately, they can be performed without converting the elements into a logarithmic representation (even though it is sometimes done when the field extension is not too large), and this is why the basis expansion representation is more widely used in practice.

1.1.5 Matrices

The problem of solving a system of linear equations leads to the concept of matrix; moreover, any linear transformation of a vector space is completely determined by a matrix. General vector spaces do not possess a multiplication operation. A vector space equipped with an additional bilinear operator defining the multiplication of two vectors is called an algebra over a field.

A field $\mathbb{F}_{2^m}$ can be seen as a vector space over $\mathbb{F}_2$ and multiplication by an element is a linear transformation on that space. We get a matrix representation by choosing a basis and expressing the linear transformation in terms of the transformation of coefficients.

For a given *companion matrix* A (ref. [27] second edition, page - 307), of the irreducible polynomial $f(x)$, we have $f(A) = 0$. The matrix A generates the cyclic group $< A >$ of order $m - 1$, which is isomorphic to $\mathbb{F}^*_{2^m}$, and the ring of matrices

$$\mathbb{F}_2[A] = \{0, I, A, \cdots, A^{m-2}\}$$

is isomorphic to the field $\mathbb{F}_{2^m}$ (I being the identity matrix).

For our implementation we needed the generator matrix of HyMES in row echelon form. We shall need to define the matrix-vector multiplication as well. Let $\alpha \in \mathbb{F}_{2^m}$ be a field element, we will consider,

- a binary vector representation of α, which is defined by the correspondence, $\alpha \Leftrightarrow \alpha_0, \alpha_1, \cdots, \alpha_{m-1}$, where $\alpha_i \in \{0, 1\}$.

- a polynomial representation, being isomorphic to the set of polynomials modulo $p(x)$ in the ring $\mathbb{F}_2[x]$ and where α is represented as an $(m-1)$-degree polynomial, $\alpha = f(x) = \Sigma_{i=0}^{m-1} \alpha_i x^i$.

- a matrix representation,

$$A \cdot g(x) = \begin{pmatrix} g(x) & \\ xg(x) & \mod f(x) \\ \vdots & \vdots \\ x^{m-1}g(x) & \mod f(x) \end{pmatrix}$$

  The binary matrix A is non-singular, and its first row provides only the binary vector representation of α.

Algorithm 1 Gaussian elimination

Require: Regular matrix $A \in \{0, 1\}^{m \times m}$.
  for each row $i = 1$ to m **do**
    for $j = i$ to m **do**
      if $A[i, j]$ is maximum and $i \neq j$ **then**
        X-OR i^{th} and j^{th} rows into i^{th} row.
        set row_find_flag
      end if
    end for
    if row_find_flag is not set **then**
      FAIL
    else
      for columns $i \neq j$ **do**
        Fill with 0
      end for
    end if
  end for
  return The matrix in RREF form.

1.1.6 Algorithmic constructions

With the theoretical background let us see the algorithms that are involved for both theory and practice. Here we briefly discuss the problems, given a prime p and an integer m, how do we actually perform the arithmetical operations of $\mathbb{F}_{p^m}$. Given a polynomial $f(x)$ of degree m with coefficients in $\mathbb{F}_{p^m}$, we wish to find a root $\alpha \in \mathbb{F}_{p^m}$ with $f(x) = 0$, if such a root does exist. This is the root-finding problem. Finally, given a polynomial $f(x) \in \mathbb{F}_{p^m}[x]$, we want to find the factorization $f = f_1 \times f_2 \times \cdots \times f_k$ into its irreducible factors $f_i(x) \in \mathbb{F}_{p^m}$. This is the factorization problem. We are primarily concerned with the binary field, thus hence forth we shall consider the constructions dedicated to them.

Addition and subtraction of two elements in $\mathbb{F}_{2^m}$ are defined as polynomial addition and subtraction on $\mathbb{F}_2$, respectively. Thus, both addition and subtraction are executed by exclusive-OR operation for every coefficient. Multiplication in $\mathbb{F}_{2^m}$ is defined as a polynomial multiplication modulo $g(x)$. The multiplicative inverse $f^{-1}(x)$ of $f(x) \in \mathbb{F}_{2^m}$ is defined as the element that satisfies $f(x) \cdot f^{-1}(x) = 1$, where "$\cdot$" denotes multiplication in $\mathbb{F}_{2^m}$.

Euclid's algorithm for polynomial calculates the Greatest Common Divisor (gcd) polynomial of two polynomials. The algorithm can be extended for calculating the two polynomials $g(x)$ and $h(x)$, that satisfy

$$GCD(a(x), b(x)) = g(x) \cdot a(x) + h(x) \cdot b(x) = d(x)$$

Here we give an overview of these algorithms.

Euclidean algorithm

Let $f(x), g(x) \in \mathbb{F}[x]$ be nonzero polynomials. We can use the division algorithm to write

$$f(x) = q(x)g(x) + r(x)$$

with $deg(r(x)) \leq deg(g(x))$ or $r(x) = 0$.

- If $r(x) = 0$, then $g(x)$ is a divisor of $f(x)$, and so $gcd(f(x), g(x)) = cg(x)$, for some $c \in \mathbb{F}$.
- If $r(x) \neq 0$, then it is easy to check that $gcd(f(x), g(x)) = gcd(g(x), r(x))$.

This step reduces the degrees of the polynomials involved, and so repeating the procedure leads to the greatest common divisor of the two polynomials in a finite number of steps. The Euclidean algorithm for polynomials is similar to the Euclidean algorithm for finding the greatest common divisor of nonzero integers. The polynomials $a(x)$ and $b(x)$ for which

$$gcd(f(x), g(x)) = a(x)f(x) + b(x)g(x)$$

can be replaced by integers.

Algorithm 2 Extended Euclidean algorithm

Require: Nonzero binary polynomials a and b; $deg(a)$ denotes the leading degree of the polynomial a.
Ensure: $deg(a) \leq deg(b)$.
  $u \leftarrow a, v \leftarrow b$.
  $g_1 \leftarrow 1, g_2 \leftarrow 0, h_1 \leftarrow 0, h_2 \leftarrow 1$.
  while $u \neq 0$ **do**
    $j \leftarrow deg(u) - deg(v)$.
    if $(j < 0)$ **then**
      $u \leftarrow v$
      $g_1 \leftarrow g_2$
      $h_1 \leftarrow h_2$
      $j \leftarrow -j$
    end if
    $u \leftarrow u + z^j v$
    $g_1 \leftarrow g_1 + z^j g_2$
    $h_1 \leftarrow h_1 + z^j h_2$
  end while
  $d \leftarrow v, g \leftarrow g_2, h \leftarrow h_2$
  return (d, g, h) such that $d = \gcd(a, b)$ and binary polynomials g, h satisfying $(ag+bh = d)$

Zech's logarithm

Zech's logarithms are used with finite fields to reduce a high-degree polynomial that is not in the field to an element in the field (thus having a lower degree). Unlike the traditional logarithm, the Zech's logarithm of a polynomial provides an equivalence, though it does not alter the value.

For any root α of $f(x) \in \mathbb{F}_q$, the product of α^a and α^b is $\alpha^a \alpha^b = \alpha^{(a+b) \bmod (q-1)}$. However, addition of field elements is not so easy when elements are given as power of a generator. Conversely, whereas addition is easy when elements are represented as polynomials, multiplication is not. For small fields, to facilitate addition of field elements represented as powers of a generator, we set up a table called a Zech's log table [7]. For each integer i, $0 \leq i \leq q-2$, we determine and tabulate the integer $j = Z(i)$ such that $1 + \alpha^i = \alpha^{Z(i)}$. Then

$$\alpha^a + \alpha^b = \alpha^a(1 + \alpha^{b-a \bmod (q-1)}) = \alpha^{a+Z(b-a)}$$

and $Z(b-a)$ can be obtained from the table.

1.2 Error correcting codes

The subject of Error Correcting Codes (ECC), describe methods to transfer data over a (likely) noisy communication channel. The message consists of k bits: m_1, m_2, ..., m_k. We discuss binary block codes, where the "block" term denotes fixed message size, here k bits. An *encoder* converts the message into a *codeword*. The codeword is transmitted over noisy communication channel that may introduce errors into the codeword. The *decoder* attempts to convert the transmitted codeword back into the original message. Each encoder/decoder scheme can handle a set of predefined error conditions. Some schemes only detect errors and some correct within certain limitations as well.

A systematic Error Correcting Code (ECC) can be considered as a way to add some redundant data to the message on the sender side. If the number of errors is within the capability of the code being used, the receiver can use this redundant information to recover the original message. Since the receiver does not have to ask the sender for retransmission of the data, it is also called forward error correction.

Definition 1.15 (Block codes) *A* block code *of size M over an alphabet with q symbols is a set of M, q-ary sequences of length n called codewords. If $q = 2$, the symbols are bits and the code is called binary.*

Hamming distance
The Hamming distance $d(u, v)$ between two words u and v, of the same length, is equal to the number of symbol places in which the words differ from one another. If u and v are of finite length n then their Hamming distance is finite. Given a code X, the minimum distance $d(X)$ of the code is the minimum Hamming distance between its codewords. It is also known as simply the distance of the code. A code X can correct t errors if and only if $d(X) \geq 2t + 1$.

1.2.1 Linear codes

Definition 1.16 (Linear codes) *A* linear (n, k) code *is a k dimensional linear subspace of $\mathbb{F}_q^n$. That is, a linear code is a nonempty set of n tuples over $\mathbb{F}_q$ (codewords) such that the sum of two codewords is a codeword, and the product of any codeword by a field element is a codeword. Any set of basis vectors for the subspace can be used as rows to form a $k \times n$ matrix G, called the generator matrix of the code. Any codeword is a linear combination of the rows of G.*

Definition 1.17 (Generator matrix) *If $\mathbf{f}_1, \mathbf{f}_2, ..., \mathbf{f}_k$ is a basis for a (n, k) code C, where*
$\mathbf{f}_1 = (f_{11}, f_{12}, ..., f_{1n})$,
$\mathbf{f}_2 = (f_{21}, f_{22}, ..., f_{2n})$,
$\vdots$
$\mathbf{f}_k = (f_{k1}, f_{k2}, ..., f_{kn})$,
then the matrix

$$F = \begin{pmatrix} \mathbf{f}_1 \\ \mathbf{f}_2 \\ \vdots \\ \mathbf{f}_k \end{pmatrix}$$
$$= \begin{pmatrix} f_{11} & f_{12} & \cdots & f_{1n} \\ f_{21} & f_{22} & \cdots & f_{2n} \\ f_{k1} & f_{k2} & \cdots & f_{kn} \end{pmatrix}$$

is called a generator matrix for C.

A code is often represented by describing a generator matrix. To compute a generator matrix of a given code C of length n, first determine a basis for the code as a vector space over $\mathbb{F}_q$, then put these basis vectors into a $k \times n$ matrix, where $k = \dim_{\mathbb{F}_q}(C)$.

Definition 1.18 (Dual code) *For a code $C \subseteq \mathbb{F}_q^n$ the set $C^\perp = \{x \in \mathbb{F}_q^n : \Sigma_{i=1}^n x_i c_i = 0; \forall c_i \in C$ is called dual code (of C).*

Parity check matrix
A parity check matrix H of a linear block code C is a generator matrix of the dual code. As such, a codeword c is in C if and only if the matrix-vector product $Hc^T = 0$.

Definition 1.19 (Syndrome) *Given the received vector y and a parity check matrix H, the syndrome of y is $S = Hy^T$.*

Observe that,

$$S = Hy^T = H(c+e)^T = Hc^T + He^T = He^T$$

i.e., the syndrome depends only on the error pattern e and not the transmitted codeword c.

1.2.2 Decoding of linear codes

Definition 1.20 (List decoding) *An algorithm which, for a given code C and a received vector y outputs the list $L(y|C,T)$ of all code vectors at distance T apart*

$$L(y|C,T) = \{c \in C : d(y,c) \leq T$$

is called a list decoding algorithm with decoding radius T.

For a code C with code distance d, a list decoding with decoding radius $T = \lfloor \frac{d-1}{2} \rfloor$ is called *bounded distance decoding*. Note that in this case list $L(y|C,T)$ is either empty or consist of a single code vector.

Definition 1.21 (MLD) *An algorithm which, for a given code C and a received vector y outputs the nearest code vector to y is called Maximum Likelihood Decoding (MLD).*

Hard problems

PKE schemes rely on hard problems. In case of McEliece system there are two hard problems that we define below (we restrict ourselves to Goppa codes)[16, 51, 8].

We will denote by $H_{n,t}$, with $n = 2^m$, the set of all parity check matrices of Goppa codes of support $\mathbb{F}_{2^m}$ and of generator polynomial of degree t. For all integers $r = tm$, a matrix/syndrome pair (H, s) of parameter (r, n) is constituted by an $r \times n$ binary matrix H and a word s of $\mathbb{F}_{2^r}$. We will denote $W_{n,t}$ the set of words of $\mathbb{F}_{2^m}$ of weight t.

Definition 1.22 (Bounded decoding problem) *A bounded decoder is a probabilistic Turing machine which takes as input a matrix/syndrome pair and outputs a string if* 0 *and* 1. *For any positive integers* $t, m, n = 2m$ *and* $r = tm$,
Input : *a matrix/syndrome pair;* (H, s).
Output : *a word* $e \in W_{n,t}$ *such that,* $He^T = s$.

The probability that any algorithm (bounded decoder) has solving the above problem is negligible as this problem was proved to be NP-complete.

Definition 1.23 (Code Distinguishing Problem) *Let* n *and* k *be two integers such that* $n \geq k$ *and* H *a parity check matrix. A code distinguisher is a probabilistic Turing machine taking as input a binary matrix and whose output is* 0 *or* 1.
Input : *a binary matrix* H.
Output : 1 *if* $H \in H_{n,t}$ i.e. *Goppa code parity-check matrix and* 0 *otherwise.*

1.2.3 Decoding of Goppa code

The usefulness of an error-correcting code would be greatly diminished if the decoding procedure was very time consuming. While the concept of decoding, *i.e.*, finding the nearest codeword to the received vector, is simple enough, the algorithms for decoding can vary widely in terms of time and memory requirements. Usually, the best (*i.e.*, fastest) decoding algorithms are those designed for specific types of codes. We shall examine some algorithms which deal with the general class of linear codes and so, will not necessarily be the best for any particular code in this class.

The decoding conventions are as follows, maximum likelihood decoding, minimum distance decoding and syndrome decoding. We shall concentrate on syndrome decoding.

Let C be a code (not necessarily linear) over a field $\mathbb{F}$. Assume that $m \in C$ is transmitted and that $r = m + e$ is received. If $C = \{m_1, m_2, \ldots, m_M\}$ is a code with M codewords, the set E of possible error patterns is $E = r - m_1, r - m_2, ..., r - m_M = r - m : v \in C = r - C$. Given the received vector r, there is a one-to-one correspondence between the possible error patterns and the codewords. To decode, we must examine the coset $r + C$ to find the appropriate error pattern (for example, the error pattern of least Hamming weight in the coset). Let C be a linear $[n, k]$ code over a finite field $\mathbb{F}$ of size q. Every coset of C has $|C| = q^k$ elements, and these cosets form a disjoint cover of $\mathbb{F}^n$, thus an $[n, k]$ linear code has q^{n-k} cosets in $\mathbb{F}^n$.

