



École Doctorale
d'Informatique,
Télécommunications
et Électronique
de Paris

Thèse

présentée pour obtenir le grade de docteur

de Télécom Paris Tech

Spécialité : Électronique et communication

Nidhal SELMANE

Attaques en fautes globales et locales sur les cryptoprocesseurs AES : mise en œuvre et contremesures

Soutenance prévue le 13 Décembre 2010 devant le jury composé de

David NACCACHE	**Rapporteurs**
Regis LEVEUGLE	
Guido BERTONI	**Examinateurs**
Jacques FOURNIER	
Habib MEHREZ	
Gilles PIRET	
Jean-Luc DANGER	**Directeurs de thèse**
Sylvain GUILLEY	

To my parents...

Acknowledgements

I would like to acknowledge all the people who supported me during my research.

First and formost, i would like to thank my thesis director Jean-luc Danger for his valuable guidance and supports, his understanding and encouraging have provided a good basis for the present thesis. I would like also to express my deep and sincere gratitude to my supervisor Sylvain Guilley, his wide knowledge and his logical way of thinking have been of great value for me.

Furthermore, I wish to express my warm and sincere thanks to the members of my reading committee, David Naccache, Regis Leveugle, Jacques Fournier and Guido Bertoni how have accepted to evaluate my work and for their detailed and constructive comments.

Moreover, During this work I have collaborated with many colleagues for whom I have great regard, and I wish to extend my warmest thanks to all those who have helped me with my work in COMELEC. Especially, i would like to mention Sami Mekki, Youssef Suissi, Shivam Basin, Hossem Magrebi, Zouha Chrif, Taoufik Chouta, Maxim Nassar, Sebastien Thomas, Oliver Meynard, Aziz ElAbid, Laurent Sauvage, Florent Falment, Sompasong Somsavaddy, Chantale Cadiat, Daniel Childz, Zouina Sahnoune, Karim Ben Kalaia, Guillome Duc, Tarik Grabaa, Lirda Naviner and Philippe Matehrat.

Finally, i would like to thank my family for their support through my education and studies. Without their encouragement and understanding it would have been impossible for me to finish this work.

For all our Freedom martyrs, Tunisia 14/01/2011

Contents

List of Figures vii

List of Tables xi

1 Résumé 1

2 Physical Attack On Cryptographic Implementation 21

2.1 Cryptography . . . 21

2.1.1 Symmetric Ciphers . . . 22

2.1.2 Asymmetric Cryptography . . . 30

2.2 Smartcard Architecture . . . 33

2.3 Side Channel Attack . . . 35

2.3.1 Timing Attack . . . 35

2.3.2 Power Analysis . . . 36

2.3.3 Electromagnetic Analysis . . . 39

2.4 Fault Attacks . . . 40

2.4.1 Power Spikes . . . 40

2.4.2 Clock Glitches . . . 40

2.4.3 Optical Attack . . . 41

2.4.4 Electromagnetic Perturbations Attack . . . 41

2.4.5 Definition of Fault Model . . . 42

2.4.6 Fault Attack on AES . . . 43

2.4.7 Summary of DFA on AES . . . 50

2.5 Conclusion . . . 51

3 Practical Attacks on AES 53

3.1 Global Attack: Setup time violation attack . . . 53

3.1.1 Attack Theory . . . 53

3.1.2 Acquisition Platform . . . 55
3.1.3 Fault Analysis . . . 56
3.1.4 Attack on ASIC . . . 58
3.1.5 Attack on FPGA . . . 61
3.2 Local Attack: Optical Fault Injection . . . 74
3.2.1 Decapsulation . . . 74
3.2.2 Practical Setup . . . 75
3.2.3 Experimental Results . . . 77
3.3 Conclusion . . . 78

4 Fault Attack Countermeasures **81**
4.1 Fault Detection . . . 82
4.1.1 Parity . . . 82
4.1.2 Concurrent Error Detection . . . 83
4.1.3 Cyclic Redundancy Check . . . 83
4.1.4 Non Linear Robust Code . . . 84
4.1.5 Double-Data-Rate as countermeasure . . . 86
4.1.6 Low cost countermeasure against setup time violation attacks . . . 86
4.2 Fault Resilience . . . 88
4.2.1 Comparison between Detection and Resilience . . . 88
4.2.2 Further Merits of the Fault Injection Resilience "FIR" . . . 90
4.2.3 Related Works . . . 90
4.2.4 Some Practical Implementations of FIR . . . 91
4.2.5 Dual-Rail with Precharge Logic as a Global Countermeasure against Implementation-Level Attacks . . . 96
4.2.6 Cost Estimation of FIR versus Traditional Approaches . . . 104
4.2.7 Associating Three Protections to Reduce the Probability of a Successful FIA . . . 107
4.2.8 Applicability of Resilience with Certification Procedures . . . 108
4.3 Case study on WDLL . . . 109
4.3.1 Wave Dynamic Differential Logic . . . 109
4.3.2 Design Flow for WDDL Implementation . . . 111
4.3.3 Experimental Results . . . 113
4.3.4 Theoretical Fault Analysis on AES in WDDL . . . 115
4.3.5 WDDL w/o EPE . . . 121

4.3.6 Analysis of the DFA Protection for DPL w/o EPE 122
4.4 Conclusion 126

5 Conclusion **127**

Bibliography **133**

List of Figures

1.1 standard de chiffrement AES . . . 3
1.2 Injection de l'erreur dans AES . . . 5
1.3 Erreur dans le tour 9 . . . 5
1.4 Erreur dans le tour 8 . . . 6
1.5 Violation de temps de setup . . . 7
1.6 Plate-forme d'injection de fautes . . . 8
1.7 Occurrence des fautes — (tension). . . . 9
1.8 Occurrence des fautes (fréquence). . . . 9
1.9 Architecture du cryptoprocesseur AES . . . 11
1.10 Occurrence des fautes simple dans Altera Stratix. . . . 12
1.11 Occurrence des fautes simple dans Xilinx Virtex5. . . . 12
1.12 Plate-forme d'injection optique . . . 13
1.13 Cartographie des fautes . . . 13
1.14 Code robuste non-linéaire . . . 15
1.15 protocole DPL . . . 16
1.16 occurrence des fautes dans une version wddl d'AES . . . 17

2.1 Data Encryption Standard . . . 24
2.2 AES encryption . . . 26
2.3 SubBytes transformation [1]. . . . 27
2.4 ShiftRow transformation [1]. . . . 27
2.5 MixColumn transformation [1]. . . . 28
2.6 AddRoundKey operation . . . 29
2.7 AES Key Schedule . . . 29
2.8 Smartcard internal architecture . . . 34
2.9 SPA on RSA implementation . . . 36

2.10 DPA on DES: Bad and correct subkey . . . 38
2.11 EMA on DES: Bad and correct subkey . . . 39
2.12 Fault effect on round 9 of AES. . . . 48
2.13 Fault effect on round 8 of AES . . . 49

3.1 Setup violation. . . . 55
3.2 Experimental faults injection platform . . . 56
3.3 AES faults analysis. . . . 57
3.4 Occurrence of Fault — (power). . . . 58
3.5 Occurrence of fault (over-clocking). . . . 58
3.6 Coverage of exploitable faults. . . . 59
3.7 Temporal localization of single faults. . . . 61
3.8 Spatial localization of single faults.The SubBytes box $s_{i,j}$ has index $4 \times i + j$ in the histogram. . . . 61
3.9 Simple AES. . . . 62
3.10 AES architecture. . . . 63
3.11 Composite Field implementation of SubBytes . . . 63
3.12 Occurrence of faults: sbox in $GF(2^4)$.(ALTERA) . . . 64
3.13 Occurrence of faults: sbox in LUT. (ALTERA) . . . 65
3.14 Occurrence of faults: sbox in RAM.(ALTERA) . . . 65
3.15 Coverage of single faults, and detail of exploitable faults in $GF(2^4)$. . . . 66
3.16 Coverage of single faults, and detail of exploitable faults in LUT. . . . 66
3.17 Coverage of single faults, and detail of exploitable faults in RAM. . . . 68
3.18 Hamming weight of exploitable faults in $GF(2^4)$. . . . 68
3.19 Exploitable errors "Round 8 and 9". . . . 69
3.20 AES architecture with critical path strictly confined in the datapath. . . . 71
3.21 Temporal localization of single fault on Altera GF(2^4). . . . 71
3.22 Occurrence of single Faults (Altera Stratix) . . . 73
3.23 Occurrence of single faults (Xilinx Virtex5) . . . 73
3.24 Optical fault injection platform . . . 76
3.25 Temporal localization with random hit . . . 77
3.26 Cartography of faults . . . 78

4.1 Parity based countermeasure . . . 83
4.2 Concurrent error detection . . . 84
4.3 Robust code countermeasure. . . . 85

4.4 Double-Data-Rate as countermeasure . . . 86
4.5 Counter-measure based on the insertion of a monitoring logic with a propagation time larger than the critical path of the rest of the circuit. . . 86
4.6 Chain Voltage/lcell. . . . 87
4.7 Suicide in case of fault detection (*top*), opposed to survival in case of fault resilience (*bottom*) protection schemes. . . . 90
4.8 Probabilistic encryption and deterministic decryption . . . 93
4.9 Two kinds of faults for 3-valued logic and for DPL, . . . 95
4.10 Susceptible organs of a smartcard in two representative sensitive operations (`EXTERNAL AUTHENTICATE` and `INTERNAL AUTHENTICATE`). Typically, the cryptography will be triple-DES or AES. . . . 97
4.11 DPL protocol. . . . 98
4.12 Two DPL w/ EE drawbacks to fight DFAs, illustrated on the example of a WDDL AND gate. In this figure and in the subsequent ones, the asterisk character (*) symbolizes the faults. . . . 101
4.13 Difference of detection and resilience working factors, represented on an example netlist. . . . 103
4.14 Memorization element in TMR. . . . 106
4.15 Memorization element in DPL. . . . 107
4.16 Multiple faults, where the false valid is not completely hidden by the `'X'` wave. . . . 108
4.17 Timing diagram for a WDDL AND gate. . . . 110
4.18 WDDL building block. . . . 111
4.19 WDDL design flow. . . . 112
4.20 occurrence of faults in wddl version . . . 113
4.21 Temporal and Spatial localization of single faults for the Wddl implementation . . . 114
4.22 WDDL implementation of the XOR gate. . . . 117
4.23 Dual-to-single rail circuitry usable in the case of a NULL0 spacer. . . . 119
4.24 WDDL w/o AND gate . . . 122
4.25 Probability that m faults injected on n wires be innocuous due to the protection conveyed by two different countermeasures: either a *detection* by an informational redundancy scheme or an *annihilation of the faulted data* by one or several VALID $\stackrel{*}{\rightarrow}$ NULL token transformations. . . . 125

LIST OF FIGURES

List of Tables

1.1 Localisation temporelle et spatial de fautes simples dans Altera Stratix . 10
1.2 Altera Vs Xilinx 12

2.1 Summary of Differential Fault Attack 51

3.1 Temporal and Spatial localization of single faults on Altera Stratix board 67
3.2 Temporal and Spatial localization of single faults on Xilinx Virtex 5 board 72
3.3 Characterization and attack results for Altera and Xilinx with the three Sbox architectures. 73

4.1 Nonlinear Robust code implementation. 85
4.2 Classical fault detection characteristics. 88
4.3 Performance overhead of different SCA+FIA countermeasures. 105
4.4 Single fault in round 10. 114
4.5 Single fault in round 9. 114
4.6 Fault strictly before round 9. 115
4.7 Modified functionality of an AND gate in the presence of erasure faults. 116
4.8 Modified functionality of an XOR gate in the presence of erasure faults. . 117
4.9 Equations for the bytes transformations $\times 01$, $\times 02$ and $\times 03$. 118
4.10 WDDL w/o area overhead 122

Chapter 1

Résumé

Introduction

Avec l'apparition des ordinateurs et des circuits intégrées, la cryptologie a connue un vrais essor. Elle est utilisée dans plusieurs domaines (carte à puce, Transaction bancaire, Télévision payante . . .). La cryptologie rassemble l'ensemble des techniques de cryptographie et de cryptanalyse.

La cryptographie est l'art de dissimuler un message en utilisant des techniques de transposition et de substitution. Le mot cryptographie vient du mot grec « Kryptos » qui veut dire « caché » et du mot « graphein » pour « écriture ». Elle respecte les principes de Kerckhoffs, qui justifient que toute information liée à un crypto système peut être publique à l'exception des clés de chiffrements.

La cryptanalyse inclut des techniques très avancées afin de retrouver ces clés pour pouvoir déchiffré les messages codés. Les attaques linéaires et différentielles en sont les exemples les plus probants. Toutefois bien que ces techniques nécessitent encore souvent de grandes quantités de paires de textes en clairs et de textes chiffrés, il existe d'autres méthodes très puissantes basées sur les "fuites d'information" involontaires. En effet un crypto système peut laisser fuir de l'information de différentes manières, c'est ainsi que des données sensibles peuvent parfois être extraites de signaux physiques émis par une machine de chiffrement. La température, la consommation, le rayonnement électromagnétique, le temps de calcul ou la lumière (infra rouge, interaction avec un laser) sont autant d'indices qui peuvent s'avérer extrêmement dangereux. On parle alors de side channel attack. Il existe plusieurs branches

de ce type d'attaque notamment Differential Power Analysis (DPA) qui exploite la consommation ou Differential Fault Analysis (DFA) qui exploite les erreurs du calcul cryptographique pour retrouver la clé du chiffrement.

La cryptanalyse par canaux cachés on longtemps été le terrain de compétence réservé des services secrets, mais depuis une dizaine d'années, avec la puissance de calcul qu'offre les ordinateurs modernes ce genre d'attaque c'est ouvert au milieux scientifique et universitaire. Ce qui constitue une vrai menace pour la sécurité des systèmes d'informations. C'est pour cette raison que les industriels et les laboratoires de recherche accorde de plus en plus d'importance à la sécurisation des circuits destiné à des applications cryptographiques

C 'est dans ce cadre que s'inscrit ma thèse qui consiste à mettre en oeuvre les attaque par injection de fautes (DFA) sur les cryptoprocesseurs AES, puis de mettre en place les contre-mesures nécessaires pour sécuriser les processeurs contre ce type d'attaque.

Le standard de chiffrement AES

L'algorithme de Chiffrement DES a été développé en 1976 par IBM. C'est un chiffrement à clé symétrique qui est encore utilisé dans plusieurs domaines (transactions bancaires...). Mais avec les progrès de l'informatique, les 2^{56} clés possibles du DES ne représente plus une barrière infranchissable. Il est désormais possible, même avec des moyens modestes, de percer les messages chiffrés par DES en un temps raisonnable. En janvier 1997, le NIST (National Institute of Standards and Technologies) des Etats-Unis lance un appel d'offres pour élaborer l'AES (Advanced Encryption standard). Le 2 octobre 2000, le NIST choisi parmi les sept candidats finals, l'algorithme*R*ijndael qui est choisi pour être le successeur de DES. C'est un algorithme de chiffrement par bloc mis au point par deux chercheurs belges, Joan Daemen et Vincent Rijmen.Le Rijndael procède par blocs de 128 bits, avec une clé de taille variable selon l'importance du message. AES est un algorithme itérative le nombre de tour dépend de la taille de la clé utilisé[1].

1. 128 bits, Nombre de tour 10.
2. 196 bits, Nombre de tour 12.
3. 256 bits, Nombre de tour 14.

La Figure 1.1 montre les différentes étapes de l'algorithme AES. Le chiffrement trans-

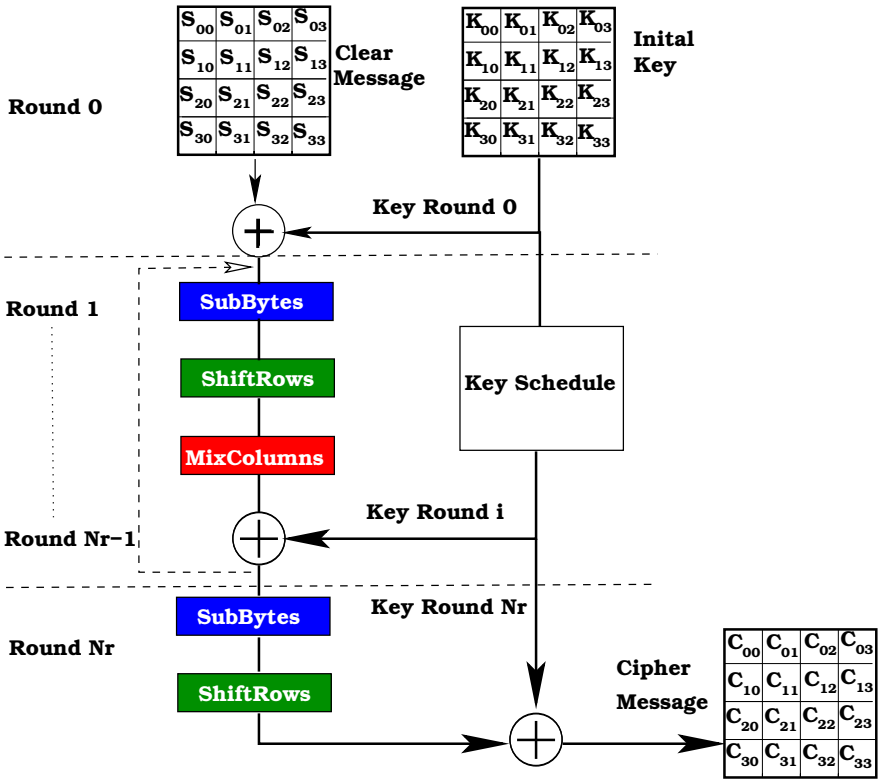


Figure 1.1: standard de chiffrement AES

forme les données contenues dans le bloc en appliquent quatre transformations sur les octets de la matrice d'état.

1. une substitution non linéaire « SubBytes ».
2. une permutation circulaire des octets au sein d'une même ligne « ShiftRows ».
3. une multiplication dans $\frac{GF(2^8)[X]}{(X^4+1)GF(2^8)[X]}$ pour chaque colonne « Mixcolumns »
4. une addition de clé « AddRoundKey ».

La transformation SubBytes

La transformation SubBytes() est une substitution non linéaire d'octets, utilisant une table S (S-box). Cette table est construite en composant deux transformations :

1. Prendre l'inverse de l'octet dans $\frac{\mathbb{Z}_2[X]}{m(X)\mathbb{Z}_2[X]}$, l'octet 0x00 étant par convention son propre inverse.

2. Lui appliquer la transformation affine suivante (dans $\frac{\mathbb{Z}_2[X]}{m(X)\mathbb{Z}_2[X]}$) :

$$pour \quad 0 \leq i < 8$$
$$b_i^{'} = b_i \oplus ..... \oplus b_{(i+6)mod8} \oplus b_{(i+7)mod8} \oplus c_i$$

La transformation ShiftRows

La transformation ShiftRows() applique une permutation circulaire sur les trois dernières lignes du bloc

$$0 < r < 4 \quad et \quad 0 < c < \mathbf{Nb}$$
$$s_{r,c}^{''} = s_{r,(c+shift(r,Nb))mod\ Nb}^{'}$$

La transformation MixColumns

La transformation MixColumns() traite chaque colonne comme un polynôme de degré 3, on calcule dans le corps galois $\mathcal{GF}_{2^8}$ le produit de ce polynôme avec un polynôme fixe $a(x)$.

$$a(x) \quad = \quad (\texttt{0x03})x^3 + (\texttt{0x01})x^2 + (\texttt{0x02})x + (\texttt{0x02})$$

La transformation AddRoundKey

La transformation AddRoundKey() addition au bloc une clé de la façon suivante :

1. une clé de tour est extraite à chaque tour, celle-ci est composée de 4 mots de 4 octets.

2. Les mots sont additionnés aux colonnes avec un simple XOR $\oplus$ sur les 16 octets.

Attaque de Piret

Comme on a vue dans la présentation de l'algorithme AES, dans le dernier tour de chiffrement on n'utilise pas la transformation MixColumns, C'est pour pouvoir décrypter le message avec la clé de la 10^{eme} ronde, inversement à l'AddRoundKey du tour initial. L'attaque de Piret exploite cette faille de l'AES. Mais elle ne fonctionne que si les deux fautes touchent un unique octet (néanmoins inconnu) de l'avant-dernier 9^{eme}**Ronde** l'avant-avant-dernier tour du chiffrement 8^{eme}**Ronde** , l'erreur doit être

injecter avant la transformation MixColumns sinon elle ne sera pas expoitable. La figure 1.2 montre l'injection de la faute dans le tour 9 du pipe du Calcul.

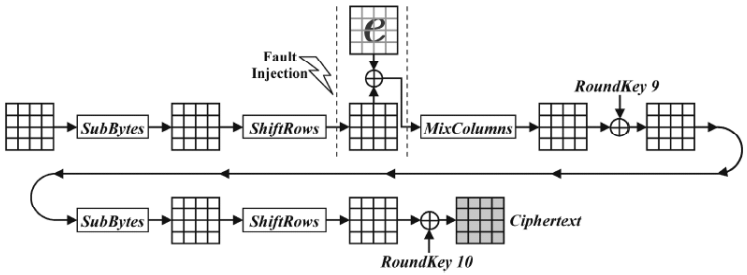


Figure 1.2: Injection de l'erreur dans AES

Analyse des effets de l'erreur

L'erreur touche un octet unique dans une colonne de la matrice d'état ShiftRows et SubBytes ne propage pas l'erreur Mais la transformation MixColumns affecte le reste des octets de la colonne. La figure 1.3 illustre la propagation d'une erreur qui tombe dans le tour 9 .

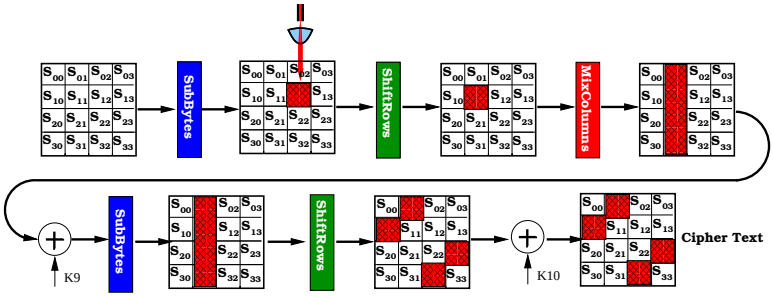


Figure 1.3: Erreur dans le tour 9

On remarque que l'erreur qui affecte l'octet S_{12} de la colonne 2 peut nous donner l'information sur quatre octets de la clés K_{01}, K_{10}, K_{23}, K_{32}. En utilisant trois autres fautes qui affecte les trois autres colonnes on peut avoir de l'information sur les 16 octets de la clé. Donc pour casser la clé, il faut avoir 4 paire de messages.

Analyse de l'information de l'erreur

Soit C un message chiffré sans erreur et D un message chiffré avec une erreur dans le tour 9 donc $E = C \oplus D$ représente l'erreur, et on trouve que dans E il y a quatre octets non nulle[**?**],qu'on peut exploiter pour trouver quatre octets de la clé comme suit:

1. Preparer une liste $\mathcal{L}_d$ qui contient 1020 différences possibles de MixColumns du Round 9 (255 x 4).
2. Faire une recherche exhaustive sur les $K^{Nr}_{0,d}, K^{Nr}_{1,(d-1)mod4}, K^{Nr}_{2,(d-2)mod4}, K^{Nr}_{3,(3-d)mod4}$.
3. Calculer $\Delta_t = SubBytes^{-1}((C \oplus K^{Nr})_{*,d}) \oplus SubBytes^{-1}((D \oplus K^{Nr})_{*,d})$
4. Voir si $\Delta_t \in \mathcal{L}_d$
5. Si oui ajouter les quatre octets de la clé à la liste $\mathcal{C}_d$ des candidats possibles
6. Retour à l'êtape 2 avec une autre paire jusqu'a avoir un seul candidat.

On remarque qu'on utilise 4 paires de message pour trouver les 16 octets de la clé. Mais on peut faire d'une pierre quatre coups en injectant l'erreur dans le Round 8, et c'est la transformation ShiftRows qui va affecter les quatre colonnes de la matrice,puis c'est la transformation MixColumns qui affecte toute la matrice comme le montre la figure 1.4.

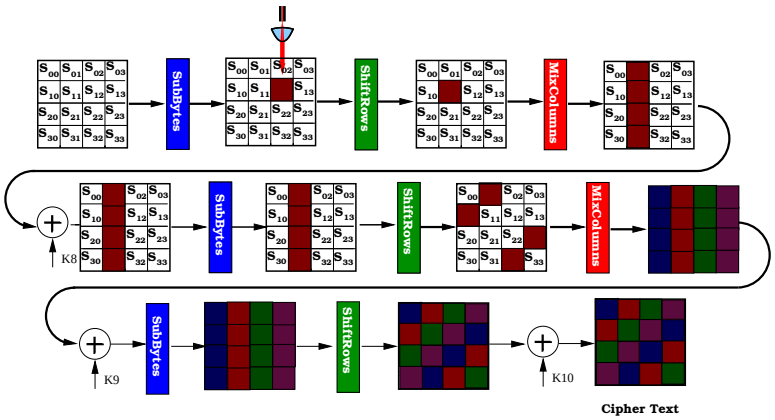


Figure 1.4: Erreur dans le tour 8

Attaque par violation de setup

Dans la logique séquentielle, un signal global, appelé l'horloge, cadence tous les calculs combinatoires. Toutes les portes devraient avoir fini de propager leurs données lorsque le front montant d'horloge arrive. Ce qu'on appelle le temps de setup, correspond à la période d'horloge la plus petite recevable.Si pour une raison quelconque la période d'horloge est inférieur au temps de setup, des erreurs dans le calcul peuvent apparaître. Puisque temps de propagation augmente avec la diminution de l'alimentation diminuer la tension d'alimentation peut provoquer des violations de temps de setup comme le montre la figure 1.5. Le délai de propagation, ainsi qu'un second élément, inhérent à l'échantillonnage des bascule D, appelé temps de stabilisation "Setup-time" définisse la fréquence maximale du circuits. En effet afin d'assurer un fonctionnement normal du circuit, la période d'horloge doit être strictement supérieur au délai de propagation maximal du circuit $T_{clk} > T_{critique} + T_{setup}$. Le sur-cadencement "overclocking" consiste à diminuer la période d'horloge et si les

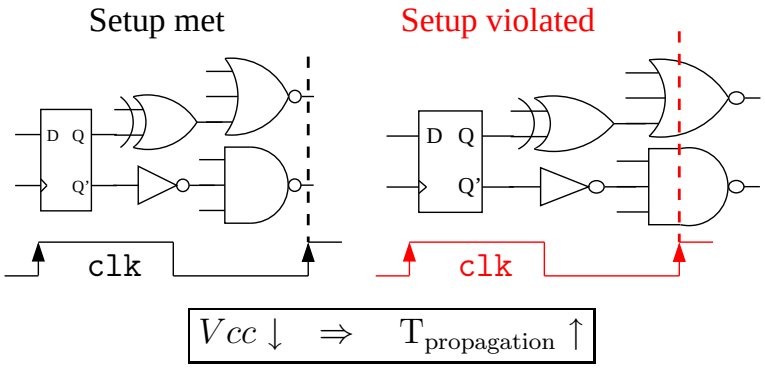


Figure 1.5: Violation de temps de setup

délais de stabilisation ne sont pas respectés. Cela a pour effet de stabiliser des fausse donnés d'ou l'injection de faute dans le système.

Plate-forme d'acquisition

A fin d'injecter les fautes on a développé une plate-forme qui permet de changer la tension d'alimentation et la fréquence d'horloge. La plate-forme consiste en une communication régulière entre un terminal RS232 et le circuit, l'alimentation et la fréquence de l'appareil sont également contrôlables à distance, de telle manière différentes valeurs

de fréquence et de tension peuvent être testés successivement. La figure 1.6 montre le dispositif d'injection de fautes. Pour chaque valeur de tension on enregistre la valeur de la clef, le message et le chiffré.

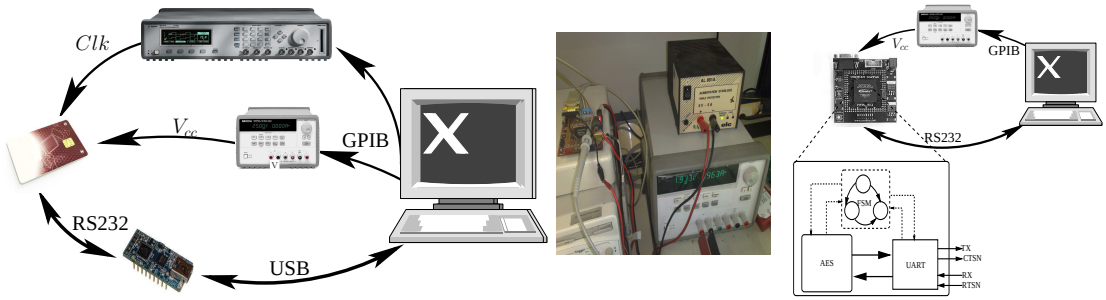


Figure 1.6: Plate-forme d'injection de fautes

Analyse

Afin d'analyser les fautes, nous supposons que le message et la clé sont connus par l'attaquant et nous utilisons une implémentation en C++ de AES adaptés pour pouvoir corrompre un octet de la matrice d'état à n'importe quel tour de chiffrement.

Attaque sur ASIC

Une première attaque sur le circuit SecMat V1 a été réaliser par:

1. **Diminution de la tension d'alimentation a une fréquence nominal de 32 Mhz**,
2. **Augmentation de la fréquence a une tension d'alimentation nominal de 1.2 V**,

On a progressivement augmenté le niveau de stress du circuit, et on remarque que l'attaquant peut choisir précisément la quantité de fautes induites dans le circuit. La figure 1.7et1.8 montre qu'il existe une plage confortable de la tension et de fréquence vulnérables où le circuit cryptographique fait sortir des résultats erronés.

Attaque sur FPGA

Architecture du SOC

A fin d'attaquer AES sur FPGA nous avons réalisé un circuit cryptographique que nous avons synthétisé sur Altéra Stratus et Filin Vortex5. Le circuit est composé de

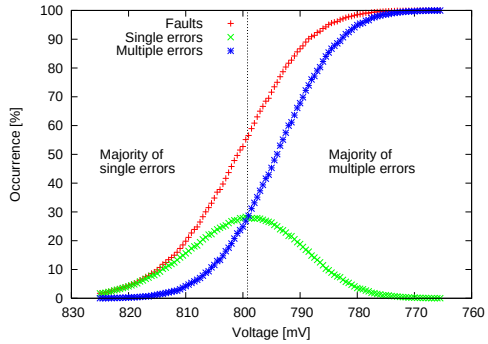


Figure 1.7: Occurrence des fautes — (tension).

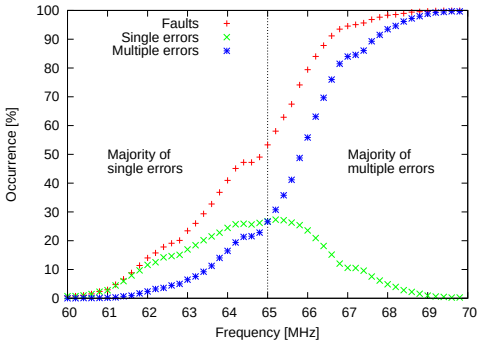


Figure 1.8: Occurrence des fautes (fréquence).

trois modules: une interface UART, un contrôleur et coprocesseur AES. Trois différentes implémentations du module Sbox de aes on été réalisés. Dans la première on a utilisé les LUT's de l'FPGA ,dans La deuxième implémentation on a utilisé la RAM de l'FPGA et dans la dernière implémentation on utilise une factorisation de la Sbox dans GF^4 . Finalement, on a réalisé l'attaque par violation de setup sur les différentes architectures.

Résultat sur Altera Stratix

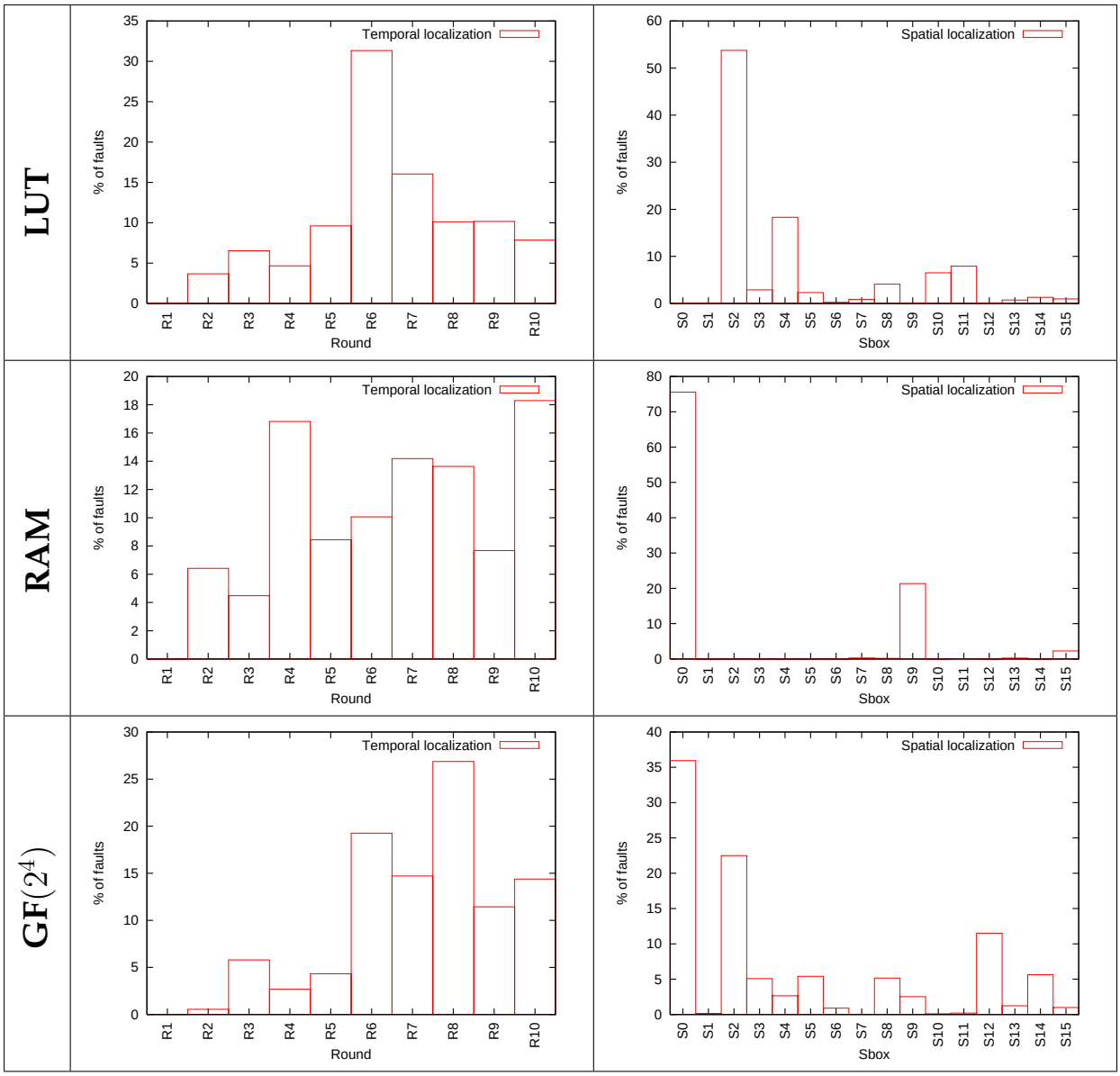
Le tableau 1.1 montrer la localisation temporelle et spatial des fautes simples dans l' FPGA Stratix d'Altera. On peut voir qu'il y a suffisamment de fautes simples qui arrivent dans les deux avant-dernier tours pour réaliser l'attaque de Gilles piret. On remarque que comme pour l'ASIC que les fautes ne sont pas uniformément répartis.

Comparaison Altera Stratix et Xilinx Virtex

La figures 1 montrent la fréquence d'apparition apparition des fautes simples dans les trois architectures d' Altera et de Xilinx. Seul les fautes simple qui affecte un octet avant la transformation Sbox sont affiché on remarque que les trois architectures sont vulnérable face au attaque par violation de setup.

En se référant au tableau 1.2 on remarque que le chemin critique dépend de l'architecture de la sbox implémentée. On peux voir aussi que l'architecture LUT a plus de fautes simple dans altera que dans Xilinx par contre on observe l'inverse pour l'implémentation

Table 1.1: Localisation temporelle et spatial de fautes simples dans Altera Stratix



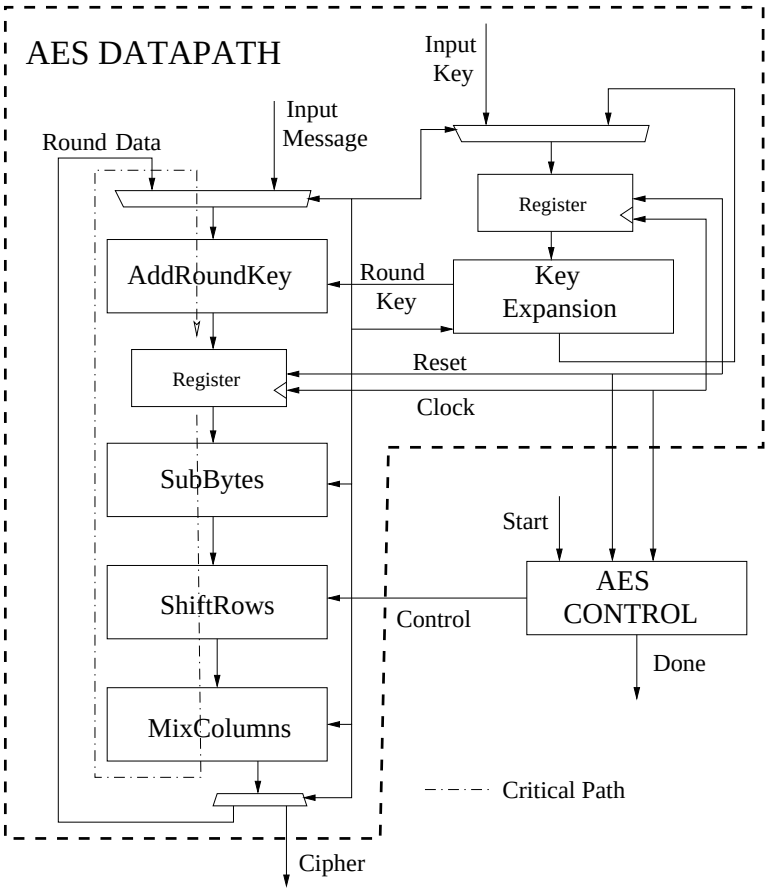


Figure 1.9: Architecture du cryptoprocesseur AES

dans la RAM. On remarque aussi que dans l' FPGA Xilinx les fautes simple commence a apparaître a des tensions plus basse que dans Altera.

Attaque optique

Plate-forme d'attaque

L'idée d'injection de fautes optiques a été présenté par S.Skorobogatov et R.Anderson en 2003 [2]. Ils ont montré qu'il est possible de modifier le contenu de la mémoire statique par la lumière. Pour réalise les différentes campagnes d'injection de fautes par tir laser nous utilisons une plate-forme laser composée d'une table XYZ, une caméra, deux lasers un vert de longueur d'onde $532\ nm$ et un autre infrarouge de longueur d'onde $1064\ nm$ et d'un générateur de basse fréquence pour contrôler la durée du tir

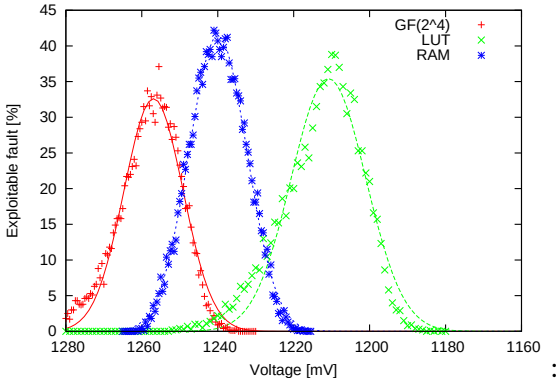


Figure 1.10: Occurrence des fautes simple dans Altera Stratix.

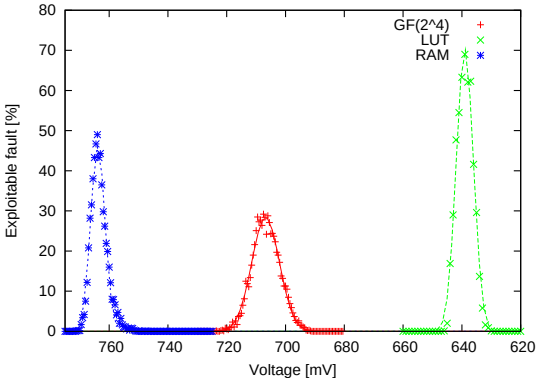


Figure 1.11: Occurrence des fautes simple dans Xilinx Virtex5.

Table 1.2: Altera Vs Xilinx

Architecture	**Temps critique(ns)**		**% faute simple**		**Surface**		**Voltage**	
FPGA	Altera	Xilinx	Altera	Xilinx	Altera	Xilinx	Altera	Xilinx
LUT	13.725	7.772	39 %	69 %	0.872	0.513	1.21	0.64
RAM	17.569	9.758	42 %	29 %	0.795	0.282	1.26	0.71
GF(2^4)	18.818	14.426	33 %	50 %	0.635	0.335	1.24	0.76

laser. L'ensemble du système est contrôlé par un programme qu'on a développé 1.12. La taille du spot laser est de 5 μm ce qui nous permet de cibler une petite zone du composant attaqué. Pour injecter les fautes une étape de préparation du composant est nécessaire c'est ce qu'on appel la décapsulation. Deux type de préparations peuvent être effectués sur le composant: une préparation chimique et une préparation mécanique. Cette étape permet au faisceau blaser d'atteindre la couche de silicium sans perdre beaucoup d'énergie. Le circuit qu'on a attaqué est un admet Armera128 qui implémente l'algorithme AES.

Résultat

On a réalisé plusieurs tentative d'injection en variant la puissance du laser et la zone d'injection finalement on réussit a trouvé une zone sensible autour d'un bus de donnée dans la flash. En utilisant un trigger qui déclenché le tir laser on a réussit a injecter

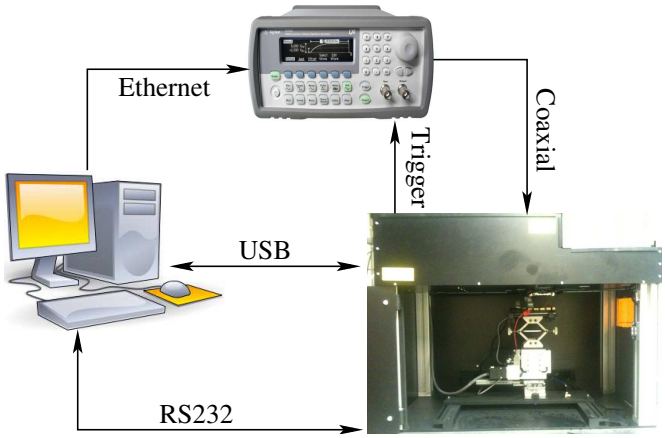


Figure 1.12: Plate-forme d'injection optique

des fautes dans l'avant dernier tour du chiffrement. Une analyse des fautes injectées a montrés que c'est des collages de type "stuck-at". la figure 1.13 montre la cartographie des fautes.

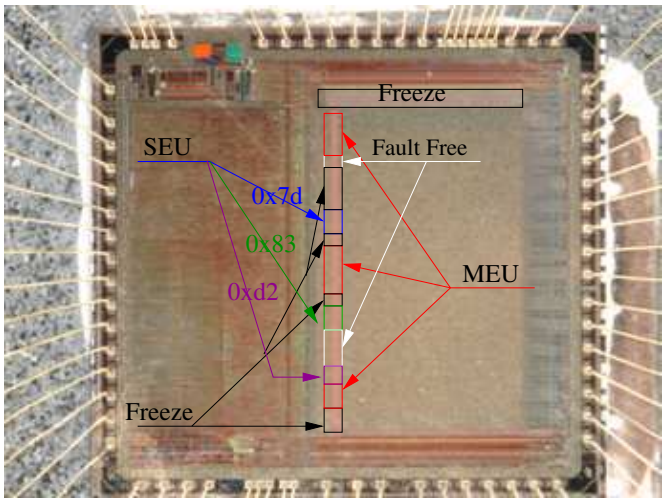


Figure 1.13: Cartographie des fautes

Contre-mesure

Vue l'importance des applications qui utilisent cette algorithme, il est nécessaire que les implémentations intègrent des solutions efficaces. Les contre-mesures sont toujours possibles et disponibles, mais elles doivent être bien pensées pour ne pas faire d'autre faille qui seront peut être exploiter dans le future. La plus part des solutions qu'on trouve dans la littérature se base sur deux principes, redondance spatiale et redondance temporelle.

Détection

C'est une contre-mesure proposée pour l'AES par Ramish. Karri [3]. Le principe est de déchiffrer chaque groupe de transformations en parallèle du chiffrement et de comparer la sortie du déchiffrement avec l'entrée avant le chiffrement. Il y a dans ce cas une double perte en terme de surface et de temps : un Sur-coût en surface de l'ordre de la duplication puisque chaque la sortie de chaque module doit être décodée, et un Sur-coût en temps puisque chaque élément doit être mémorisé pour être comparé avec le même élément codé et décodé.

Code robuste non-linéaire

Cette technique consiste à calculer pour chaque colonne de la clé et du message une signature sur 32 bits qui va nous permettre de détecter les erreurs, La signature est calculée à chaque cycle en utilisant une colonne de la clé du ronde. Le calcul de la signature ce fait en utilisant un prédicteur linéaire, puis un compresseur linéaire qui permet de réduire la signature de 32 bit à S_l bits tel que $S_l < 32$. La signature est cubé X^3 dans $\mathcal{GF}_2$ pour produire une signature non linéaire comme la sortie d'une ronde de l'AES. Une fois la signature originale obtenue on utilise un réseau de détection d'erreur(EDN) qui va permettre de calculer à partir de la sortie du ronde une autre signature qui sera comparer avec la signature final et ainsi détecter les erreurs [4]. Cette méthode de détection est très efficace car elle permet de protéger l'algorithme contre les attaques qui vise le datapath comme celle de Piret et les attaque qui vise l'expansion de la clé et elle permet de détecter les erreurs à la volée. Mais elle induit une augmentation considérable dans la surface du circuit. La figure 1.14 illustre cette contre mesure:

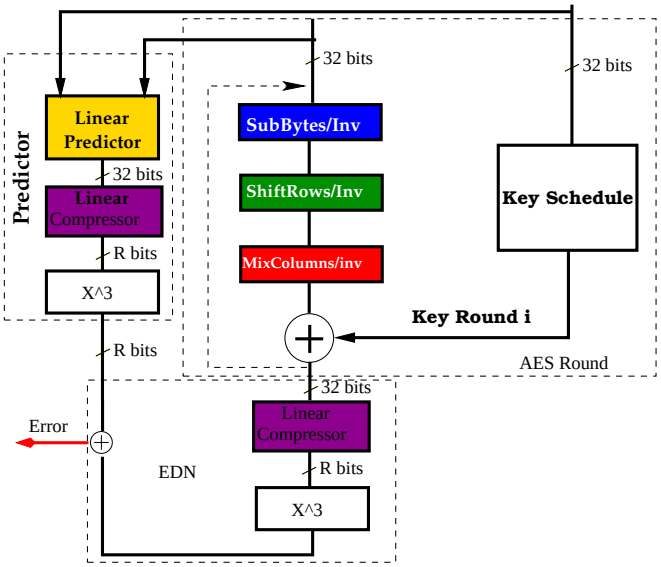


Figure 1.14: Code robuste non-linéaire

Résilience

Une autre stratégie pour contrer les attaques en fautes est la résilience, elle consiste a laisser la faute se propager dans le circuit tout en empêchant l'attaquant d'avoir un résultat exploitable pour pouvoir monter une DFA a fin de retrouvé la clé de chiffrement. Contrairement aux techniques classique de détection d'erreur, cette nouvelle approche autorise le circuit à continuer son calcul et a fournir un résultat faux,tant que ce résultat ne porte pas d'information utile pour retrouvé le secret. on propose dans cette thèse deux solution pour contré les attaques en fautes.

Niveau protocole

A fin de réaliser une attaque en faute, l'attaquant a besoin de chiffré deux fois le même message avec la même clé. Donc si on arrive a l'empêcher d'avoir ce couple de chiffrement faux et correcte. il ne sera pas capable de faire une DFA même si il réussi a injecter une faute dans le calcul. Pour implémenter ce niveau de résilience il suffit de modifier le message en lu ajoutant de l'aléa comme le montre les algorithmes suivants:

Algorithm 1: Chiffrement probabiliste

1 Générer un message aléatoire r de même taille que x.
2 Envoyer le couple $(y = \text{AES}_k(x \oplus r), r)$.

Algorithm 2: Déchiffrement Déterministe.

1 Déchiffrer y avec k: $z = \text{AES}_k^{-1}(y)$.
2 Envoyer $z \oplus r = x$.

Niveau logique

Une autre méthode pour implémenter la résilience c'est d'utiliser la logique DPL. En effet cette technique est utilisée généralement comme contre-mesure contre les attaques passives possède des propriétés de résilience. En effet il y a seulement deux états valide $(0, 1)$ et $(1, 0)$ donc si on injecte une faute simple on obtient deux états invalides $NULL0$ $(0, 0)$ et $NULL1$ $(1, 1)$ comme le montre la figure 1.15. Vue que les algorithmes cryptographiques possèdent une grande propriété de diffusion, ces états invalides se propagent dans toute la net-liste pour effacer tout l'information utile ce qui empêche un attaquant de réaliser une DFA.

A fin de validé cette contre mesure on a réalisé une implémentation en WDDL d'un AES puis une attaque par violation de temps de setup sur le circuit. Une observation intéressante est que a chaque fois qu'un octet est affecté par une fautes, un octet nul apparaît dans le texte chiffré a la place du bon octet. Cela signifie que même après avoir réussi à injecter la faute lors du chiffrement et de connaître précisément l'emplacement de la faute, la sortie ne donne aucune information qui peut être exploiter pour récupérer la clé de chiffrement. Par conséquent, une conception WDDL

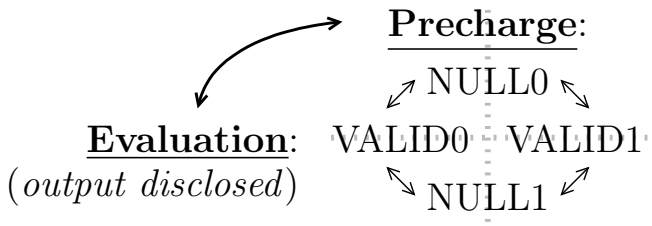


Figure 1.15: protocole DPL

est naturellement protégé contre les attaques par violation de temps de setup 1.16. Par

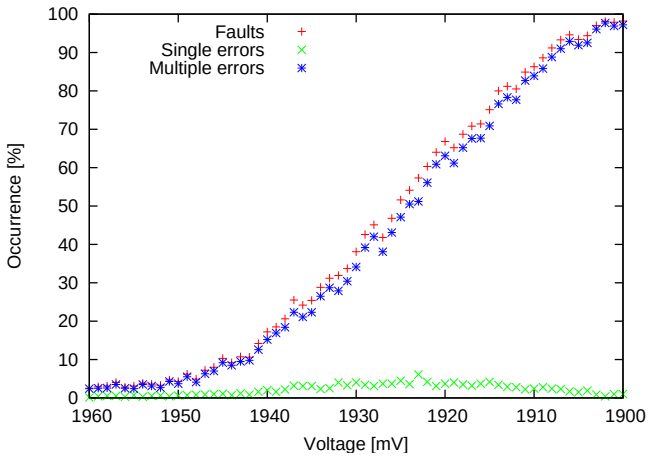


Figure 1.16: occurrence des fautes dans une version wddl d'AES

contre si l'attaquant arrive a injecter une faute symétrique $(1,0) \rightarrow (0,1)$ la logique DPL n'est pas capable d'assurer la protection du circuit mais en pratique il est presque impossible d'introduire ce type de fautes.

conclusion

Dans cette thèse, nous présentons différents aspects d'attaque sur les implémentation cryptographique de l'algorithme de chiffrement AES, ainsi qu'une étude sur les contre-mesures possibles.

La première méthode d'injection de fautes est une nouvelle technique pour injecter des fautes globales cette technique est non-invasive basée sur la violation temps de setup. Nous avons démontré que cette méthode globale permet l'injection de fautes aléatoires dans le circuit . Malgré que nous ne contrôlons pas le temps et le emplacement de l'injection de la fautes, On arrive a obtenir suffisamment de faute pour réaliser les attaques en fautes. Nous avons montré que cette attaque peut être réalisé sur les circuits ASIC et FPGA. On a aussi réalisé une attaque locale sur un microprocesseur Atmel ATmega128 en utilisant un laser.

Nous présentons aussi dans cette thèse, une nouvelle approche pour contré les attaque en fautes basé sur la résilience. La résilience n'impose aucune destruction des secrets dans le cas d'une attaque par faute. Dans une implémentation protégée par résilience, quand une faute est injecté avec succès mais n'a pas de conséquence dans le calcul, le circuit ne présente aucune réaction à la faute par contre un circuit protégé par un système de détection arrête automatiquement le calcul même si la faute n'a pas d'effet. Dans une implémentation résiliente même si la faute est injectée lors du calcul l'attaquant ne peut pas exploiter le résultat a fin d'exécuter une attaque DFA.

Plusieurs méthodes concrètes pour mettre en oeuvre la résilience pour les chiffrements symétriques sont proposées, parmi lesquelles un mode aléatoire de fonctionnement qui convient pour des cartes à puce a faible coût. Nous proposons d'utiliser les logiques DPL comme méthode de protection. Ces logiques protègent simultanément contre les attaques par observation et par perturbation, et sont moins coûteux que la détection basée sur les codes.

General Introduction

Nowadays, digital information is more and more important in our information society and it is necessary to protect such sensitive information using cryptographic algorithms. Those cryptographic systems are often implemented in hardware to increase the throughput of information. When cryptographic systems can be accessed physically no one is sure of the security of transferred information. Indeed attacks that target directly the physical implementations can be devised.

This kind of attacks is known as "side channel attacks" (SCA). They can be classified in two types: active and passive, both of which providing enough information to fully compromise the security. The devices that are concerned are, for instance, smartcards (pay-TV cards, SIMs, *etc.*) or handheld terminals (mobile phones, PDAs, *etc.*).

The first type of SCA is called passive attack and consists in observing physical emanations of the system, like power (Differential Power analysis, or DPA [5]) or ElectroMagnetic field (ElectroMagnetic Analysis, or EMA [6]). An off-line analysis of the physical measurements allow to extract the full key, by correlation or pattern matching techniques.

The second type of attacks is called active attack and consists in injecting faults during the execution of a cryptographic algorithm. Faults attacks can of course be used to obtain DoS (Denial of Services). But the real strength of fault attacks is that they enable an attacker to retrieve secret information concealed within the device [7]. From the knowledge of one or multiple couples {correct ciphertext, faulted ciphertext}, some hypotheses on the secret key can be discarded. This generic attack strategy is referred to as DFA (Differential Fault Analysis). Although active attacks were reported later (in 2001 [8]) than passive attacks (in 1998 [5]), many attacks have been published, which show how very few errors can break even the most secure cryptosystems: The

most astounding results are the Bellcore attacks against RSA, where a single faulty signature may reveal the secret key, and AES, where only two well localized faults can break the cipher.

There are several techniques known for fault injections in a system: The variations of the supply voltage, the clock frequency, the temperature variation, or the irradiation by a laser beam will most probably lead to a wrong computation result that can be exploited to realize DFA.

This kind of attack represents a greater threat for the implementation of cryptographic algorithms such as the AES and RSA than passive attacks.

This thesis concentrates on one specific side-channel namely faults. Here, an adversary induces faults into a device, while it executes a known program, and observes the reaction. The adversary has to tamper with an attacked device in order to create faults, thereby opening the desired side-channel.

The main propose of this work is to validate the feasibility of fault attacks and to implement some countermeasure to protect crypto processors. This thesis work is recorded in three chapters. The outline of the thesis is as follows:

- In chapter **1** we introduce cryptography, side channel attacks and state of art of fault attacks.
- In chapter **2** we give a new method to inject global faults in both ASICs and FPGAs, then we show local optical semi invasive attacks on software implementation of AES.
- In chapter **3** we present fault attack countermeasure and discuss a new method of protection based on resilience, then we study the resilience on WDDL.
- Finally, in chapter **4** we conclude by an overall review on this work and open some perspectives to improve it.

Chapter 2

Physical Attack On Cryptographic Implementation

Standardized cryptographic algorithms are basically secure against algorithmic attacks. But once such algorithms are implemented, either on dedicated hardware or as software on a micro-controller, different physical properties of the algorithm can be observed. Over the years, sophisticated attacks have been developed that enabled attackers to break cryptographic devices by such observations. In this chapter, we introduce the basic principles of cryptographic algorithms and we show how its physical implementation can be exploited.

First, we will describe some general principles, such as symmetric ciphers and public-key schemes. Then we will give an overview of cryptanalytic techniques aimed at breaking the most used cryptosystems. We will focus our attention on active attacks. In practice, we present attack methodologies based on intentional injection of errors during the computation process, describing how we can inject such faults and the most common attacks to exploit faulted results.

2.1 Cryptography

The prefix of the “cryptology” stems from the Greek root crypto that means “hiding”. Cryptology include two branches “cryptography” and “cryptanalysis”. Cryptography provides methods to transform legible information (plaintext) into a form that is protected (ciphertext) with the help of a secret information (cipherkey). Any information related to cryptographic system can be public except the key. While cryptanalysis provides methods to extract the “cipherkey”. Cryptography is used nowadays

in a large variety of domains. Information is now created, transmitted and stored in an electronic form and this sensitive data must be protected from unauthorized use. This can be easily accomplished through cryptographic systems which can be roughly divided into two main classes: symmetric ciphers and public-key algorithms. The symmetric algorithms use the same private key to cipher and decipher, on the other hand, public key cryptography requires a pair of keys private for decipher and public to cipher message .

2.1.1 Symmetric Ciphers

Symmetric ciphers are the oldest and most common algorithms in cryptography. One of the simplest forms is known as the Caesar cipher "reputedly used by Julius Caesar to conceal messages" in which the process is simply one of shifting the alphabet by so many places in one direction or another. Symmetric ciphers accept a plain input text and a secret key and return an encrypted output text.

Usually, the algorithm is made of two distinct processes: a data-path, where the initial input is processed and mixed with the key and a key schedule, which is used to obtain the whole needed key material starting from the secret key. The algorithms are mostly iterative, which means that a few simple operations are repeated for a certain number of times; at the same time, for each iteration (called round) the key schedule computes a round key which will be used only once.

Decryption is computed using the same encryption key: usually, the decryption process is the execution in the reverse order of the inverse functions of the encryption data-path, although in some cases the same algorithm can be used (these are called involution ciphers such as NOEKEON [9]). The key schedule, however, must still provide key material in the reverse order, which means that the key must be completely unrolled before decryption can start. Sometimes, the key schedule is executed before the encryption and the key material is stored into memory for future use; alternatively, if allowed by the cipher structure, it can be executed in parallel with the encryption process (on-the-fly).

Symmetric ciphers are partitioned into two categories. When the input data is processed one bit or byte at a time, then the algorithm is called stream cipher. When the input is a block of few bytes, it is a block cipher. The variety of symmetric block ciphers

available in the literature, in the industry and on the market, is huge. They constitute the simplest way to protect a transmission, which led many parties to implement their own proprietary encryption scheme and keep it jealously secret. In the most ancient one, the robustness of the encryption scheme relies on the secrecy of the key, but also on the secrecy of the algorithm. Such practice is against the Kirchhoff's principle, stating that the security must rely only on the key. If the algorithms specifications leak and become public, chances are that the scheme is no longer secure: this happened with the Content Scrambling System [10], the protection scheme used on commercial DVDs. Luckily, the recent trend is to develop public algorithms, which are therefore given to the community of researchers, who can find possible weaknesses.

Symmetric cryptography normally requires the key to be shared and simultaneously kept secret within a restricted group. It is simply not possible for a person who views the encrypted data with a symmetric cipher to be able to do so without having access to the key used to encrypt it in the first place. If such a secret key falls into the wrong hands, then the security of the data encrypted using that key is immediately and completely compromised. Hence, what all systems in this group of secret key methods share is the problem of key management.

The most common symmetric block ciphers are DES and AES.

2.1.1.1 Data Encryption Standard

As presented in the FIPS standard[1] this algorithm is designed to encipher and decipher blocks of data consisting of 64 bits under control of a 64-bit key of 56 bits of entropy. Deciphering must be accomplished by using the same key as for enciphering, but with the schedule of addressing the key bits altered so that the deciphering process is the reverse of the enciphering process.

[1]Federal Information Processing Standards (FIPS) are publicly announced standards developed by the United States Federal government for use by all non-military government agencies and by government contractors. Many FIPS standards are modified versions of standards used in the wider community (ANSI, IEEE, ISO, etc.) Some FIPS standards were originally developed by the U.S. government. For instance, standards for encoding data (e.g. country codes), but more significantly some encryption standards, such as the Data Encryption Standard (FIPS 46) and the Advanced Encryption Standard (FIPS 197).

A block to be enciphered is subjected to an initial permutation IP, then to a complex key-dependent computation and finally to a permutation which is the inverse of the initial permutation IP^{-1}. Figure 2.1 shows the DES algorithms.

The key-dependent computation can be simply defined in terms of a function f, called the cipher function, and a function KS, called the key schedule [11].

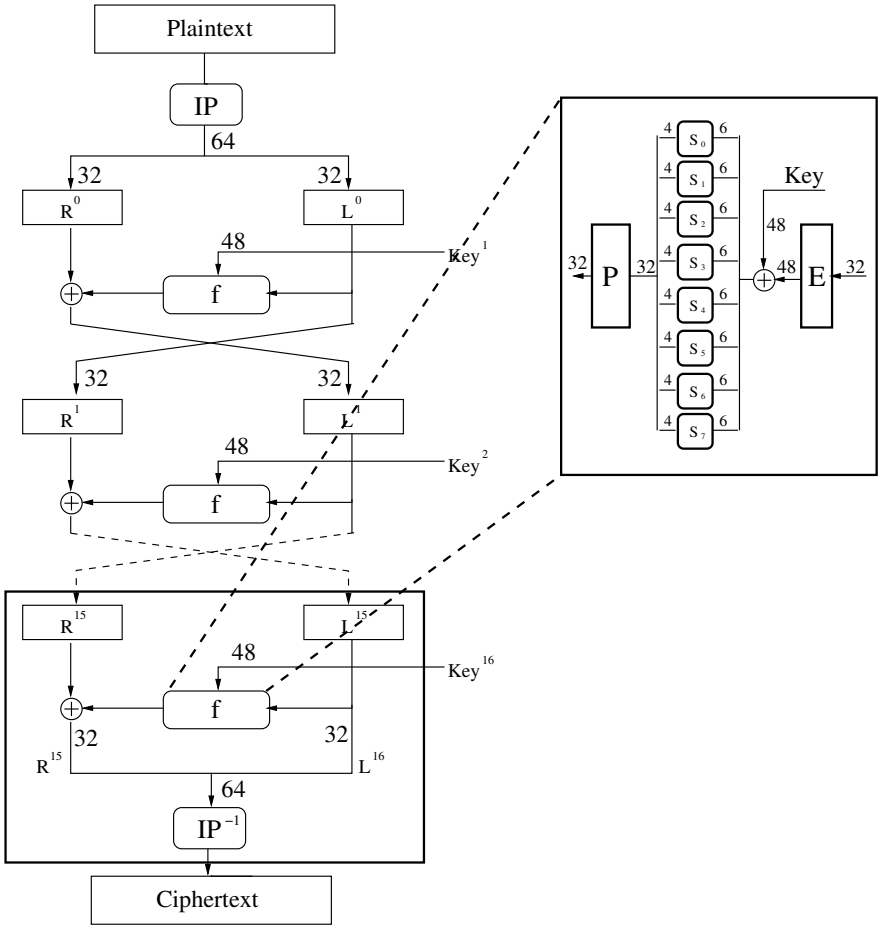


Figure 2.1: Data Encryption Standard

Data can be recovered from cipher only by using exactly the same key used to encipher it. Unauthorized recipients of the cipher who know the algorithm but do not have the correct key cannot derive the original data algorithmically. However, anyone who does have the key and the algorithm can easily decipher the cipher and obtain the original data. A standard algorithm based on a secure key thus provides

a basis for exchanging encrypted computer data by issuing the key used to encipher it to those authorized to have the data. Selection of a different key causes the cipher that is produced for any given set of inputs to be different. But with the increase of the computation speed in new computers this key can be found in few minutes using brute force attack (which means a trial of all possible values of the key). In fact, in June 1997 the DES was cracked by a federation of hackers that were using a network of normal computers and it took 23 hour and 15 minutes [12]. This motivated the need for a more robust encryption mechanism has been performed especially that this algorithm is used in important economic transactions and governmental communications. The first idea was to use 3DES which consists in using a call of DES, then $-DES^{-}1$, and finally DES. But being aware of the weakness of the DES, the NIST[2] made an invitation to tender to work out a new standard.