Recall the definition of syndrome 1.19. Here we briefly mention the steps involved in *Syndrome decoding* method.

1. If the generator matrix takes the form $G = [I_k|R]$, work out the parity check matrix $H = [R^T|I_{n-k}]$.

2. Enumerate the cosets of C in $\mathbb{F}$ and chose coset leaders σ_i for $i = 1, \ldots, 2^{n-k}$

3. Compute the syndromes $s(\sigma)$ for each coset leader and construct the syndrome lookup table.

4. Compute the syndrome $s(r)$ and from the table determine which σ_i satisfies $s(r) = s(\sigma_i)$.

5. Decode the message as the first k bits of $y + \sigma_i$.

1.2.4 Binary Goppa codes

Let m be a positive integer, and let n and t be two positive integers such that $n \leq 2^m$ and $t < n/m$. A binary Goppa code $\Gamma(L, g)$ is defined by an ordered subset $L = (\alpha_1, \ldots, \alpha_n)$ of $\mathbb{F}_{2^m}$ of cardinality n, called *support*, and an irreducible[1] monic polynomial $g(z)$ of degree t in $\mathbb{F}_{2^m}[z]$, called *generator*. It consists of all words $a = (a_1, \ldots, a_n) \in \mathbb{F}_2^n$ such that

$$R_a(z) \triangleq \sum_{j=1}^{n} \frac{a_j}{z - \alpha_j} \bmod g(z) = 0. \tag{1.1}$$

This code is linear, has dimension[2] $k \geq n - tm$ and minimum distance $2t + 1$ at least. We denote $\mathcal{G}_{m,n,t}$ the set of all binary Goppa codes with a support of cardinality n in $\mathbb{F}_{2^m}$ and an irreducible generator of degree t over $\mathbb{F}_{2^m}$.

Decoding of binary Goppa codes:

Here we give an outline which we shall see in Chapter 2, in implementation and algorithm oriented details. We use Patterson algorithm for decoding.

Since the received word $y = x + e = e \bmod g(z)$, the syndrome of a received word, with reference to equation 1.1 can be written as,

$$s(z) = \Sigma_{j=1}^{n} \frac{y}{z - \alpha_j} \bmod g(z) = \Sigma_{j=1}^{n} \frac{e}{z - \alpha_j} \bmod g(z) \tag{1.2}$$

To recover the error positions we have to solve the key equation $\sigma(z) \cdot s(z) = e(z)$, where $\sigma(z)$ is the error locator polynomial and $e(z)$ is the error polynomial.

Note that it can be shown that $e(z) = \sigma(z)'$ is the formal derivative of the error-locator polynomial and by splitting $\sigma(z)$ into even and odd polynomial parts $\sigma(z) = u(z)^2 + z \cdot v(z)^2$

[1] square-free without roots in L in the most general definition

[2] for parameters suitable with the McEliece system, the equality always holds

, we finally determine the following equation which needs to be solved to determine error positions:

$$s(z)(u(z)^2 + z \cdot v(z)^2) = b(z)^2 \bmod g(z) \tag{1.3}$$

To solve equation 1.3 for a given codeword, we first compute an inverse polynomial $T_i(z) = s(z)^{-1} \bmod g(z)$. It follows that $(T(z) + z)v(z)^2 = u(z)^2 \bmod g(z)$. Then, for all $1 \leq i \leq t-1$, the polynomial $T_i(z) \in \mathbb{F}_{2^m}[z]$ of degree at most $t-1$ such that

$$T_i(z)^2 = z^i \bmod g(z), \text{ we also denote } T_i(z) = \sqrt{z^i} \bmod g(z)$$

We have $z^{2^{mt}} = z \bmod g(z)$, thus we can obtain $T_1(z) = z^{2^{mt-1}} \bmod g(z)$ by squaring polynomials modulo $g(z)$. For odd values, we have $T_{2i}(z) = z^i$ and for even values $T_{2i+1} = z^i T_1(z) \bmod g(z)$.

Algorithm 3 Decoding of binary Goppa code

Require: Received vector y, the Goppa code $\Gamma(L, g)$.
Compute syndrome $s(z)$ for the word y.
Compute $T(z) = s(z)^{-1} \bmod g(z)$
Compute $S(z) = \sqrt{T(z) + z}$
Compute $u(z)$ and $v(z)$ with Fill with $u(z) = v(z)S(z) \bmod g(z)$
Compute the locator polynomial $\sigma(z) = u(z)^2 + zv(z)^2$
Find the roots of $\sigma(z)$
return The error positions.

Different key equations

Usually the *key equation* is the equation connecting the *syndrome* and the *locator polynomial*.

Let, $b = (b_1, b_2, \ldots, b_n)$ be the received vector. $L = (\alpha_1, \alpha_2, \ldots, \alpha_n)$ be the support. $g(x) \in \mathbb{F}_{2^m}[z]$ be the degree t generator. We have $b = a + e$, a being a code word and e an error of weight t.

$$R(z)\sigma(z) = \sigma'(z) \bmod g(z). \tag{1.4}$$

If the key equation is of the form 1.4, we have the degrees as,

polynomial	degree of the polynomial
$R(z)$	$t-1$
$\sigma(z)$	t
$\sigma'(z)$	$t-1$

We solve the equation with Patterson algorithm 1.2.6.

When,

$$R(z) = \sum_{i=1}^{n} \frac{b_i}{z - \alpha_i} \bmod g(z) = \sum_{i=1}^{n} \frac{e_i}{z - \alpha_i} \bmod g(z)$$

and

$$\sigma(z) = \prod_{i=1}^{n}(z - \alpha_i)^{e_i}$$

and degree of $\sigma(z) = t$.

$$R(z)\sigma(z) = \sigma'(z) \bmod g^2(z) \tag{1.5}$$

If the key equation is of the form 1.5, we have the degrees as,

polynomial	degree of the polynomial
$R(z)$	$2t-1$
$\sigma(z)$	t
$\sigma'(z)$	$t-1$

We solve the equation with Euclidean algorithm 1.1.6. Note that, we can solve this equation as $g(z)$ is square free.

Let, H be a parity check matrix defined for Goppa code as,

$$\mathbf{H} = \begin{pmatrix} 1 & 1 & \dots & 1 \\ \alpha_1 & \alpha_2 & \dots & \alpha_n \\ \vdots & \vdots & \vdots & \vdots \\ \alpha_1^{2t-1} & \alpha_2^{2t-1} & \dots & \alpha_n^{2t-1} \end{pmatrix} \qquad \begin{pmatrix} \frac{1}{g(\alpha_1)^2} & \dots \\ \dots & \dots \\ \ddots & \frac{1}{g(\alpha_n)^2} \end{pmatrix}$$

We have the syndrome $S(x)$ of the form,

$$S(x) = \sum_{i=0}^{2t-1} S_i x^i$$

where $S_i = e^i \sum_{j=1}^{n} \frac{\alpha_j^i}{g(\alpha_j)^2} = b^i \sum_{j=1}^{n} \frac{\alpha_j^i}{g(\alpha_j)^2}$. Thus we get, $\sigma(x) = \prod_{j=1}^{n}(1 - x\alpha_j)^{e^j}$. The evaluator polynomial

$$w(x) = \sum_{j=1}^{n} \frac{e_j}{g(\alpha_j)^2} \frac{\sigma(x)}{1 - x\alpha_j}$$

and $\frac{1}{1-x\alpha_j} = \sum_{\ell \geq 0}(x\alpha_j)^\ell$, Thus,

$$w(x) = \sigma(x) \sum_{i=1} \frac{e_j}{g(\alpha_j)^2} \sum_{\ell \geq 0}(x\alpha_j)^\ell \tag{1.6}$$

$$= \sigma(x) \sum_{\ell \geq 0}(\sum_{j=1}^{n} \frac{e_i \alpha_j^\ell}{g(\alpha_j)^2})x^\ell \tag{1.7}$$

$$= \sigma(x) \sum_{\ell \geq 0} S_\ell x^\ell \tag{1.8}$$

$$= \sigma(x)S(x) \bmod x^{2t} \tag{1.9}$$

Hence we get the third equation,

$$S(x)\sigma(x) = w(x) \bmod x^{2t} \tag{1.10}$$

If the key equation is of the form 1.10, we have the degrees as,

polynomial	degree of the polynomial
$S(x)$	$2t-1$
$\sigma(x)$	t
$w(x)$	$t-1$

We solve the equation with Berlekemp-Massey and Euclidean algorithm 1.1.6.

1.2.5 Root finding of error locator polynomial

As mentioned, the most time consuming part of decoding of binary Goppa codes is the root finding of error locator polynomial. That was the initial motivation for us to investigate the possible ways to make the part faster. As for many equivalent classes of codes the problem is the same.

Several approaches toward root finding in characteristic 2 are possible, their efficiency depends on the size of the parameters m and t.

- Chien search (described later 1.2.6) computes roots by evaluating artfully the polynomial in all points of L. This method is recommended for hardware implementations and coding theory applications in which m is small.
- BTA is a recursive algorithm using trace function properties. It is a faster method for secure parameters for McEliece-type cryptosystems.
- Equal-degree factorization is an algorithm of Cantor and Zassenhaus [58, Chapter 14]. Its scope is more general but under some adaptations, it enables to find roots of polynomials in characteristic 2 (this specific case is treated in Exercise 14.16 in [58]). Then it has similarities with BTA (use of trace function, computations of gcds and a recursive structure) but seems slightly more expensive. Its time complexity is $\mathcal{O}((m+\log t)\, t^2 \log t)$ operations in $\mathbb{F}_{2^m}$.
- Zinoviev procedures are dedicated to root finding for polynomials of degree less than 10. For $m \geq 11$, they are (theoretically at least) more efficient than Chien search.

Definition 1.24 *An affine polynomial has the form:*

$$A(z) = L(z) + c$$

where L is a linearized polynomial over $\mathbb{F}_{q^m}$ and $c \in \mathbb{F}_{q^m}$.

Definition 1.25 *A linearized polynomial over $\mathbb{F}_{q^m}$ is a polynomial of the form:*

$$L(z) = \sum_{i=0}^{n} l_i \cdot z^{q^i}.$$

with $l_i \in \mathbb{F}_{q^m}$ and $l_n = 1$.

In our case, $q = 2$. The *Trace polynomial* is an example of linearized polynomial.

Finding Roots of an Affine Polynomial

Let us have an affine polynomial $A(z) = L(z) + c = \sum_{i=0}^{m-1} l_i \cdot z^{2^i} + c$.
Consider $(\alpha_1, \cdots, \alpha_m)$ is a $\mathbb{F}_2$-basis of $\mathbb{F}_{2^m}$, $(l_i)_{1 \leq i \leq m}$, c and x are elements of $\mathbb{F}_{2^m}$. Guess $x = (x_1, \cdots, x_m)$ is a root of A. Finding zeroes of an affine polynomial is equivalent to solving a linear system. Indeed, we have:

$$\begin{aligned} A(x) = 0 &\Leftrightarrow L(x) = c \\ &\Leftrightarrow \sum_{i=1}^{m} x_i \cdot L(\alpha_i) = \sum_{i=1}^{m} c_i \cdot \alpha_i \quad \text{(using linearity of } L) \\ &\Leftrightarrow \sum_{i=1}^{m} \sum_{j=1}^{m} x_i \; l_{i,j} \cdot \alpha_i = \sum_{i=1}^{m} c_i \cdot \alpha_i \quad \text{(linear system in } x_i). \end{aligned}$$

We use the following notation,

- σ_e is the error locator polynomial of the error word e.
- d_{max} is the maximum degree up to which we use Zinoviev procedures.
- D_{Zin} is the set of degrees for which we apply Zinoviev procedures.
- $(\beta_1, \cdots, \beta_m)$=$(\alpha, \alpha^2, \cdots, \alpha^m)$ is a fixed polynomial basis of $\mathbb{F}_{2^m}$ over $\mathbb{F}_2$ where α is a primitive element of $\mathbb{F}_{2^m}$.

1.2.6 Algorithmic constructions

Berlekamp trace algorithm :
Berlekamp trace algorithm was originally published in [5]. This algorithm is very efficient for finite fields with small characteristic. The trace function $Tr(\cdot)$ of $\mathbb{F}_{2^m}$ over $\mathbb{F}_2$ is defined by

$$Tr(z) = z + z^2 + z^{2^2} + ... + z^{2^{m-1}}$$

it maps the field $\mathbb{F}_{2^m}$ onto it's ground field $\mathbb{F}_2$. A key property of the trace function is that if $(\beta_1, ..., \beta_m)$ is any basis of $\mathbb{F}_{2^m}$ over $\mathbb{F}_2$, then every element $\alpha \in \mathbb{F}_{2^m}$ is uniquely represented by the binary m-tuple

$$(Tr(\beta_1 \cdot \alpha), ..., Tr(\beta_m \cdot \alpha)) .$$

The basic idea of the Berlekamp trace algorithm is that any $f(z) \in \mathbb{F}_{2^m}[z]$, with $f(z) \mid z^{2^m} - z$, splits into two polynomials

$$g(z) = \gcd(f(z), Tr(\beta \cdot z)) \text{ and } h(z) = \gcd(f(z), 1 + Tr(\beta \cdot z)).$$

```
BTA(σ, i)
    if deg(σ) ≤ 1 then
        return rootof(σ)
    σ_0 ← gcd(σ(z), Tr(β_i · z))
    σ_1 ← gcd(σ(z), 1 + Tr(β_i · z))
    return BTA(σ_0, i + 1), BTA(σ_1, i + 1)
```

```
Berlekamp_trace_algorithm(σ)
    return BTA(σ, 1)
```

Figure 1.1: Pseudo code for the Berlekamp trace algorithm

The above property of the trace ensures that if β iterates through the basis $(\beta_1, ..., \beta_m)$, we can separate all the roots of $f(z)$ (see Figure 1.1).