2.1.1.2 Advanced Encryption Standard

In 2 October 2000, the NIST has chosen between the seven final candidates, and that was the ''Rijndael'' algorithm that won the competition and became the new Encryption standard algorithm AES [1].

AES is an encryption algorithm invented by two Belgians researchers Joan Daemen and Vincent Rijmen. The AES algorithm proceeds by block of 128 bits, and a key of variable length. The length 128, 192, 256 allows a trade off between security and efficiency.

AES is an iterative algorithm, the number of rounds depend on the length of the key, for a 128-bit key length the number of round is equal to 10 rounds, 12 for 192-bit key and 14 for 256-bit key.

Furthermore, AES encryption and decryption are based on four different transformations that are performed repeatedly in a certain sequence. Each transformation maps a 128-bit input state into a 128-bit output state.The transformations are grouped

[2]The National Institute of Standards and Technology (NIST), known between 1901 and 1988 as the National Bureau of Standards (NBS), is a non-regulatory agency of the United States Department of Commerce. The institute mission is to promote U.S. innovation and industrial competitiveness by advancing measurement science, standards, and technology in ways that enhance economic security and improve quality of life.

in rounds. The rounds are slightly different for encryption and decryption. These transformations are described in the Figure 2.2.

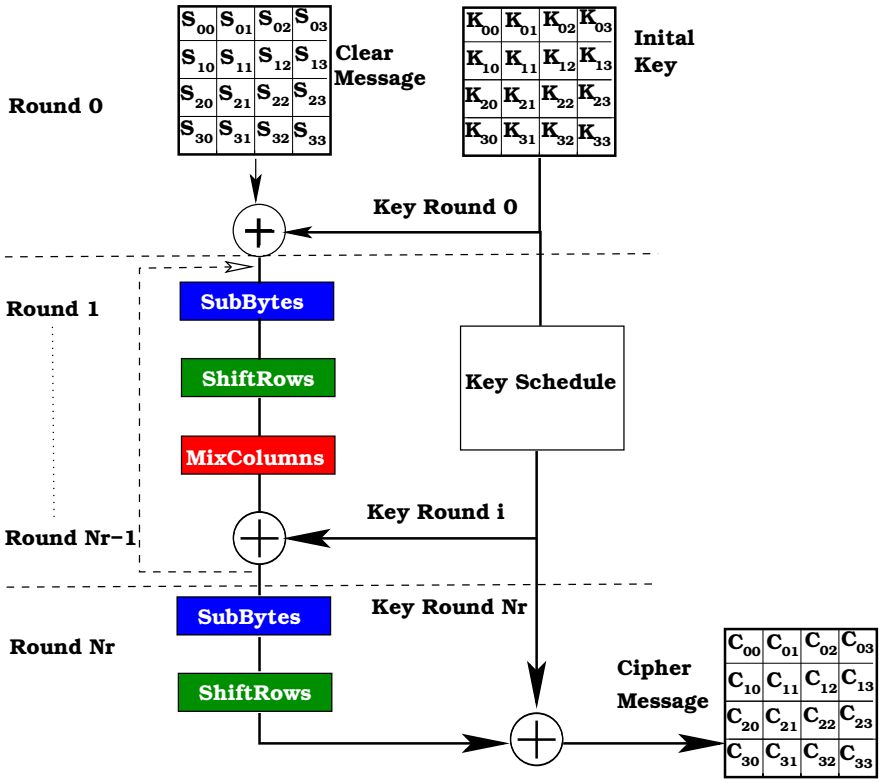


Figure 2.2: AES encryption

In the AES, the 128-bit data block is considered as a 4×4 array of bytes called state matrix. The algorithm consists of an initial data/key addition, 9 full rounds (when the key length is 128 bits), and a final (modified) round. A separate key scheduling module is used to generate all the sub-keys, or round keys, from the initial key; a sub-key is also represented as 4×4 array of bytes. The full Rijndael round involves four steps:

1. a non-linear substitution that is applied on the state matrix: « SubBytes ».
2. A circular bytes permutation within the same line: « ShiftRows »
3. A multiplication in $GF(2^8)$ for each column: « MixColumns »

4. A simple XOR with the output of the key register: « AddRoundKey ».

SubBytes Transformation

The SubBytes transformation replaces each byte in a block by its substitute from an S-box as shown in figure 2.3. The Sbox is an invertible substitution table which is constructed by a composition of two transformations:

- First, each byte $A_{i,j}$ is replaced with its reciprocal in $GF(2^8)$ (except that 0, which has no reciprocal, is replaced by itself).
- Then, an affine transformation f is applied.

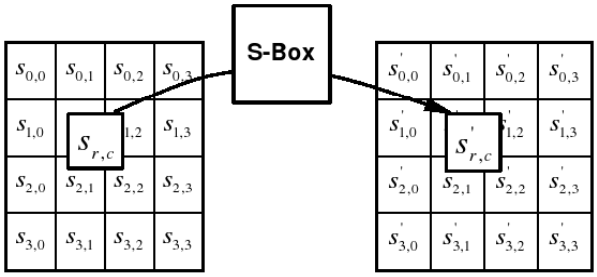


Figure 2.3: SubBytes transformation [1].

The S-box is usually implemented as a look-up table consisting of 256 entries, each entry is 8 bits wide, but it also can be computed "on-a-fly".

ShiftRows Transformation

Next comes the ShiftRows transformation, each row in a 4×4 array of bytes of data is shifted 0, 1, 2 or 3 bytes to the left in a round fashion, producing a new 4×4 array of bytes as shown in figure 2.4.

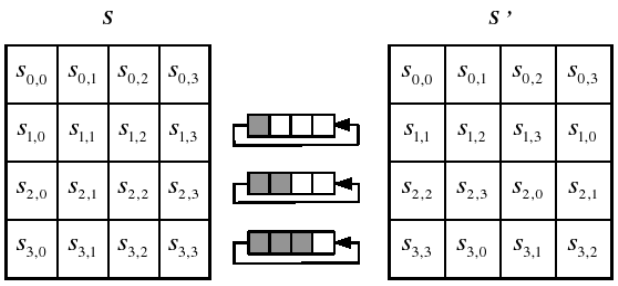


Figure 2.4: ShiftRow transformation [1].

MixColumns Transformation

The MixColumns transformation, operates on each column individually as shown in figure 2.5. Each byte is mapped into a new value that is a function of all four bytes in the column. The transformation can be defined as a matrix multiplication on the state, each column is treated as a polynomial over $GF(2^8)$ and is then multiplied modulo $x^4 + 1$ with a fixed polynomial a(x):

$$a(x) = (\texttt{0x03})x^3 + (\texttt{0x01})x^2 + (\texttt{0x02})x + (\texttt{0x02})$$

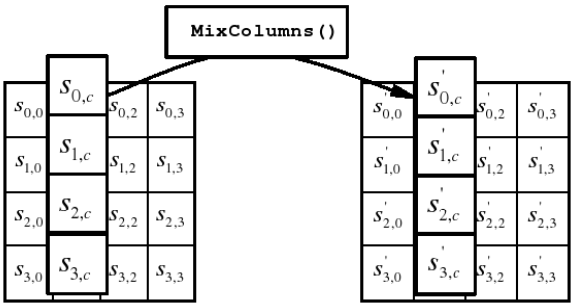


Figure 2.5: MixColumn transformation [1].

AddRoundKey Transformation

The final transformation is AddRoundKey, it simply XOR-es the result with the sub-key for the current round as shown in the following figure 2.6

The Key Schedule

Each round accepts a round key derived from the initial secret key by means of the Key Schedule process as described in figure 2.7.

- "Rotword" operation takes a 32-bit word and rotates it by eight bits to the left.
- "Subword" operation uses S-Box table to replace each byte of the columns.
- "Rcon" is a table of constants depending on the round number.

More precisely, if k^0 is the secret key and k^i is the i^{th} round key, then Key Schedule computes $k^i = KS_i(k^{i-1})$ as a function of the previous round key. The functions KS_i themselves depend on the round and on the size of the key. However, the KS_i do not differ much from each other, and for a key size of 128 bits they are all identical.

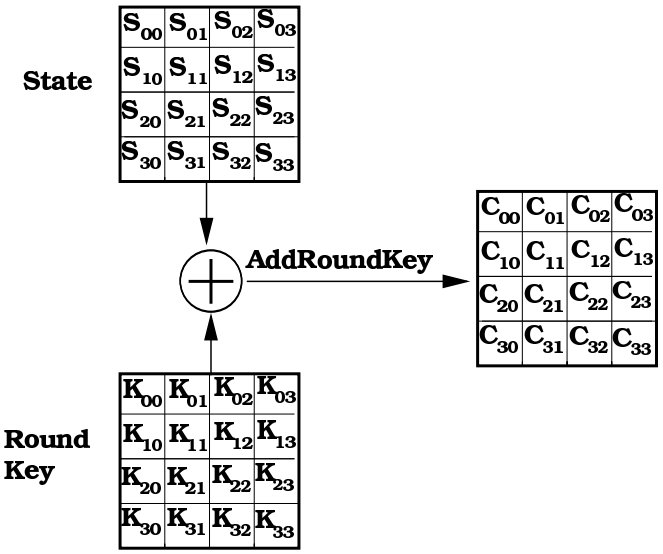


Figure 2.6: AddRoundKey operation

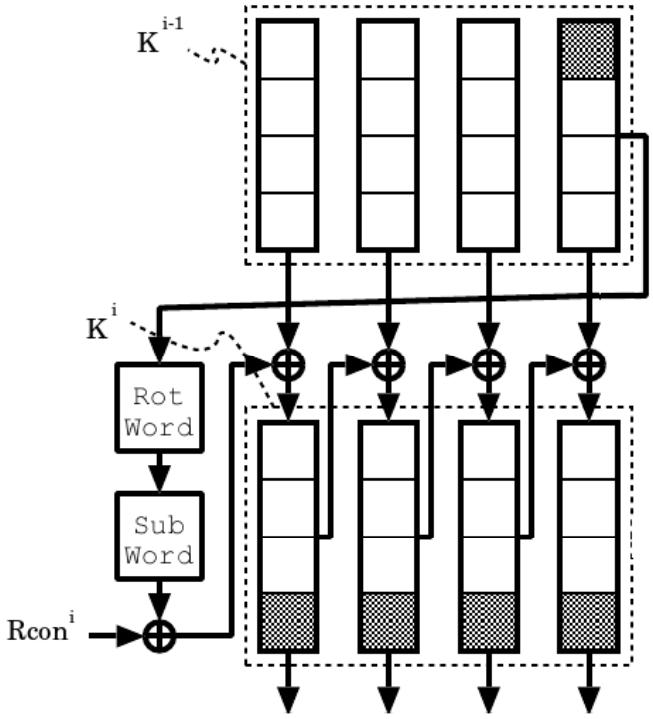


Figure 2.7: AES Key Schedule

2.1.2 Asymmetric Cryptography

Symmetric ciphers are highly efficient from the computational point of view, but they have an important issue: key management is extremely inefficient. First of all, the secret key must be exchanged securely and the encryption of the secret key is not an option, since it would represent the same problem again and again. If the key gets leaked, then it should be revoked and new one should be shared. In addition to this, each different secure link requires its own key: every pair of users should be assigned a unique key, known only to the authorized owners, which means that the overall number of keys would grow exponentially. Even if the key is shared within a single group of people, there would be no way to identify correctly the sender within the group, or have a subset of authorized receivers.

Public-key (asymmetric) cryptosystems are the solution to these issues. Each user has a pair of keys: a secret key (the private key) and a second public key. The keys of each pair are related to each other: it is easy to compute the public key from the private key, but the inverse is computationally infeasible. Thus, each user can generate a (random) public key, which will be posted publicly and then compute his secret key; other users will be able to know only the public key, and will be unable to invert the process to obtain the user's secret key. What is encrypted with one of the keys, can be decrypted only using the other: for instance, if we use our public key to encrypt a message, it will be decrypted only by our private key. This scheme allows to provide some very important properties for secure communications:
Confidentiality: it is the guarantee that the message will not be read by an unauthorized user; it can be achieved by encrypting the message with the public key of the receiver, thus we can guarantee that only his private key will allow decryption.
Authentication: it is the proof of the sender's identity, certifying that the sender of the message is actually the one who claims to be; it can be achieved by encrypting the message with the private key of the sender. It will be decrypted only with the public key of the sender, revealing his identity.
Non-repudiation:it is strictly related to the previous concept and means that the sender can not deny having sent the message. It is based on the assumption that the private key is known only to its legitimate owner and that it can not be inferred from the public key. Thus, the message could not be sent by any other user.
Integrity: it guarantees that the message was not modified or tampered with, and it

is exactly the message that was transmitted at the source. It is usually achieved by attaching a digest of the message itself, usually the result from a commonly shared hashing algorithm; then, the digest only can be encrypted with the sender's private key. At the reception point, the receiver computes the digest of the message; then he decrypts the digest he got by using the sender's public key and compares the results, proving that they were not modified. The system relies on the security of the hashing algorithm, i.e., the complexity of creating different messages with the same digest (collision attacks).

2.1.2.1 RSA

The most used public-key crypto system is RSA [13], based on modular exponentiation in finite ring $\mathbb{Z}_n$. Encryption is computed by exponentiating the message with the secret or the public key, decryption is computed again by exponentiation with the other exponent. RSA is based on the problem of factoring the product of two large primes.

Algorithm 3: RSA Algorithm

1. Choose two distinct large primes p and q of the same bit length.
2. Compute $N = p.q$ as the RSA modulus.
3. Let $\varphi(N) = (p-1).(q-1)$ denote Euler's totient function.
4. Choose a public key $3 < e < \varphi(N) - 2$ coprime to $\varphi(N)$
5. Compute as the secret key the unique integer $1 < d < \varphi(N)$ such that $e.d = 1 \, mod \, \varphi(N)$.

- Encryption a message $M \in \mathbb{Z}_n$ Compute $C = M^e \, mod \, N$.
- Decrypting a ciphertext $C \in \mathbb{Z}_n$ Compute $M = C^d \, mod \, N$.
- Signature of a message $M \in \mathbb{Z}_n$, the signature S is created as $S = M^d \, mod \, N$.
- Verification of a signature $S \in \mathbb{Z}_n$ of a message M $S^e = M \, mod \, N$.

2.1.2.2 Elliptic Curve Cryptography

Another method to define public key algorithms is to use elliptic curves [14]. In contrast to RSA, computations take place in a finite additive group. An elliptic curve E over field **K** is defined by the Weierstrass equation: $E : y^2 + a_1xy + a_3y = x^3 + a_2x^2 + a_4x + a_6$. The set of points (x,y) $\in K^2$ as a solution of the equation E, together with the point at infinity O form and additive abelian group. The point O is the neutral element of the group. It is denoted as E($\mathbb{F}_p$). The group operation is called addition for two distinct points and doubling otherwise. An elliptic curve group operation consists of many field operations. For a field $\mathbb{F}_p$ with a characteristic other than 2 the equation E can be simplified to $E : y^2 = x^3 + ax + b \;\; a, b \in K$. In order to encrypt message using ECC we have to chose and elliptic curve E defined over a prime filed $\mathbb{F}_p$ such that the

order of E is divisible by a large prime q, then we chose a base point P on E of order q. Obviously the choice of E and P is crucial for the security of the system. The order of the base point P must be a large prime.

Algorithm 4: ECC Algorithm

1. Choose an elliptic curve **E** defined over $\mathbb{F}_p$.
2. Choose a base point **P** on **E** of order q.
3. Choose the secret key k $\in \{1, 2, ..., q-1\}$
4. Compute public key **Q**=k.**P** on **E**.

Encryption

- Choose a random session key r $\in \{1, 2, ..., q-1\}$
- Compute the two points **R**=r.**P**=(x_1, y_1) and r.**Q**=(x_2, y_2).
- Compute $C = x_2 + M$ where $M \in \mathbb{Z}_p$ is the message to be encrypted.
- Send out (**R**,C) =(x_1, y_1, C)

Decryption

- Compute $(x_2', y_2') = k.\mathbf{R}$
- Recover $M = C - x_2'$

2.2 Smartcard Architecture

The trend for miniaturisation and portability of every computing device has led to the development of smartcards which is a small computing device as large as a credit card and equipped with processing unit, some memory and dedicated processors for cryptographic computations. The smartcard has a microprocessor embedded in it that, when coupled with a reader, has the processing power to serve many different applications. By the means of cryptographic algorithm, smart cards make personal data available only to the appropriate users.

Since its commercial launch in 1992, the smart card has taken full advantage of the miniaturization of circuits and, although the maximum area of the chip is standardized to 25 mm^2, the circuits have evolved to more computing capabilities. Smart cards are used in combination with terminals; such as bank cards, prepaid phone cards or SIM, whose terminal is the mobile phone. Most of the time they are links in the chain of custody of a larger system. The industry has set standard ISO/IEC 7816 to facilitate interoperability of the smartcard and FIPS-140 to ensure the security of this component of cryptographic modules.

The integrated circuit as shown in figure 2.8, may contain a microprocessor (CPU) capable of processing this information and specialized cryptographic hardware that uses algorithms such as RSA, 3DES or AES. Smartcard is mainly used as means of personal identification(identity card ,SIM card), payment service(credit card) or for prepaid services (phone card, pay-TV).

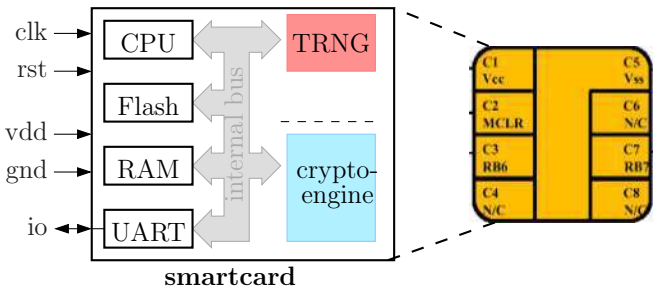


Figure 2.8: Smartcard internal architecture

The smartcard have in general five pin-out:

- Vdd is the supply voltage that drives the chips at 5V, 3V or 1.8V.
- Gnd is the substrate or ground reference voltage against which the Vdd potential is measured.
- Reset is the signal line that is used to initiate the state of the integrated circuit after power on.
- IO input/output connector. This is the signal line by which the chip receives commands and interchanges data with the outside world.

- CLK is the clock signal is used drive the logic of the IC and is also used as the reference for the serial communications link.

Smartcards are extremely customizable and can be applied to a large variety of application domains, but on the other hand they are extremely vulnerable to physical attacks based on side channel information leakage. Next section will be dedicated to present physical attacks known as Side Channel attack "SCA".

2.3 Side Channel Attack

Cryptosystems are usually implemented as embedded devices or software algorithms. Even if the cryptosystem is secure and without any apparent flaw, its implementation may reveal some useful information about the secret key in an indirect way. Kocher in [15] and in [16] published two novel attack techniques, witch exploit side channel leakage of cryptographic devices. Computation requires time, consumes power and causes electromagnetic radiations: all these are possible sources of information related to secret key. These techniques are rather powerful, since they allow to reduce the complexity of an attack to several orders of magnitude, on the other hand, they require physical access to the device to collect the necessary measurements, while the computation phase of the attack can be done off-line.

2.3.1 Timing Attack

The timing attack was predefined opposed by Paul Kocher in 1996 [15]. The simplest concept of timing attacks is that the attackers exploit the observed different execution time between different instructions to see if some specific instruction was executed or skipped by Mich the attackers can extract the secret key.

Consider the flow of any program, programs always spend much time in comparison and conditional-jump. Each condition leads to distinct path. Therefore, if the attacker inputs different messages into a program, the time consumed will be slightly different for different inputs. Thus, the attacker can observe the variance in execution time, and then obtain the secret key. For example,In RSA the execution time for the square-and-multiply algorithm used in modular exponentiation depends linearly on the number of '1' bits in the key. While the number of '1' bits alone is not nearly enough information to make finding the key trivially easy, repeated executions with

the same key and different inputs can be used to perform statistical correlation analysis of timing information to recover the key completely.

2.3.2 Power Analysis

Cryptographic devices are implemented using semiconductor logic gate, which are constructed out of CMOS. The electron flow across the silicon substrate when charge is applied to a CMOS's gate consumes power and produces electromagnetic radiation. Further more, most power is consumed when the gate is in transition. This power variation can be recorded as a trace of the executed data.

Simple Power Analysis

Simple power analysis [16] involves directly observing a device power consumption. When the instruction processes different secret values, the corresponding power consumptions is different for each value. If the difference is discernable, then the attacker could analyze these power consumptions of instruction and extract the involved secret key. The SPA cannot only analyze the impressible instructions, but also reveal the sequence of instructions executed, it can be used to break cryptographic implementations in which the execution path depends on the data being processed such as naive implementations of RSA, whose electromagnetic trace is shown in Figure 2.9.

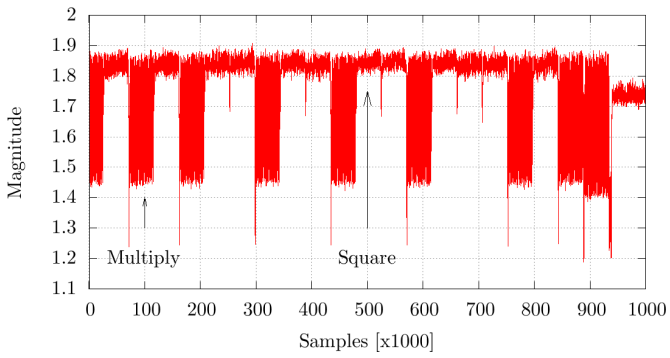


Figure 2.9: SPA on RSA implementation

Differential Power Analysis

Differential power attack "DPA" [16] exploits biases in the varying power consumption of microprocessors or other hardware while performing operations using secret

keys. DPA attacks involves signal processing, that can extract secrets from measurements which contain too much noise to be analyzed using Simple Power Analysis. DPA can attack on either first or the last round of Symmetric key algorithm and requires the knowledge of either the plaintext or the ciphertext. The side channel exploited is the difference between the power consumed by a single gate when its output rises or falls. The attack exploits the fact that in CMOS logic, a gate only dissipates energy when it change states.

DPA uses a model of the attacked device, which is used to predict several values of the side-channel output. Then a set of power traces is collected by computing many encryptions. The power traces are then partitioned into two sets, according to the model. If the model and the partitioning are good, then a difference should emerge from the analysis, revealing information on secret key as shown in Algorithm 5.

In order to mount DPA an attacker proceeds in two phases. First, a large number of power consumption traces for different plaintexts are recorded. The second step consists in extracting the secret key by applying statistical techniques. The attacker carefully uses the measuring equipment to reduce external noise and repeatedly process the same data a sufficient number of times, then average the correct data to reduce the algorithmic noise.

The basic concept of DPA is that the plaintexts (or the ciphertexts) are divided in two parts according to the related key value guessed. If the related key value is guessed correctly, the power consumptions of the corresponding plaintexts (or the ciphertexts) can be divided into two parts correctly . The average power consumption of one part will be slightly more than the other. Then, some obvious biases in the total differences of these two parts will appear in the total difference trace. If the key value is guessed incorrectly, traces are randomly spread into both set which the averages are very close to each others. Hence, the total difference of these divided two parts will nearly be null in the total differential trace as shown in Figure 2.10.

Algorithm 5: DPA Algorithm

1. Encrypt many randomly selected plaintexts "M"
2. Collect their corresponding power traces "T"
3. Choose selection function "L"
4. Make guess on the key "K"

For (i = 0 to keylength)
Collect the average power trace: $T(\{L_i(M,K)=0\})$
Collect the average power trace: $T(\{L_i(M,K)=1\})$
Compute the differential trace: $D_i = T(\{L_i(M,K)=1\}) - T(\{L_i(M,K)=0\})$
If D_i has a positive spike
$K_i = 1$
Else
$K_i = 0$
End For
Output: Cipherkey

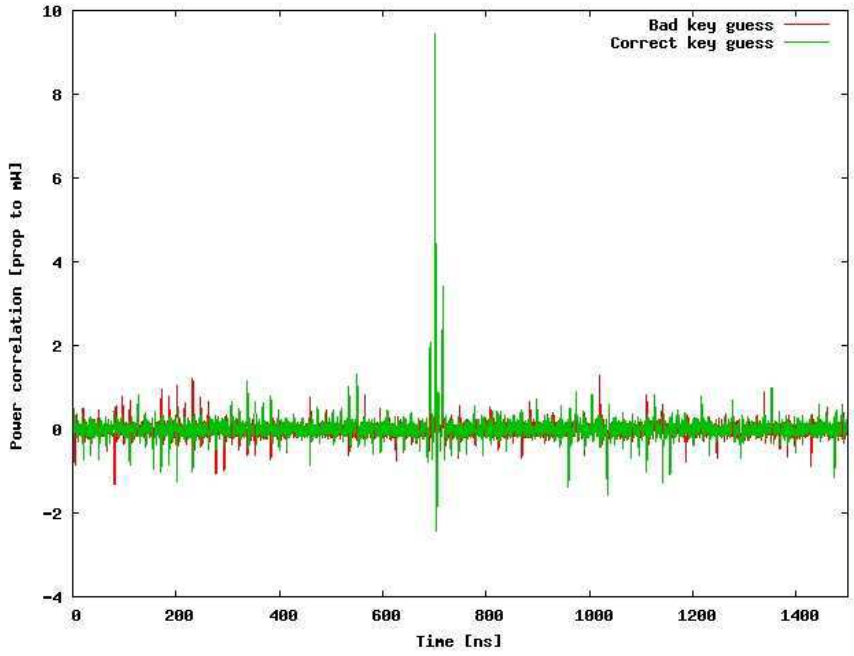


Figure 2.10: DPA on DES: Bad and correct subkey

2.3.3 Electromagnetic Analysis

Electromagnetic analysis "EMA" as presented by Quisquater and Samyde [17] are very similar to DPA, but they exploit the information leaked by Electromagnetic radiations.Since any electrical current flowing through a conductor induces electromagnetic (EM) emanations, it seems natural to look for the same phenomenon in the vicinity of a semiconductor. In fact, when CMOS gates consume power, the current pulse cause variation in the EM field surrounding the chip. Power consumption of the device varies while data are being processed, so does the em field and one may legitimately expect to extract secret information from a relevant EM analysis. Quisquater and Samyde showed that it is possible to measure the Electromagnetic radiation from a smart card as shown in Figure 2.11. Their measurement setup consisted of a sensor which was a simple flat coil, a spectrum analyser or an oscilloscope and a Faraday cage. In their article they introduced the terms Simple EMA "SEMA" and Differential EMA "DEMA".This attacks exploits correlations between secret data and variations in power radiations emitted by the cryptographic devices.

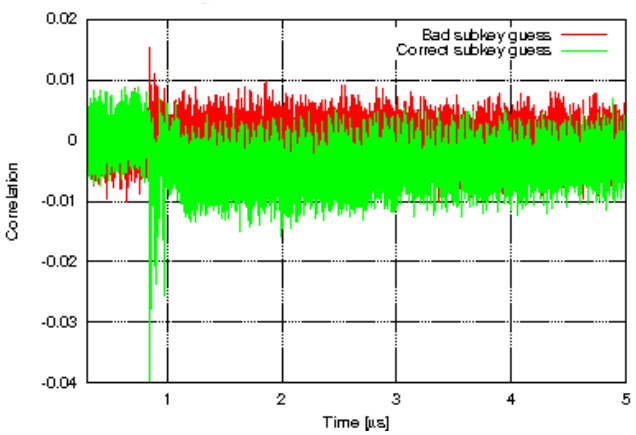


Figure 2.11: EMA on DES: Bad and correct subkey

2.4 Fault Attacks

Fault attacks exploit the physical properties of devices. Theoretical fault attacks are based on fault models. In this section, we will give an overview of actual physical methods to induce practical faults. This will show that there are numerous ways to induce faults into physical devices as shown in "The Sorcerer's Apprentice Guide to Fault Attacks" [18].

2.4.1 Power Spikes

Embedded system needs power supply which is provided externally. An adversary with laboratory equipment is capable of both tampering with the power feeding as well as measuring power consumption (thus enabling power attacks).

It has been defined by standards [19], that a smartcard must tolerate a certain variation in the power supply VCC of $\pm 10\%$ of the standard voltage. However, if the variation is significantly higher than 10%, the embedded system is no longer required to work properly. In fact, short massive variations of the power supply, which are called spikes, can be used to induce errors into the computation of the smartcard. Variations in supply voltage during execution may cause a processor to misinterpret or skip instructions. These methods do not require a modification of the device itself but provoke faults by modifying the working conditions. Power spikes are cited as standard way to induce faults by various authors [20, 21].

2.4.2 Clock Glitches

Similar to the power supply, an adversary is able to tamper the clock signal. The clock signal CLK has to meet various electrical constraints. For instance the voltage for the CLK signal at high level V_{IH} may range from 0.7 VCC to VCC and the low level V_{IL} from 0 to 0.5 VCC, where VCC is the power supply voltage. The embedded system must also work properly with deviations of clock rise and clock fall times of 9% from the standard period clock cycle[19]. . The the adversary may provide the embedded system with an external clock signal, which incorporates short massive deviations from the standard voltage, which are beyond the required tolerance bounds. Such signals are called glitches.

Glitches can be defined by a huge range of different parameters [22], and they can be used to both induce memory faults as well as causing a faulty execution behaviour (code change attacks). Hence, the possible effects are the same as for spike attacks. However, clock-signal glitches are the simplest and most practical attacks as presented by Agoyan and al. [23]. Details about glitches can be found in [21, 24, 25]

2.4.3 Optical Attack

If an embedded system is depackaged, such that the silicon layer is visible, it is possible to use a laser cutter (red or green laser) or focused UV light in order to change the state of internal signals (transient faults) or even destroy them (destructive faults) [22]. This allows to induce a great variety of faults. Memory cells used for EEPROM memory and semiconductor transistors have been found to be sensitive to coherent light, i.e lasers, and ionizing radiation such as cosmic rays. This is due to photoelectric effects and already works for white light.

By using lasers or focused ultraviolet light, EEPROM bits can be erased, as shown in [2]. This happens if the photon energy of the applied kind of light exceeds the semiconductor band gap. Modern green or red lasers can be focused on relatively small regions of a chip, such that faults can be targeted fairly well. Agoyan et al. showed in there article [26] that reproducible single-bit faults on SRAM, often considered unfeasible, can be obtained.

2.4.4 Electromagnetic Perturbations Attack

Changes in the external electrical field have been considered as a possible method for inducing faults into embedded systems [24, 27]. Here, faults are induced by placing the device in an Electro-Magnetic field, which may influence the transistors and memory cells. However, the main problem using such an approach is to target specific bits or variables stored in the card.

The use of Eddy currents to induce faults has been motivated by electromagnetic analysis of smartcards, which has been proposed as a passive side-channel for these devices. If the passive probing antenna is subjected to an alternating current, the resulting Eddy currents can be strong enough to interfere with the operation of a transistor or memory block.

The property exploited for fault attacks is the fact that Eddy currents can modify the number of electrons inside a transistors oxide grid [28]. This changes the threshold voltage of the transistor such that it cannot be switched during the electromagnetic perturbation. Depending on the actual transistor, this can be used to ensure that a memory cell contains the value 0 or 1. This effect can be used to induce transient or permanent faults.

Inducing Eddy currents does not require to depackage a chip, hence, attacks can easily be carried. However, this requires that an adversary knows the layout of an attacked device in order to control the targeted area to attack. It has been shown by Samyde in [27] that Eddy currents can be used to induce faults very precisely, such that individual chosen bits can be set or reset.

2.4.5 Definition of Fault Model

Fault attacks are based on tampering with a device in such a way that the device performs abnormally. The reaction of the device, can be a faulty result or an error message, or some form of safety or urgency behaviour as a destruction of the device. An adversary can take advantage of the induced modification to find the secrets hidden in a device. As cryptographic algorithms are generally public according to the Kerckhoffs's principle, an adversary has a minimum knowledge of the algorithm implementation and thus, can assume a set variables witch depends on the secret key. This allows the adversary to determine what kind of error will provoke the reaction which may be observable. For example if a single bit in the secret key is flipped during an attack, and if the device does not detect this fault, a faulty result with a specific pattern is returned. By comparing this faulty result with the correct one, an adversary might be able to deduce one bit of the secret key. An adversary may also target the flow of operations, such that certain operations are repeated or skipped. To achieve and exploit a desired effect, he needs to have the knowledge about how a certain physical attack will affect the logical flow of the attacked algorithm.

There has been a large number of different fault attacks in the literature [7, 29, 30, 31]. They differ by the faults models which is a set of different parameters like the fault type (transient vs permanent), the number of bits affected, the probability to get an exploitable fault, the duration of the fault, the location of the fault, the time when

the fault is applied and the preparation work that has to be done. We propose three main types of parameters which define the faults models:

- **Spatial parameters:** The stress is applied on the whole device “global faults” or a small region “local faults”.
- **Temporal parameters:** The time when the fault occurs can be fully synchronized or completely random.
- **Number of affected signals:** We differentiate between a “single faulty bit”,“few faulty bits” e.g. in the same Byte “single faulty byte” or random number of faulty bits”multiple faults”.

2.4.5.1 Transient Faults

This type of fault corresponds to a fault which is short-lived, such that after a given amount of time, the effect vanishes and the correct value is present again. This type of fault induces "soft errors" in the device. It is generally assumed that during the decay of a transient fault, there are no intermediate states, i.e., there is only a unique faulty value and a correct value. If a transient fault occurs in a memory block ”RAM”, the fault can be memorized. In this case the fault impact of the modified variable lasts until that variable is explicitly overwritten. For instance after a reset of the device the fault can disappear as the volatile memory can be reinitialized.

2.4.5.2 Permanent Faults

Permanent faults modify definitively the behavior of the device. They are also called destructive as once injected, there is no way to return to the initial state. This may be caused by a fault injection in a non-volatile memory ”ROM” or the cutting of a wire inside the chip by means of a laser cutter or a Focus Ion Beam (FIB) machine. This fault are equivalent to Stuck-At faults which are the fault models used for the ASIC integrity test after manufacturing.

2.4.6 Fault Attack on AES

2.4.6.1 Blömer and Seifert

In [7],Blömer and Seifert present an attack that allows the adversary to retrieve the whole key when the block length is greater than or equal to the key size “128 bits”.

The attack consists in encrypting a null block, i.e., a plain text where each bit is '0'. Thus, after the initial key addition, the temporary state stores the whole encryption key: $s_{ij} = 0 \oplus K_{ij} = K_{ij}\ for\ i, j\ \in \{0, 1, 2, 3\}$ Before computing any other operation (namely, the SubBytes), the attacker tries to set a specific bit S_{ij}^l to 0. After that, the encryption process can continue without any other interference. If $K_{ij}^l = 0$, then setting the corresponding S_{ij}^l value to 0 does not affect the final result, which is still the correct encrypted output; on the other hand, if $K_{ij}^l = 1$, resetting the bit to 0 actually affects the temporary state and results in an incorrect ciphertext.

Even if the cryptographic device is able to detect the corruption and resets itself, the adversary knows that the fault has an impact, thus revealing the value of the bit key K_{ij}^l. In conclusion, the correct encryption key can be found with only 128 bit faults, provided that they are precisely located. The attack, however, can also be applied when the key is longer than the data block. Once the first 128 bits of the key have been recovered using the above technique, the encryption process can be simulated up to the next AddRoundKey operation in Round 1. Hence, we can choose the plaintext such that the state consists of all zeros before AddRoundKey is applied. The round key is added to the state, then the attacker tries to reset S_{ij}^l to 0 and completes the encryption. Similarly, if the result is correct, the corresponding bit of the key is 0; if an error is detected at the output, then that specific bit of the key is 1. This attacks is also known as safe error attacks.

2.4.6.2 Giraud

In 2003, Christophe Giraud proposed a couple of attacks against AES [29] . In the first attack, the model of the fault is restricted to the single-bit fault . To find the key the fault must occur just before the subByte of the last round of AES, the attacker chooses a message and enciphers it twice, one time to get the correct ciphertext C and then a faulted ciphertext C^* and i the faulted byte number. If one bit error is injected, we can denote A the correct byte and A' the faulted one and we have the following equation:

$$\begin{cases} \delta &= C^i \oplus C^{*i}\,, \\ \delta &= (SubBytes[A] \oplus K^{10}) \oplus (SubBytes[A'] \oplus K^{10})\,. \end{cases}$$

Next step is to do an exhaustive search for the couple of (A, A') that match the two conditions:

1. $\delta = SubBytes[A] \oplus SubBytes[A']$

2. $A \oplus A' = 2^i with\ i \in \{0, \ldots, 7\}$

The exhaustive search will allow to compute a list of possible Keys bytes $K^i = SubBytes[A] \oplus C^i$. Therefore, each pair of (A, A') will generate one possible key that we put in the list α of possible keys if $|\alpha| > 1$ repeat the same operation with new message to build a new list of possible message β then take the intersection of the two sets $\alpha \cap \beta$ and repeat the operation until we have only one candidate left. With probability of about 97%, three faults are enough to recover one byte of K^10.

Algorithm 6: Giraud one bit DFA algorithm

Input : C, C^*
Output: $K^i\ where\ i\ is\ the\ faulted\ byte\ \in \{0, \ldots, 15\}$

```
Create two empty set α and β
Compute δ = C^i ⊕ C^{*i}.
while |α| ≠ 1 do
    Clear β
    Look up for A and A'
    if SubBytes[A] ⊕ SubBytes[A'] = δ and A ⊕ A' = 2^j with j ∈ {0,...,7} then
            β ← SubBytes[A] ⊕ C
        if |α| = 0 then
            α ← α ∪ β
        else
            α ← α ∩ β
        end if
    end if
end while
```

In the second attack, the injected error may affect a whole byte. The attacker addresses the Key Schedule unit, in order to inject the fault, at first, in the last word of the penultimate round key; those ciphertexts which are not suitable for the first step of the attack are discarded and a new attempt is made. By comparing the correct and corrupted results, it is possible to detect the position and actual e_j value of the fault by solving the following equation in the unknown e_j : $C_k \oplus C_k^* = SubBytes(K_j^9) \oplus SubBytes(K_j^9 \oplus e_j)$

Then, solving the equation in K_j^9 allows to identify the key candidates; the correct value can be found by cross-comparing with the results from another faulty ciphertext. Using only 32 faulty ciphertexts (on average), the last 4 bytes of the penultimate round

key (from K^9_{12} to K^9_{15}) can be detected. The process is then repeated by attacking the last word of K^8. The only difference is that it might be more difficult to retrieve the error value and hence more faulty ciphertexts might be required. However, the values $K^8_{12} - K^8_{15}$ can be recovered with few ciphertexts (the author claims 19 ciphertexts for a 99% chance to succeed). These values can be used to compute $K^9_8 - K^9_{11}$. Finally, injecting the fault in the temporary state at the beginning of the 9th round allows to guess the error value; the error value can be used to recover the affected state bytes (by exhaustive search), which in turn can be used to simulate the cipher and retrieve the remaining key values. The author claims that this attack can break the AES-128 by using less than 250 faulty ciphertexts.

2.4.6.3 Chen and Yen

In [31], Chen and Yen published their attack against AES. The attack is very similar to the second attack of Giraud [29], although they claim a reduced complexity and hence less ciphertexts to recover the key. First, a fault in injected into one of the last four bytes of K^9 , which is the penultimate round key; this means that, due to the Key Schedule routine, the last round key will have 5 corrupted bytes which will appear in the final output. This allows to collect some condition statements on some bytes of the last round key; by intersecting a few statements coming from different faults, the exact values of the key can be found. Then, an error is injected in the antepenultimate round key (namely k^8). Only three bytes are useful targets; when those bytes are hit more bytes of the key material can be guessed.

So far, the attack is just like that proposal by Giraud. Here the authors inject again the error in the antepenultimate round key, but in a different position (namely, in the penultimate word of the antepenultimate key) This leads to 6 or 7 corrupted bytes in the final output, that can be used to guess 8 additional key bytes, as in the previous steps. In conclusion, 13 bytes of the final round key can be guessed; the remaining 3 bytes can be recovered without much effort by exhaustive search. The initial secret key can be revealed just by computing the Inverse Key Schedule. In conclusion, the attack is able to break the AES-128 key by using less than 44 fault injections, when the fault can be induced with accuracy.

2.4.6.4 Dusart,Letrourneux and Vivolo

In [32], the authors presented a very simple and effective attack to retrieve the last round key. Since the AES Key Schedule is invertible, it is then possible to compute the initial secret key going backwards. Moreover, the fault model is highly practical, since the initial assumption is that a random error is injected before the last MixColumns operation. Timing must be accurate, while location knowledge is not essential, since it can be easily gathered by comparing a pair of faulty and correct ciphertexts: in the corrupted result, only 4 bytes differ and their location depends on the fault location. A random byte fault injected before the MixColumns is spread immediately to the other bytes of the same column. The subsequent key addition and the final round exploit only local operations, modifying the value of bytes independently. The only exploitable operation is the SubBytes, since the non linearity allows to make some assumptions on the error value.

The attack hence relies on solving a system of equations describing the final steps of the algorithm, the equations depend on the values of the state, of the round key and of the injected error: $SubBytes(x) + SubBytes(x + c.\epsilon) = \epsilon'$ where c is one of the MixColumns coefficients and depends on the specific byte we are considering. By solving each equation in ϵ, we obtain some sets of possible error values, which can be intersected, thus obtaining a smaller set. Then, for each error value, the corresponding key bytes can be guessed: it can be proved that the number of possible key byte values ranges from 2 to 4, depending on the fault value. However, after few iterations, intersecting the set solutions allows to retrieve 4 bytes of the key quickly; the complete AES-128 key can be recovered by using less than 50 ciphertexts.

2.4.6.5 Piret and Quisquater

In [30], Piret and Quisquater describe a further attack on AES. In the initial description, they follow the same path as Dusart [32]. Injecting the fault between the last and the penultimate MixColumns, it is possible to obtain a set of candidates for 4 key bytes. In principle, a couple of well located faults allow to find the unique correct candidate.

They present, however, a further attack: to analyze the faulty ciphertext when the error is injected before the penultimate MixColumns. When injecting the error, the subsequent MixColumns spreads the infection over the whole column, thus affecting

4 bytes as shown in Figure 2.12, this sequence of events is shown in Figure 2.12, the non-linear substitution layer (SubBytes) is computed, then the bytes are shuffled according to the ShiftRows operation. At this point, we still have only 4 corrupted bytes, but they are all scattered over different columns. Again, the last MixColumns finally spreads the 4 errors over the whole state, thus infecting all 16 bytes, by exploiting this, it is possible to retrieve the whole AES-128 key with only a couple of faults using the algorithm 7.

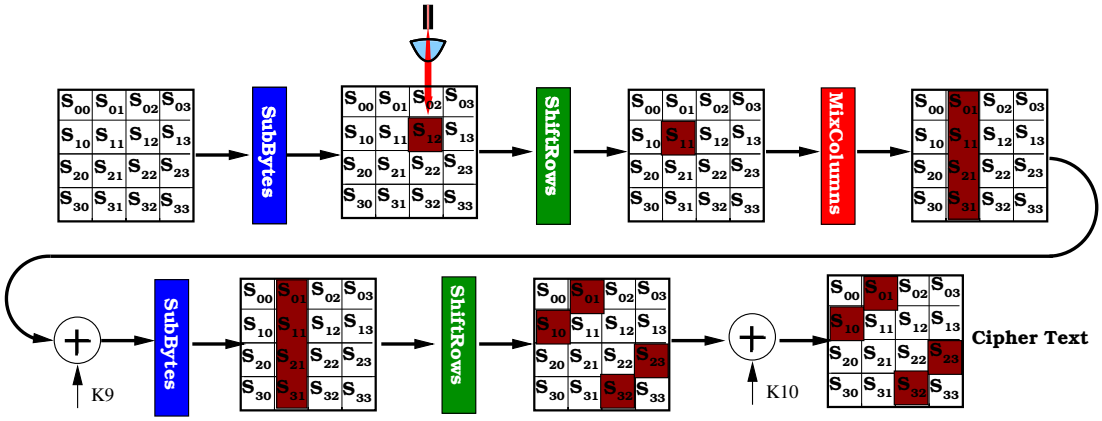


Figure 2.12: Fault effect on round 9 of AES.

Algorithm 7: Piret's DFA Algorithm.

1. Prepare a list L_D that contain 1020(255×4) different possibility of Mixcolumn of the Round 9.
2. Make an exhaustive search on the $K_{0,d}^{Nround}$, $K_{1,(1-d)mod[4]}^{Nround}$, $K_{2,(2-d)mod[4]}^{Nround}$ and $K_{3,(3-d)mod[4]}^{Nround}$.
3. Compute Δ_t =SubBytes^{-1}((C$\oplus K^{Nround})_{*,d}) \oplus SubBytes^{-1}((D \oplus K^{Nround})_{*,d})$.
4. Verify if $\Delta_t \in L_D$.
5. If yes we will add the four bytes of the key to the list C_d of the potential candidates.
6. Return to the second step with another pair of fault until we get only one candidate.

An error in round 8 yield to four errors in round 9, thus we just need two well located faults to recover the hole key. The propagation of the a faults in round 8 is show in Figure2.13.

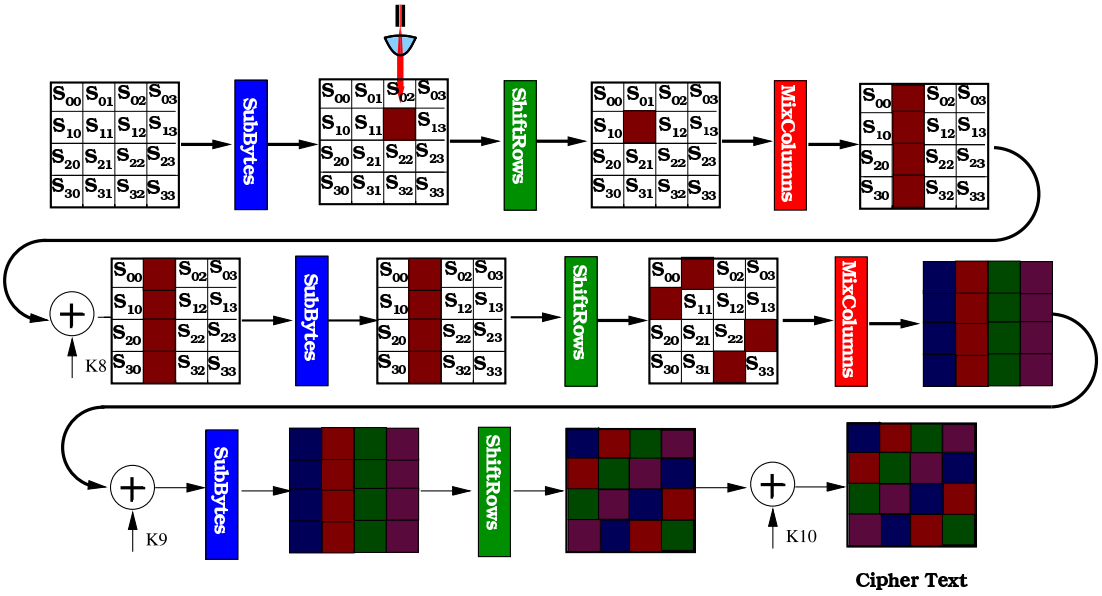


Figure 2.13: Fault effect on round 8 of AES .

2.4.6.6 Moradi, Shalmani and Salmasizadeh

In CHES 2006, Amir moradi presented new attacks against AES [33]. The authors proposed new fault models that cover all faults that can occur in the 9th round of encryption. The biggest advantage of these attack methods is the high coverage rate of used fault models. With the first fault model the author needs only 6 faulty ciphertexts in average for discovering the main key and 1495 faulty ciphertexts for the second one.

2.4.6.7 Tusntall and Mukhopadhyay

In [34] the authors present an enhanced Differential Fault Attack that can be applied to the AES using a single fault. The single fault affects the input of the antepenultimate round. The AES key can be found using a two stage algorithm. If we note x_i, correct byte of the ciphertext and x'_i faulted byte of the ciphertext then we can derive sixteen equations by using MixColumns. Once resolved an attacker could expect to have

2^{32} key hypotheses for the secret key. In order to further reduce keys hypotheses the relationship between the ninth round and the last round can be used. Using the Inverse MixColumns operation and the equation found in the first step, the authors define a set of four new equations. Finally all the key hypotheses generated by the first step are tested using the new set of equations, if the hypotheses satisfies the new set then it can be recorded as a key candidate, else it discarded. Using the second step an attacker would expect to have 2^8 possible keys, and can easily perform a brute-force attack to find the cipher key.

2.4.6.8 Differential Behavioral Analysis

Differential Behavioral Analysis "DBA" as described in [35] is an hybrid attack between differential power analysis "DPA" and a safe-error attack. The model of the fault used is byte-wise "stuck-at" as for Safe error attack. Then the probabilistic treatment of DPA is applied. DBA is similar to DPA but instead of power we use the behaviour of the circuit when a fault is injected as side channel.

2.4.6.9 Fault Sensitivity Analysis

In CHES 2010, a new fault attack was presented known as fault sensitivity analysis [36]. Unlike most existing fault based attack FSA does not use values of faulted ciphertexts. In FSA, starting from a condition where a correct ciphertext is obtained, the attacker gradually increases the intensity to which he disturbs the power supply or external clock. While stressing the device, there must be a moment where the success rate of fault injection is non zero and a moment where the success rate is 1. This information is recorded as new side channel information and using correlation between the data and the success rate, the attacker can retrieve secret information.

2.4.7 Summary of DFA on AES

Table 2.1 presents a comparison between different attacks. The comparison criteria are:

- **The fault model**. It corresponds to the number of bit which are necessary to modify. the location
- **The fault target**. The datapath is generally the target but the key expansion path can also be targeted.

- **The number of faults**. This indicate the number of faults which are needed to make sure the attack will succeed.

Attack	Fault model	Fault target	# of faults
Tusntall and Mukhopadhyay [34]	Single Byte	Datapath	1
Piret and Quisquater [30]	Single Byte	Datapath	2
Giraud 1 [29]	Single Bit	Datapath	35
Dusart et al. [32]	Single Byte	Datapath	50
Blömer 1 [7]	Single Bit	Datapath	128
Blömer 2 [7]	Single Byte	Datapath	256
Moradi et al. [33]	Multiple Bytes	Datapath	1495
Takahashi et al. [37]	Single Byte	KeySchedule	7/1
Chen and yen [31]	Single Byte	KeySchedule	44
Giraud 2 [29]	Single Byte	KeySchedule	248

Table 2.1: Summary of Differential Fault Attack

Among these attacks the Piret and Tunstall proposals present the minimum number of faults. This is particularly important to increase the attack feasibility as it could be difficult to get different errors from the same fault injection. Moreover they target the datapath which is easier to disturb than the keypath for some attacks. For instance the setup violation attack, which is presented in chapter 2, targets the logic on critical paths which are more numerous in the datapath than in the keypath.

2.5 Conclusion

In this chapter, the general concepts about cryptography and the main attack techniques used to recover the secret keys have been introduced. A specific attention has been given to the Differential Fault Analysis "DFA" as it is one of the most effective attacks against implementation of cryptographic algorithm. A synthesis of different DFA targeting the AES algorithm is presented. Among them it has been shown that the attack of Piret and Quisquater is particularly powerful as only two faults are necessary to retrieve the 128-bit AES key. This attack type has been chosen to perform the practical attacks presented in chapter 2. The Tunstall attack would have been interesting to carry out but this new attack was not yet known during the Phd period. As the question about the feasibility of such attack arises, chapter 2 shows the implementation instances by using both non-invasive and invasive methods. These methods

inject respectively global and local faults and they clearly show the efficiency of the DFA attack with a relatively cheap equipment.

Chapter 3

Practical Attacks on AES

In this chapter, the DFA on AES experiments are presented. The attacks have been carried out and analyzed in detail on two different platforms which have been set up specifically for this purpose.

The first platform allows to inject faults by using a global and non-invasive technique based on under-powering or overclocking the device. This simple technique is cheap to carry out and had never been used for the time being. It provides ways to subtly analyze attacks on targets in embedded systems based on FPGAs or ASICs. The smartcard is not the primary target of this attack as some voltage protections exist against spikes attacks. The attack impact is to provoke setup time violations on Flip-Flop, thus generating faults. A complete study of the temporal and spatial localization of single faults has been done. Three target types have been used: custom ASIC (SECMAT project) Altera Stratix and Xilinx Virtex5.

The second platform allows to inject faults by using a semi-invasive method based on a laser beam. With this more costly technique it becomes possible to inject well located faults on a device. The target is an ATMega128 microcontroller running a software implementation of AES.

3.1 Global Attack: Setup time violation attack

3.1.1 Attack Theory

In CMOS logic [38], every gate has a propagation delay, that is inherent to the transition switching time and to the existence of capacitance (most often unwanted, *i.e.* parasitic) around it. Thus, every stimulus at the input of a logic gate requires some

time to produce its effect at the output. The propagation delay depends on many factors.

The most important one is the function of the gate. For instance, an AND gate with one input equal to 0 will evaluate to 0 irrespective of the second input arrival date. This "early evaluation" would not have happened with the same data configuration on an OR gate. Indeed, the OR gate cannot decide of the final result before being acquainted on the value of its latest input if the fastest ones are equal to 0.

At the second order, the propagation time also depends on the shape of the input signals. An input signal transiting slowly will take a longer time to propagate through the gate. A signal coming from a long line is likely to have a smooth edge, whereas a well buffered signal that has a short interconnect length will probably have a steep transition slope. Thus the routing also influences the propagation time. The cross-coupling of the wires is also a subtle effect that affects the signal's speed and transition speed: depending on the neighbour wires, any signal can be either slowed down or sped up, or have its waveform be modified.

Additionally, we mention that some gates can answer in a non-deterministic time. One representative example is the XOR gate that is placed in the situation of an arbiter: this occurs when the two inputs change almost simultaneously from (0, 1) to (1, 0). Then, depending on the relative delay between the two inputs, the gate can start to glitch. The answer time is also dependant on thermal fluctuations.

Eventually, the CMOS gates do glitch, which means that some transitions can be generated even if they are not the final values. Once generated, the glitches propagate and lead to other glitches. The final value of the computation is taken only after no other changes occur. Indeed, the digital computations can be modeled as the transmission of events on an oriented graph: the signals propositions being causal, every computation in a combinational netlist necessarily converges.

To summarize, the propagation time in CMOS logics depends highly on the data and on the routing. Furthermore, the longest path for the same input data might not be the same, because of the non-deterministic behavior of some gates.

In sequential logic, one global signal, called the clock, cadences all the combinational computation in parallel, at the same place. One implicit condition is that all the gates are expected to have finished propagating their data when the clock rising edge arrives. Of particular importance is the longest path, that determines the maximal clock frequency. As discussed in the previous subsection, this critical path is data-dependent and corresponds to the clock smallest admissible period. If for whatever reason the clock period is lower than the setup time, then faults can occur. When either the clock accelerates or the data signals decelerate, the setup time can eventually be violated. The propagation time increases with the decrease of the power supply and faults are caused by an early latching of a combinatorial function as shown in figure 3.1.

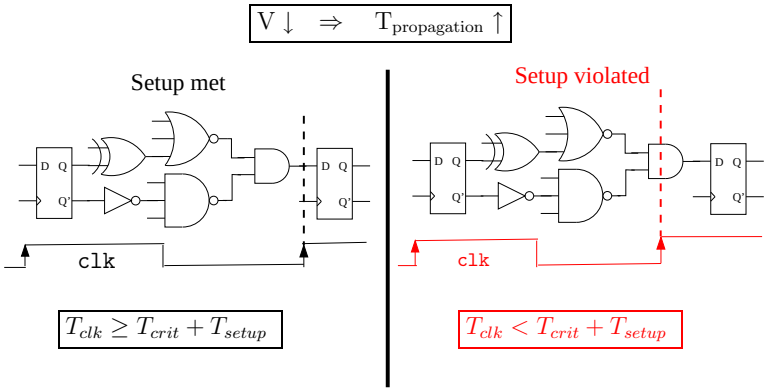


Figure 3.1: Setup violation.

3.1.2 Acquisition Platform

The setup is composed of two parts:

1. A communication channel between the analyzer (personnel computer) and the circuit under test. This allows the user to inject the ciphering orders and receive back the results.

2. A remotely controlled power supply and clock synthesizer which provides respectively the supply voltage and the clock. Hence various values of frequency and voltage can be accurately adjusted to feed the attacked circuit.

Figure 3.2 sketches the experimental setup.

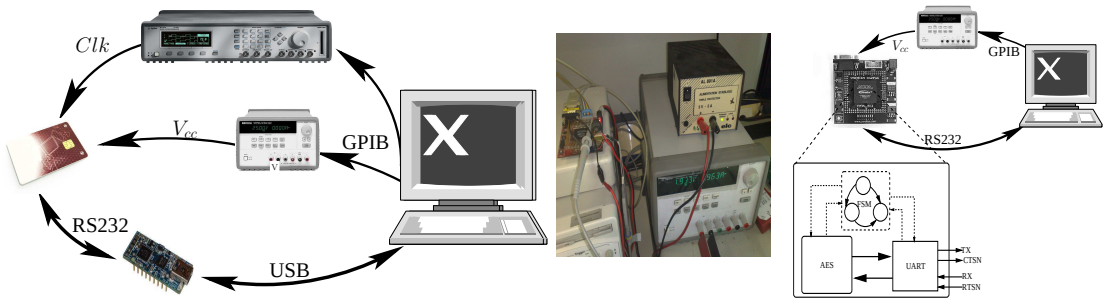


Figure 3.2: Experimental faults injection platform

The power supply and the frequency generator can be controlled through a GPIB port. The power supply can deliver a voltage with an accuracy of half a millivolt. The frequency generator can deliver a frequency with an accuracy of one MHz. In order to collect data we have conducted the flowing acquisition campaign:

- The triples {message, key, ciphertext} are recorded for N encryptions at each values of Vcc or F (2.000.000 ciphertexts for ASIC and 100.000 ciphertexts for FPGA's).
- In view to simulate an attack, we decided to keep the key at a constant value.
- To collect representative results, the input message varies randomly at each encryption.

3.1.3 Fault Analysis

This software tool is very important to know where and when the fault has been generated. The only assumption to do is that the message and the key are known by the attacker, and that they are not faulted. In other words, the faults concern only the encryption, and thus can affect only the ciphertext. For each target (ASIC and FPGAs) a classical AES architecture of the "encrypt" function is used. The rounds of the AES algorithm are implemented with a unique loop as this serial approach provides a significant cost reduction in hardware. Therefore setup violation can occur at each AES round.

In the "encrypt" function, the fault value XORed is with the message and permits any fault injection at any round of the AES encryption. The array `fault_t` is defined as:

```
// Mask ready to be applied
// to every round' state
typedef reg_128 fault_t[10];
```

where `reg_128` is typedef'ed for `char[4][4]`. To simulate a single Byte fault injection, the `fault_t f` is initialized to full zero, with the exception of a single byte (`char`) within the `reg_128` state for a single round index. The fault analysis consists in calling this function for all the possible faults values, and compare with the ciphertext obtained from the experimental platform. Indeed, the only evidence that a fault has occurred experimentally is a faulted ciphertext. The exhaustive search of single "byte-flip" requires $\underbrace{(2^8-1)}_{single\ Byte\ fault} \times \underbrace{16}_{Bytes} \times \underbrace{10}_{Rounds} = 40\,800$ calls to the "encrypt" routine. This allows to:

- Find the experimental faulty ciphertext. In this case, the fault is called "covered", or
- Declare that the fault is "uncovered", if the faulty ciphertext matches none of the function "encrypt". This fault is called "uncovered".

The purpose of this implementation is to identify the typology of faults that occurs in the device, as shown in Figure 3.3. In this figure, the AES function with two arguments is the regular implementation of AES-128, whereas the AES function with three arguments is the "encrypt" routine detailed previously.

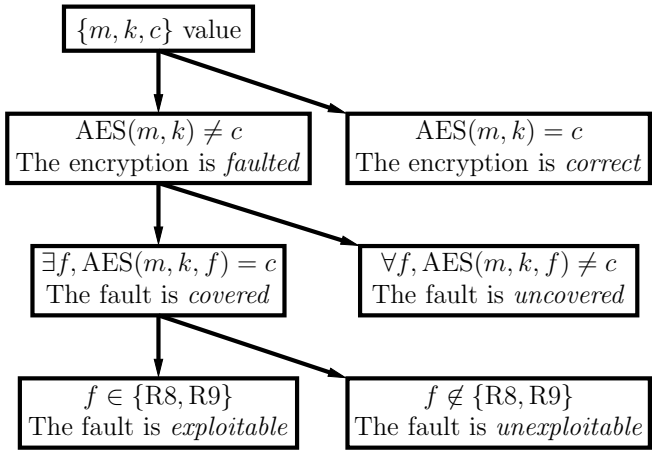


Figure 3.3: AES faults analysis.

3.1.4 Attack on ASIC

In this section, we present some result on power and frequency attack on SecMat v1 [39] witch is an academic ASIC designed to evaluate side channel attacks and their possible countermeasures. It contains two different implementation of non-protected DES, an implementation of protected DES and an unprotected implementation of AES along with peripherals like CPU, RAM, *etc*. The AES is a "low area" pipelined implementation which computes column by column, *i.e.* 4 cycles per round or 50 cycles per encryption. The size of the circuit is 4.0 mm^2 for 2.0 million transistors.

We considered soft stresses:

1. **Supply deprivation**, based on the power voltage decrease at nominal clock frequency (32 MHz),
2. **Over-clocking**, based on clock frequency increase at nominal power voltage (1.2 volt).

We gradually raise the stress level, and show that an attacker can accurately choose the quantity of faults induced within the device. The figures 3.4 and 3.5 show that there is comfortable range of vulnerable voltage and frequency where the cryptographic device outputs faulty results while not crashing.

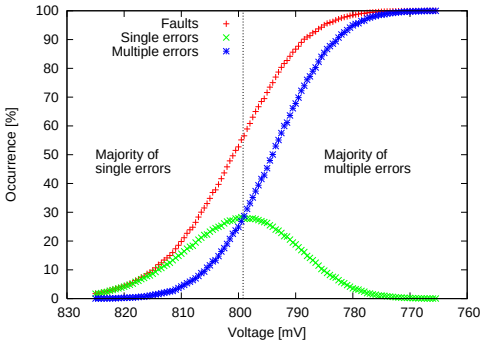


Figure 3.4: Occurrence of Fault — (power).

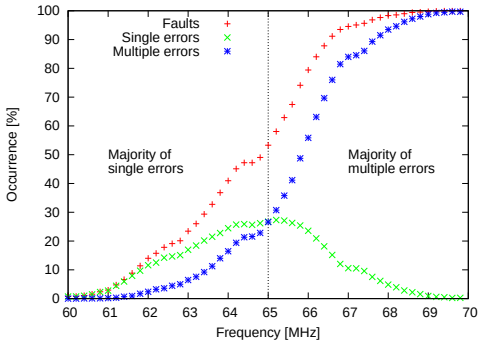


Figure 3.5: Occurrence of fault (over-clocking).

Given the resemblance of the fault occurrence profile, we conclude that the two studied stress modalities generate faults of the same type. The errors are caused by a setup time violation: the output of a combinatorial logic block is sampled in its

downstream synchronous register prematurely. In the case of supply deprivation, the combinatorial logic becomes slower at constant clock frequency, whereas in the case of over-clocking, the combinatorial logic keeps its natural speed but the registers latches more often. With both fault injection paths, a critical path is violated, resulting in single errors for low level of stresses: in Figures 3.4 and 3.4, single errors are dominant above $\approx$ 800 mV and below $\approx$ 65 MHz.

3.1.4.1 Experimental Evaluation of Piret's DFA

The Figure 3.6 presents the coverage of single faults. The coverage is defined as being the ratio between single and detected faults:

$$Coverage = \frac{\sum Single\ Faults}{\sum Detected\ Faults}.$$

A fault is said "detected" when the ciphertext has not the expected value, i.e. $AES(m,k) \neq c$ where m is the plaintext, k the key and c the expected ciphertext. A fault is said "single" when only one Byte is faulted among the intermediate variables of AES. To apply the Piret's algorithm only single faults have to be injected either in the penultimate or antepenultimate round.

The first faults (for the highest voltage values), are almost all single. This can be seen in Figure 3.6 by the fact that the coverage is close to one hundred percent. As the voltage decreases, the coverage degrades, attesting the gradual appearance of multiple faults. We can also compute the coverage $Coverage_{Ri}$ in specific round Ri:

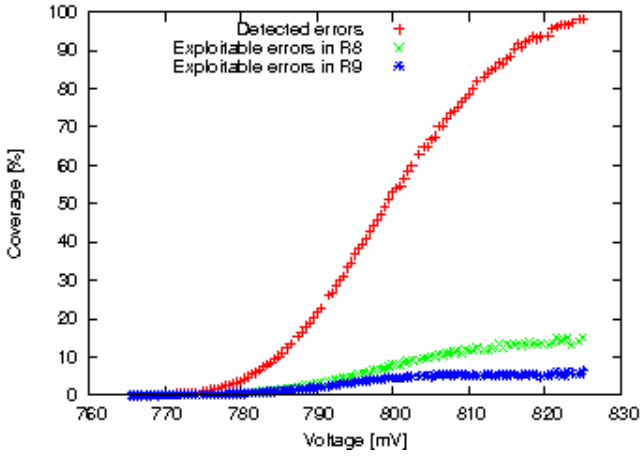


Figure 3.6: Coverage of exploitable faults.

$$Coverage_{Ri} = \frac{\sum Single\ Faults\ \in Round^i}{\sum Detected\ Faults}$$

and can we observe that, amongst the "covered" (i.e. single) faults, 16 % fall in the round 8 and 4 % in the round 9, where Piret's DFA can exploit them. As a consequence, **Piret's DFA is practical**.