Patterson algorithm :

The Patterson algorithm [45] solves the Goppa code key equation: given $R(z)$ and $g(z)$ in $\mathbb{F}_{2^m}[z]$, with $g(z)$ of degree t respectively, find $\sigma(z)$ of degree t such that

$$R(z)\sigma(z) = \frac{d}{dz}\sigma(z) \bmod g(z)$$

We write $\sigma(z) = \sigma_0(z)^2 + z\sigma_1(z)^2$. Since $\frac{d}{dz}\sigma(z) = \sigma_1(z)^2$, we have

$$(1 + zR(z))\sigma_1(z)^2 = R(z)\sigma_0(z)^2 \bmod g(z).$$

Because $g(z)$ is irreducible, $R(z)$ can be inverted modulo $g(z)$. We put $h(z) = z + R(z)^{-1} \bmod g(z)$ and we have

$$h(z)\sigma_1(z)^2 = \sigma_o(z)^2 \bmod g(z).$$

The mapping $f(z) \mapsto f(z)^2 \bmod g(z)$ is bijective and linear over $\mathbb{F}_2^{tm}$, there is a unique polynomial $S(z)$ such that $S(z)^2 = h(z) \bmod g(z)$. We have

$$S(z)\sigma_1(z) = \sigma_0(z) \bmod g(z).$$

The polynomial $\sigma_0(z), \sigma_1(z)$ are the unique solution of the equation

$$\begin{cases} S(z)\sigma_1(z) = \sigma_0(z) \bmod g(z) \\ \deg \sigma_0 \leq t/2 \\ \deg \sigma_1 \leq (t-1)/2 \end{cases} \tag{1.11}$$

The three steps of the algorithm are the following

1. Compute $h(z) = z + R(z)^{-1} \bmod g(z)$ using the extended Euclidian algorithm.
2. Compute $S(z) = \sqrt{h(z)} \bmod g(z)$

If $s(z)$ such that $s(z)^2 = z \bmod g(z)$ has been precomputed and $h(z) = h_0 + h_1 z + \ldots + h_{t-1} z^{t-1}$, we have

$$S(z) = \sum_{i=0}^{(t-1)/2} h_{2i}^{2^{m-1}} z^i + \sum_{i=0}^{t/2-1} h_{2i+1}^{2^{m-1}} z^i s(z)$$

3. Compute $(\sigma_0(z), \sigma_1(z))$ as in (1.11) using the extended Euclidian algorithm.

The polynomial $\sigma(z) = \sigma_0(z)^2 + z\sigma_1(z)^2$ is returned.

Zinoviev Procedures (1996):

Zinoviev methods [60] find the monic affine multiple of smallest degree of any polynomial f of degree $d \leq 10$ over $\mathbb{F}_{2^m}$. At step $i \geq 0$, we compute a multiple of f of degree $2^{\lceil \log_2 d \rceil + i}$ and we try to decimate the non-linear terms by solving a homogeneous system of linear equations. If the system has no solution, we go up step $i + 1$. Besides, an algorithm proposed by Berlekamp, Rumsey and Solomon in [9] ensures to find an affine multiple of degree 2^{d-1} and thus guarantee Zinoviev methods terminate, in the worst case, at step $d - 1 - \lceil \log_2 d \rceil$. After that, finding roots of an affine polynomial is easier than in the general case (see Appendix 1.2.5). For this, we only have to solve a linear system of order m over $\mathbb{F}_2$. Then, we have to determine the roots of f, among the roots of the affine polynomial we have found. We just evaluate f in those points to do so.

Consider q is a prime power and m is a positive integer. Let us give the useful definitions:

Chien Procedure (1964)

Chien search is a recursive algorithm. It is a clever exhaustive search. Let $f(x) = a_0 + a_1 \cdot x + \cdots + a_t \cdot x^t$ be a polynomial over $\mathbb{F}_{2^m}$ and let α be a generator of the multiplicative group $\mathbb{F}_{2^m}^*$.

$$\begin{aligned} f(\alpha^i) &= a_0 + a_1 \cdot \alpha^i + \cdots + a_t \cdot (\alpha^i)^t \\ f(\alpha^{i+1}) &= a_0 + a_1 \cdot \alpha^{i+1} + \cdots + a_t \cdot (\alpha^{i+1})^t \\ &= a_0 + a_1 \cdot \alpha^i \cdot \alpha + \cdots + a_t \cdot (\alpha^i)^t \cdot \alpha^t \end{aligned}$$

Set $a_{i,j} = a_j(\alpha^i)^j$. It is easy to obtain $f(\alpha^{i+1})$ from $f(\alpha^i)$ since we have that $a_{i+1,j} = a_{i,j} \cdot \alpha^j$. Moreover, if $\sum_{j=0}^{t} a_{i,j} = 0$, then α^i is a root of f.

Algorithm 4 simplified BTZ without precomputation - BTZ(f, d, i)

First call: $f \leftarrow \sigma_e$; $d \leftarrow d_{max} \in \{2, \ldots, 10\}$; $i \leftarrow 1$.
if degree$(f) \leq d$ **then**
  return ZINOVIEV(f, d);
else
  $g \leftarrow \gcd(f, \mathrm{Tr}(\beta_i \cdot z))$;
  $h \leftarrow f/g$;
  return BTZ$(g, d, i+1) \cup$ BTZ$(h, d, i+1)$;
end if

Algorithm 5 BTZ with precomputation - BTZ(f, D, i)

First call: $f \leftarrow \sigma_e$; $D \leftarrow D_{Zin} \subset \{2, \ldots, 10\}$; $i \leftarrow 1$.
{precomputation phase}
for $1 \leq i \leq m$ **do**
  $T_i \leftarrow \mathrm{Tr}(\beta_i \cdot z)) \bmod f$;
end for
$i \leftarrow 1$;
{computation phase}
if degree$(f) \in D$ **then**
  return ZINOVIEV(f, d);
else
  $T \leftarrow T_i \bmod f$;
  $g \leftarrow \gcd(f, T)$;
  $h \leftarrow f/g$;
  $i \leftarrow i + 1$;
  return BTZ$(g, d, T) \cup$ BTZ(h, d, T);
end if

Part II

The Hybrid McEliece Encryption Scheme(HyMES)

Chapter 2

Implementation of Hybrid McEliece Encryption Scheme (HyMES)

In this chapter we describe the theory and implementation of HyMES, proposed by Nicolas Sendrier and myself. The system is functional and the full source code can be found at `www.http://www-roc.inria.fr/secret/CBCrypto/index.php?pg=hymes`, the system is licenced with LGPL.

To follow the logical development we shall start with a brief description of original McEliece crypto-system and then we shall advance to the theory and implementation aspects of HyMES.

2.1 Original McEliece crypto-system

Introduced by Robert J. McEliece in 1978 [40], the scheme is supposedly the fastest among public key crypto-systems and is still considered secure with reasonable cryptographic parameters. While the hovering existence of quantum computers pose threat to RSA-like schemes, McEliece system is believed to be immune. The system has *good security reduction*, is *efficient* but *large public key size*. There are several attempts made to reduce the key size to a reasonable standard, following P. Gaborit's paper in 2005 [24, 4, 42]. There is a recent attack proposed by Faugere *et al.* [21]. The idea behind this scheme is to first select a particular code for which an efficient decoding algorithm is known, and then to disguise the code as a general linear code. Since the problem of decoding an arbitrary linear code is NP-hard, a description of the original code can serve as the private key, while a description of the transformed code serves as the public key.

The McEliece scheme originally designed with classical binary Goppa codes has resisted cryptanalysis to date, though the original parameters are proved to be weak [11]. It is also notable as being the one of the first public-key encryption schemes to use randomization in the encryption process.

For each irreducible polynomial $g(x)$ over $\mathbb{F}_{2^m}$ of degree t, there exists a binary irreducible Goppa code of length $n = 2^m$ and dimension $k \geq n - mt$, capable of correcting

any pattern of $\leq t$ errors. As it is a linear code, it can be described by its $k \times n$ generator matrix G. With the aid of a regular $k \times k$ scrambler matrix S and an $n \times n$ permutation matrix P, a new generator matrix $\hat{G}$ is constructed that hides the structure of G.

$$\hat{G} = S \cdot G \cdot P$$

The public key consists of $\hat{G}$ and the matrices S and P together with $g(x)$ form the secret key. The new matrix $\hat{G}$ looks like a generator matrix of an arbitrary linear code, which is assumed to be difficult to decode if the trapdoor information (*i.e.* the structure of the Goppa code) is unknown. The encryption operation consists of multiplication of the k-bit message vector x by $\hat{G}$ and the modulo 2 addition of an error vector e with Hamming weight t.

$$c = x \cdot \hat{G} \oplus e$$

The first step of the decryption is the computation of $c \cdot P^{-1}$. Subsequently the decoding scheme makes it possible to recover $x \cdot S$ from $c \cdot P^{-1} = (x \cdot S \cdot G) \oplus (e \cdot P^{-1})$. The message x is finally constructed by a multiplication with S^{-1}. Bellow we give formal description of the system.

2.1.1 System description

As for any public key encryption schemes, McEliece scheme also can be described in three successive stages, **Key-generation**, **Encryption** and **Decryption**.

Key-generation

The objective of this stage is to generate the set of keys that will be used for encryption and decryption, the public-key and the private-key respectively.

1. Users select a binary (n, k)-binary Goppa code Γ capable of correcting t errors. This code possesses an efficient decoding algorithm.

2. Alice generates a $k \times n$ generator matrix G for the code Γ.

3. Select a random $k \times k$ binary non-singular matrix S.

4. Select a random $n \times n$ permutation matrix P.

5. Compute the $k \times n$ matrix $\hat{G} = S \cdot G \cdot P$.

6. Alice's public key is $(\hat{G}, t)$; her private key is (S, G, P).

Encryption

Suppose Bob wishes to send a message x to Alice whose public key is $(\hat{G}, t)$:

1. Encode the message as a binary string of length k.
2. Compute the vector $c' = x\hat{G}$.
3. Generate a random n-bit vector e containing at most t ones.
4. Compute the ciphertext as $c = c' + e$.

Decryption

1. Compute the inverse of P, P^{-1}.
2. Compute $\hat{c} = cP^{-1}$.
3. Use the decoding algorithm for the code C to decode $\hat{c}$ to $\hat{x}$.
4. Compute $x = \hat{x}S^{-1}$.

2.1.2 Security

As we stated earlier, the McEliece cryptosystem is considered to be fairly secure. For reasonable parameters the system resisted cryptanalysis to date. The two main types of attack approaches are noted bellow.

The first type of attacks, known as *structural attacks* or *key attacks* try to reconstruct a decoder for the code generated by the public-key $\hat{G}$ by studying its structure. From the very construction of the system, the code generated by the public key $\hat{G}$ is equivalent to Γ. The best known attacks consist in enumerating the codes in the family to find a code which is equivalent to Γ [49, 37, 38, 23].

The second type of attacks aim to decode the intercepted cipher text related to the public code generated by the public-key, known as *decoding attacks* or *message attacks*. Since Γ is equivalent to the code generated by the matrix $\hat{G}$, both codes have the same error-correcting capability. The cost of the attack depends on the parameters of the code, its length, its dimension and its error-correcting capability. It implies that the parameters of the system have to be chosen carefully. These attacks have been thoroughly studied in [33, 14, 11].

2.1.3 Strengths and Drawbacks

McEliece is strong against cryptanalysis. Attempts have been made to cryptanalyze McEliece, but none have been really successful. This public-key cipher runs much faster than any algorithm relying on number theory.

The size of the public key ($\hat{G}$) is quite large. Using the Goppa code with parameters suggested by McEliece ($m = 10, t = 50$), the public key would consist of 2^{19} bits. The encrypted message is much longer than the plain text message.

Please note the above mentioned two drawbacks are addressed primarily during the course of this thesis.

2.1.4 Niederreiter cryptosystem

A dual variant of the McEliece scheme was proposed a few years later by Harald Niederreiter [43].

This is a knapsack-type cryptosystem which employs a t error correcting (n, k) linear code C over $\mathbb{F}_q$. Let H be an $(n-k) \times n$ parity check matrix of C, M any $(n-k) \times (n-k)$ non-singular matrix, and P any $n \times n$ permutation matrix, all over $\mathbb{F}_q$. The system is described as bellow.

- *Private Key:* H, M, and P.
- *Public Key:* $H' = MHP$ and t.
- *Messages:* n dimensional vectors y over $\mathbb{F}_q$ with weight t.
- *Encryption:* $z = yH'^T$; z, the ciphertext of dimension $n - k$.
- *Decryption:* Since $z = y(MHP)^T, z(M^T)^{-1} = (yP^T)H^T$. Use a fast decoding algorithm for C to find yP^T and thus y.

2.2 Aim for implementation and system description

Though considered fast and secure, the McEliece encryption scheme has received little interest in practice. Mainly due to the large public key size, the scheme's practical usability was in question and has never been thoroughly studied. Until recently, the existence of quantum computers imminent in the future, RSA-like schemes are no more the primary choice. People are searching for secure alternative PKCs for practical applications. The Ecrypt network of excellence project eBATS `http://www.ecrypt.eu.org/ebats/`, tried to benchmark many different crypto-systems. We submitted the McEliece PKC in the said endeavor for benchmarking. While doing so, many theoretical issues came up. We tried to gain in key size and execution time. We reduced the key size by using the generator matrix in systematic form and increased the transmission rate by encoding information in the error. We prove that under semantically secure conversion the scheme still preserve security of the original system. We came up with a few tweaks in algorithmic constructions as well. Our primary intention being to able to design a research tool which can serve as the basic standard for practical McEliece like schemes.

2.2.1 The *Hybrid* McEliece Scheme

As we already stated this is a scheme with two main modifications. Let us describe the system and then we shall see how the changes from the original contribute towards our goals of addressing main two drawbacks of the original system.

Recall that for each irreducible polynomial $g(x)$ over $\mathbb{F}_{2^m}$ of degree t, there exists a binary irreducible Goppa code $\Gamma(L, g)$ of length $n = 2^m$ and dimension $k \geq n - mt$, capable of correcting any pattern of $\leq t$ errors. We chose the subset $L = \{\alpha_1, \cdots, \alpha_n\}$ such that $g(\alpha_i) \neq 0; \forall \alpha_i \in L$ and call it *support*. We construct a *generator matrix* G for the code $\Gamma(L, g)$ and bring it to reduced row echelon form $G = (Id \mid R)$, Id being the identity matrix. Depending upon the hard problem, *indistinguishability of Goppa codes*, we will prove that upon publishing R as the *public key* we still will keep the security of original scheme. The code definition $\Gamma(L, g)$ (see §1.2.4) and thus the decoder associated with it serves as the *private key*. The encryption operation consists of multiplication of the k-bit message vector x by R, concatenating the message with the resultant vector and the modulo 2 addition of an error vector e.

$$c = x \parallel x \cdot R \oplus e$$

As we use message bits to be encoded as error, this e has a special form (it is of predefined exact weight), which we obtain through *constant weight encoding* (see at the end of this section). Again we will prove under semantically secure conversion we do not lose security for this change.

The first step of the decryption is the inverse operation of the *constant weight encoding* and subsequently the decoding scheme $\Psi_{L,g}$ for $\Gamma(L, g)$ makes it possible to recover x. Bellow we give formal description of the system.

Let $\varphi : \{0,1\}^\ell \to W_{n,t}$ be an injective mapping where $W_{n,t}$ denotes the set of words of length n and Hamming weight t. Both φ and φ^{-1} should be easy to compute and the integer ℓ should be close to $\log_2 \binom{n}{t}$. As the original scheme, we use Goppa codes for HyMES.

Let two m and t be two integers. Let $n = 2^m$ and $k = n - tm$.

Key-generation

1. Sender select a binary (n, k)-binary Goppa code $\Gamma(L, g)$ capable of correcting t errors with an efficient decoding algorithm.