Furthermore, we are now able to quantify the experimental efficiency of Piret's DFA. We can state that the average number $N_{experimental}$ of faults to collect experimentally (at Vcc=820 mV) in order to successfully mount Piret's DFA is:

- $N_{experimental} = \lceil N_{theory}/0.16 \rceil = \lceil 2/0.16 \rceil = 13$ for an attack in round 8, and
- $N_{experimental} = \lceil N_{theory}/0.04 \rceil = \lceil 8/0.04 \rceil = 200$ for an attack in round 9.

This proves that, in practice, the key can be found with only 13 ciphertexts. This collection is typically realised in a couple of seconds.

3.1.4.2 Spatio-Temporal Characterization of Faults

The "covered" faults are analyzed in terms of spatio-temporal locality (% of encrypted messages):

- in which rounds (from R1 to R10) are they more likely? (refer to Figure 3.7) and
- in which byte of state are they more likely? (refer to Figure 3.8).

We can see that the first round (R1) is never affected by faults. This observation was indeed predictable, since the first is special: it consists of the AddRoundKey transformation alone. Therefore, the critical path is not in this round.

It can seem counter-intuitive that faults are not uniformly distributed between rounds. In a static timing analysis (STA) of a design, the critical path is the same for every iteration. Therefore, one might expect that if a critical path is violated at one round, then all the rounds (8 or 9) will be faulty. However, we observe single errors localized at a given round. The reason could be that the critical path is highly data-dependant.

From the analysis of Figure 3.7 and 3.8, we have shown that the faults are not uniformly distributed over time and space. This observation, albeit not general since our setup is very particular, can be a valuable information for the designers in charge of implementing counter-measures.

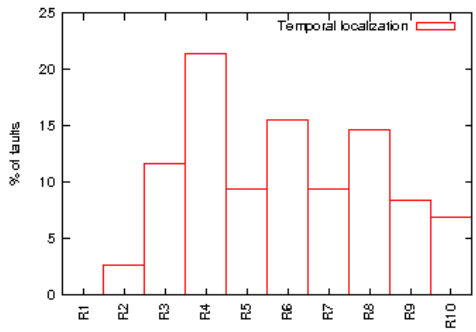


Figure 3.7: Temporal localization of single faults.

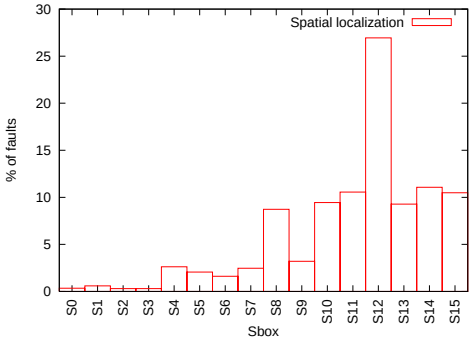


Figure 3.8: Spatial localization of single faults.The SubBytes box $s_{i,j}$ has index $4 \times i + j$ in the histogram.

One of the reason that could explain the non-uniformity of faults can be the distribution of faults over the time, if we suppose that the occurrence of faults follows a Gaussian distribution centred in $\mu \approx \frac{T_c}{2}$ where T_c is the encryption time and with a random variance σ. Then we can find that the probability to have a X a single bytes fault in one round is:
$P\{X|R_i\} = (1 - \frac{1}{2}(1 + erf(\frac{\frac{i}{10}T_c - \mu}{\sigma\sqrt{2}}))).(1 - \frac{1}{2}(1 + erf(\frac{\frac{9-i}{10}T_c - \mu}{\sigma\sqrt{2}})))$ where **erf** is the error function and i $\in \{0 \dots 9\}$ is the round number.

3.1.5 Attack on FPGA

3.1.5.1 SoC Architecture

In order to attack AES on FPGA we implement a cryptographic SOC that we synthesize on Altera Stratix and Xilinx Virtex5 FPGAs. The encryption platform is composed of three modules: UART interface, a controller and AES co-processor as shown in Figure 3.9. This design communicates with a monitoring PC via RS-232 cable. The critical path of the design is located in the cryptographic module to avoid the crash of the system when we decrease the power supply.

3.1.5.2 AES architecture

Figure 3.10 shows the architecture of a simple, non protected AES co-processor. The AES co-processor is designed to have a parallel architecture. It performs each round of AES in each clock cycle. The four sub-rounds are SubBytes, ShiftRows, MixColumns

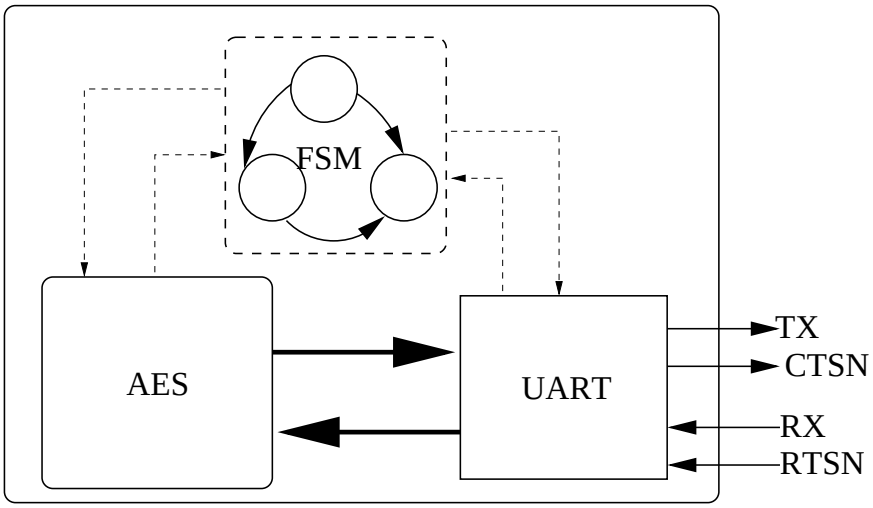


Figure 3.9: Simple AES.

and AddRoundKey. These sub-rounds along with some multiplexers and key scheduler comprise the datapath. The key scheduler or expander calculates a key for each round which is then used in the datapath.

The most important module in the AES is SubBytes, therefore we try to implement this component in three different ways. In the first architecture we implement Sbox as look-Up Table (LUT) [1]. This asynchronous table is a non-linear byte substitution indexed by the most significant nibble and least significant nibble of the input. The second implementation also uses the same design but the table is made synchronous and is sensitive to falling edge of the clock. Such implementations are automatically moved to the block RAM by the synthesis tool. At each falling edge, RAM samples the input address. In the third implementation, we optimize sbox using composite field. In fact, The Rijndael Sbox is a nonlinear transformation over the finite field $GF(2^8)$ each element of the 256 field can be represented as a 2 dimensional vector over $GF(2^4) : a = a_h * x + a_l$ with a $\in GF((2^4)^2)$ and $a_h, a_l \in GF(2^4)$. We can define an isomorphic mapping of the field elements which are represented with respect to $GF(2^4)$. Using subfield we can build optimized sbox as described in [40]. Figure 3.11 shows the architecture of the composite field sbox. We use the same SubBytes for the Datapath and the Key schedule.

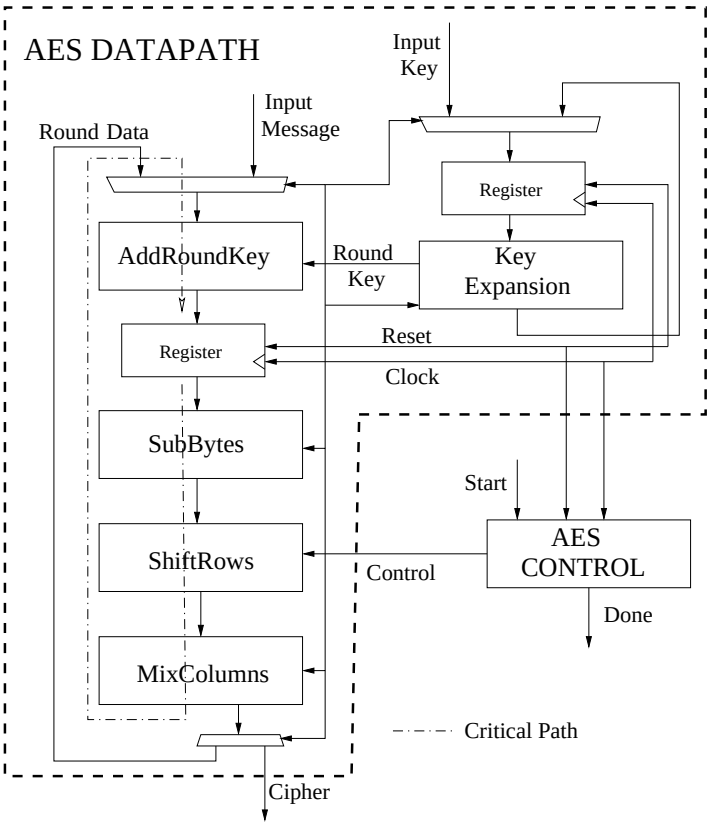


Figure 3.10: AES architecture.

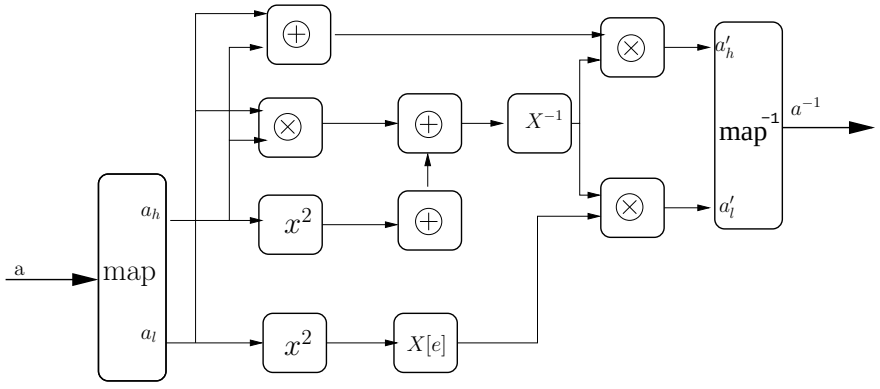


Figure 3.11: Composite Field implementation of SubBytes

3.1.5.3 Experimental Results on ALTERA Stratix

In our architecture, the delays in the datapath are greater than in the key schedule. As we focus on non-invasive attacks, we assume local faults cannot be injected directly into the keypath. Moreover, as global perturbations will not affect the keypath either, we assume that the key schedule block is fault-free. At higher voltages $V \in [1.28V \ldots 1.275]$ only single faults occur. As we decrease the voltage beyond a certain threshold, setup time is violated on multiple paths and faults become multiple (uncovered). It is straightforward to adapt the results obtained in this section to other attacks, such as attacks on the key schedule [29].

Figures 3.12, 3.13, 3.14 show the occurrence of faults in the three architectures on the Altera board. Faults are partitioned into single *i.e.* faults which affect one byte on the AES state before the SubBytes transformation in the datapath or multiple *i.e.* faults affecting multiple byte or occur in the keypath. Single faults have a "bell-shape" distribution. This behavior is compatible with a fault model where errors are caused by a setup violation on critical combinatorial path. It is well established that propagation time of a signal on a particular path increases with decreasing supply voltage [41]. Thus at lower voltages it is more likely that a critical path is violated to generate frequent single faults. Nevertheless, below a threshold, multiple critical paths are violated. Hence an augmentation of multiple faults, and a subsequent diminution of single faults.

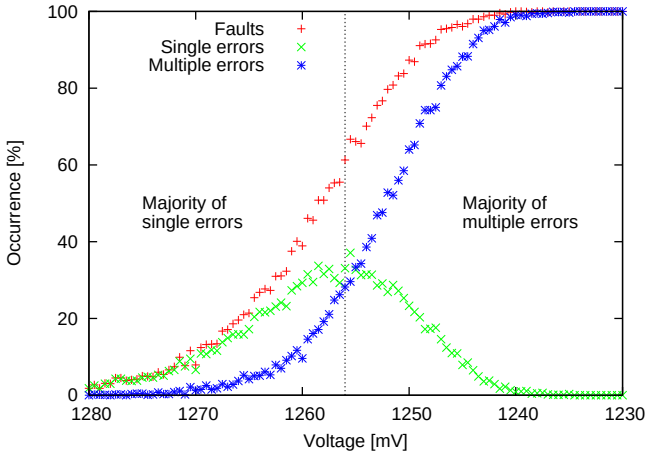


Figure 3.12: Occurrence of faults: sbox in $GF(2^4)$.(ALTERA)

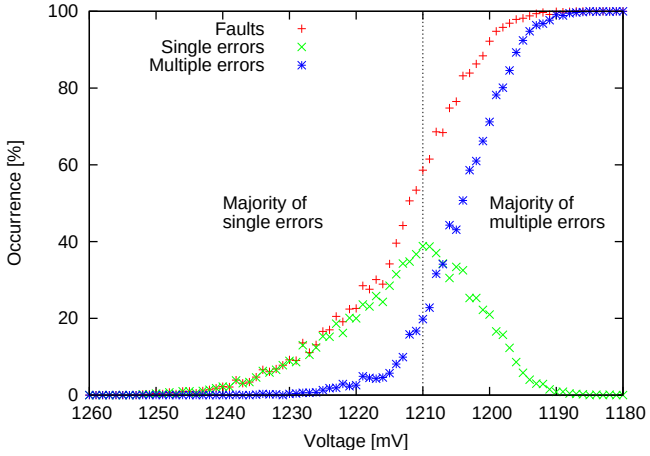


Figure 3.13: Occurrence of faults: sbox in LUT. (ALTERA)

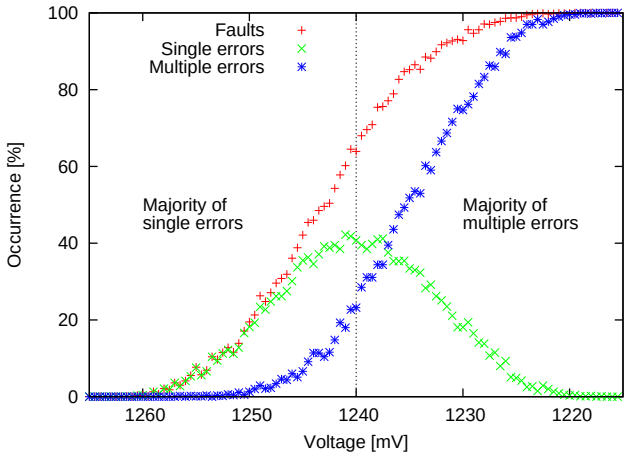


Figure 3.14: Occurrence of faults: sbox in RAM.(ALTERA)

Figures 3.15, 3.16, 3.17 present the coverage of single faults, *i.e.* the ratio between single and detected faults. The first faults , are almost all single as the coverage is close to one hundred percent. As the voltage decreases, the coverage degrades, attesting the gradual appearance of multiple faults. In our experiments, we use the "Piret's Attack" [30] to exploit the faults. As per this attack, single faults affecting only the two penultimate rounds are used for retrieving the key. From here, we address such faults as exploitable faults.

Figure 3.18, shows the Hamming weights of the exploitable byte-flips for the $GF(2^4)$

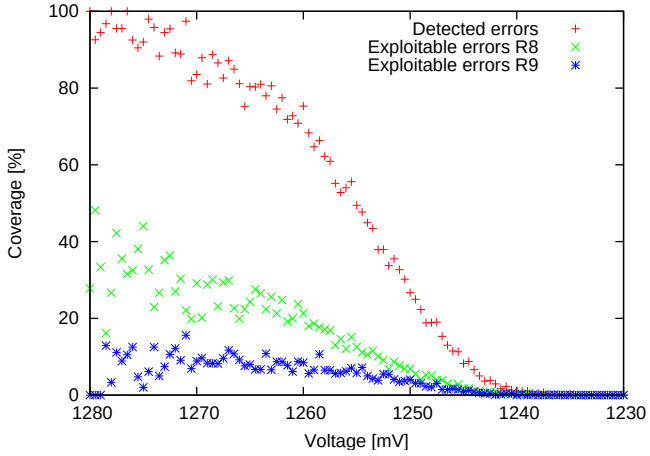


Figure 3.15: Coverage of single faults, and detail of exploitable faults in $GF(2^4)$.

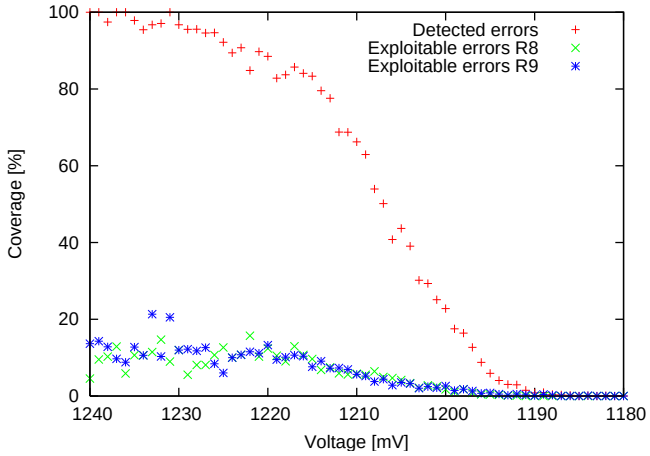
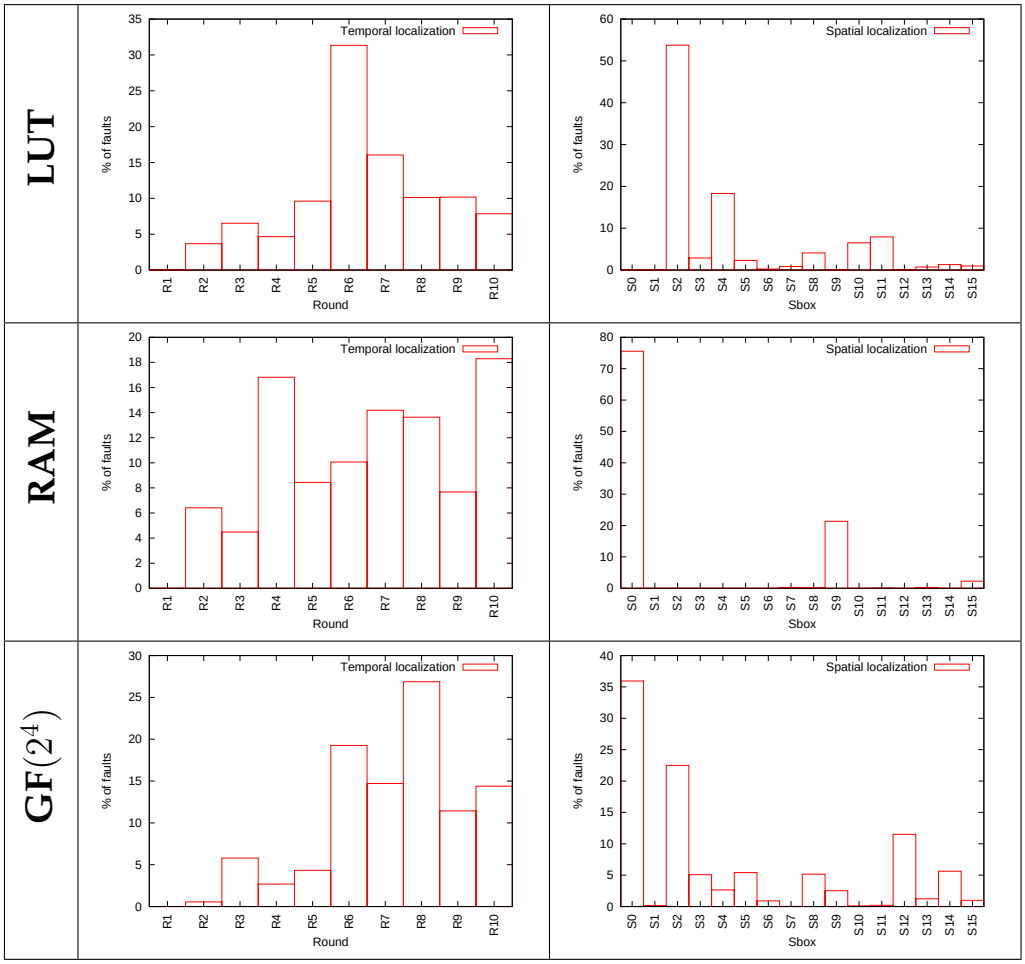


Figure 3.16: Coverage of single faults, and detail of exploitable faults in LUT.

architecture. One interesting observation from this figure is that most of the faults occurring in the circuit are a single bit fault (Hamming Weight of the fault=1). This information allows attacker to mount some of the published "Bit Fault" attacks [29].

Table 3.1 shows the temporal localization of the single faults in the Altera Stratix FPGA. In temporal localization the single faults are classified according to the round of occurrence. When using Piret's attack, the attacker is interested by the single faults occurring in the two penultimate rounds only. As for the ASIC attack, we can see that

Table 3.1: Temporal and Spatial localization of single faults on Altera Stratix board



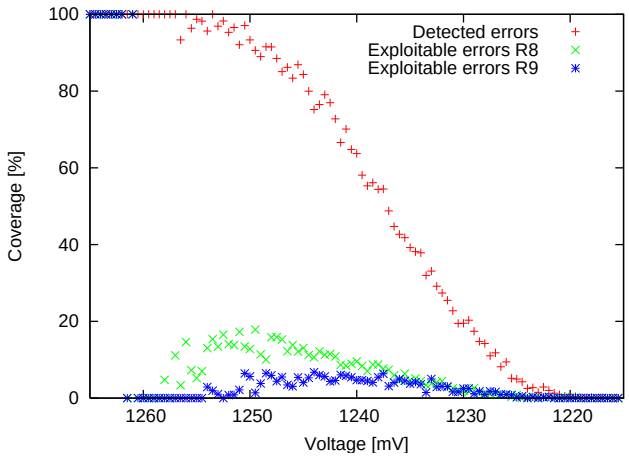


Figure 3.17: Coverage of single faults, and detail of exploitable faults in RAM.

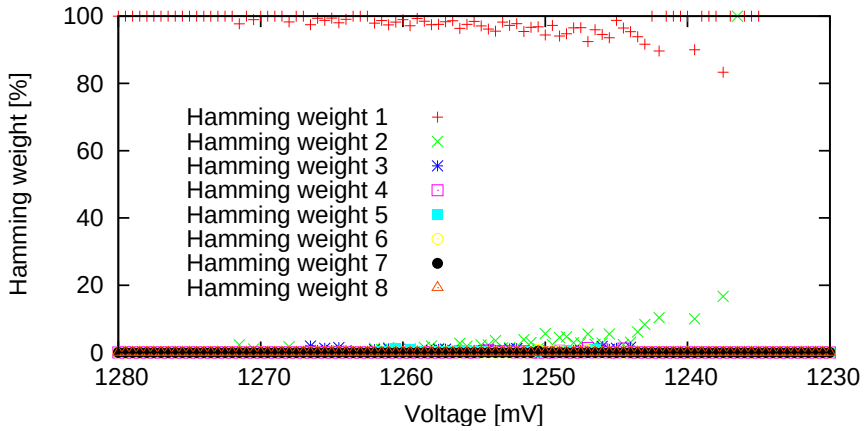


Figure 3.18: Hamming weight of exploitable faults in $GF(2^4)$.

there is no fault in the first round. This is because the first round in AES is comprised only of the AddRoundKey operation resulting in a fairly small timing path. This also proves that the communication between the FPGA and PC is fault free.

Table 3.1 shows that each sbox has different number of faults. Since the computation time of logic gates is data-dependent, there is an uneven temporal and spatial distribution of the faults. The spatial localization of the faults (*i.e.* the classification of faults according to the affected Sbox) is shown in Table 3.1. As shown in the histograms the

attacker can find sufficient number of faults to successfully mount the attack.

3.1.5.4 Security Evaluation of the three architecture against DFA

In this section, we compare the three architectures with respect to security. Figures 3.12, 3.13, 3.14 show the occurrences of faults in different architectures. For the sake of comparison, we plot the exploitable faults on the same diagram. In figure 3.19 we see that the peak of the bell shaped distribution is highest for the architecture with sbox in GF(2^4). It shows that around 13% of the single faults are exploitable. On the other hand, less than 6% of the faults are exploitable when the sbox is implemented as a table in LUT. This means that we need half the amount of faulty ciphertext when attacking sbox in GF(2^4) than needed for LUT. These results are in accordance with the results obtained in [42], where authors have used the timing analysis of post map netlist of AES co-processor. The Authors demonstrate that it is more difficult to attack sbox in LUT than a sbox in GF(2^4) because a higher attack frame means higher probability of creating a single fault. Thus, we have practically prove the results which were stated theoretically in [42]. Voltage is another parameter for security. A sbox which gets faulty at lower voltage is more secure because it is more likely that some other part of the design stops working at lower voltages.

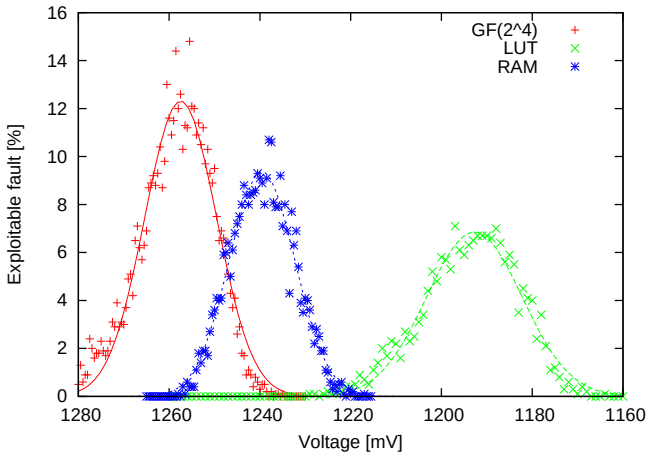


Figure 3.19: Exploitable errors "Round 8 and 9".

Recently, few methods have been reported [43] which suggest to synthesize the bulky parts of AES like SubBytes & MixColumns into the peripherals like block RAM,

DSPs, etc. These methods reduce the logic utilization in the FPGA and hence are cost effective. We also tested an sbox implementation in the RAM. The results as shown in Figure 3.19 is that 9% of the faulty ciphertexts are exploitable for RAM implementation against 6% in case of sbox in LUT. So we see there is a trade-off between cost & security. It has always been known that higher security comes at higher cost; this rule of thumb also applies to AES.

Figure 3.19 shows that sbox in RAM is more secure than the sbox in GF(2^4). The timing information on critical path in the datapath suggests that sbox in RAM should be less secure than sbox in GF(2^4). Contrary results can be explained by following arguments. The clock period is 20 ns.

The RAM is sampled on the falling edge while the state register is sampled at the rising edge. The critical path calculated is between these two edges which is just for negative half of the clock cycle. The Altera CAD tool "Quartus" normalizes the timing depending on the duty cycle of the clock and displays in terms of one full cycle. This fact was confirmed when reduction in the duty cycle of the clock resulted in a higher maximal frequency of operation. In practice, the timing of the path is less than $19.818ns$ and should be interpreted separately for each half of the clock cycle. This is also corroborated by figure 3.19. In case of sbox in GF(2^4), timing information given by "Quartus" is trustworthy as all the operations are sensitive to the rising edge of the clock. When the sbox is implemented in GF(2^4), as shown in the architecture, the worst-case critical path in the datapath will begin from and end at the state register which stores round data. In case of sbox in RAM, the worst-case critical path is between the output of RAM & state register. As compared to RAM, apart from operators of ShiftRows, MixColumns & AddRoundKey, the sbox in GF(2^4) also uses combinatorial operators to implement the sbox as well. Since occurrence of fault is a dynamic parameter, the presence of larger number of combinatorial components may increase the probability of fault occurrence in this architecture. However nothing definite can be concluded as the construction of RAM and LUTs are different. Due to the difference in construction, the delays due to under-powering will evolve differently.

In order to check the impact of the controller on the critical path of the previous experiments, we implemented a new architecture ($GF(2^4)$): the control signals are computed prior to every round. This architecture is depicted in figure 3.20.

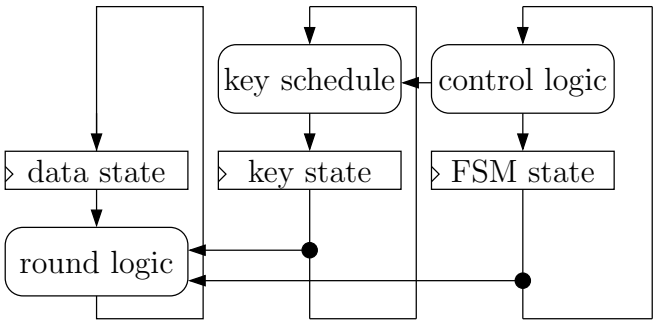


Figure 3.20: AES architecture with critical path strictly confined in the datapath.

We check that the one hundred first critical paths originate from and finish in the datapath. We reproduce the same fault campaign for Altera FPGA with the GF(2^4) architecture of SubBytes. The fault occurrence per round is shown in Figure 3.21. This figure clearly shows that the faults are not uniformly distributed despite the decoupling of the datapath from the control. This observation seems to be specific to hardware implementations: similar result obtained in a CPU show uniform distribution [44]. This is certainly due to the greatest complexity of algorithms implemented on hardware, were 128 bits are processed in parallel as opposed to 8 bits in a CPU.

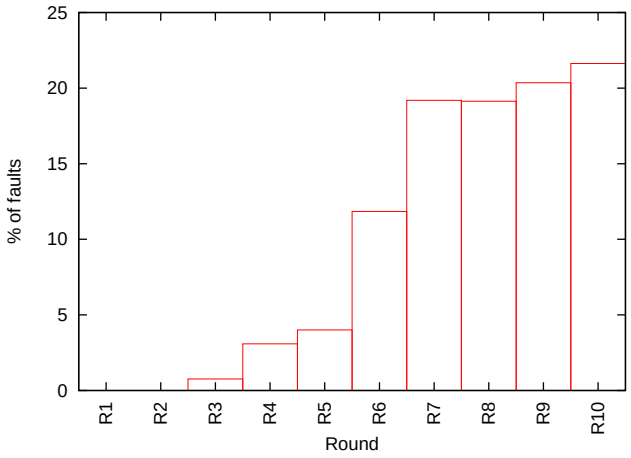
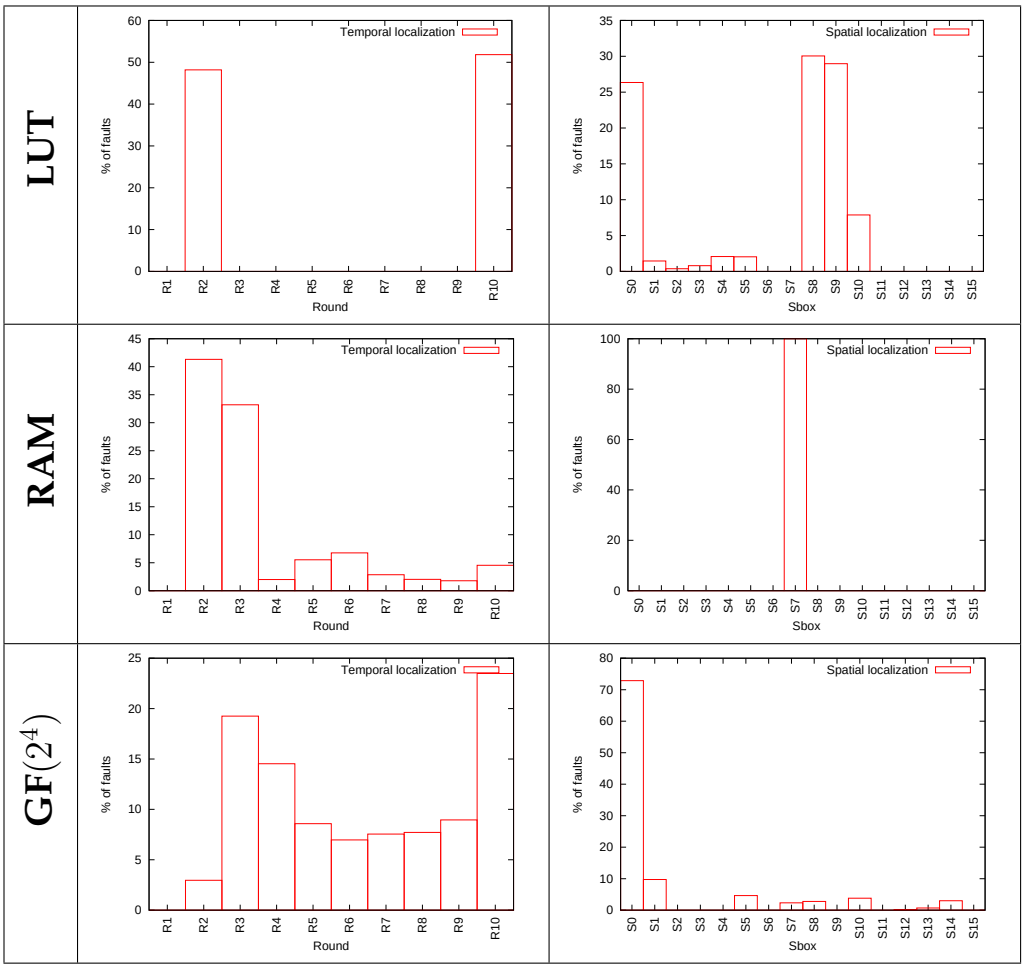


Figure 3.21: Temporal localization of single fault on Altera GF(2^4).

Table 3.2: Temporal and Spatial localization of single faults on Xilinx Virtex 5 board



3.1.5.5 Experimental Results on Xilinx Virtex5

Table 3.2 shows the temporal localization of the single faults for the Xilinx Virtex5 FPGA. We can exploit faults that occur in the RAM and the GF(2^4) implementation. But there is no exploitable faults for the LUT implementation. We can argue this is due to the fact that the critical path is shorter in LUT implementation, hence the controller can also be affected by the setup time violation and the fault is detected as multiple.

Table 3.3: Characterization and attack results for Altera and Xilinx with the three Sbox architectures.

Architecture	Critical path(ns)		% of Sing. faults		PDF		Voltage(V)	
FPGA	Altera	Xilinx	Altera	Xilinx	Altera	Xilinx	Altera	Xilinx
LUT	13.725	7.772	39 %	69 %	0.872	0.513	1.21	0.64
RAM	17.569	9.758	42 %	50 %	0.795	0.282	1.26	0.76
GF(2^4)	18.818	14.426	33 %	29 %	0.635	0.335	1.24	0.71

3.1.5.6 Security Comparison between ALTERA and XILINX

Figure 3.22 and Figure 3.23 shows the occurrence of single faults in the three architectures for Altera and Xilinx. Faults are partitioned into single *i.e.* faults which affect one byte on the AES state before the SubBytes transformation in the datapath or multiple *i.e.* faults affecting multiple byte or occur in the keypath. Single faults have a "bell-shape" distribution.

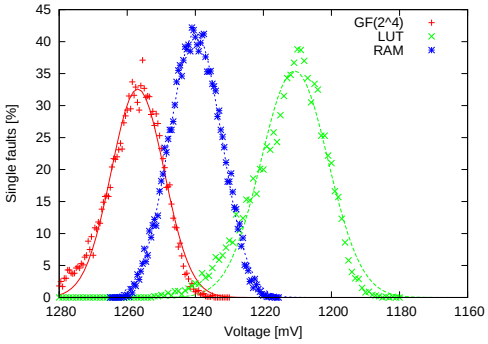


Figure 3.22: Occurrence of single Faults (Altera Stratix)

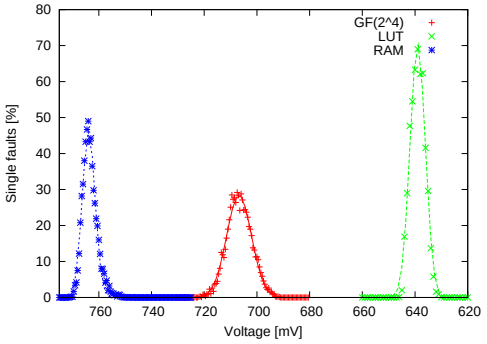


Figure 3.23: Occurrence of single faults (Xilinx Virtex5)

Now referring to table 3.3, we see that the critical path are different for the same implementations on different FPGAs but stay in the same order for the two FPGAs. We also see that the LUT implementation has more exploitable faults in Altera FPGAs than in Xilinx and the opposite for RAM implementations. From this we can deduce that the attacker has to change his strategy according the target and their sensitivity to setup time violations. This fact is confirmed by GF implementations where the

exploitable fault are increased if the RAM is not targeted. Here we introduce the concept of probability density function ($PDF(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{(x-\mu)^2}{2\sigma^2}}$) of the fault. The PDF determines the area under the fault curve of each implementation and thus the overall robustness. The higher the PDF, the higher are the possibilities for an attacker to succeed. We see that overall, the implementations on Xilinx have a smaller PDF and hence are more robust. The reason behind this is that faults occur in Xilinx at a much lower voltage to make non-cryptographic parts of the circuit non-functional.

We have shown that global attack apply also to FPGA without deprogramming them, in fact the setup violation does not affect the configuration. Local fault discussed in the section 3.2 are more powerful in so far as they can target an accurate area. However,they can affect the configuration of FPGA [45], therefore in the next section we focus on local attack on ASIC "ATMega128".

3.2 Local Attack: Optical Fault Injection

The idea of optical fault injection was presented by S. Skorobogatov and R. Anderson in 2002 [2]. They showed that it is possible to change the content of a static memory by light. In this section, we present optical fault injection attack using synchronized laser in order to set up and optimize the conditions for a successful fault attack. Later, in 2010 Canivet and Leveugle showed practical laser fault attacks on SRAM-based FPGAs [46] and Agoyan et al. presented in [26] practical laser attack on a microcontroller. More precisely, as compared to global attacks (such as Setup violation attack), laser fault injection do target the intended logic and not only the module containing the critical timing path.

3.2.1 Decapsulation

In order to access the surface of a chip in a common plastic package, a decapsulation procedure is applied. Generally speaking, a chip can be accessed from the front side or from the rear side. A decapsulation from the front side gives access to the passivation that covers the metal layers. Accessing the substrate is possible from the rear side.

3.2.1.1 Front-Side Decapsulation

The plastic of the package over the chip has to be removed to gain access to the chips passivation. If the chip is no damaged and the bonding wires are intact, the decapsulation has not impact on the functionality of the device. The plastic can be etched by fuming nitric acid (> 95%), which is a mixture of concentrated nitric acid (HNO_3) and nitrogen dioxide (NO_2). In order to concentrate the etching process on a region above the chip and to minimize the amount of plastic that has to be etched away, a hole is milled into the package beforehand.

The milling step has to be done carefully to prevent damaging the bonding wires of the chip. Afterwards, a pipette is used to fill the hole with acid, which causes the plastic to carbonize. Since nitric acid also affects the pins of the package, it is necessary that the acid does not accidentally touch them. After an etching period of a few seconds, the acid and the solvents are removed in acetone by ultrasonic treatment. Etching and cleaning processes are repeated several times until the chip is exposed to the needs of the attack. Heating the acid and the device accelerates the etching process and decreases the number of repetitions.

3.2.1.2 Rear-Side Decapsulation

Applying a decapsulation procedure from the rear side does not involve any chemicals. A mill and a screwdriver are enough to access the substrate. First, the device package is milled down to the copper plate, which is part of a typical plastic package. The chip is mounted onto the plate during the packaging process. Second, the plate is removed using a screwdriver. The residues of the glue that stuck chip and plate together are removed by milling. The substrate is thinned carefully, using the mill, if required.

3.2.2 Practical Setup

In order to realise the attack we developed a software able to control an XYZ table, a camera and a frequency generator which control the laser pulse. The optical fault injection platform is showed in figure 3.24. The laser we are equipped with is an acoustic diode-pumped solid-state laser working on two different wavelengths, namely green

(at 532 nm) and infra-red (at 1064 nm). We use green laser to attack front-side decapsulated circuits, while infra-red laser is generally used to attack back-side decapsulated circuits.

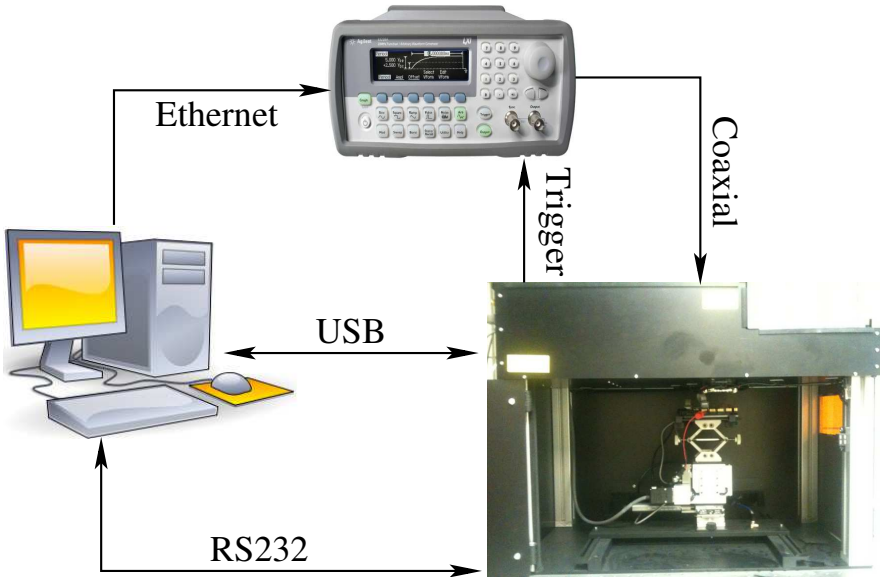


Figure 3.24: Optical fault injection platform

The attacked device is the ATMEL ATMega128 microcontroller, in which a software AES-128 was implemented using the SubBytes table stored in flash memory. Once the front side decapsulation done thanks to the CMP "Centre Microélectronique de Provence" we can see as shown in figure 3.26 the circuit is not a flip-chip, the next step to successfully inject fault's consist in localizing an area where the laser pulse can be injected without perturbing the whole circuit. To perform a local attack our approach is to scan the whole circuit by steps of $1\mu m$ and on every scanned point we change the laser power and the pulse width while it's ciphering. In fact using a laser at a high power level can irreparably damage the device. Indeed several zones (a data or address bus of the EEPROM) on the device are very sensitive and lead to complete freeze black zone in figure 3.26. Only some areas happen to lead to exploitable faults.

3.2.3 Experimental Results

A first random attack with a laser of 1ms period and 2 μs pulse width on the sensitive area, shows that we can induce single byte fault as needed for G. Piret and J.-J. Quisquater attack [30]. The temporal location is shown in figure 3.25 where we can see that almost all the rounds can be hit with a random localization.

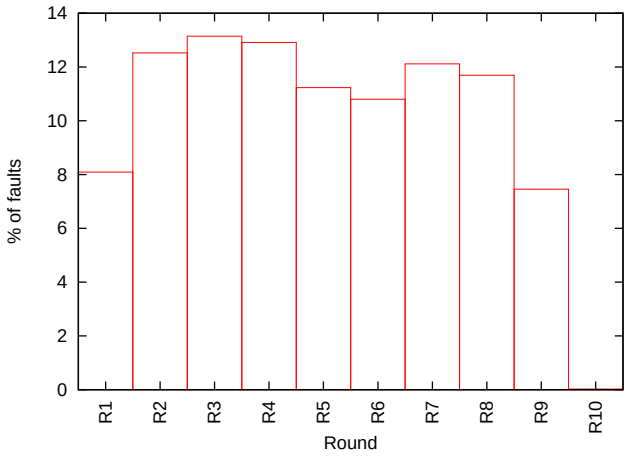


Figure 3.25: Temporal localization with random hit

To get a fault in a precise round the laser pulse needs to be synchronized. Consequently the AES code programmed within the ATMega is complemented with instructions allowing to output a trigger in order to launch the laser at a precise timing.

With the help of this synchronization we noticed that all the injected faults are exploitable except one among 10000. We attacked first the Byte S0 but by changing the synchronization at the beginning of the 8^{th} round it has been shown that other bytes can be affected as well. We have characterized the faults by testing all the hypotheses of the single byte error with the same software described in 3.1.3.

In order to realise a cartography of faults, we can collect 100 faulted ciphertext for each point that we analyse to find occurrence of single faults. The result is shown in figure 3.26, the cartography of the faults seems to be around an address or a data bus of the Flash memory. Another important point is that the power level $\approx 10mW$ of the laser does not affect the value of the fault injected, only the change of spatial location around the sensitive area has an impact on the value of the faulted Byte.

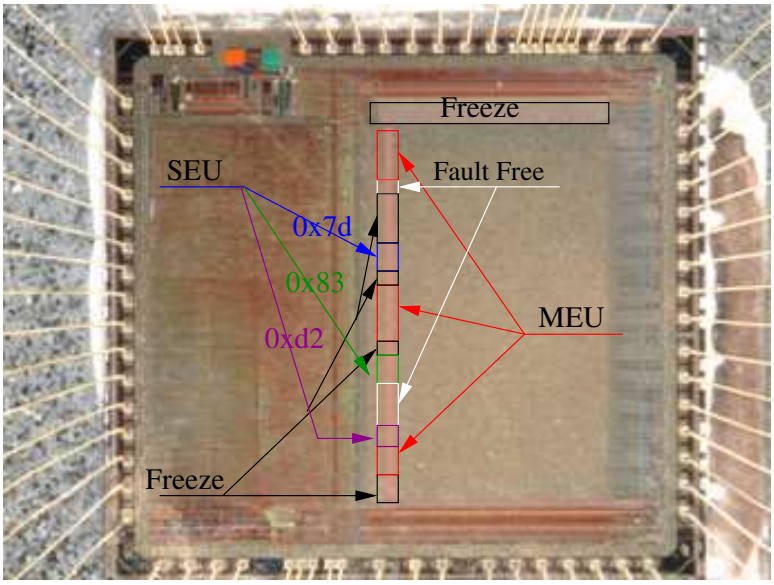


Figure 3.26: Cartography of faults

3.3 Conclusion

In this chapter practical attacks have been implemented and allowed us to understand the fault injection mechanism for both global and local faults. Two platforms have been specifically designed for this objective, one being dedicated for global fault attacks, the other for the local fault attacks. On each platform an home-made analysis software has been developed. It permits the control and the evaluation of the fault attacks.

The global fault injection is performed on the first platform. It is based on setup time violation caused by underpowering or overclocking the targeted device. This fault injection method is new, although particularly simple, in the academic community. In this method, the time and location of the occurrence of the faults are not controlled, making the attack particularly easy and cheap. It has been shown that both ASIC and FPGAs are vulnerable to this attack by using the Piret's algorithm. The single fault distribution among the rounds and the Sboxes is dependant on the targeted device implementation. The faults may be unexploitable if it does not occur at the correct round. The experiments have shown that this happens seldomly. This attack is relatively easy to thwart as simple countermeasures, as the use of voltage sensors, should detect such attacks. It is the case in smartcards but not in many other electronic

systems embedding cryptography.

The local fault injection method is based on a laser platform with a target device being a processor running a software implementation of AES. The laser power, the shot location and time to trigger the shot were controllable. This allowed to perform a fault cartography to determine the exploitable locations as well as the fault types. Successful attack on AES have been obtained by using transient ”stuck-at” faults on a specific area of the device.

In the next chapter we will present fault attack countermeasures. In particular a new protection strategy based on resilience rather than detection will be introduced.

Chapter 4

Fault Attack Countermeasures

In this chapter, we will discuss fault attack countermeasures for the Advanced Encryption Standard. In the early years of fault tolerance in secure embedded systems, analogue solutions were used. They consist in disseminating voltage, temperature, light sensors on the surface of the chip, and any miscellaneous combination thereof. The problem with this approach is that it requires a mixed design, which is much more complicated from a CAD perspective than a purely digital design. Also, the analogue parts are consuming a lot of power and area in the design. Those practical and economical reasons explain why the analogue this solution is no longer sufficient against the new threats.

Therefore modern designs resort to all-digital detection mechanisms. The generic ones exploit some artificial redundancy. It can be either implemented in time, space or information (code-based). All those strategies have been compared in [47], and shown to be roughly alike. Depending on the cryptographic scheme to protect, some dedicated countermeasures can also be implemented. The idea is to exploit some identities of the algorithm in order to detect possible errors with a high probability. In a typical encryption: the encrypted message can be decrypted and tested against the original plaintext. The same applies to digital signatures: the signature can be verified before being outputted. We underline that these verifications can represent a weakness *per se*,notably in front of so-called *safe errors* attacks [48].

However, the resilience against faults attacks has seldom be proposed. At the opposite, resilience in observation attacks is a hot topic. Following the proposal of Paul C. Kocher at CHES 2006 [49] to update the keys on a frequent and regular basis, ideas for side-channel resilient schemes have come up, as illustrated for instance by the

"Provable Security against Physical Attacks" workshop [50]. Actually, many techniques of reliability have been ported *as such* to security applications. Nonetheless the objectives of reliability and security differ:

- Reliability requires ideally that either the computations are correct or that an alarm is raised;
- Security requires that the computation result, if erroneous, carries no information about secret involved in the computation. This is a more flexible requirement than for reliability. On the one hand, it allows the system to output a false result C^* instead of the correct one C, as long as it reveals no information about the secret K. A formalization of security models under fault attacks can be done, and has actually already been initiated for instance by this paper [51]. From an information-theoretic perspective, the requirement can be stated as "*the mutual information between* (C, C^*) *and* K *is null*". Rising an alarm can even be a vulnerability in some contexts. For instance, the differential behavior analysis (DBA [35]) manages to extract a key simply by knowing whether or not the computation went well, provided the fault model is of "stuck-at" type and roughly reproducible. Fault Injection Resilience "FIR" has no concept of alarm, hence is immune against such attack methods.

4.1 Fault Detection

4.1.1 Parity

In [52] the authors describe a solution for the low cost concurrent error detection in the substitution-permutation network. The detection scheme is based on single parity bit which is propagated through the non-linear and the linear layer of the ciphers. Prediction through the linear layer can be very simple, reducing itself to a bunch of XOR as shown in figure [52]. The prediction for the non-linear layer "SubBytes" can be obtained by extending the tables storing the output values. This method can detect only single faults, moreover, the fault coverage is not impressive, since it about 96.3% for single stuck-at faults and the overhead is about 18%. In conclusion, this countermeasure may not be considered as sufficient to protect against a malicious attacker. This why we can add more parity bits in order to detect multiple faults as proposed in [53] to improve fault coverage, but faults of even order may still be masked with a non-negligible probability, the area overhead is 33%.

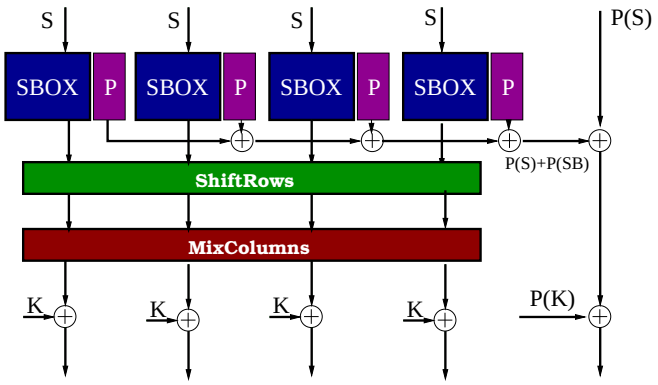


Figure 4.1: Parity based countermeasure

4.1.2 Concurrent Error Detection

In 2002 Karri et al. have proposed a solution [3] based on the involution property (inverse relationships that exist between encryption and decryption) to check if the condition $f^{-1}(f(x)) = x$ is respected through the cipher. We can implement CED at algorithmic level, round level or operation level as shown in Figure. 4.2. The most straightforward methods of performing CED are Hardware Redundancy and Time Redundancy. In Hardware Redundancy, two copies of the hardware are used concurrently to perform the same computation on the same data. At the end of each computation, the results are compared and any discrepancy is reported as an error. The advantage of this technique is that it has minimum error detection latency and it can detect both transient and permanent faults. A drawback of this technique is that it entails at least 100% hardware overhead. In Time Redundancy, the same hardware is used to perform both the normal and re-computation using the same input data. The advantage of this technique is that it uses minimum hardware overhead, in the other hand its drawbacks are that it entails 100% time overhead.

4.1.3 Cyclic Redundancy Check

In this solution [54], fault detection is based on the cyclic redundancy check (CRC) over $GF(2^8)$. This approach used the linear behavior of each operation in AES to design a detection scheme. This scheme only uses a $(n+1, n)$ CRC to detect the errors, where n is $4, 8, 16$ (depending on the implemented in 8-bit, 32-bit, or 128-bit architecture) and the parity of the output of each operation is predicted. Because AES is

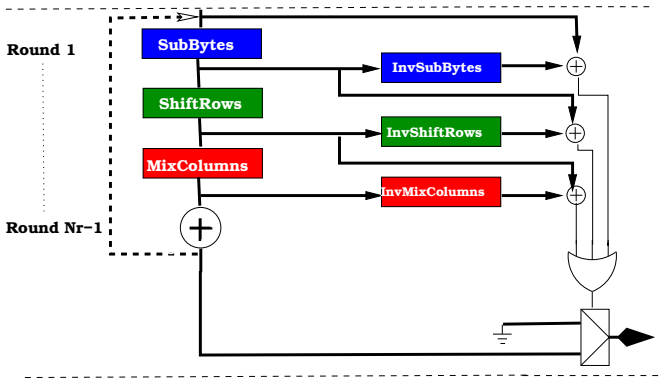


Figure 4.2: Concurrent error detection

byte-oriented and its constants are ingeniously designed, the parity of the output can be predicted from a linear combination of the parity of the input. In most cases, the parity is the summation of the input data; also, the schemes are highly scalable and are suitable for 8-bit, 32-bit, or 128-bit architecture. This is important because many AES designs are either 8-bit or 32-bit architecture. Another advantage of the proposed approaches is that the parity calculation between the encryption and the decryption is symmetric because the parity generation in encryption is quite similar to the one in decryption. This will bring some benefits while integrating encryption and decryption into one circuit. This method can also be used to protect the KeySchedule.

4.1.4 Non Linear Robust Code

Karpovsky et al presented in [55] two different solutions. In the first architecture, AES is devised into 2 parts non-linear and linear, where the non-linear block consists in the multiplicative of the Rijndael S-box. In order to detect a fault in this part we perform an inverse multiplication of the Sbox output, then we check only few bits(2 or 3) of the result in order to reduce the area overhead. The non-linear block is checked by computing a partial GF product: in practice, the input and the output of the inversion functional unit are multiplied together, and it is verified that the result is the field unit. The linear layer is protected by exploiting that the sum of the bytes of a single column is not affected by the Mixcolumns transformation, hence a 8-bit signature is used for each column.

In the second proposal [4], a robust non-linear code is described, based on the addition of two cubic networks, computing $y(x) = x^3$ in $GF(2^8)$, to the previous

*****	Slice	Frequency	Hardware Overhead
Non Protected	3856	53.29 Mhz	0%
Robust Nonlinear	5205	51.98 Mhz	35%

Table 4.1: Nonlinear Robust code implementation.

linear scheme. The method allows to produce r-bit signatures as shown in figure 4.3 to detect errors. This solution provides good error detection properties against faults of all multiplicities 2^{2-r}. The overall hardware overhead cost is about 35 % for a piplined architecture of AES that we have implemented on Altera Stratix FPGA the result is shown in Table 4.1.4. For parallel architecture like the one presented in [4] the total area overhead is about 77%.

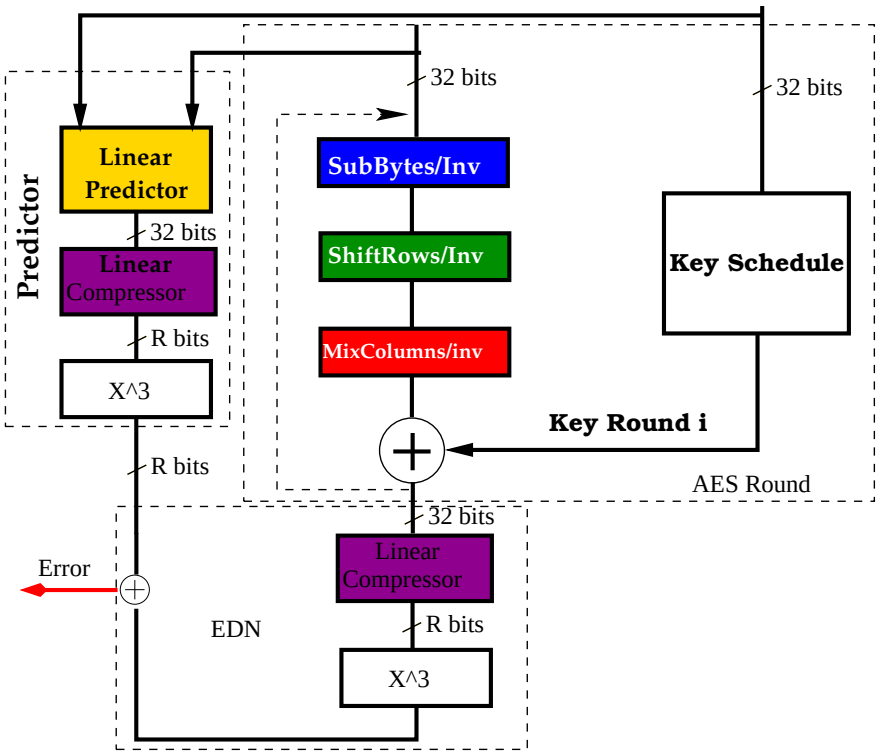


Figure 4.3: Robust code countermeasure.

4.1.5 Double-Data-Rate as countermeasure

In 2008 Maistri and Leveugle proposed a new countermeasure [56] against fault attack on AES. This countermeasure is based on time redundancy using Double Data Rate computation. This countermeasure uses both the rising and the falling clock edges to compute twice the ciphertext. In fact, registers are duplicated in order to create two parallel data paths, controlled by the clock edges, while the operation logic is shared between the two paths as shown in figure 4.4. The area overhead is 36 % for a piplined architecture and the throughput reduction is between 15% and 55%.

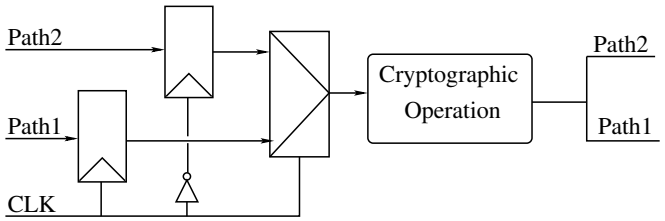


Figure 4.4: Double-Data-Rate as countermeasure

4.1.6 Low cost countermeasure against setup time violation attacks

A straightforward countermeasure against non-invasive global attacks on various circuits consists in inserting into the circuit some logic in charge of detecting abnormal situations before the critical parts of the designs become faulty. For instance, the figure 4.5 presents a setup consisting of a series of buffers followed by an inverter, making up a delay line, inserted between two registers. The source register value passes through the series of buffers, inverted by the inverter and then stored in the destination register at the next clock cycle.

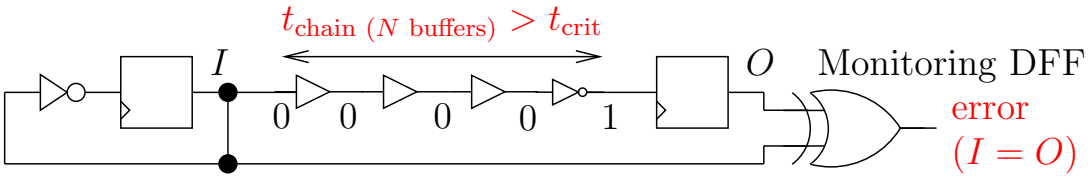


Figure 4.5: Counter-measure based on the insertion of a monitoring logic with a propagation time larger than the critical path of the rest of the circuit.

The source registers also receives the complement of its current value every clock cycle. For a circuit working in nominal conditions, the outputs of the two registers should be complementary. If this condition is violated, the circuit is faulty and should be immediately stopped. The number of buffers are chosen such that the delay of this chain becomes a little greater than the critical path of the targeted circuit. If the operating voltage is reduced, the delay chain will be violated before the actual circuit and an alarm is raised before the cryptographic parts of the design become faulty.

The chain should be implemented in such a way that it operates at the same clock as the protected circuit and driven by the same source voltage. We implemented this countermeasure on an Altera Stratix FPGA. Instead of using RTL buffer, we used "lcell", the stratix primitive cell. The advantage of using lcell is that the user is sure that synthesis tool will not remove or shorten the length of the chain while optimization.

We analyzed the chain in order to find a relation between the length of the chain and the faulty voltage. Figure 4.6 shows the voltage of the setup violation in function of the lcell number used in the chain. It is clear that the violation voltage increases more or less linearly with the number of buffers.

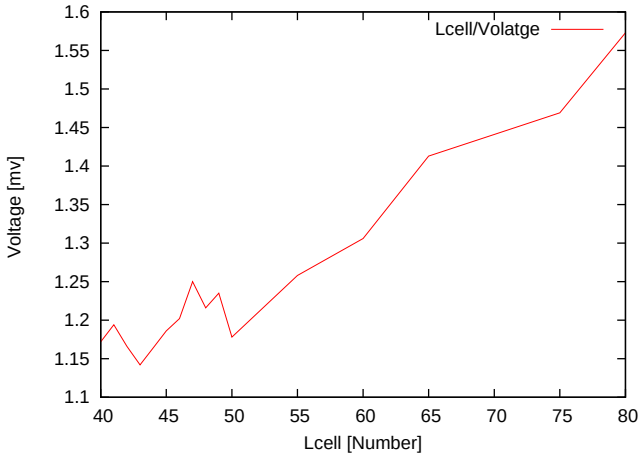


Figure 4.6: Chain Voltage/lcell.

Table 4.2: Classical fault detection characteristics.

		Ciphertext incorrect?	
		Yes	**No**
Alarm raised?	**Yes**	Safe	Problem of **availability**
	No	Problem of **security**	Safe

4.2 Fault Resilience

4.2.1 Comparison between Detection and Resilience

Neither detection nor resilience schemes are able to withstand all the faults. Indeed, whatever the protection mechanism, we can theoretically build an attack (possibly adaptative) able to replace an authentic value with another one. The goal of the countermeasure is to make this substitution very chancy.

In this subsection, we investigate the side-effects of the countermeasures. The detection strategy suffers from two drawbacks illustrated in Tab. 4.2. First of all, the device can raise an alarm even if the result is correct. This is the case when the fault happens on a variable that does not impact the output. This situation is of course not true in general, otherwise the variable could have been removed from the implementation. However, in the course of a specific computation, this is possible. One trivial example is the result of an `AND` gate, that has zero for one input, and that is faulted on its second input. The fault will not be propagated and the result will be correct irrespective of the fault taking place or not. However, if a detection mechanism raises an alarm, then the whole computation will be stopped and adequate actions will be undertaken, thus causing a denial of service (DoS) despite the absence of a security problem. Second, detection mechanisms do not cover all the possible faults, and some faults can propagate without being detected.

On the contrary, an ideal resilient scheme will feature:

- **an optimal availability**: false detections do not exist, since errors are not caught but propagated.

- **an optimal security**: the fault generates a wave of erroneous data independent of the previous pristine (and sensitive) values. Therefore no sensitive information is propagated.

Also, in terms of coding and deployment guidelines, the advantages of resilience as opposed to resistance (fault detection) are manifold. We can really claim that resilience is a new security approach to protect cryptography, because of these typical improvements:

- In traditional designs, miscellaneous checks are scattered in the code. For instance, ratification counters and baits are usual tricks to detect "blind attacks". No such extra operations are required in the context of fault resilience, since it is not catastrophic that the IC fails. To be perfectly clear, such subterfuges are more *palliative* than *curative*. They notably hinder automatic or formal code expertise, although some applications would demand such a high confidence evaluation level.

- When using detection, faults can also occur in the detection logic. But then, the problem becomes eventually insolvable, since more and more logic is necessary (by recursion, we need detection logic for the detection logic, itself being protected by detection, *etc*.)

- On top of that, the resilience relieves the designer from having to deal with the reactions to the threat. These features are all in one very annoying for the chip manufacturer; if they are activated unexpectedly they possibly ruin the device, causing large costs to replace the defective cards. Now, the secure chip manufacturers are often balancing between activating the maximum level of countermeasures and risking card auto-scuttling (false positive)[1]. Such a dilemma does not exist with fault resilience. The card starts to produce faulty results while under stress (either because of an attack or because of a natural hazard), but returns to its nominal operating conditions as soon as the stress disappears. Thus the risk of having a permanent damage due to a false alarm is merely nonexistent. This point is exemplified in Fig. 4.7

[1] Remember that early countermeasures against faults were intended to make up for the poor quality card readers, that inappropriately injected unwanted electrical glitches in the smartcards! Also, Ross Anderson and Markus Kuhn explained in [57] that the wild fluctuations in clock frequency that frequently occur when a card is powered up and the supply circuit is stabilising, caused so many false alarms that the [detection] feature is no longer used by the card's operating system.