2. Sender generates a $k \times (n - k)$ binary matrix R, such that $(Id \mid R)$ is a generator matrix of $\Gamma(L, g)$

3. Sender's *public key* is R; her *private key* is the pair (L, g) and thus the decoder.

Encryption

When a sender sends a message x to a reciever whose public key is R, he,

1. Encodes the message as a binary string of length k.

2. Encodes error of length ℓ into word of length n and weight t using the mapping φ.

3. Computes the vector xR.

4. Compute the ciphertext as $y = (x \parallel xR) + \varphi(e)$.

$$\begin{array}{ccc} \{0,1\}^k \times \{0,1\}^\ell & \rightarrow & \{0,1\}^n \\ (x,e) & \mapsto & (x \parallel xR) \oplus \varphi(e) \end{array}$$

Decryption

1. Use the decoding algorithm $\Psi_{L,g}$ for the code $\Gamma(L,g)$ to decode y to retrieve $e = \Psi_{L,g}(y)$ and $y - e = x \parallel *$. Where $*$ denotes the remaining part of the message.

$$\begin{array}{ccc} \{0,1\}^n & \rightarrow & \{0,1\}^k \times \{0,1\}^\ell \\ y & \mapsto & (x, \varphi^{-1}(\varphi(e))) \end{array}$$

2.2.2 Security analysis of the changes made

Let us emphasize on the main two differences compared with the original system:

- We use the error to encode information bits.
- We use a public key in *row echelon* form.

Those changes will improve the credentiality of the system by addressing two main drawbacks (public key size and information rate) of the original McEliece scheme. Here we prove these changes have no impact on the security of the system.

Cryptographic security

The first reductional proof of security for the McEliece encryption scheme was given by Kobara and Imai in [31]. In the same paper, several semantically secure conversions, generic and ad-hoc, are proposed. The purpose of those conversion is to transform a One Way Encryption (OWE) scheme, the weakest notion of security, into a scheme resistant to adaptative chosen ciphertext attack (IND-CCA2), the strongest notion of security (in the random oracle model).

In this section, we prove that under two algorithmic assumptions (the hardness of decoding and the pseudo-randomness of Goppa codes), the hybrid version of McEliece encryption scheme is one way.

One way encryption schemes

We consider a public key encryption scheme where the public key is chosen uniformly in the space $\mathcal{K}$. Let $\mathcal{P}$ and $\mathcal{C}$ denote respectively the plaintext and ciphertext spaces. We consider the sample space $\Omega = \mathcal{P} \times \mathcal{K}$ equipped with the uniform distribution P_Ω. An adversary $\mathcal{A}$ for this encryption scheme is a mapping $\mathcal{C} \times \mathcal{K} \to \mathcal{P}$. It is successful for $(x, K) \in \Omega$ if $\mathcal{A}(E_K(x), K) = x$, where $E_K(x)$ denotes the encryption of x with the public key K. The success probability of $\mathcal{A}$ for this cryptosystem is equal to

$$P_\Omega(\mathcal{A}(E_K(x), K) = x).$$

Definition 2.1 (OWE) *A public key encryption scheme is a* One Way Encryption *scheme if the probability of success of any of its adversary running in polynomial time is negligible.*

In practice, one needs more than just an OWE scheme. For instance, McEliece encryption scheme, though it is OWE, is vulnerable to many attacks [12, 15, 26, 57]. On the other hand, if we admit the existence of perfect hash functions, there are generic conversions (see for instance [3, 46]) which, starting from an OWE scheme, provide a scheme resistant against adaptative chosen ciphertext attack.

Those generic conversions as well as other specific ones exist for the original McEliece encryption scheme (see [31]).

Security assumptions

Let m and t be two positive integers, let $n = 2^m$ and $k = 2^m - tm$. We denote $\{0,1\}^{k \times n}$ the set of binary $k \times n$ matrices and by $\mathbf{G}_{m,t}$ the subset consisting of all generator matrices of a binary irreducible t-error correcting Goppa code of length n and support $\mathbf{F}_{2^m}$ (up to a permutation). Finally, recall that $W_{n,t}$ denotes the binary words of weight t and length n.

Definition 2.2 *Let m and t be two positive integers, let $n = 2^m$ and $k = 2^m - tm$. Let P_{Ω_0} be the uniform distribution over the sample space*

$$\Omega_0 = \{0,1\}^k \times W_{n,t} \times \{0,1\}^{k \times n}$$

- *An* adversary *is a procedure $\mathcal{A} : \{0,1\}^n \times \{0,1\}^{k \times n} \to W_{n,t}$. We denote $|\mathcal{A}|$ its maximal running time.*
- *The* success probability *of an adversary $\mathcal{A}$ is defined as*

$$\mathrm{Succ}(\mathcal{A}) = P_{\Omega_0}\left(\mathcal{A}(xG + e, G) = e\right).$$

- *The* success probability over $\Omega' \subset \Omega_0$ *of an adversary $\mathcal{A}$ is defined as*

$$\mathrm{Succ}(\mathcal{A} \mid \Omega') = P_{\Omega_0}\left(\mathcal{A}(xG + e, G) = e \mid (x, e, G) \in \Omega'\right).$$

- *We call (T, ε)-adversary over Ω' an adversary $\mathcal{A}$ such that $|\mathcal{A}| \leq T$ and* $\mathrm{Succ}(\mathcal{A} \mid \Omega') \geq \varepsilon$.

- *A* distinguisher $\mathcal{D}$ *is a mapping* $\{0,1\}^{k\times n} \to \{\text{true}, \text{false}\}$. *We denote* $|\mathcal{D}|$ *its maximal running time.*
- *The* advantage *of a distinguisher* $\mathcal{D}$ *for* $\mathbf{S} \subset \{0,1\}^{k\times n}$ *is defined as*

$$\text{Adv}(\mathcal{D}, \mathbf{S}) = \left| P_{\Omega_0}(\mathcal{D}(G) \mid G \in \mathbf{S}) - P_{\Omega_0}(\mathcal{D}(G)) \right|.$$

- *We call* (T, ε)*-distinguisher over* $\mathbf{S}$ *a distinguisher* $\mathcal{D}$ *such that* $|\mathcal{D}| \leq T$ *and* $\text{Adv}(\mathcal{D}, \mathbf{S}) \geq \varepsilon$.

The first assumption states the difficulty of decoding in the average case in a linear code whose parameters are those of a binary Goppa codes.

Assumption 1 *For all* (T, ε)*-adversary over* Ω_0*, the ratio* T/ε *is not upper bounded by a polymonial in* n.

The worst-case is known to be difficult (the associated decision problem is NP-complete) in the general case [8] (Syndrome Decoding) and in the bounded case [22] (Goppa Parameterized Bounded Decoding). The status of the average case is unknown, but it is believed to be difficult [2].

The second assumption states that there exists no efficient distinguisher for Goppa codes. In other words, the generator matrix of a Goppa code looks random.

Assumption 2 *For all* (T, ε)*-distinguisher over* $\mathbf{G}_{m,t}$*, the ratio* T/ε *is not upper bounded by a polymonial in* n.

There is no formal result to assess this assumption. However, there is no known invariant for linear code, computable in polynomial time, which behave differently for random codes and for binary Goppa codes.

The hybrid McEliece encryption scheme is one way

We use the notations and definitions of the previous section. The public key is a binary $k \times (n-k)$ matrix R. We consider a public injective mapping $\varphi : \{0,1\}^{\ell} \to W_{n,t}$. The hybrid McEliece encryption is defined as

$$\begin{array}{ccc} \{0,1\}^k \times \{0,1\}^{\ell} & \longrightarrow & \{0,1\}^n \\ (x, e) & \longmapsto & (x \parallel xR) + \varphi(e) \end{array}$$

Theorem 2.3 *Under Assumption 1 and Assumption 2, the hybrid McEliece system is a OWE scheme.*

Before proving the theorem, we will prove some intermediate results in the form of three lemmas. We will use the following notations:

- $\mathbf{S}_{k\times n}$ the binary systematic $k \times n$ matrices (*i.e.* of the form $(Id \mid R)$),

- $\mathbf{G}'_{m,t} = \mathbf{G}_{m,t} \cap \mathbf{S}_{k\times n}$ the systematic generator matrices of Goppa codes,
- $\mathcal{E} = \mathrm{Im}(\varphi) \subset W_{n,t}$ the image of $\{0,1\}^\ell$ by φ. In practice $\mathcal{E}$ can be any subset of $W_{n,t}$.
- We consider the three following decreasing subsets of $\Omega_0 = \{0,1\}^k \times W_{n,t} \times \{0,1\}^{k\times n}$

$$\begin{array}{rcccl} \Omega_1 & = & \{0,1\}^k \times \mathcal{E} \times \{0,1\}^{k\times n} & = & \{(x,e,G) \in \Omega_0 \mid e \in \mathcal{E}\} \\ \Omega_2 & = & \{0,1\}^k \times \mathcal{E} \times \mathbf{G}_{m,t} & = & \{(x,e,G) \in \Omega_1 \mid G \in \mathbf{G}_{m,t}\} \\ \Omega_3 & = & \{0,1\}^k \times \mathcal{E} \times \mathbf{G}'_{m,t} & = & \{(x,e,G) \in \Omega_2 \mid G \in \mathbf{S}_{k\times n}\} \end{array}$$

The success probability of an adversary $\mathcal{A}$ for the hybrid McEliece scheme is equal to

$$\mathrm{Succ}(\mathcal{A} \mid \Omega_3) = P_{\Omega_0}\left(\mathcal{A}(xG+e,G) = e \mid G \in \mathbf{G}'_{m,t}, e \in \mathrm{Im}(\varphi)\right).$$

Lemma 2.4 *Any (T,ε)-adversary over Ω_1 is a $\left(T, \varepsilon|\mathcal{E}|/\binom{n}{t}\right)$-adversary over Ω_0.*

Proof.
(of Lemma 2.4) Let $\mathcal{A}$ denote the (T,ε)-adversary over Ω_1 of the statement. By definition, it is such that

$$\mathrm{Succ}(\mathcal{A} \mid e \in \mathcal{E}) = \mathrm{Succ}(\mathcal{A} \mid \Omega_1) \geq \varepsilon.$$

We have

$$\begin{array}{rcl} \mathrm{Succ}(\mathcal{A}) & = & P_{\Omega_0}(\mathcal{A}(xG+e,G) = e) \geq P_{\Omega_0}(\mathcal{A}(xG+e,G) = e, e \in \mathcal{E}) \\ & \geq & P_{\Omega_0}(\mathcal{A}(xG+e,G) = e \mid e \in \mathcal{E}) P_{\Omega_0}(e \in \mathcal{E}) \\ & \geq & \mathrm{Succ}(\mathcal{A} \mid e \in \mathcal{E}) P_{\Omega_0}(e \in \mathcal{E}) \geq \varepsilon \dfrac{|\mathcal{E}|}{\binom{n}{t}} \end{array}$$

which proves the lemma. ◇

Lemma 2.5 *If there exists a (T,ε)-adversary over Ω_2 then*

- *either there exists $(T,\varepsilon/2)$-adversary $\mathcal{A}$ over Ω_1,*
- *or there exists $(T + O(n^2), \varepsilon/2)$-distinguisher for $\mathbf{G}_{m,t}$.*

Proof.
(of Lemma 2.5) Let $\mathcal{A}$ denote the (T,ε)-adversary over Ω_2 of the statement. We consider the distinguisher $\mathcal{D}$ defined for all $G \in \{0,1\}^{k\times n}$ by $\mathcal{D}(G) = (\mathcal{A}(xG+e,G) = e)$ where (x,e) is randomly and uniformly chosen in $\{0,1\}^k \times \mathcal{E}$. We have

$$\left\{ \begin{array}{rcl} P_{\Omega_0}(\mathcal{D}(G)) & = & \mathrm{Succ}(\mathcal{A} \mid e \in \mathcal{E}) \\ P_{\Omega_0}(\mathcal{D}(G) \mid \Omega_2) & = & \mathrm{Succ}(\mathcal{A} \mid e \in \mathcal{E}, G \in \mathbf{G}_{m,t}) \end{array} \right.$$

We have the advantage of the distinguisher as,

$$\mathrm{Adv}(\mathcal{D}, \mathbf{G}_{m,t}) = \mid P_{\Omega_0}(\mathcal{D}(G)) - P_{\Omega_0}(\mathcal{D}(G)) \mid \Omega_2 \mid$$

From which we easily derive

$$\mathrm{Succ}(\mathcal{A} \mid \Omega_2) \leq \mathrm{Succ}(\mathcal{A} \mid \Omega_1) + \mathrm{Adv}(\mathcal{D}, \mathbf{G}_{m,t}). \tag{2.1}$$

To run $\mathcal{D}$, one has to compute the ciphertext $xG + e$ which has a cost upper bounded by $O(n^2)$ and to make one call to $\mathcal{A}$. So we have $|\mathcal{D}| \leq T + O(n^2)$. By definition of $\mathcal{A}$, we have $\mathrm{Succ}(\mathcal{A} \mid \Omega_2) \geq \varepsilon$. Thus at least one of the two right-hand side terms of the inequality (2.1) is greater than $\varepsilon/2$. This implies that either $\mathcal{A}$ verifies

$$\mathrm{Succ}(\mathcal{A} \mid \Omega_1) \geq \frac{\varepsilon}{2}$$

or $\mathcal{D}$ verifies

$$\mathrm{Adv}(\mathcal{D}, \mathbf{G}_{m,t}) \geq \frac{\varepsilon}{2},$$

which proves the lemma. ◇

Lemma 2.6 *If there exists a (T, ε)-adversary over Ω_3 then*

- *either there exists $(T + O(n^3), \lambda\varepsilon/2)$-adversary over Ω_2,*
- *or there a exists $(O(n^3), \lambda/2)$-distinguisher for $\mathbf{G}_{m,t}$,*

where $\lambda \geq 0.288$ is the probability for a binary $k \times k$ matrix to be non-singular.

Proof.
(of Lemma 2.6) We denote $\mathrm{Syst}(G)$ a procedure which returns on any input $G = (U \mid V) \in \{0,1\}^{k \times n}$ such that U is non-singular the matrix $(Id \mid U^{-1}V) \in \mathbf{S}_{k \times n}$. On other inputs, Syst() leave G unchanged.

Let $\mathcal{A}$ denote the (T, ε)-adversary over Ω_3 of the statement.

We define the adversary $\mathcal{A}'$ as $\mathcal{A}'(y, G) = \mathcal{A}(y, \mathrm{Syst}(G))$. We define the distinguisher $\mathcal{D}$ which returns true on input G if and only if $\mathrm{Syst}(G) \in \mathbf{S}_{k \times n}$. The running time of Syst() is upper bounded by $O(n^3)$, thus $|\mathcal{A}'| \leq T + O(n^3)$ and $|\mathcal{D}| = O(n^3)$.