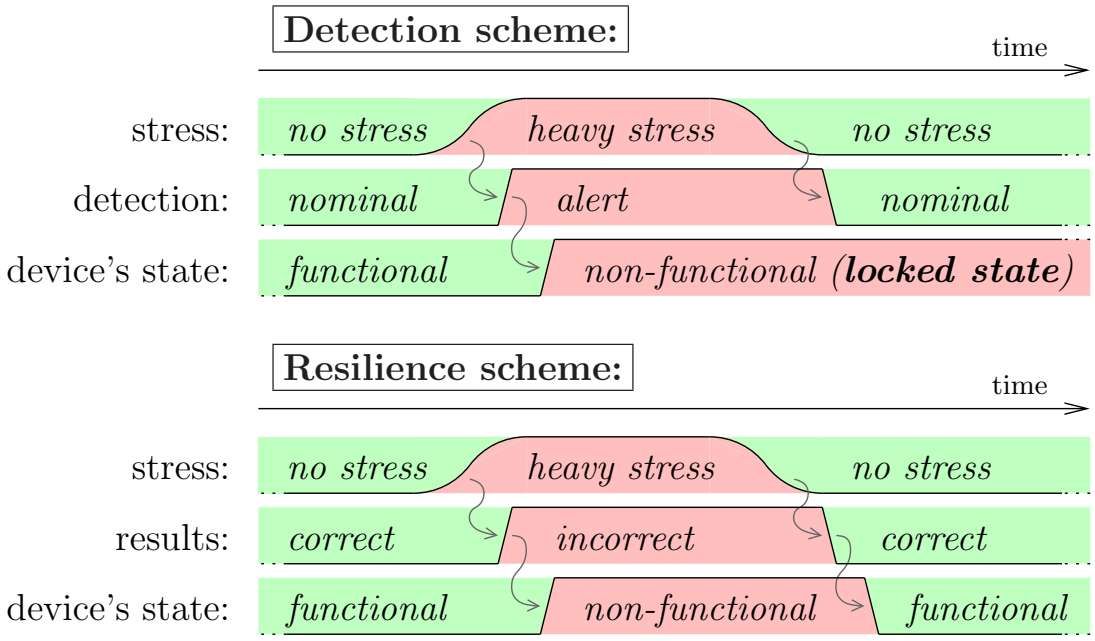


Figure 4.7: Suicide in case of fault detection (*top*), opposed to survival in case of fault resilience (*bottom*) protection schemes.

4.2.2 Further Merits of the Fault Injection Resilience "FIR"

One feature that gives to FIR a remarkable strength is its agnosticism with respect to attacks. By making any faults independent at its source and during its propagation independent of the previous values, it merely prevents any attack at their root. Therefore, new scenario schemes not envisioned yet are thwarted proactively, which provides a forward security. Typically, most – if not all – attacks studied so far are differential: they assume the attacker knows couples of correct & faulted computations. Now, higher-order attacks could as well be possible: they would imply more than one faulty result. Additionally, faulted ciphertext-only attacks could also be devised. FIR fights all those future new threats that a pure DFA counter-measure would maybe fail to cover.

4.2.3 Related Works

Earlier publications have noticed the interest of allowing cryptographic devices to output faulty results, without jeopardizing their security. However, all those results focused on asymmetric cryptography, and more specifically on RSA. A fault tolerant

RSA with CRT[1] algorithm is given and formally proved in [58]. This article introduces the concepts of "*fault infective CRT computation*" and "*fault infective CRT recombination*". the algorithm is designed to have the errors occurring during the "mod p" half propagate in the "mod q" half, and *vice-versa*, thus denying the Bellcore [59] attack. This idea is definitely a FIR, albeit crafted to the case of RSA and more specifically against the Bellcore attack, whereas, the FIR scheme we present is algorithm-agnostic.

Other formal ways to secure sensitive algorithms have been proposed. For instance, the paper [60] about "Algorithmic Tamper-Proof" (ATP) explains how to protect an implementation, by the specification of security requirements on the circuit and by restricting the power of the attacker. A cryptographic module implementing the FIR is definitely not protected in the context described in paper [60]. We would like to make clear that the FIR notion introduced applies to a system that has a trusted environment: the asset at risk is therefore only the cryptographic core. In other terms, the two methods ATP and FIR do not consider the same security boundary.

4.2.4 Some Practical Implementations of FIR

The purpose of this section is to provide with some resilient cryptographic schemes. For the sake of clarity, we focus on the protection of symmetric block encryption modules. Indeed, as they are deprived by construction from any algebraic properties, they are also the most difficult ones to protect. The state-of-the-art in asymmetric algorithms protection is very well advanced and formally proved. An overview, on the example of RSA, can for instance be found in these papers [61, 62].

In the subsection 4.2.4.1, we provide with a protocol-level resilient scheme. The subsection 4.2.4.2 rather introduces two gate-level solutions.

4.2.4.1 Formal Counter-Measures against Fault Injection Attacks

A differential fault analysis (DFA [63]) requires the same plaintext to be encrypted twice with the same key. Common attack scenarios consider the case where the attacker is able to inject one fault in only one of the encryptions. Then, she can deduce information about the key using a DFA. Thus, DFAs are made impossible if an attacker

[1]The computations "mod $n = p \cdot q$" are done separately "mod p" and "mod q", and then combined back. This processing – possible only for the owner of the private key – speeds up the overall computation by a factor of four.

is not able to request twice the same encryption. It is possible to devise such a scheme, as typified by algorithm (8).

Algorithm 8: Probabilistic Encryption Algorithm built on top of AES.

Input : A plaintext x to be encrypted with the key k.
Output: A ciphertext with a random number.

1 Determine a random number r of the same size as x.
2 Return the couple $(y = \text{AES}_k(x \oplus r), r)$.

This algorithm (8) is considered as secure against DFA because the probability that two encryptions are generated with the same plaintext is roughly speaking $2^{n/2}$, where n is the entropy of x or r. Indeed, this is a classical instance of the birthday paradox.

We mention additionally that the scheme of algorithm (8) protects against a broader class of attacks than only the DFAs. It is a random encryption scheme, that has the remarkable property that the attacker cannot decide if the encryption is actually faulty or not.

Unfortunately, this scheme is not secure in decryption. As a matter of fact, the decryption algorithm corresponding to (8) is given in algorithm (9). This algorithm can be called repeatedly without the AES inputs being modified:it is deterministic.

Algorithm 9: Deterministic Decryption Algorithm matching algorithm (8).

Input : A ciphertext under the form $(y = \text{AES}_k(x \oplus r), r)$ to be decrypted by the AES key k.
Output: The plaintext x.

1 Decrypt y with key k: $z = \text{AES}_k^{-1}(y)$.
2 Return $z \oplus r = x$.

This situation can however be exploited to protect low cost embedded systems, such as smartcards or RFID tags, that communicate with a larger device, such as a reader. It is fairly easy to protect the reader against fault attacks by "physical tamper-proof measures". For instance, the reader's electronic circuits can be imprisoned into a mold, protected with a pasted metallic cover and sealed into a box equipped with intrusion detection sensors. The same level of sophistication is impossible for smartcard or tags modules, because their form factor is extremely constrained in size (due to stringent

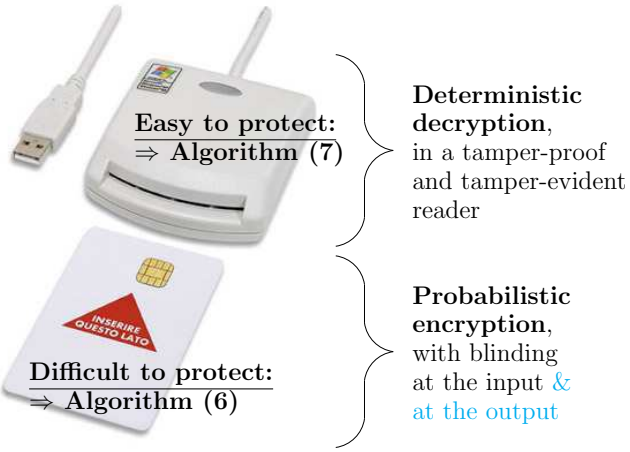


Figure 4.8: Probabilistic encryption and deterministic decryption

requirements about the mechanical strength edicted by ISO 7816-1). Therefore, the attacker will most certainly prefer to attack the embedded system to extract the shared secret key. Thus, if the reader plays the decryption (9) and the embedded system the encryption (8), the unbalance between the tamper-resistance of the two devices is made up by the opposite unbalance of the algorithm, in terms of resistance against DFA. This strategy of reinforcing the security by algorithmic means of the weakest element in the security chain is illustrated in Fig. 4.8.

Notice that if a handy homomorphous encryption algorithm HEA is available, a completely secure encryption/decryption scheme can be devised. Let us denote by $\mathrm{HDA} = \mathrm{HEA}^{-1}$ the corresponding decryption algorithm and $\times$ the composition law in the group of homomorphy:

$$\forall y_1, y_2, \;\; \mathrm{HDA}(y_1 \times y_2) = \mathrm{HDA}(y_1) \times \mathrm{HDA}(y_2)\,.$$

The encryption proceeds as per algorithm (8) using HEA instead of AES, whereas the decryption consists in algorithm (10). This scheme can use for instance RSA or Paillier's cryptosystem [64] as underlying encryption primitive.

The resilient algorithms presented in this subsection 4.2.4.1 have the drawback that the size of the ciphertext is doubled. This can be a limitation for instance in contactless cards authentication, where the transmission time must remain short. Also in wireless sensor networks the increasing of the data transmitted means a very high cost in terms of power.

Algorithm 10: Probabilistic Decryption Algorithm matching (8) with HEA instead of AES as underlying cipher.

Input : A ciphertext under the form $(y = \text{HEA}_k(x \oplus r), r)$ to be decrypted by the HEA key k.
Output: The plaintext x.

1 Determine a random number s of the same size as y or r.
2 Return $\text{HDA}_k(y \times s)/\text{HDA}_k(s) \oplus r = x$.

Nonetheless the algorithm (8) can be made more bandwidth and power-efficient if the message x to encrypt is cut into several blocks.

In this case, alternative encoding, such as the probabilistic all-or-nothing transform (AONT) described in [65, 66], could be taken advantage of. With respect to other probabilistic symmetric encryption schemes (most of the times, the encryption involves a random IV – which is short for *initialization vector*), this AONT scheme is original in the sense that the randomness is not disclosed along with the ciphertext. This denies the possibility to conduct a side-channel attack on the first round(s) of the encryption algorithm. A similar scheme has also been described in [67]. As such, this all-or-nothing scheme (in general, but also under the form of its "Probabilistic Signature Scheme", *aka* PSS, avatar [68]) is an implementation of FIR. In addition, it reduces the number of blocks to be exchanged to the number of plaintext blocks plus one. In summary, algorithm (8) combined with [65] has the benefit of bringing a SCA-resistance in addition to the FIA-resilience. Certainly, this suggestion of protocol-level countermeasure can be optimized, but we leave this topic open for future works [69].

4.2.4.2 Multi-Valued and Redundant Representation Logics

Multi-valued logics allow to encode more than one bit with one electrical state. It is for instance used in some power-constant logic styles [70]. Let us consider the case of an equipotential holding three states, denoted `0`, `1/2` and `1`, amongst which only the two `0` and `1` are functional. Then, if a fault turns a valid value into `1/2`, the provenance has been forgotten.

The same goes for redundant logics, such as the m-out-of-n representations (for $0 < m < n$). For instance, the 1-out-of-2 representation, also known as dual-rail with precharge logic (DPL), admits two valid states, denoted by `01` and `10`, and two invalid

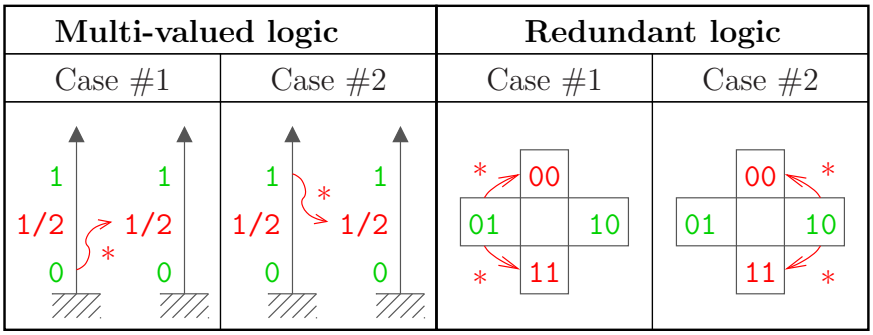


Figure 4.9: Two kinds of faults for 3-valued logic and for DPL,

states, denoted by `00` and `11`. In the case one fault turns a valid token into an invalid one, the value before the fault is lost. The effect of faults on these two logic styles is summed up in Fig. 4.9. It clearly appears that the state after the fault is decorrelated from the initial state, thereby establishing the resilience, for the relevant cases where the data is sensitive.

Now, the resilience only works in the case the attacker fails to inject "false valid" faults, *i.e* $0 \overset{*}{\leftrightarrow} 1$ faults in multi-valued logic or $01 \overset{*}{\leftrightarrow} 10$ faults in DPL. Let us assume this situation is rare. It seems all the more difficult to achieve in DPL because the attacker must produce two antinomic concerted faults.

As will be exposed into greater details in Sec. 4.2.5, the resilience will build up each time a valid false is produced along with invalid faults. In this case, the two faults will propagate, and if the logic favors the generation of invalid instead of valid states, then the diffusion of the netlist will encourage the invalid states to hide the false valid states. This case is optimal if the logic meets this requirement:

> "if any input is invalid, so is the output".

This behavior is "saturating"; the faults will percolate in the netlist and the invalid values will saturate most of the nets, thereby absorbing all the false valids that are crossed. So the resilience is amplified by the diffusion in the netlist and the collaborative behavior of gates to favor invalid values propagation. This phenomenon of invalid values (dominant) suppressing false valid values (recessive) is further detailed in the next section 4.2.5.

4.2.5 Dual-Rail with Precharge Logic as a Global Countermeasure against Implementation-Level Attacks

DPL styles are solutions primarily designed to protect a cryptographic implementation against side-channel attacks. However, it has been noticed that these styles can also natively withstand some perturbation attacks [71, 72, 73, 74]. Unlike traditional counter-measures against fault attacks, the DPL does not implement a protection, but is rather resilient. This means that faults are not caught, but rather left free to cascade their effect, knowing that eventually their observable consequences will not be harmful.

4.2.5.1 Requirements for Simultaneous SCA and FIA Protection

In order to better illustrate the close relationship between observation and perturbation attacks, we need to notice that security perimeters depend on the application. For instance, in an ISO/IEC 7816 compliant smartcard, several security violation situations can be encountered.

- The critical part is the memory in case of an external authentication. Indeed, if the memory can be corrupted, then any rogue reader can be forced to be seen as authentic. Here, there is no secret to retrieve, but simply an invalid state to be setup by force.
- However, during an internal authentication, the smartcard uses its cryptographic secret. Therefore, the risk for the smartcard is to have its key retrieved illegitimately. Differential fault attacks and side-channel attacks are two tools available to recover the key. In addition, as the protection against attacks is expansive, the designer will try to partition the cryptographic block at risk. Typically, when it implements symmetrical encryption, this block can be split into:
  - a control part, subject to fault attacks, such as round reduction attacks [75], but leaking no sensitive information as the algorithm is supposed to be known by the attacker (common assumption with Kerckhoffs' law), and
  - a data processing part, subject to both fault attacks, such as DFAs [30, 63], and side-channel attacks, such as DPA [16].

The overall requirement for security against implementation-level attacks in a smartcard is depicted in Fig. 4.10. This block-diagram shows in red the security boundary

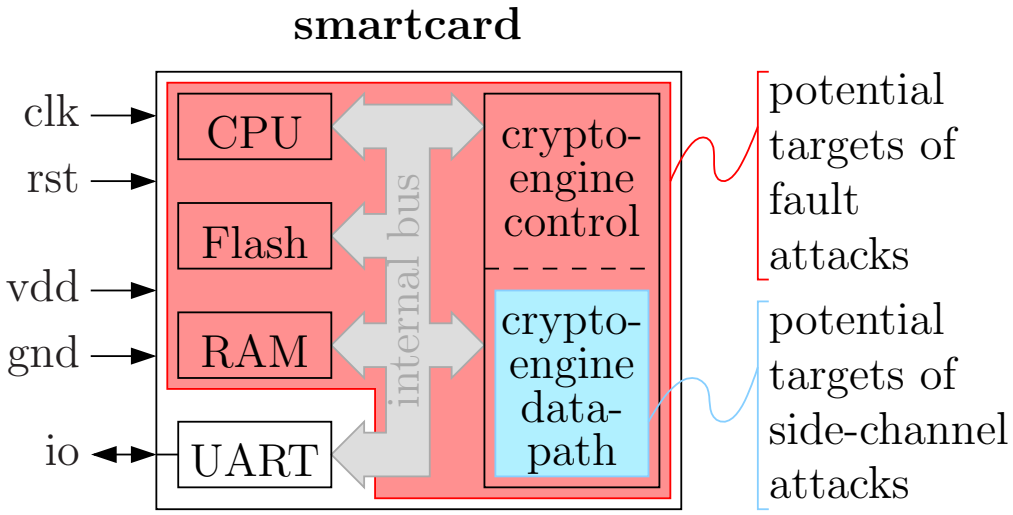


Figure 4.10: Susceptible organs of a smartcard in two representative sensitive operations (`EXTERNAL AUTHENTICATE` and `INTERNAL AUTHENTICATE`). Typically, the cryptography will be triple-DES or AES.

for fault attacks and in cyan that for SCAs. It appears clearly that some organs shall be protected only against fault attacks, but that all the organs that shall be protected against SCA must also be protected against FIA. This is an advanced question, all the more important as it is in this part of the design that the largest overheads are expected.

The countermeasures against SCA include:

- Information masking, implemented with random splitting of data into shares,
- Information hiding, implemented with DPL.

More information about these two categories of protection against SCAs can be found in the "DPA book" [76], respectively at chapter 7 and 9.

Amongst this array of possible protections, DPLs [77, 78] are of particular interest because they have native protections against DFAs.

We will thus focus in this section on the combined DFA and SCA protection of the datapath of cryptographic modules; The type of fault attacks we consider are those described in [79], the two most famous of them being that of Biham & Shamir [63] (DES) or Piret & Quisquater [30] (AES), enhanced by Tunstall in [34].

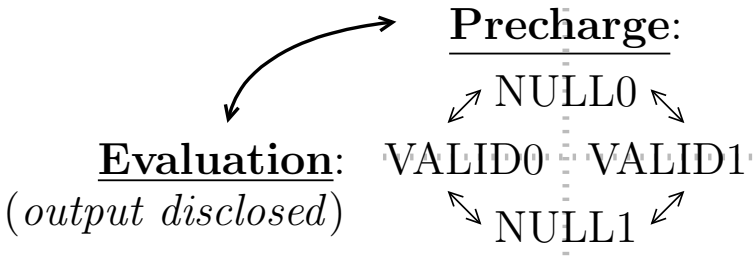


Figure 4.11: DPL protocol.

Another motivation to focus on the crypto-datapath is that it is usually the most complex design part; therefore it represents the largest area and contains the longest critical timing paths. This explains that local faults are more likely to target the datapath because of its predominant surface, and that global faults also affect preferentially the datapath that is most tight in meeting the setup time constraint.

4.2.5.2 Previous Art about DPL in the Presence of Faults

We use the following notations for the DPL representation. Every logical variable a is represented by a couple (a_f, a_t) of wires, that carry two values. The semantic of the four possible combinations is detailed below.

- a is VALID if $\boxed{a_f \oplus a_t = 1}$. More precisely, VALID $\doteq \{\text{VALID0}, \text{VALID1}\}$ or VALID $\doteq \{(1,0),(0,1)\}$.
- a is NULL if $\boxed{a_f \oplus a_t = 0}$. More precisely, NULL $\doteq \{\text{NULL0}, \text{NULL1}\}$ or NULL $\doteq \{(0,0),(1,1)\}$.

The two NULL states are used alternatively with the VALID ones as precharge stage, so that the next evaluation starts afresh from a known state. The DPL protocol is recalled in Fig. 4.11.

There are two flavors of DPL, depending they feature the early propagation effect (named EPE in the literature, discovered independently by [80, 81]) or are protected against it. The definition of those variants can be summarized by the following conditions to be fulfilled by all the instances f:

- **DPL w/ EPE**: $\exists a$ VALID, $f(a, \text{NULL}) = \text{VALID}$;
- **DPL w/o EPE**: $\forall a$ VALID, $f(a, \text{NULL}) = \text{NULL}$.

In DPL, only results on *evaluation* are observable, because *return to precharge* faults are not outputted. We adopt the following faults typology on DPL:

- **Asymmetric faults**: {VALID0, VALID1} $\xrightarrow{\downarrow}$ NULL0, triggered by **global** perturbations (*e.g.* caused by a setup time violation due to power/clock glitch, overclocking or under-powering);
- **Symmetric faults**: {VALID0, VALID1} $\xrightarrow{\downarrow \text{ or } \uparrow}$ {NULL0, NULL1}, triggered by **local** perturbations (*e.g.* caused by injection of high energy laser light, electromagnetic field or particles beam).

4.2.5.3 *DPL w/ EPE* is Protected against Multiple *Asymmetrical* Faults

WDDL [82] is a typical DPL w/ EPE style. In this logic, the `AND` function is defined as: $(y_f, y_t) \doteq (a_f + b_f, a_t \cdot b_t)$. We use the following color code in Boolean truth tables:

- gray: the regular truth table in the absence of faults (*i.e.* the intended functionality),
- purple: anticipated values (evaluation even if not all inputs are valid).

Otherwise, green and red still represent respectively correct and incorrect behaviors or properties.

As shown below, WDDL can propagate correct valid results in the presence of asymmetrical faults.

b \ a	VALID0	VALID1	NULL0
VALID0	VALID0	VALID0	VALID0 (**EPE**)
VALID1	VALID0	VALID1	NULL0
NULL0	VALID0 (**EPE**)	NULL0	NULL0

This behavior is positively resilient. It is that of the `U`nitialized value in VHDL enumerated type `ieee.std_logic_1164.std_ulogic`, recalled below:

b \ a	`'0'`	`'1'`	`'U'`
`'0'`	`'0'`	`'0'`	`'0'`
`'1'`	`'0'`	`'1'`	`'U'`
`'U'`	`'0'`	`'U'`	`'U'`

where the tokens {VALID0, VALID1, NULL0} implement respectively the items {`'0'`, `'1'`, `'U'`}.

These conclusions can be challenged in the case of a coupling of the fault injection analysis with a side-channel analysis. For instance, the fault sensitivity analysis (FSA [36]) can, under some circumstances, exploit the unbalance within the two wires making up a dual-rail pair. However, the FSA has only been demonstrated as partially successful on a WDDL chip.

Actually, this FIA-resistance solution has already been sketched in [83]. This article introduces two methods to protect circuits against FIAs.

The first one consists in resisting to an arbitrary number of "stuck-at-0"[1]. Those "*reset faults*" correspond to our "*asymmetric faults*". However, this publication is overly conservative; invalid tokens are generated even if the data is not tainted. Also, the authors of [83] add a series of cascade gates at the output of the circuit. Their role is to turn all other valid tokens to invalid ones. Additionally, they request that the circuit commits suicide at this point (when the ciphertext is all NULL, noted "$\perp$" in [83]). Our key remark is that those two requirements are actually overkill. Indeed, the overall security is not jeopardized if some valid and some invalid tokens are outputted; therefore, we can save the cascade stage. In addition, we insist that it is then useless to permanently destroy the circuit: as we know the attacker only gets faulted crypto results that do not convey any information about the sensitive variables, it is safe to continue without erasing the secrets, that are merely not compromised. Therefore, the scheme we present is more user-friendly, in the sense it keeps the application up-and-running unless a fault is indeed influencing the result.

The second countermeasure against arbitrary faults in [83] is more *ad hoc*, since one needs to know the maximum number of faults an attacker can inject to dimension the level of protection (based on an adaptively sized countermeasure). In the next paragraph, we study FIR in the presence of multiple symmetric faults.

4.2.5.4 *DPL w/ EPE* is not Protected against Multiple *Symmetric* Faults

To start with, we assume neither $a \xrightarrow{*} \bar{a}$ nor $b \xrightarrow{*} \bar{b}$ happens. However, even in this favorable case, WDDL can generate incorrect false results. They are presented by skulls (symbol: ☠) in the following table.

[1] ...or equivalently "stuck-at-1" for all the faults.



Figure 4.12: Two DPL w/ EE drawbacks to fight DFAs, illustrated on the example of a WDDL AND gate. In this figure and in the subsequent ones, the asterisk character (*) symbolizes the faults.

b \ a	VALID0	VALID1	NULL0	NULL1
VALID0	VALID0	VALID0	VALID0 (**EPE**)	VALID0 (**EPE**)
VALID1	VALID0	VALID1	NULL0	NULL1
NULL0	VALID0 (**EPE**)	NULL0	NULL0	VALID0 (☠)
NULL1	VALID0 (**EPE**)	NULL1	VALID0 (☠)	NULL1

For instance, the twain simultaneous errors:

1. $a = \text{VALID1} \xrightarrow{*\uparrow} a = \text{NULL1}$ and
2. $b = \text{VALID1} \xrightarrow{*\downarrow} b = \text{NULL0}$

trigger a dreadful transformation: VALID1 $\xrightarrow{*}$ VALID0.

Therefore, because of EPE, logical inversions $f(a,b) \xrightarrow{*} \overline{f(a,b)}$ can occur, which makes FIAs (such as DFAs) possible.

The implication is that DPL in itself does not provide a good protection against symmetrical faults. As a matter of fact, it can filter out a NULL (see Figure 4.12(a)) and generate a faulted VALID from NULL tokens (see Figure 4.12(b)). In contrast, the DPL styles that are EE-free propagate the NULL unconditionally; this feature is even part and parcel of the DPL w/o EE specification. Additionally, the NULL (behaving like an `'X'`) always absorbs other VALID faults, as shown in Figure 4.16.

4.2.5.5 *DPL w/o EPE* is Protected in front of Multiple *Symmetric* Faults

Now, the DPL w/o EPE styles are protected against multiple symmetric (hence asymmetric) faults. This is shown in the table below.

b \ a	VALID0	VALID1	NULL0	NULL1
VALID0	VALID0	VALID0	NULL0	NULL1
VALID1	VALID0	VALID1	NULL0	NULL1
NULL0	NULL0	NULL0	NULL0	NULL1
NULL1	NULL1	NULL1	NULL0	NULL1

Remark that if we call:

- `'0'`: VALID0,
- `'1'`: VALID1,
- `'X'`: NULL $= \{\text{NULL0}, \text{NULL1}\}$,

then we have the same behavior (*i.e. "propagate always"*) as VHDL. This is illustrated below:

b \ a	`'0'`	`'1'`	`'X'`
`'0'`	`'0'`	`'0'`	`'X'`
`'1'`	`'0'`	`'1'`	`'X'`
`'X'`	`'X'`	`'X'`	`'X'`

Finally, we note that even if a few mutations $a \xrightarrow{*} \overline{a}$ exist for some variables a, it is very likely that the `'X'` wave caused by $a \xrightarrow{*}$NULL *eats* them. As detailed in the next sub-section, the recessivity of `'X'` over NULL, coupled with the avalanche of `'X'` caused by the diffusion property of the logic, accounts for that.

4.2.5.6 Revisiting the Comparison Resilience vs. Detection

One can argue that the DPL used as a FIR is in fact a very low-grain fault detection scheme. Indeed, FIR shares with the detection strategy the fact that redundancy is required. However, it is coupled to a diffusion that makes the detection at one stage take advantage of the rest of the stages. This detection is propagated in a wave, that constitutes a collaborative strategy that is absent from the pure detection schemes. This difference is illustrated in Fig. 4.13. In traditional detection schemes, the computation (noted: C) and the detection (noted: D) logics are dissociated. In particular, the detection blocks do not communicate. In the DPL FIR scheme, the computation and the detection are merged (noted: C+D) and this information propagates downwards the netlist.

There are two properties of DPL that help resilience:

- The **redundancy** of the netlist. At an n-bit output of a combinational block, only 2^n amongst the 2^{2n} possible ones are valid.

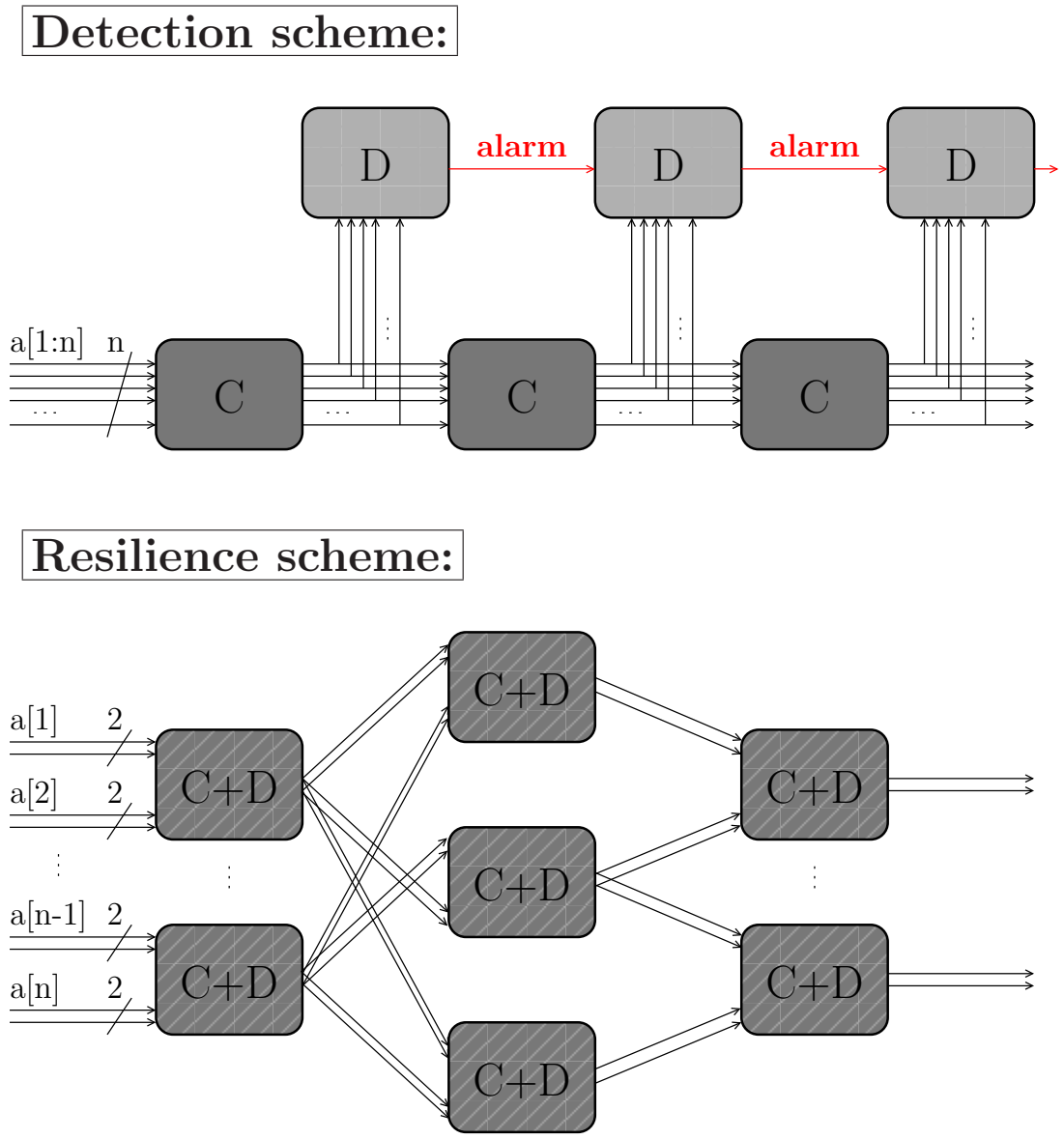


Figure 4.13: Difference of detection and resilience working factors, represented on an example netlist.

- The **diffusion** within the netlist, which is characteristic to the cryptographic algorithms. This property is especially true at the netlist level for logics free from EPE [74]. Indeed, the fanout of each gate is double w.r.t. separable logics such as WDDL [82].

Current detections schemes work independently of the computation and in a non-collaborative way. At the opposite, FIR consists in intricating the detection agents with the computation and to tightly interconnect them. The objective is to trigger a proliferation of tamper-evidence logic markers (NULL tokens).