If $\mathcal{A}'$ succeeds with $(x, e, G) \in \Omega_2$ and $\mathrm{Syst(G)} \in \mathbf{S}_{k \times n}$, then $\mathcal{A}$ succeeds with $(x', e, \mathrm{Syst}(G)) \in \Omega_3$ for some x'. We have

$$\mathrm{Succ}(\mathcal{A}' \mid \Omega_2, \mathrm{Syst}(G) \in \mathbf{S}_{k \times n}) \geq \mathrm{Succ}(\mathcal{A} \mid \Omega_3) \geq \varepsilon$$

and (note that the events "$e \in \mathcal{E}$" and "$\mathrm{Syst}(G) \in \mathbf{G}'_{m,t}$" are independent)

$$\begin{aligned}
\mathrm{Succ}(\mathcal{A} \mid \Omega_3) &\leq \mathrm{Succ}(\mathcal{A}' \mid \Omega_2, \mathrm{Syst}(G) \in \mathbf{S}_{k \times n}) \\
&\leq \mathrm{Succ}(\mathcal{A}' \mid e \in \mathcal{E}, \mathrm{Syst}(G) \in \mathbf{G}'_{m,t}) \\
&\leq \frac{P_{\Omega_0}(\mathcal{A}'(xG + e, G) = e, e \in \mathcal{E}, \mathrm{Syst}(G) \in \mathbf{G}'_{m,t})}{P_{\Omega_0}(e \in \mathcal{E}, \mathrm{Syst}(G) \in \mathbf{G}'_{m,t})} \\
&\leq \frac{P_{\Omega_0}(\mathcal{A}'(xG + e, G) = e, e \in \mathcal{E}, G \in \mathbf{G}_{m,t})}{P_{\Omega_0}(e \in \mathcal{E}) P_{\Omega_0}(\mathrm{Syst}(G) \in \mathbf{G}'_{m,t})} \\
&\leq \mathrm{Succ}(\mathcal{A}' \mid e \in \mathcal{E}, G \in \mathbf{G}_{m,t}) \frac{P_{\Omega_0}(G \in \mathbf{G}_{m,t})}{P_{\Omega_0}(\mathrm{Syst}(G) \in \mathbf{G}'_{m,t})} \\
&\leq \frac{\mathrm{Succ}(\mathcal{A}' \mid \Omega_2)}{P_{\Omega_0}(\mathrm{Syst}(G) \in \mathbf{S}_{k \times n} \mid G \in \mathbf{G}_{m,t})}.
\end{aligned}$$

We consider now the distinguisher $\mathcal{D}$. By definition, we have

$$P_{\Omega_0}(\mathrm{Syst}(G) \in \mathbf{S}_{k\times n} \mid G \in \mathbf{G}_{m,t}) \;\; \geq \;\; P_{\Omega_0}(\mathrm{Syst}(G) \in \mathbf{S}_{k\times n}) - \mathrm{Adv}(\mathcal{D}, \mathbf{G}_{m,t}).$$

We also have $\lambda = P_{\Omega_0}(\mathrm{Syst}(G) \in \mathbf{S}_{k\times n})$ the proportion of non-singular binary $k \times k$ matrices. Putting everything together, we get

$$\varepsilon \leq \mathrm{Succ}(\mathcal{A} \mid \Omega_3) \leq \frac{\mathrm{Succ}(\mathcal{A}' \mid \Omega_2)}{\lambda - \mathrm{Adv}(\mathcal{D}, \mathbf{G}_{m,t})}$$

and

$$\mathrm{Succ}(\mathcal{A}' \mid \Omega_2) \geq \lambda\varepsilon - \varepsilon\mathrm{Adv}(\mathcal{D}, \mathbf{G}_{m,t}).$$

We easily conclude that, if $\mathcal{D}$ has an advantage smaller than $\lambda/2$ for $\mathbf{G}_{m,t}$ then $\mathcal{A}'$ has a success probability over Ω_2 greater than $\lambda\varepsilon/2$. $\diamond$

Constant weight encoding

For producing the injective mapping $\varphi : \{0,1\}^\ell \to W_{n,t}$ we need for the hybrid scheme is not an easy task. Existing solutions [19, 50, 52] are all based on a (source) encoder $W_{n,t} \to \{0,1\}^*$ whose decoder is used for processing binary data. Unfortunately they all have either a high computation cost, or a variable length encoder.

Here, we use another encoder which uses a new recursive dichotomic model for the constant weight words. Let $x = (x^L \parallel x^R) \in W_{n,t}$, with $n = 2^m$, where x^L and x^R have length $n/2 = 2^{m-1}$ and $i = w_H(x^L)$, we define

$$F_{m,t}(x) = \begin{cases} \mathbf{nil} & \text{if } i \in \{0, 2^m\} \\ i, F_{m-1,i}(x^L), F_{m-1,t-i}(x^R) & \text{else} \end{cases}$$

where $a, \mathbf{nil} = \mathbf{nil}, a = a$. Any element of $W_{n,t}$ is uniquely transformed into a finite sequence of integers. If $x \in W_{n,t}$ is chosen randomly and uniformly then the distribution of the head element i of $F_{m,t}(x)$ is

$$\mathrm{Prob}(i) = \frac{\binom{n/2}{i}\binom{n/2}{t-i}}{\binom{n}{t}}, i \in \{0, 1, 2, \ldots, t\}.$$

The sequences of integers produced by $F_{m,t}$ can be modeled by a stochastic process with the above probabilities. We use an adaptative arithmetic source encoder to encode them. This allows us to produce a nearly optimal encoder from which we build a fast and efficient mapping $\varphi : \{0,1\}^\ell \to W_{n,t}$. For values of (m,t) of practical cryptographic interest we always have $\ell \geq \lfloor \log_2 \binom{n}{t} \rfloor - 1$.

2.3 Implementation of HyMES

The full project, wrote in C language, which is freely available at our website, [1] under LGPL, can broadly be categorized into four set of functions, *Library functions, Key generation functions, Encryption functions* and *Decryption functions*. Let us start describing each in detail.

2.3.1 Library functions

We needed to define Galois field and functions to operate on the field elements, the polynomial representation of the extension field and matrix functions.

Galois field

Recall the definition of field (see §1.1), here as we are interested in binary field *i.e.* a field with characteristic 2. The usual *XOR* operation serve the purpose of addition. However for the extension field $\mathbb{F}_{2^m}$, we needed the operations defined. The addition is still simple *XOR* and we define the multiplication with the help of *logarithmic* and *exponential* tables.

Let $x = \alpha^i$, $x \in \mathbb{F}_{2^n}$, then i is the discrete logarithm of x, with respect to α. Once the construction of the field is determined, the primitive element α is fixed. Then, an exponentiation function *expf* can be defined to represent elements with the power value. For example, $x = \alpha^i$ is represented as,

$$x = exp(i);\ x \neq 0$$

Let y be another element in $\mathbb{F}_{2^n}$, $y = \alpha^j$. Then the multiplication $mul(x, y)$ of x and y can be represented as,

$$mul(x, y) = \alpha^i \alpha^j = \alpha^{i+j}$$

From the property of primitive element, $\alpha^{2^m-1} = 1$. Hence, we have

$$mul(x, y) = \alpha^{(i+j) \bmod (2^m-1)}$$

where, mod means modular operation. With the definition of the exponentiation and the discrete logarithm functions, we can derive the multiplication function as,

$$mul(x, y) = exp(log(x) + log(y)) \bmod (2^m - 1)$$

The pre-computation approach is to compute all possible values of the exp and log functions based on the primitive element and store them in a table. The exponentiation and discrete logarithm operations are just table look-ups thereafter. The size of the table equals to $2^n - 1$, which is also the number of non-zero elements in the field. For example, $\mathbb{F}(256)$ has table size of 255.

[1] `http://www-roc.inria.fr/secret/CBCrypto/index.php?pg=hymes`

We represents the field elements by integers. Let $x \in [0, 2^m[$ is an integer and $\gamma \in \mathbb{F}_{2^m}$ is the corresponding field element. Thus, for a primitive α, we have,

$$x = \Sigma_{i=0}^{m-1} x_i 2^i \leftrightarrow \gamma = \Sigma_{i=0}^{m-1} x_i \alpha^i$$

where $x_i \in \mathbb{F}_2$ and i is integer.

Let us denote $\gamma = [[x]]$ as the notation for a field element γ represented by an integer x in machine. The basic arithmetic operations are thus defined as bellow,
Let, for integers x, y and z the corresponding field elements are $[[x]]$, $[[y]]$ and $[[z]]$ and x_i denoted the i^{th} bit position of integer x.
Addition - $[[x]] + [[y]] = [[z]]$ is the X-OR operation on 2 integers x and y and the result is stored in z. **Multiplication -** Similarly, for $[[x]][[y]] = [[z]]$, we look-up the corresponding entries in the *log* table, add them (X-OR) taking modulo $(2^n - 1)$ and look-up the corresponding value in the *exp* table.

Polynomial functions

The polynomial representation of fields (see §1.1.2)should allow the arithmetic operations over the extension field with ease. We take the coefficients as field elements and define multiplication, addition, gcd functions. This polynomial representation helps us mainly for finding gcd and *root finding of error locator polynomial*. For gcd we use Extended Euclidean algorithm (see algorithm 2) and root finding is detailed in part III.

Addition is field addition described above performed for each coefficient. Similar for *multiplication*. *Quotient* and *remainder* are found applying field arithmetic modulo *irreducible* polynomial.

Let g be irreducible polynomial of degree t. The more dedicated functions are,
square - is implemented with all the computed values $z^{2i} \bmod g(z) \forall i \leq deg(g)$ for a polynomial z. We adjust the coefficients.

Algorithm 6 Squaring a polynomial

Require: A polynomial z, irreducible polynomial g.
  for $i = 0$ to $t/2 - 1$ **do**
    compute $square[i] = z^{2i}$
  end for
  for $i = t/2$ to $t - 1$ **do**
    compute $square[i] = z^2 \cdot square[i-1] \bmod g$
  end for
  adjust coefficients for all the terms.
  return *square*, the array containing all the square values.

square root - Finding square root is due to the equation

$$z^{2^{mt-1}} = \sqrt{z} \bmod g(z)$$

Here we pre-compute all values $z^{i/2} \bmod g(z) \forall$ odd $i \leq deg(g)$.
irreducibility testing - is defined by the property, there exists no $i < t$ such that $g(z)|z^{2^{mi}} - z$, or $z^{2^{mi}} \bmod g(z) \neq z$. $g(z)|z^{2^{mi}} - z \Leftrightarrow g(z)$ has a factor of degree i with coefficients in $\mathbb{F}_{2^m}$.

We solve $z^{2^{mi}} \bmod g(z) \forall i \leq t/2$. If we find no result of the above equation equal to z then $g(z)$ is irreducible.

Matrix functions

The matrix structure is defined as a continuous array with maximum column number to mark the beginning of the next row. We use a compact representation that actually maps *binary positions* of a matrix by *bit positions*. This feature enables efficient row operations. We have functions to do *Gaussian elimination* (see algorithm 1), matrix - matrix and matrix - vector multiplications.

2.3.2 Key generation

Recall the description of Goppa code in §1.2.4. Let $\Gamma(L, g) \in \mathcal{G}_{m,n,t}$. We have $L = (\alpha_1, \ldots, \alpha_n)$ with all the α_j distinct in $\mathbb{F}_{2^m}$ and $g(z) \in \mathbb{F}_{2^m}[z]$ monic irreducible of degree t. Let $H^{(0)}$ be a matrix of size $t \times n$ in $\mathbb{F}_{2^m}$ whose entry in row i, $0 \leq i < t$ and column j, $1 \leq j \leq n$, is $H^{(0)}_{i,j} = \alpha_j^i / g(\alpha_j)$.

Let $(\beta_1, \ldots, \beta_m)$ denote a basis of $\mathbb{F}_{2^m}$ over $\mathbb{F}_2$. Any $\gamma \in \mathbb{F}_{2^m}$ can be uniquely written as $\gamma = b_1(\gamma)\beta_1 + \ldots + b_m(\gamma)\beta_m$ with $b_\ell(\gamma) \in \mathbb{F}_2$ for $\ell = 1, \ldots, m$. The $tm \times n$ binary matrix H of general term

$$H_{im+\ell,j} = b_\ell(H^{(0)}_{i,j}), 0 \leq i < t, 1 \leq \ell \leq m, 1 \leq j \leq n$$

is a parity check matrix of binary Goppa code$\Gamma(L, g)$. We apply a Gaussian elimination (algorithm 1) on H to obtain a systematic form $H_S = (R \mid I)$. If the last tm columns of H are singular, we permute columns and change L in such a way that H_S is a parity check matrix of $\Gamma(L, g)$. A generator matrix of $\Gamma(L, g)$ is then given by the matrix $(I \mid R^T)$ (where I is the identity matrix and R^T the transpose of R). The public key will be the $k \times tm$ matrix R^T (where $k = n - tm$).

A t-error correcting binary Goppa code of length n over $\mathbb{F}_{2^m}$ is defined by

- a *support* $L = (\alpha_1, \ldots, \alpha_n)$ with all the α_j in $\mathbb{F}_{2^m}$
- a generator $g(z) \in \mathbb{F}_{2^m}[z]$ monic irreducible of degree t.

It has dimension $k \geq n - tm$ (equality holds with an overwhelming probability for the all parameters considered here). We denote $r = n - k = tm$ the co-dimension.

We generate L containing all the field elements randomly. We generate monic irreducible polynomial $g(z)$ of degree t by randomly picking $g(z)$ and checking it's irreducibility (see 2.3.1).

For a given code $\Gamma(L, g)$ the public key is generated as follows

- For all $0 \leq i < t$, all $1 \leq j \leq n$ and all $1 \leq \ell \leq m$ compute

$$H_{im+\ell,j} = b_\ell \left(\frac{\alpha_j^i}{g(\alpha_j)} \right) \tag{2.2}$$

  where $b_\ell(\gamma) \in \mathbb{F}_2$ is the ℓ-th coordinate of $\gamma \in \mathbb{F}_{2^m}$ in some basis of $\mathbb{F}_{2^m}$ over $\mathbb{F}_2$.

- compute a systematic form $H_S = (R^T \mid I_r)$ of the $r \times n$ matrix H. If the last r columns of H are singular, we permute columns and change L in such a way that H_S has the prescribed form and is a parity check matrix of $\Gamma(L, g)$. I_r is the $r \times r$ identity matrix, R is a $k \times r$ binary matrix and R^T its transpose.

- publish R as the public key ($G = (I_k \mid R)$ is a generator matrix of $\Gamma(L, g)$).

$R_a(z)$ the *rational function* of the code $\Gamma(L, g)$, can also be defined in H (see [39], Chapter 12, page 339).

We apply Gaussian elimination 1 to the matrix 2.2 and we post the redundancy part R as public key.

2.3.3 Encryption

Let $f_K()$ be a stream cipher parameterized by a key K. For any binary string x, we denote $f_K(x)$ its encryption. We assume it has the same length as x and that $f_K(f_K(x)) = x$. Please refer §2.1.1 to compare the original McEliece encryption process with ours.

Let $W_{n,t}$ denote the set of binary words of length n and Hamming weight t. Let $x \in \{0,1\}^k$ be the cleartext

- We pick message of length $k + \ell$ and split it into two parts $x \to x_k + x_\ell$.

- we convert the x_ℓ into the error e at random in $W_{n,t}$. We use constant weight encoding (see §2.2.2) for producing the error vector.

- compute $x' = f_e(x_k)$, the stream cipher acts as the randomizer.

- compute $y' = (x' \mid x'R^T)$, we concatenate x' with the matrix-vector product of x' and R^T, the transpose of the generator matrix in reduced row echelon form.

- the ciphertext is $y = y' + e$

Vector matrix multiplication (binary), error generation.

2.3.4 Decryption

The decryption uses Patterson algorithm 1.2.6. The main part of the decryption process is the decoding of binary Goppa codes (see §1.2.4).

As we described the three steps of decoding are *syndrome computation*, *solving the key equation* and *finding the roots of the error-locator polynomial* (see decoding algorithm 3).

To compute the syndrome we precompute for all α_j, $1 \leq j \leq n$

$$f_{\alpha_j}(z) = \frac{1}{z - \alpha_j} \bmod g(z)$$

as this set is not very big for cryptographic parameters, this helps us to speed up the process considerably.

Let the received vector $y = (y_1, y_2, \cdots, y_n)$.

Then we compute the syndrome as,

$$R_e(z) = \sum_{j=1}^{n} \frac{y_j}{z - \alpha_j} \bmod g(z) = \sum_{j=1}^{n} y_j f_{\alpha_j}(z)$$

To solve the key equation 1.4 we pre compute all the square root values for $i = 0, 1, \ldots, t-1$,

$$T_i(z) = \sqrt{z^i} \bmod g(z)$$

We use the methods described in §2.3.1 to compute the square roots.

let $h(z) = h_0 + h_1 z + \ldots + h_{t-1} z^{t-1}$, we solve,

$$S(z) = \sqrt{h(z)} \bmod g(z) = \sum_{i=0}^{t-1} h_i^{2^{m-1}} T_i(z)$$

where $h(z)$ is computed with Euclidean algorithm 1.1.6 as

$$h(z) = z + \frac{1}{R_e(z)} \bmod g(z)$$

We solve the equation

$$u(z) = v(z)S(z) \bmod g(z)$$

to find to find $u(z)$ and $v(z)$ of degree at most $t/2$ and $(t-1)/2$ respectively.

We can now find the locator polynomial as,

$$\sigma(z) = u(z)^2 + zv(z)^2$$

The final step of decoding *i.e.* finding roots of the error locator polynomial, thus finding the error vector e, is detailed in the next chapter (see §3).

We retrieve the plain text x_k as, $y' = y + e = (x' \mid x'R^T)$ with x' of length k, and compute the cleartext $x_k = f_e(x')$ and $x_\ell = \phi_{-1}(e)$. Finally we get $x = x_k + x_\ell$.

2.4 Simulation results

We implemented the hybrid version of McEliece encryption scheme in C programming language. In Figures 2.1 and 2.2 we plot the running time per plaintext byte versus the logarithm in base 2 of the work factor of the best known attack [14].

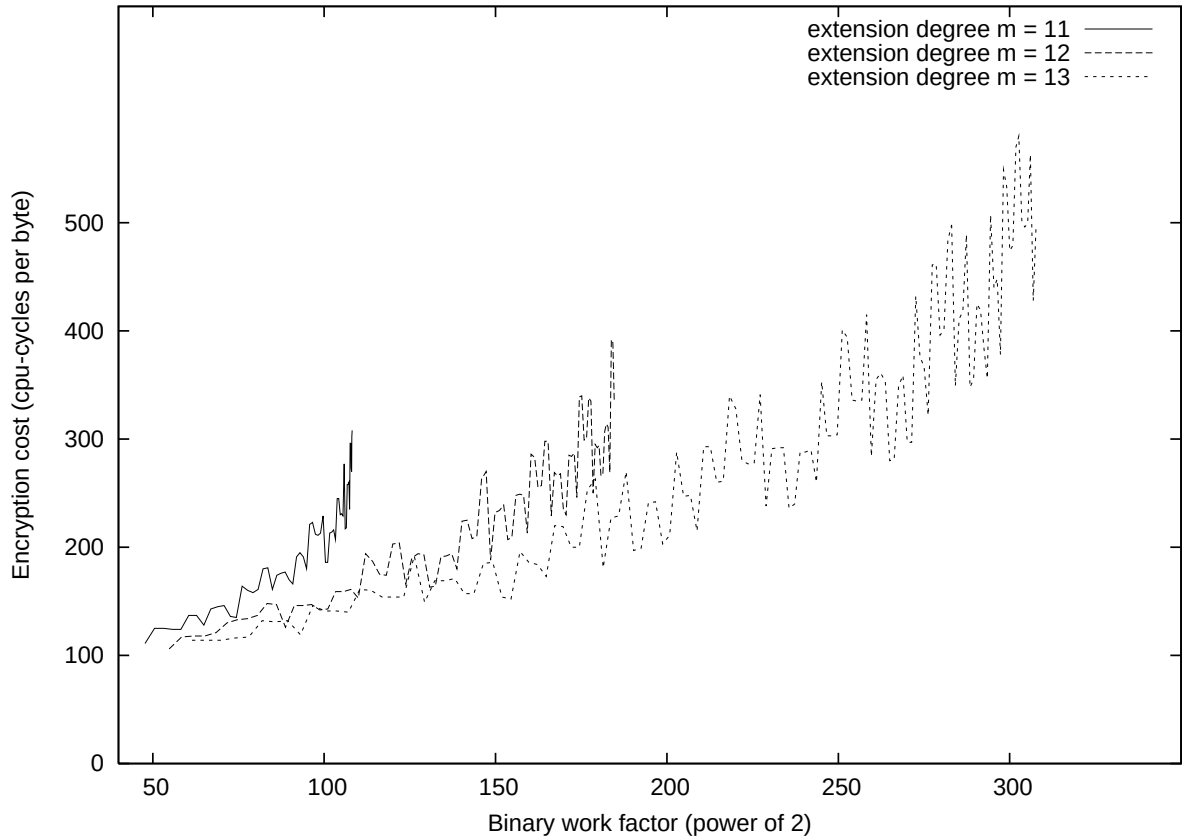

Figure 2.1: Encryption cost vs binary work factor for different extension degrees

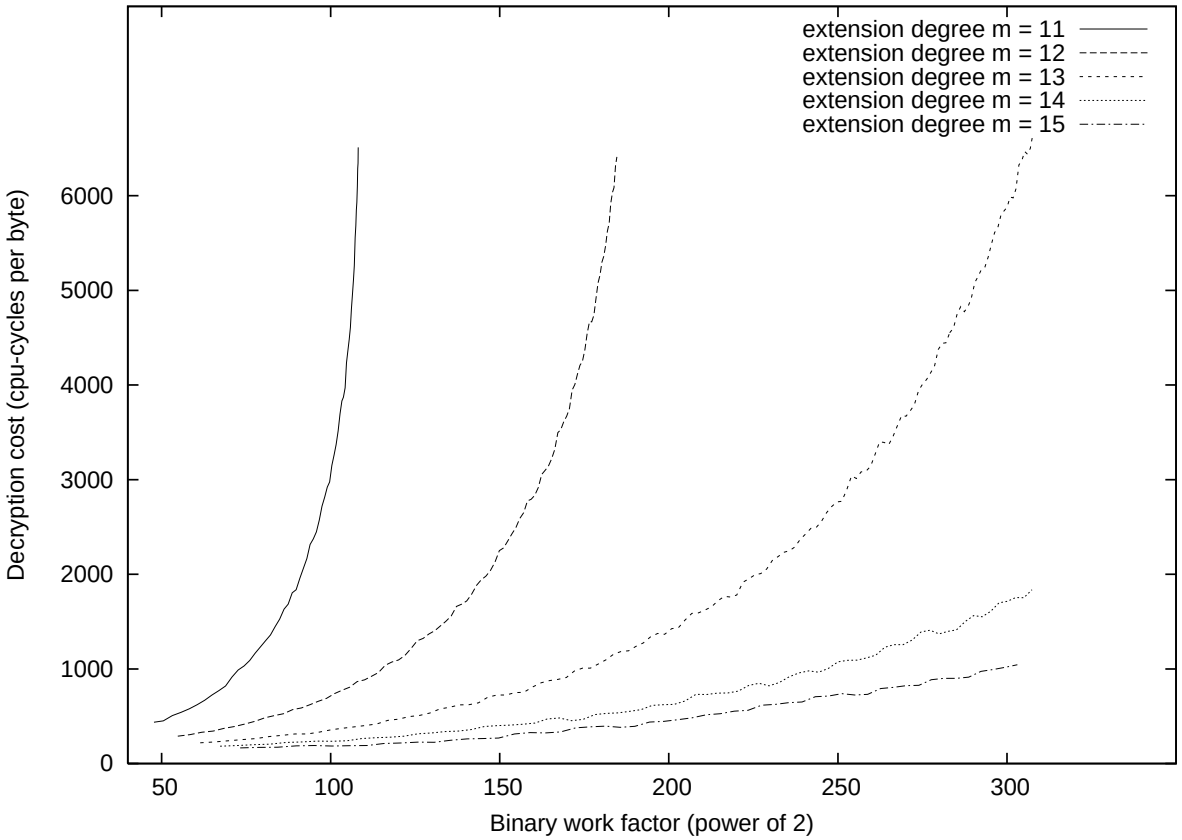

Figure 2.2: Decryption cost vs binary work factor for different extension degrees

Various values of t were tried for an extension degree $11 \leq m \leq 15$. As expected, for a fixed m, the performance gets better for smaller values of t. However, for a fixed security level, the best performance is not obtained for the smallest block size (i.e. extension degree). On the contrary the system works better for higher extension degrees. However, for $m \geq 13$ encryption speed for fixed security becomes steady. See Figure 2.1 and Figure 2.2.

	cycles/byte			
(m, t)	encrypt	decrypt	key size	security
$(10, 50)$	243	7938	32 kB	60
$(11, 32)$	178	1848	73 kB	88
$(11, 40)$	223	2577	86 kB	96
$(12, 21)$	126	573	118 kB	88
$(12, 41)$	164	1412	212 kB	130
$(13, 18)$	119	312	227 kB	93
$(13, 29)$	149	535	360 kB	129
$(14, 15)$	132	229	415 kB	91
$(15, 13)$	132	186	775 kB	90
$(16, 12)$	132	166	1532 kB	91

Table 2.1: McEliece: selected parameters at a glance

	cycles/byte	
	encrypt	decrypt
RSA 1024 (1)	800	23100
RSA 2048 (1)	834	55922
NTRU (2)	4753	8445

(1) RSA encryption (with malleability defense) using OpenSSL.
(2) `ntru-enc 1 ees787ep1` NTRU encryption with $N = 787$ and $q = 587$. Software written by Mark Etzel (NTRU Cryptosystem).

Table 2.2: Performance of some other public key systems (EBATS source)

2.4.1 Comparison with other systems

Our simulations were performed on a machine featuring an Intel Core 2 processor with dual core. It ran on a 32 bits operating system and a single core. The C program was compiled with `icc` Intel compiler with the options `-g -static -O -ipo -xP`. Timings are given in Table 2.1. Compared with the state of the art implementation of other public key encryption schemes (see Table 2.2), McEliece encryption gains an order of magnitude for both encryption and decryption.

The source used for Table 2.2 is an EBATS preliminary report[2] of March 2007.

[2] `http://www.ecrypt.eu.org/ebats/D.VAM.9-1.1.pdf`

Part III

Efficient root finding of polynomials over finite fields

Chapter 3

Efficient root finding of polynomials over finite fields

Root finding of polynomials over finite fields is a classical algebraic algorithmic problem. It is considered as one of the most time-consuming sub-process of the decoding process of Reed-Solomon, BCH and Goppa codes. There are some well known approaches for finding roots of the so-called error-locator polynomial. The most widely known root finding algorithm is Chien search method [17], which is a simple substitution of all elements of the field into the polynomial, so it has very high time complexity for the case of large fields and polynomials of high degree. Berlekamp Trace Algorithm (BTA) [6] is another well known method. It is a recursive method based on the trace function properties.

McEliece cryptosystem is considered as one of the fastest public key schemes and is still esteemed secure for reasonable parameters. The classical [40] (described in §2.1) and hybrid [13] McEliece schemes use binary Goppa codes [25]. The decryption process employs an algebraic decoding algorithm which is often broken up in three parts namely: syndrome computation, finding the solution of the key equation, and the root finding of the error locator polynomial. This last step takes theoretically three fourth of the total decryption time.

In this chapter, we present a hybrid method involving BTA and a method proposed by Zinoviev [60]. Zinoviev proposed direct root finding procedures for polynomials with degree at most 10. Our idea is to compute directly the roots with Zinoviev procedures up to some degree and to use BTA for greater degrees. Moreover, we improve Zinoviev procedures for polynomials of degree 2 and 3 with time-memory tradeoffs. We analyze both the theoretical complexities and the experimental complexities of our proposal. We obtain a theoretical gain of 93% over Chien method and 46% over BTA. Experimental results confirm theory up to degree 4 at least. For instance with $m = 11$, $t = 32$ and $d_{max} = 4$, our method takes 60% of the total decryption time with respect to 72% for BTA and 87% for Chien.

3.1 Efficient Polynomial Root Finding Problem

Let $m > 0$ be the extension degree of the two-element field $\mathbb{F}_2$. We consider a univariate monic polynomial f, of degree $t > 0$, in the polynomial ring $\mathbb{F}_{2^m}[x]$. Without loss of generality, we assume that f has no multiple root and that f factorizes into linear factors over the binary field $\mathbb{F}_{2^m}$ (*e.g.* see [36]). We are looking for an efficient way, in terms of time-complexity, to find all zeroes of f.

3.2 Background

For the description of binary Goppa codes see §1.2.4

Algebraic decoding algorithm

Let e, y, z be n-length binary vectors. We have to find z the sent codeword knowing $y = z + e$ where y is the received word and e the error word.

Algebraic decoding is carried out in three steps :

1. Syndrome computation

$$R_y(z) = R_e(z) = \sum_{i=1}^{n} \frac{e_i}{z - \alpha_i} \text{ over } \mathbb{F}_{2^m}[z]/(g(z))$$

2. Solving the key equation to obtain the *error locator polynomial* σ_e (with the Berlekamp-Massey algorithm or the Extended Euclidean algorithm).

$$R_e(x).\sigma_e(x) = \sigma'_e(x) \text{ over } \mathbb{F}_{2^m}[x]/g(x)$$

   Notation: σ'_e denotes the formal derivative of σ_e.

3. Error locator polynomial root finding

$$\sigma_e(z) := \prod_{i=1}^{n} (z - \alpha_i)^{e_i};\ e_i \neq 0 \Leftrightarrow \sigma(\alpha_i) = 0$$

3.3 Why root finding is important?

Let $\mathbb{F}_{2^m}$ be the extension field of degree m of the two-element field $\mathbb{F}_2$. We consider a univariate monic polynomial f, of degree $t > 0$, in the polynomial ring $\mathbb{F}_{2^m}[z]$. Without loss of generality, we assume that f has no multiple root and that f factorizes into linear factors over the binary field $\mathbb{F}_{2^m}$ (*e.g.* see [35]). Our goal is to find an efficient way, in terms of time and space complexity, to find all the zeroes of f.

We are specifically interested in the said problem in the McEliece context.

The efficiency of the root finding algorithms is a problem that we study in code-based cryptography. McEliece-type cryptosystems are often based on binary Goppa codes (presented in Appendix 1.2.4). Their decryption algorithm employs an algebraic decoding process to recover the original message from the cyphertext. The most time-consuming stage, in the implementation of algebraic decoding of binary Goppa codes, with practical parameters, is the root finding of the error locating polynomial. This polynomial fulfills the above mentioned properties.

In this article, we consider an n-lenght binary Goppa code that corrects up to t errors (the algorithm used is given in Appendix 1.2.4). Let us recall, in practice, parameters are chosen such that: $n = 2^m$ and $mt \leq n$.

Decryption Complexity

Theoretical Complexity = number of arithmetic operations in $\mathbb{F}_{2^m}$ required to decrypt in the worst case.

- Syndrome computation $\mathcal{O}(nt)$
- Key equation solving $\mathcal{O}(t^2)$
- Error locator polynomial root finding
  - BTA $\mathcal{O}(mt^2)$
  - Chien search $\mathcal{O}(nt)$

Experimental Complexity = average running time for the decryption.

We give below the percentage of the total time spent in each stage of the decryption algorithm.

- Syndrome computation 11.3%
- Key equation solving 12.0%
- Error locator polynomial root finding
  - BTA 72.1%
  - Chien search 85.6% [1]
- Other tasks 4.6%

[1] The percentages given are associated with BTA, if we take Chien search, the numbers for syndrome computation and key equation solving should vary accordingly.

Asymptotically, syndrome computation is the leading cost. For recommended parameters (*i.e.* $m = 11$, $t = 32$), the most time-consuming step in the decryption (decoding) consists in finding roots of σ_e. Differences that can appear between theoretical and experimental complexities would be due to several reasons:

- tweaks of implementation (quality of implementation, used processors and compilers);
- cost of conditional tests and memory accesses (that are neglected in our theoretical study but that can be noteworthy in practice);
- not so good approximations (*e.g.* we approximate the cost of an inversion of a binary matrix of order m to m^2 field operations, we thus overlook a small multiplicative constant);
- weighting[2] of arithmetic operations in the field.

3.4 Our Proposal

3.4.1 Speed up McEliece Decryption

The drawback of BTA is the large number of recursive calls when the system parameters grow. We reduce it by mixing BTA and Zinoviev procedures that are ad-hoc methods for finding roots of polynomials of degree ≤ 10. This is a classical technique employed to decrease the number of recursive levels, *e.g.* the well-known Quicksort algorithm is optimized with analagous methods. We call this process BTZ (Berlekamp Trace - Zinoviev) in the scope of this document. BTZ depends on a parameter d_{max} that is the maximum degree up to which we use Zinoviev methods.

3.4.2 Implementation Tweaks

In our implementation, we use a polynomial basis to represent the field elements. We implemented Zinoviev procedures with time-memory tradeoffs for polynomials of degree 2 and 3, that enable to perform better than with the original procedures. We explain these new methods and give their complexities in the following.

Time-Memory Tradeoff for Degree 2.

We want to solve the equation: $z^2 + az + b = 0$ for a, $b \in \mathbb{F}_{2^m}$. If the solutions exist, we denote them z_1 and z_2. First, we make a change of variable. We set $z = ax$. We obtain the equation $x^2 + x + b/a^2 = 0$. It costs one division and one squaring in $\mathbb{F}_{2^m}$.

[2] In our study, we distinguish two cases: in the first one, addition and multiplication both cost one operation in $\mathbb{F}_{2^m}$, we denote it $K(+) = K(\times) = 1$ where K is the cost function of an arithmetic operation, in the second one, $K(+) = 1$ and $K(\times) = m$.

Let i be an element of $\mathbb{F}_{2^m}$ and f_i be the mapping:

$$\begin{aligned} f_i : \ \mathbb{F}_{2^m} &\to \mathbb{F}_{2^m} \\ x &\mapsto x^2 + x + i \end{aligned}$$

The equation $f_i(x) = 0$ has two solutions in $\mathbb{F}_{2^m}$ if and only if $\mathrm{Tr}(i) = 0$ (a proof is given in [9]), else this equation has no solution in $\mathbb{F}_{2^m}$.

Let T be a table containing elements of $\mathbb{F}_{2^m}$.

In other words, $T[i]$ contains one of the two elements of the kernel of f_i, if i is in the image of $x \mapsto x^2 + x$. Note that $j + 1$ is the second element of this kernel.

Now, we read from the table, the element $T[b/a^2]$. We invert the change of variable by computing: $z_1 = ab$. Then, we compute the second solution: $z_2 = ab + a$. This process costs one multiplication and one addition in $\mathbb{F}_{2^m}$. Thus, we have solved our problem within four operations in $\mathbb{F}_{2^m}$ with a memory of 2^m field elements. It is useful for small m.

Time-Memory Tradeoff for Degree 3.

In this case, the equation to solve is: $z^3 + az^2 + bz + c = 0$ where a, b, $c \in \mathbb{F}_{2^m}$. We obtain an affine multiple of degree 4 by multiplying the polynomial by $z + a$. It costs two multiplications, one squaring and two additions in $\mathbb{F}_{2^m}$. The substitution $z = \sqrt{(a^2 + b)}x$ ($m - 1$ repeated squarings[3]) and a normalization (two divisions) enable to obtain an equation of the form: $x^4 + x^2 + dx = e$, with d, $e \in \mathbb{F}_{2^m}$. Let us denote f the mapping $x \mapsto x^4 + x^2 + dx$. By construction a is a root of $f(x) = e$, we want to find the other three. The mapping f has a kernel of dimension two (except if $d = 0$, in this case, the dimension is one). We have only to store two elements which form a basis of the kernel of $x \mapsto x^4 + x^2 + dx$ in a table for all $d \in \mathbb{F}_{2^m}$. This requires a storage memory of 2×2^m field elements. Notice, this step does not depend on the coefficient e. Let us call the lookup table T and the two elements stored in the table for a given d: λ_1 and λ_2. Then, we have $T[d] = (\lambda_1, \lambda_2)$. As f is linear, the three other roots of f are: $a + \lambda_1$, $a + \lambda_2$ and $a + \lambda_1 + \lambda_2$. We obtain them with three additions. Lastly, we invert the substitution (three multiplications) and thus the problem is solved. Here, we used the fact that we know that a is a root of f to find the other ones. We cannot use anymore this extra information for polynomials of degree 4 onwards.

3.4.3 How do we Compute Theoretical Complexity?

We will not give here the complexity recurrence formula, that we use to compute the number of operations required to process BTZ for the sake of clarity. Instead, we prefer to explain how we obtain it.

[3]Let us mention that one addition and one multiplication with a constant, are enough using the method proposed in [29]. We do not take it into account in Table 3.1.

	Addition	Mult.	Division	Squaring	Matrix Inv.	Total Cost
Z_2 precomput.	m^2	m^2	0	$m(m-1)$	1	$4m^2-m$
Z_2 w/o tradeoff	m	1	1	1	0	$m+3$
Z_2 w/ tradeoff	1	1	1	1	0	4
Z_3 w/o tradeoff	$2(m+1)$	$3m$	0	m	1	m^2+6m+2
Z_3 w/ tradeoff	5	5	2	m	0	$m+12$
Z_4	$2m+9$	$3(m+1)$	8	m	1	$m^2+6m+20$
Z_5	$4m+101$	$7m+104$	1	0	1	$m^2+11m+206$

Table 3.1: Number of operations over $\mathbb{F}_{2^m}$ in Zinoviev procedures

About BTA.

The polynomials with which we deal in BTA are monic, without multiple root and can be factorized into linear factors over $\mathbb{F}_{2^m}$. These polynomials form a set $\mathcal{P}$. Such polynomials of degree d are entirely determined by their d roots. Hence, there are $\binom{2^m}{d}$ such polynomials.

Moreover, we know that for all $\beta \in B$, a $\mathbb{F}_2$-basis of $\mathbb{F}_{2^m}$, half of the elements of $\mathbb{F}_{2^m}$ have $\mathrm{Tr}(\beta \cdot z) = 0$ and that for the other half, $\mathrm{Tr}(\beta \cdot z) = 1$.

For each step of Algorithm **??** (see Appendix **??**), we compute the gcd of a polynomial with $\mathrm{Tr}(\beta \cdot z)$. The polynomial that we obtain contains the roots, such that $\mathrm{Tr}(\beta \cdot z) = 0$. We then make a Euclidean division that gives us another polynomial of degree $d-i$, which contains the roots, such that $\mathrm{Tr}(\beta \cdot z) = 1$.

About Zinoviev Procedures.

In Table 3.1, we assume that all the arithmetic operations on the field have unitary cost and that one binary matrix inversion of order m costs m^2 additions in $\mathbb{F}_{2^m}$. Let Z_d denote the Zinoviev procedure for degree d where d varies from 2 to 5. For $d = 2$ and $d = 3$, we consider two possibilities: with or without the time-memory tradeoff. When we use the tradeoff, the space complexity is exponential in m, that is, it is in the order of the size of the field *i.e.* 2^m, up to a multiplicative constant factor. For greater degrees ($6 \leq d \leq 10$), the time complexity is $\mathcal{O}(m^2 + dm + d2^d)$. It is exponential in d since in the worst case, the affine multiple has degree equal to 2^{d-1}. Therefore, in the last step of Zinoviev procedures, we would have to evaluate the polynomial in 2^{d-1} points for finding its roots. This is a reason for which we do not use Zinoviev methods for higher degrees than 10. One observes, in Table 3.1, that the most expensive part of the Zinoviev procedures is the binary matrix inversion which enables to find the roots of the affine polynomial (see Section 1.2.6 for more details).

3.5 Simulation Results

In Table 3.2, we provide experimental data for finding the roots of a polynomial of degree $t = 32$ over $\mathbb{F}_{2^m} = \mathbb{F}_{2048}$. BTZ_d means that we use BTZ with $d_{max} = d$, for all suitable d.

$n = 2048$, $t = 32$, $m = 11$		Chien	BTA	BTZ_2	BTZ_3	BTZ_4
# CPU cycles per byte for	root finding	3200	1300	900	800	800
	decrypting	3700	1800	1400	1300	1300
percentage [4] of time spent for	syndrome computation	5	10	13	14	14
	solving key equation	7	14	18	19	19
	root finding	87	72	65	61	60

Table 3.2: Experimental data for finding the roots of a polynomial of degree $t = 32$ over $\mathbb{F}_{2^m} = \mathbb{F}_{2048}$.

In Table 3.3, we present the theoretical number of field operations for correcting $t = 32$ errors of a $n = 2048$-length word over $\mathbb{F}_{2^{11}}$ using BTZ with $d_{max} = 5$ and time-memory tradeoff (see Section 3.4.2). For information, BTZ without the tradeoff gives very close *theoretical* results.

Let us depict the gain of BTZ over BTA and Chien procedure in terms of percentage of number of operations according to the parameter d_{max} and the polynomial degree t. The results (given in Figure 3.2 and Figure 3.3) take into account the time-memory tradeoff for degree 2 and 3. For the sake of readibility and relevance in cryptographic applications, we restrict to the case $m = 11$, $K(\times) = m$, $t \leq 100$ and $d_{max} \leq 6$. Additionally, we present similar results for greater degrees in Appendix 3.5. Nevertheless, we have also computed these results for $m = 8, 11, 12, 13, 14, 15, 16, 20, 30, 40$, $K(\times) = 1$, $t \leq 300$, $d_{max} \leq 10$ and for both *with* and *without* the tradeoff. These data give the information that the higher is t, the higher is the optimal d_{max}. One can deduce from the two following graphs that for $m = 11$ and $t = 32$, the recommended theoretical value for d_{max} is 5. Indeed, we have a substantial gain of 46% over BTA and 93% over Chien method for this value of d_{max}.

$\text{BTZ}_{d_{max}}$ **vs.** BTA **for** $m = 11$**,** $30 \leq t \leq 300$

[4] Remaining percentages correspond to other minor tasks.

$n = 2048$, $t = 32$, $m = 11$	Chien	BTA	BTZ_2	BTZ_3	BTZ_4	BTZ_5	BTZ_6
$K(+) = K(\times) = 1$	129k	16k	13k	11k	10k	10k	10k
$K(+) = 1$, $K(\times) = m$	764k	91k	65k	54k	50k	47k	48k

Table 3.3: Theoretical number of field operations for correcting $t = 32$ errors of a $n = 2048$-length word over $\mathbb{F}_{2^{11}}$ using BTZ with $d_{max} = 5$ and time-memory tradeoff.

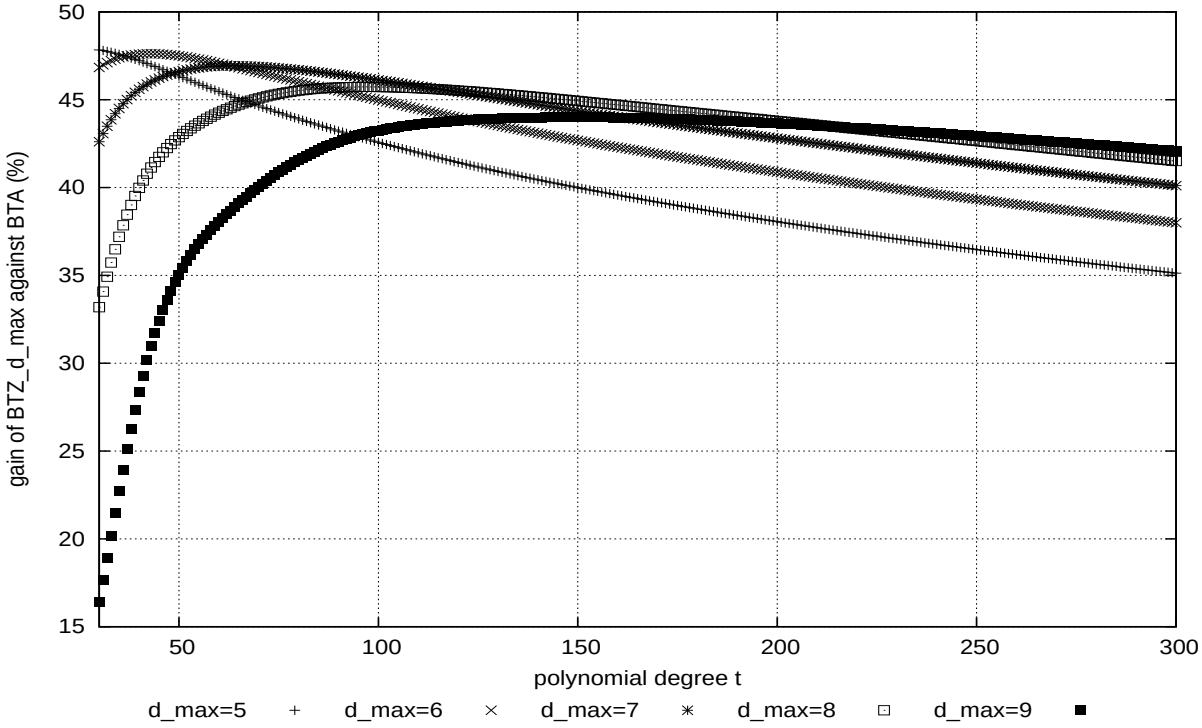

Figure 3.1: $\mathrm{BTZ}_{d_{max}}$ vs. BTA; $m = 11$; $K(+) = 1$; $K(\times) = m$; with time-memory tradeoff

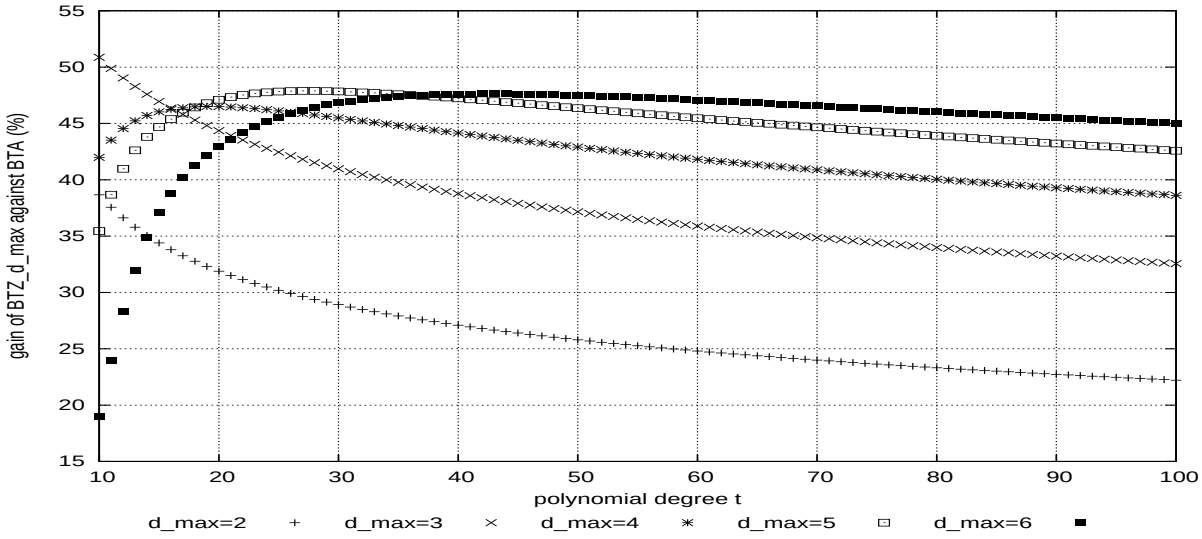

Figure 3.2: $\mathrm{BTZ}_{d_{max}}$ vs. BTA; $m = 11$; $K(+) = 1$; $K(\times) = m$; with time-memory tradeoff

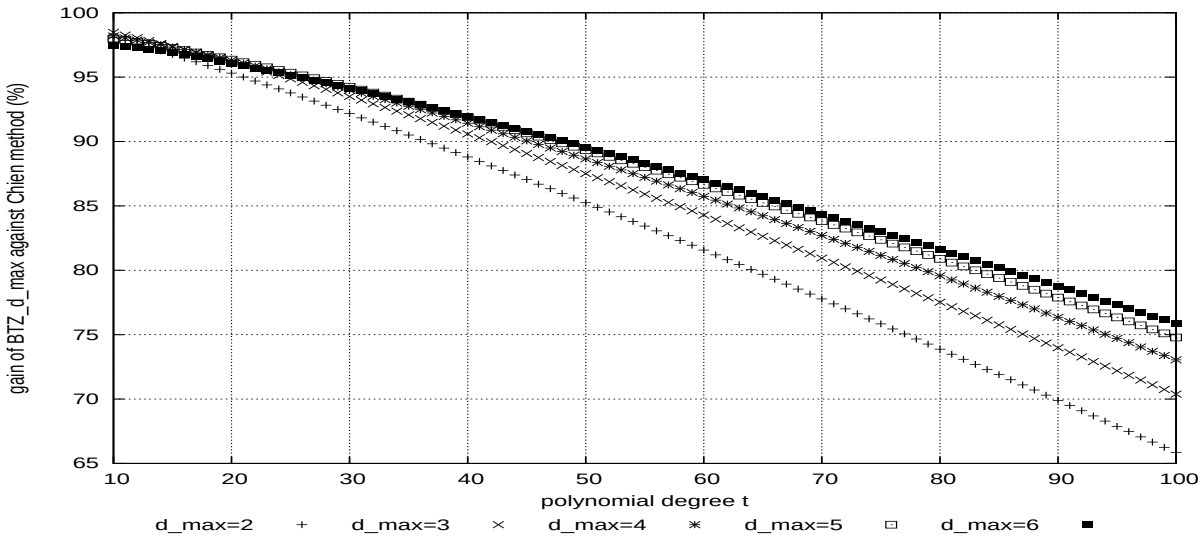

Figure 3.3: $\mathrm{BTZ}_{d_{max}}$ vs. Chien; $m = 11$; $K(+) = 1$; $K(\times) = m$; with time-memory tradeoff

Bibliography

[1] C. Adams and H. Meijer. Security-related comments regarding McEliece's public-key cryptosystem. In C. Pomerance, editor, *Advances in Cryptology - CRYPTO'87*, number 293 in LNCS, pages 224–228. Springer-Verlag, 1987.

[2] A. Barg. Complexity issues in coding theory. In V. S. Pless and W. C. Huffman, editors, *Handbook of Coding theory*, volume I, chapter 7, pages 649–754. North-Holland, 1998.

[3] M. Bellare and P. Rogaway. Optimal asymetric encryption. In A. De Santis, editor, *EUROCRYPT'94*, number 950 in LNCS, pages 92–111. Springer-Verlag, 1995.

[4] Thierry P. Berger, Pierre-Louis Cayrel, Philippe Gaborit, and Ayoub Otmani. Reducing key length of the mceliece cryptosystem. pages 77–97, 2009.

[5] E. R. Berlekamp. Factoring polynomials over large finite fields. *Mathematics of Computation*, 24(111):713–715, 1970.

[6] E. R. Berlekamp. Factoring polynomials over large finite fields. In *SYMSAC '71: Proceedings of the second ACM symposium on Symbolic and algebraic manipulation*, page 223, New York, USA, 1971. ACM.

[7] E. R. Berlekamp. *Algebraic Coding Theory*. Aegen Park Press, 1984.

[8] E. R. Berlekamp, R. J. McEliece, and H. C. van Tilborg. On the inherent intractability of certain coding problems. *IEEE Transactions on Information Theory*, 24(3), May 1978.

[9] E. R. Berlekamp, H. Rumsey, and G. Solomon. On the solution of algebraic equations over finite fields. volume 10, pages 553–564, June 1967.

[10] Elwyn R. Berlekamp. Bit-serial reed - solomon encoders. *IEEE Transactions on Information Theory*, 28(6):869–874, 1982.

[11] Daniel J. Bernstein, Tanja Lange, and Christiane Peters. Attacking and defending the mceliece cryptosystem. pages 31–46, 2008.

[12] T. Berson. Failure of the McEliece public-key cryptosystem under message-resend and related-message attack. In B. Kalisky, editor, *CRYPTO'97*, number 1294 in LNCS, pages 213–220. Springer-Verlag, 1997.

[13] Bhaskar Biswas and Nicolas Sendrier. McEliece cryptosystem implementation: Theory and Practice. In *PQCrypto*, pages 47–62, 2008.

[14] A. Canteaut and F. Chabaud. A new algorithm for finding minimum-weight words in a linear code: Application to McEliece's cryptosystem and to narrow-sense BCH codes of length 511. *IEEE Transactions on Information Theory*, 44(1):367–378, January 1998.

[15] A. Canteaut and N. Sendrier. Cryptanalysis of the original McEliece cryptosystem. In *Advances in Cryptology - ASIACRYPT'98*, number 1514 in LNCS, pages 187–199. Springer-Verlag, 1998.

[16] Pierre-Louis Cayrel, Philippe Gaborit, David Galindo, and Marc Girault. Improved identity-based identification using correcting codes. *CoRR*, abs/0903.0069, 2009.

[17] R. T. Chien. Cyclic decoding procedures for Bose-Chaudhuri-Hocquenghem codes. volume 10, pages 357–363, 1964.

[18] Benny Chor and Ronald L. Rivest. A knapsack-type public key cryptosystem based on arithmetic in finite fields. *IEEE Transactions on Information Theory*, 34(5):901–909, 1988.

[19] T. Cover. Enumerative source encoding. *IEEE Transactions on Information Theory*, 19(1):73–77, January 1973.

[20] W. Diffie and M. Hellman. New directions in cryptography. *IEEE Transactions on Information Theory*, 22(6):644–654, November 1976.

[21] Jean-Charles FaugÃ¨re, Ayoub Otmani, Ludovic Perret, and Jean-Pierre Tillich. Algebraic cryptanalysis of mceliece variants with compact keys. 2010.

[22] Matthieu Finiasz. *Nouvelles constructions utilisant des codes correcteurs d'erreurs en cryptographie Ã clef publique*. Th'ese de doctorat, École Polytechnique, October 2004.

[23] Matthieu Finiasz and Nicolas Sendrier. Security bounds for the design of code-based cryptosystems. 5912:88–105, 2009.

[24] Philippe Gaborit. Shorter keys for code based cryptography. *International Workshop on Coding and Cryptography - WCC, Bergen, Norway*, 2005.

[25] V.D. Goppa. A new class of linear error-correcting codes. In *Probl. Inform. Transm.*, volume 6, pages 207–212, 1970.

[26] C. Hall, I. Goldberg, and B. Schneier. Reaction attacks against several public-key cryptosystems. In *Proc. of ICICS'99*, number 1726 in LNCS, pages 2–12. Springer-Verlag, 1999.

[27] I. Herstein. *Topics in Algebra*. John Wiley, New York, 1975.

[28] I. S. Hsu, T. K. Truong, L. J. Deutsch, and I. S. Reed. A comparison of vlsi architecture of finite field multipliers using dual, normal, or standard bases. *IEEE Trans. Comput.*, 37(6):735–739, 1988.

[29] K. Huber. Note on decoding binary Goppa codes. *Electronics Letters*, 32(2):102–103, August 2002.

[30] H. Janwa and O. Moreno. McEliece public key cryptosystems using algebraic-geometric codes. *Design, Codes and Cryptography*, 8:293–307, 1996.

[31] K. Kobara and H. Imai. Semantically secure McEliece public-key cryptosystems - Conversions for McEliece PKC-. In K. Kim, editor, *PKC'2001*, number 1992 in LNCS, pages 19–35. Springer-Verlag, 2001.

[32] N. Koblitz. Elliptic curve cryptosystems. *Mathematic of Computation*, 48(177):203–209, 1987.

[33] P. J. Lee and E. F. Brickell. An observation on the security of McEliece's public-key cryptosystem. In C. G. Günther, editor, *Advances in Cryptology - EUROCRYPT'88*, number 330 in LNCS, pages 275–280. Springer-Verlag, 1988.

[34] J. S. Leon. A probabilistic algorithm for computing minimum weights of large error-correcting codes. *IEEE Transactions on Information Theory*, 34(5):1354–1359, September 1988.

[35] R. Lidl and H. Niederreiter. *Finite Fields*. Cambridge University Press, 1983.

[36] Rudolf Lidl and Harald Niederreiter. *Finite Fields (Encyclopedia of Mathematics and its Applications)*. Cambridge University Press, October 1996.

[37] P. Loidreau and N. Sendrier. Some weak keys in McEliece public-key cryptosystem. In *IEEE Conference, ISIT'98*, Cambridge, MA, USA, August 1998.

[38] P. Loidreau and N. Sendrier. Weak keys in McEliece public-key cryptosystem. *IEEE Transactions on Information Theory*, 47(3):1207–1212, April 2001.

[39] F. J. MacWilliams and N. J. A. Sloane. *The Theory of Error-Correcting Codes*. North-Holland, 1977.

[40] R. J. McEliece. A public-key cryptosystem based on algebraic coding theory. *DSN Prog. Rep.*, Jet Prop. Lab., California Inst. Technol., Pasadena, CA, pages 114–116, January 1978.

[41] Ralph C. Merkle, Student Member, Ieee, Martin E. Hellman, and Senior Member. Hiding information and signatures in trapdoor knapsacks. *IEEE Transactions On Information Theory*, 24:525–530, 1978.

[42] Rafael Misoczki and Paulo S. L. M. Barreto. Compact mceliece keys from goppa codes. pages 376–392, 2009.

[43] H. Niederreiter. Knapsack-type crytosystems and algebraic coding theory. *Prob. Contr. Inform. Theory*, 15(2):157–166, 1986.

[44] City CA) Omura, Jimmy K. (Culver and CH) Massey, James L. (Zurich. Computational method and apparatus for finite field arithmetic. (4587627), May 1986.

[45] N. J. Patterson. The algebraic decoding of Goppa codes. *IEEE Transactions on Information Theory*, 21(2):203–207, March 1975.

[46] D. Pointcheval. Chosen-ciphertext security for any one-way cryptosystem. In H. Imai and Y. Zheng, editors, *PKC 2000*, number 1751 in LNCS, pages 129–146. Springer-Verlag, 2000.

[47] R. L. Rivest, A. Shamir, and L. M. Adleman. A method for obtaining digital signatures and public-key cryptosystems. *Communications of the ACM*, 21(2):120–126, February 1978.

[48] N. Sendrier. On the structure of a randomly permuted concatenated code. In P. Charpin, editor, *Livre des résumé – EUROCODE 94*, pages 169–173, Abbaye de la Bussière sur Ouche, France, October 1994. INRIA.

[49] N. Sendrier. Finding the permutation between equivalent codes: the support splitting algorithm. *IEEE Transactions on Information Theory*, 46(4):1193–1203, July 2000.

[50] N. Sendrier. *Cryptosystèmes à clé publique basés sur les codes correcteurs d'erreurs.* Mémoire d'habilitation à diriger des recherches, Université Paris 6, March 2002.

[51] N. Sendrier. On the security of the McEliece public-key cryptosystem. In M. Blaum, P.G. Farrell, and H. van Tilborg, editors, *Information, Coding and Mathematics*, pages 141–163. Kluwer, 2002. Proceedings of Workshop honoring Prof. Bob McEliece on his 60th birthday.

[52] N. Sendrier. Encoding information into constant weight words. In *IEEE Conference, ISIT'2005*, Adelaide, Australia, September 2005.

[53] Peter W. Shor. Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer. *SIAM J. Comput.*, 26(5):1484–1509, 1997.

[54] V. M. Sidel'nikov. A public-key cryptosystem based on Reed-Muller codes. *Discrete Mathematics and Applications*, 4(3):191–207, 1994.

[55] V. M. Sidel'nikov and S. O. Shestakov. On cryptosystem based on generalized Reed-Solomon codes. *Discrete mathematics (in russian)*, 4(3):57–63, 1992.

[56] J. Stern. A method for finding codewords of small weight. In G. Cohen and J. Wolfmann, editors, *Coding theory and applications*, number 388 in LNCS, pages 106–113. Springer-Verlag, 1989.

[57] H.M. Sun. Further cryptanalysis of the McEliece public-key cryptosystem. *IEEE Trans. on communication letters*, 4(1):18–19, January 2000.

[58] J. von zur Gathen and J. Gerhard. *Modern computer algebra*. Cambridge Univ. Press, 2003.

[59] Charles C. Wang, T. K. Truong, Howard M. Shao, Leslie J. Deutsch, Jim K. Omura, and Irving S. Reed. Vlsi architectures for computing multiplications and inverses in gf(2m). *IEEE Trans. Comput.*, 34(8):709–717, 1985.

[60] V.A. Zinoviev. On the solution of equations of degree ≤ 10 over finite fields GF(2^q). 1996.