4.2.6 Cost Estimation of FIR versus Traditional Approaches

The traditional approach to counteract implementation-level attacks is a composition:

- first use detection schemes, that can be inserted early at the RTL of the algorithm [84];
- then map this FIA-aware RTL description into a SCA-proof logic style. Indeed, the detection logic manipulates sensitive variables, and might itself leak secrets [85]. Therefore, it deserves a protection against SCAs. In a similar fashion, the study reported in paper [86] confirms that gate-level countermeasures against FIAs do not reduce the information leakage.

This implies that the overhead of the FIA and SCA countermeasures get multiplied.

A typical overhead for FIA countermeasures can be found in [47]. Let us consider the case of a non-linear code, such as [87], that is suited to detect multiple faults. Its overhead is 77 % in area and 15 % in throughput.

As such, those performance losses are more affordable than those required to thwart SCAs. For instance, WDDL incures an increase of 3.1 in area and 3.9 in throughput [88].

The combination of [87] and [88] results in an increase of 5.5 in area and 4.5 in throughput.

Those results are to be contrasted with the FIR approach using an EPE-proof DPL style. This style already merges FIA and SCA countermeasures. The reported overheads for two of those logics are given in Tab. 4.3.

Table 4.3: Performance overhead of different SCA+FIA countermeasures.

Strategy	**Detection + DPL**	**Resilience = DPL**	
Countermeasure	[87] + [88]	DRSL [101]	IWDDL [102]
Area	5.49 ×	2.56 ×	4.34 ×
Throughput	4.49 ×	2.00 ×	1.53 ×

It clearly appears that using a symbiotic SCA+FIA countermeasure is more efficient than combining two countermeasures one on top of each other. We notice that those alternative "DPL without EPE" logics yield similar performances: iMDPL [89], STTL [90, 91], SecLib [92, 93, 94, 95] and WDDL w/o EPE [74] and BCDL [96, 97].

We also attract the reader's attention on the fact that asynchronous logics, especially the quasi-delay insensitive (QDI) style [71, 75], can be implemented in DPL [98]. Now, asynchronous logic is designed to remain functional irrespective of the environmental variations. Concrete work [99] on this topic had been carried out in the framework of the G3Card project [100]. However, the G3Card consortium only detects NULL1 as an error marker in a DPL protocol where the only allowed spacer is NULL0. This signalization is restrictive and do not consider propagation of errors; instead, an instantaneous detection is suggested, which seams hard to put in practice in *real-time* given that such checks shall be done for each and every gate of the design. Moreover, asynchronous QDI logics have a drawback in terms of resilience: each gate being sequential in nature (due to the necessary handshakes with the upstream fanin and downstream fanout gates), a fault can cause a deadlock, should the fault cause a protocol violation (*i.e.* the transitions depicted in Fig. 4.11 are not respected). To relieve the circuit from this deadlock, the asynchronous circuit shall be reset. Thus the resilience provided by an asynchronous circuit is in-between the two cases illustrated in Fig. 4.7. The card is not destroyed permanently, since a reinitialization relaunches it; however, the system must detect that the logic hung (perhaps with the help of a watchdog) in order to restart it. Despite of these discrepancies with the FIR concepts, we note that QDI still increases the number of situations where the circuit remains functional, while remaining "resilient" if the external conditions are too harsh.

Eventually, we wish to underline that these overheads are not that dramatic when contrasted with those encountered in other domains that also require dependability

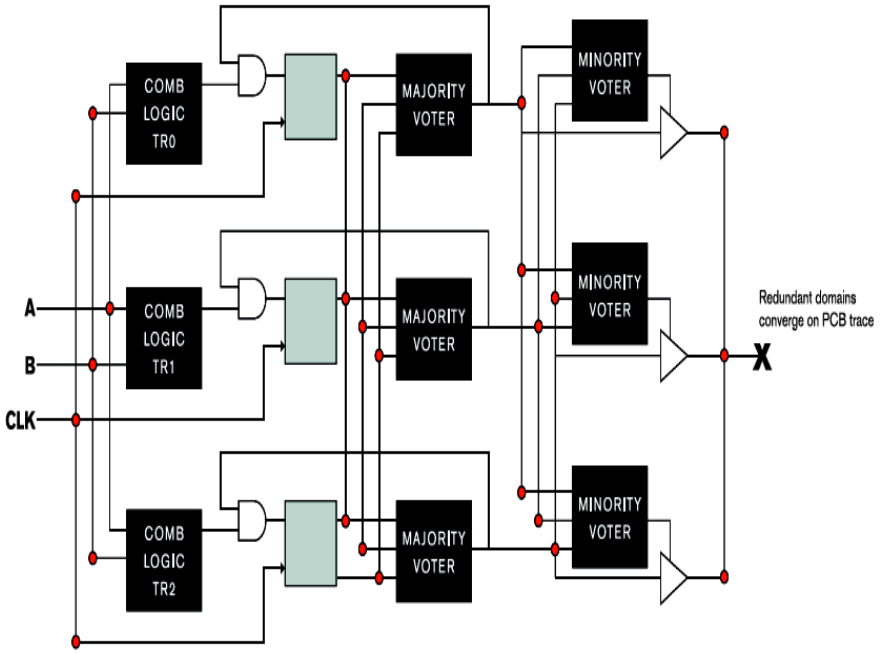


Figure 4.14: Memorization element in TMR.

features. Typically, the avionic industry makes use of techniques such as triple modular redundancy (TMR) to thwart single event upsets (SEUs). An example of a memorization element in TMR style is given in Fig. 4.14 as implemented in Xilinx "XTMR" solution [103].. The amount of logic involved in this structure is by far larger than that required in the DPL counter-part, depicted in Fig. 4.15 although four times larger than an unprotected flip-flop, this structure is nevertheless much smaller than that involved in TMR logic (see Fig. 4.14). This structure has two stages to accompany the evaluation $\leftrightarrow$ dynamic of the DPL protocol. We notably insist that such a construction is naturally immune to the attack presented in [104], that exploits an optimization of some DPL style: when the redundant dual-rail state is stored as one single bit, an exploitable leakage appears at the flip-flop level. To conclude this comparison between figures 4.14 and 4.15, we emphasize that the overhead figures shall not be considered in absolute, but relatively to the protection goal that is intended to be achieved.

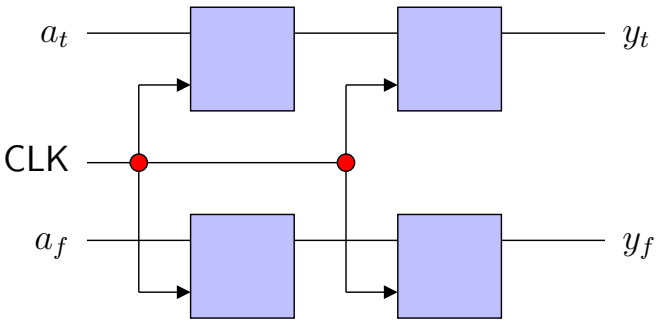


Figure 4.15: Memorization element in DPL.

4.2.7 Associating Three Protections to Reduce the Probability of a Successful FIA

Some faults in DPL circuits do not disclose any information about the faulted sensitive variable. However, in the case "false valid" are generated, the problem becomes different. This can happen in two problematic cases:

1. When the absorbing fault is too deep in the logic cone w.r.t. the false valid, as shown in Fig. 4.16, where f is a block with perfect diffusion, such as a substitution box implemented in logic. In this case, if the logic cone covered by the `'X'` happen to yield a correct value, then a valid fault is generated; unless the `'X'` are checked for at the output.

2. When a valid false occurs on one column of the state matrix alone, but that an `'X'` is generated on another column (knowing the two columns are not interfering in AES last round). In this case also, the faulty behavior can be observed by checking the validity of all the output bits.

To fight these remaining risks, three protections can be associated so as to increase the security level:

1. DPL, as detailed in the previous section.

2. Detection of NULLs at the end of the computation.

3. Regular detection schemes, such as coding.

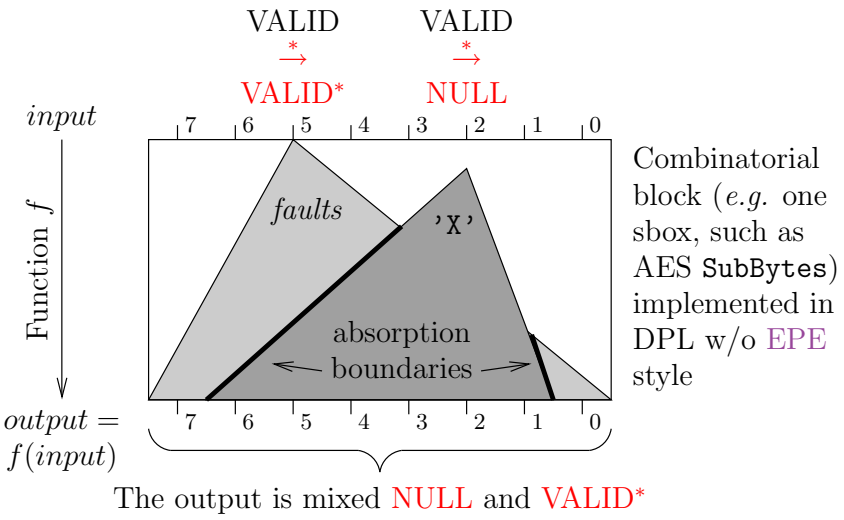


Figure 4.16: Multiple faults, where the false valid is not completely hidden by the `'X'` wave.

4.2.8 Applicability of Resilience with Certification Procedures

The two main certification schemes of security products are the FIPS 140 and the common criteria. We examine in this section if the resilience can be applied with the current version of those standard, or if the standards are too conservative.

4.2.8.1 NIST FIPS 140-3

The FIPS 140 [105, 106] formulates security requirements for cryptographic modules. It defines four levels of security, the highest of which is referred to as "security level 4". The functional security objectives of FIPS 140 are defined in §3. It includes those two requirements:

1. to detect errors in the operation of the cryptographic module and
2. to prevent the compromise or the modification of sensitive data and SSPs (Sensitive Security Parameters) resulting from these errors.

The "resilience" protection discussed in this thesis definitely fulfills the second requirement. However, not all resilient schemes comply with the first requirement. For instance, using the randomized homomorphic encryption (Algorithm 8), the errors cannot be detected. The partial resilience of dual-rail type countermeasure can allow

a detection of the fault. However, the security of this scheme is ensured even if there is no detection. This means that FIPS-140 standards 2 & 3 are not resilience-ready, although they express this idea.

More precisely, the exact statement of the requirements is detailed in §4.5.5 (140-2 [105]) or §4.6.5 (140-3 [106]). For the security level 4, the cryptographic module shall either employ environmental failure protection (EFP) features or undergo environmental failure testing (EFT). The EFP consists in a constant monitoring of the environment (temperature and voltage) whereas EFP is an a priori characterization of the perturbation consequences. In both cases, the protection circuitry shall either (1) shutdown the module to prevent further operation or (2) immediately zeroize all secret plaintext and private cryptographic keys and SSPs.

Such authoritative and irremediable actions could have been prevented using a resilience scheme, without compromizing the device security. Therefore, we find that FIPS 140-{2,3} standards are too strict, resulting in potential inconveniences from the user perspective if non malicious faults causes the module shutdown or zeroization.

4.2.8.2 Common Criteria

The Common Criteria (CC) [107] is a framework that permits comparability between results of independent security evaluations. It is an international standard ISO/IEC 15408:2005. The CC in themselves do not specify security requirements. Instead, a "target of evaluation" (TOE) must meet "security targets" (ST). One or more "protection profiles" (PP) must be respected by the ST. The security requirements are expressed in the PPs, whose structure is standardized but whose content is up to the designer. This flexibility allows a designer to tailor the PP to his (or that of his client) security objectives. Therefore, the CC readily accepts the resilience as a solution against fault attacks.

4.3 Case study on WDLL

4.3.1 Wave Dynamic Differential Logic

Power consumption of a standard CMOS cell is dependent on the transition of its input. Thus for a DPA-resistant design, a possible solution could be to introduce a

family of DPA resistant cells. In a WDDL cell [82], one transition per cycle is observed, which is favourable for a DPA resistant logic style.

WDDL uses true and false representations of each signal (I/O of each cell). To make the power consumption fairly uncorrelated to the processed data, it is necessary that there should be the same number of transitions every cycle. This condition is fulfilled by alternate cycles of precharge and evaluation. In the precharge phase all the signals are charged to the same level (*e.g.* 0 in WDDL) and during evaluation exactly one of the two complementary outputs is evaluated (=1). Figure 4.17 shows the timing diagram of WDDL AND gate. We can see that during precharge all signals are put to logic 0. During evaluation, exactly one of the two complementary inputs and outputs evaluates to 1.

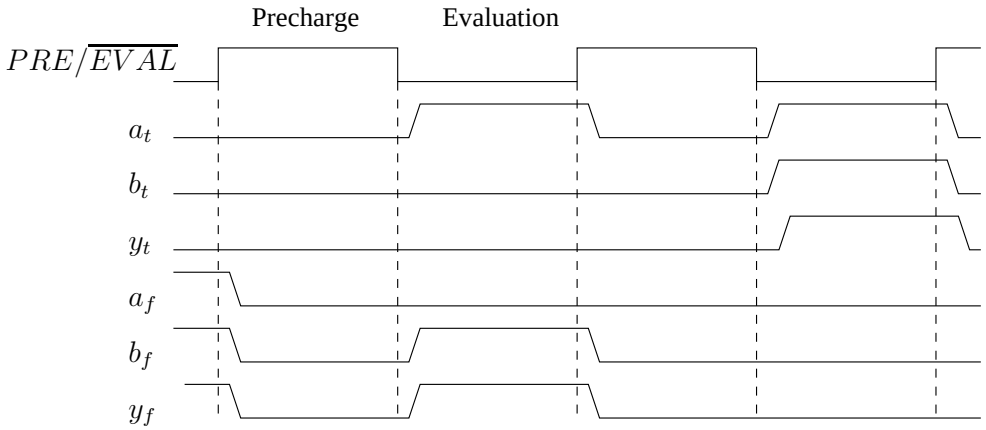


Figure 4.17: Timing diagram for a WDDL AND gate.

In DPL, glitches make the design vulnerable to attacks [108]. Indeed, without special attention, if the inputs arrive at different moments, glitches can be observed. To avoid glitches it is necessary that all the gates in the design should be positive in nature. To ensure this in WDDL, the design is synthesized with a library consisting of only positive gates (like AND, OR) [109]. As shown in figure 4.18, a WDDL AND gate consists of an AND gate (G) and a complementary OR gate (G^*, satisfying $G^*(x) \doteq \overline{G(\overline{x})}$).

For sequential circuits, each flip-flop is replaced by a pair a flip-flops. This double flip-flop allows the precharge wave to propagate through the whole design as all the gates are positive. It has to be noted that inverters in WDDL are implemented by crossing the true and false signals of the same variable.

A point worth noting in figure 4.18 is that one flip-flop in the single-rail design is replaced by four flip-flops in the WDDL design. This is explained as follows. During the precharge phase, the combinatorial part of the circuit will be discharged to 0 and this 0 is stored to the first of the two flip-flops. The second flip-flop will store the result of the last computation. In the evaluation phase, the value stored in the second flip-flop serves as input and the output is stored in the first flip-flop. In the mean while, the zero stored in the first flip-flop is shifted to the second flip-flop to allow proper precharge of the circuit ahead in the next cycle. This phenomenon happens in both true and false rail. Thus the number of flip-flops is quadrupled in the WDDL design.

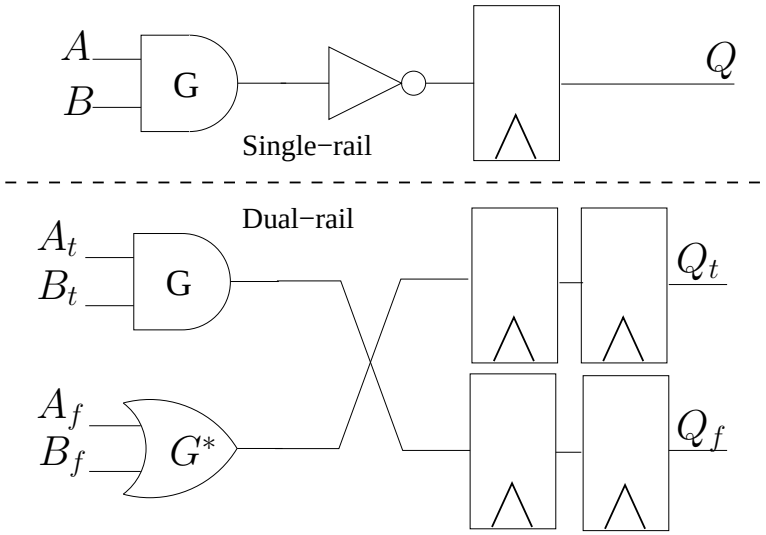


Figure 4.18: WDDL building block.

4.3.2 Design Flow for WDDL Implementation

As every digital system, cryptographic coprocessors can be separated into control and datapath parts. As the secret key is used only in the datapath part, leakage from the

control part is not crucial. Thus to assure security of the design it is sufficient to implement only the datapath in WDDL. This will also save area as WDDL takes more area on the FPGA than a single-rail design. The design flow to implement a cryptographic coprocessor on an FPGA is shown in figure 4.19. The datapath is first synthesized using an ASIC synthesizer taking advantage of a library with only positive LUTs (the FPGA synthesis tool does not provide enough options to limit the library therefore we use an ASIC synthesizer). As the number of positive functions with four inputs is fairly large (166), the library size is reduced by keeping only one function for any equivalence class where the inputs or the output are logically inversed and the inputs are swapped. Indeed, the inversions are dealt with externally from the LUT with wire-crossings (typical transformation of WDDL), and the FPGA mapper tools are able to change the LUT mask to make up for input pins permutations. Then the output netlist is processed using a custom tool (called vDuplicate in figure 4.19) which converts a single-rail netlist into a WDDL netlist. The controller is then connected to the WDDL datapath using a wrapper. The FPGA vendor tool does synthesis, mapping, placing & routing for the whole design on the FPGA.

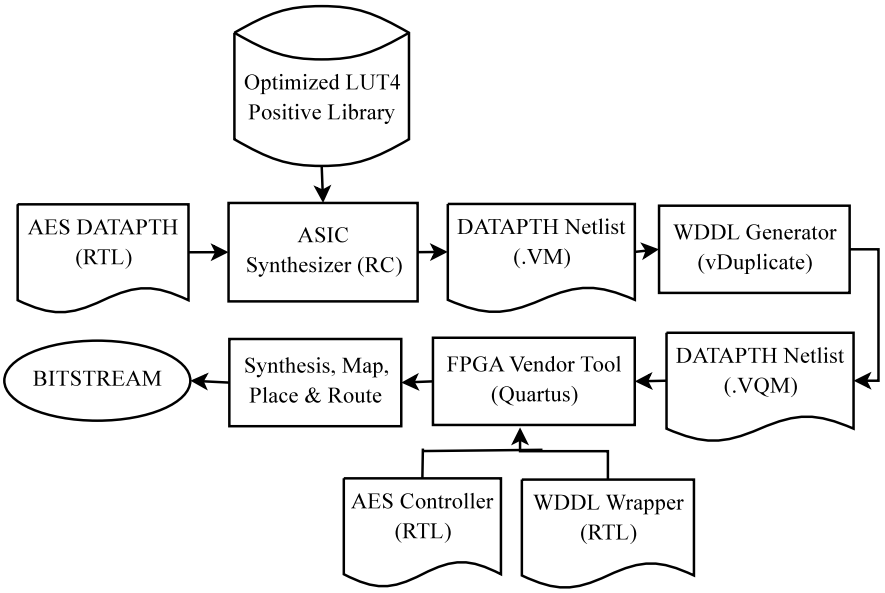


Figure 4.19: WDDL design flow.

4.3.3 Experimental Results

In order to evaluate DPL, we carried out a setup time violation attack on the WDDL impelementation of AES. As for the setup in chapter 2, we keep the key fixed and we change randomly messages. We decrease the voltage by a step of 0.5 mV and for each voltage the triple K,M,C is recorded. We collect 120.000 encryptions. For the WDDL version of AES, the SuBbytes was implemented in GF^4. Once analysis done, we can see the results in the figure 4.20. Less than 2% of the detected faults are single and

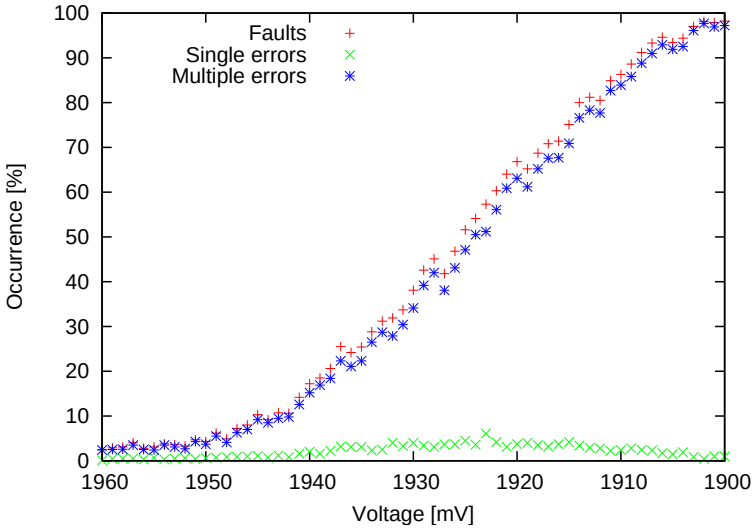


Figure 4.20: occurrence of faults in wddl version

all of them fall in the last round of AES as shown in figure 4.21. These faults are not exploitable and thus the key cannot be retrieved using Piret attack. The reason why all the faults are seen in the last round is as follows. When an XOR gate is implemented using positive logic, it is a combination of AND, OR gates and inverters (for inverted inputs). These inverters yield a mixture of true and false part of the design as per the definition of XOR. Thus a fault occurring in a true part is further corrupted by mixing with the false part and vice versa. The MixColumns operation involves a lot of XOR operations.

Therefore a MixColumns operation after a fault will corrupt the fault which cannot be detected. Since the last round does not have MixColumns, the faults are detected but not exploitable. One interesting observation was that every time a byte is affected

by a fault, a null byte in the ciphertext was reflected at its expected place. This means that even after successfully injecting the fault during encryption and precisely knowing the location of the fault, the output does not give any information which can be acted upon to retrieve the hidden secrets. Therefore, a WDDL design is naturally secure against setup time violation faults.

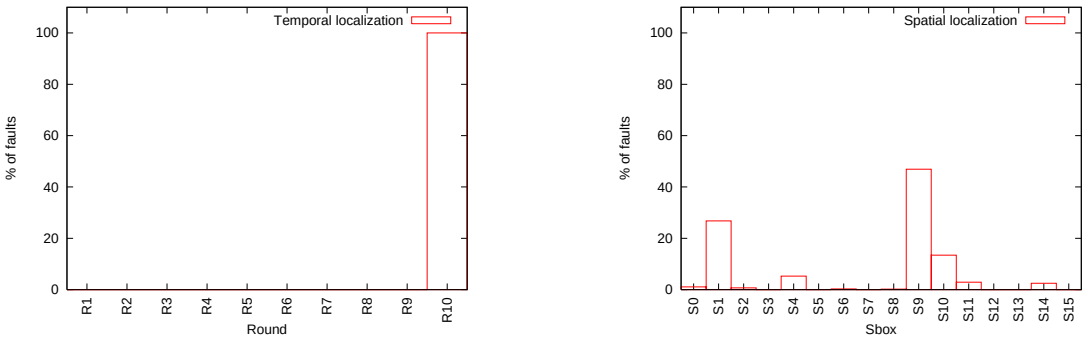


Figure 4.21: Temporal and Spatial localization of single faults for the Wddl implementation

The output of faulted ciphertexts is illustrated in Tables 4.4, 4.5, 4.6.

Table 4.4: Single fault in round 10.

key	00000000000000000000000000000000
message	093c7b78f4fa44baff2f67fc2d259dd0
ciphertext	96296994aba80db3ea81b491230985db
ciphertext*	96296994aba80db3ea81b49123090 0db

Table 4.5: Single fault in round 9.

key	00000000000000000000000000000000
message	c4968c64c72bbcb88acb744253f51be7
ciphertext	43720bee23f577a8311bf769f58e97e7
ciphertext*	00720bee23f57700311b0069f5 00 97e7

Table 4.6: Fault strictly before round 9.

key	00000000000000000000000000000000
message	be6d1ddeb2406e9a8546efc65284c4e7
ciphertext	fa73bc0ffb30e9209ec8bfe8f77b96f4
ciphertext*	00000000000000000000000000000000

4.3.4 Theoretical Fault Analysis on AES in WDDL

The purpose of this section is to show that the fault model corresponding to a setup violation time has the consequence that all DFAs on AES in WDDL are impractical.

In an under-powering or overclocking attack, faults arise from a setup time violation [110, 111]. Authors of paper [42] argue that the effect of a glitch on the power supply increases the propagation times of all the signals, which makes this disturbance similar in effect to under-powering the global chip. As the WDDL protocol with a (0, 0) spacer starts in evaluation step with all the nodes voltage equal to zero, the evaluation consists in propagating rising transitions along exactly half of the wires. If by any means, an attacker manages to trigger a setup time violation, the consequence is an asymmetric bit flip: only 1 to 0 errors are considered. Therefore, the consequence of the fault is to leave (at least) one dual-rail signal in its (0, 0) precharge state, while the others couples of wire are in legal (0, 1) or (1, 0) evaluation state.

The error is likely to happen for a few dual-rail signals if the stress level is low. This invalid data representation will then propagate through the next round logic. Four cases are possible:

1. the protocol error can turn into functional errors on the data or not, and
2. the protocol errors can vanish while flowing through the combinatorial logic (self protocol healing), or, at the opposite, be amplified.

The next subsection shows that functional errors occur, corresponding to bits erasure. In addition, the erasure rate increases: one single error at the entrance of a round will trigger many invalid precharge bits to be generated, and we show that in a reasonable cryptographic algorithm (no computation is done uselessly), the erasure rate increases. The consequence is that, after some percolation in the combinatorial logic, most of the values are erased.

4.3.4.1 Propagation of Faults

We start this analysis by the example of two representative gates: the AND and the XOR functions each having two inputs that we note a and b. We assume in this study that the fault occurs on input a. In evaluation, instead of having $(a_t, a_f) = (0, 1)$ when $a = 0$ and $(a_t, a_f) = (1, 0)$ otherwise, we simply have $a_t = a_f = 0$, which can also be expressed as a=NULL. The logic that implements the AND gate is $(c_t, c_f) = (a_t \cdot b_t, a_f + b_f)$. When a is faulty, the Tab. 4.7 function degenerates to AND$(a^*, b) = 0$ if $b = 0$, and NULL otherwise.

Table 4.7: Modified functionality of an AND gate in the presence of erasure faults.

Correct computation								
a	b	a_t	a_f	b_t	b_f	c_t	c_f	c
0	0	0	1	0	1	0	1	0
0	1	0	1	1	0	0	1	0
1	0	1	0	0	1	0	1	0
1	1	1	0	1	0	1	0	1
Faulted computation								
a	b	a_t	a_f	b_t	b_f	c_t	c_f	c
NULL	0	0	0	0	1	0	1	0
NULL	1	0	0	1	0	0	0	NULL

The same analysis can be carried out for the WDDL XOR gate in figure 4.22. The logic that implements the WDDL XOR gate is $(c_t, c_f) = (a_t \cdot b_f + a_f \cdot b_t, (a_f + b_t) \cdot (a_t + b_f))$. This equation shows that if we have a faulty input $(a_t = a_f = 0)$ then the output will be NULL $(c_t = c_f = 0)$. Thus the XOR gate has a maximum error propagation since the error is propagated for any value of b as shown in table 4.8

Now, for any function f, we have this property: The output of f is correct when f does not depend on the faulty input, and erased otherwise.

Let f be a positive Boolean function with inputs (a, b) then its WDDL equivalent F can be defined as:

$$\begin{cases} F_t(a_t, b_t) &= f(a_t, b_t)\,, \\ F_f(a_f, b_f) &= \overline{f(\overline{a_f}, \overline{b_f})}\,. \end{cases}$$

Table 4.8: Modified functionality of an XOR gate in the presence of erasure faults.

Faulted computation								
a	b	a_t	a_f	b_t	b_f	c_t	c_f	c
NULL	0	0	0	0	1	0	0	NULL
NULL	1	0	0	1	0	0	0	NULL

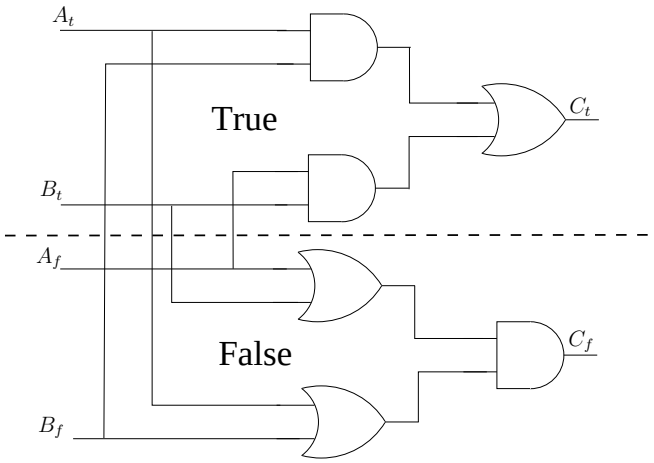


Figure 4.22: WDDL implementation of the XOR gate.

The proof is straightforward. If the output does not depend on the faulty input, the computation is correct for both the true and the false outputs, because the protocol violation does not impact the result. On the contrary, for the configuration of non-faulty inputs b such as F depends on the faulty input bit, then we have four cases:

1. $F_t = F_f = 1$: impossible since F is positive and the inputs are lower than a legal value, that is either (1, 0) or (0, 1),

2. $F_t = 1$ and $F_f = 0$. In this case, $1 = f(0, b)$ [equation for F_t] and $0 = \overline{f(\overline{0}, \overline{\overline{b}})} = \overline{f(1, b)}$ [equation for F_f] , *i.e.* $1 = f(1, b)$. Therefore $f(0, b) = f(1, b)$. However, we assumed that F does depend on the first faulty input, hence a contradiction.

3. $F_t = 0$ and $F_f = 1$: for the same reason, this case is incompatible with the input configuration such that F does depend on the faulty input.

4. Consequently, the only possibility is that $F_t = F_f = 0$, hence a NULL propagation.

Let us now study a random function, modeling the byte substitution table (SubBytes) of the AES. If there is a NULL fault at the input, then:

- for one half of the input data, a specific output bit will depend on this input, and
- for the other half, the targeted output bit does not depend on the input.

Therefore, statistically, one half of the output bits are erased to NULL. Notice that this result is independent of the exact functional decomposition in a positive dual gates netlist. Similarly, if two inputs are erased, then 3/4 of the outputs will also be NULL. And of course, when seven or eight errors are presented at the input, all the output bits become NULL.

We have already shown in section 4.3.4.1 that with XOR gates the fault propagation is maximal. The MixColumns transformation is a multiplication of a polynomial over $GF(2^8)$ with the fixed polynomial $a(x)$[4.1], reduced modulo $x^4 + 1$.

$$a(x) = (\texttt{0x03})x^3 + (\texttt{0x01})x^2 + (\texttt{0x01})x + (\texttt{0x02}) \tag{4.1}$$

The equations for the byte multiplications involved in this multiplication are written down in Tab. 4.9. Hence we see that the MixColumns operation is implemented as a tree of XOR gates. This ensures a maximum propagation of NULL.

Table 4.9: Equations for the bytes transformations $\times 01$, $\times 02$ and $\times 03$.

a'	$a \times 01$	$a \times 02$	$a \times 03$
a'_7	a_7	a_6	$a_7 \oplus a_6$
a'_6	a_6	a_5	$a_6 \oplus a_5$
a'_5	a_5	a_4	$a_5 \oplus a_4$
a'_4	a_4	$a_3 \oplus a_7$	$a_4 \oplus a_3 \oplus a_7$
a'_3	a_3	$a_2 \oplus a_7$	$a_3 \oplus a_2 \oplus a_7$
a'_2	a_2	a_1	$a_2 \oplus a_1$
a'_1	a_1	$a_0 \oplus a_7$	$a_1 \oplus a_0 \oplus a_7$
a'_0	a_0	a_7	$a_0 \oplus a_7$

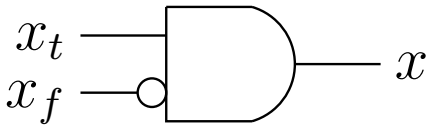


Figure 4.23: Dual-to-single rail circuitry usable in the case of a NULL0 spacer.

In an SPN (substitution permutation network) like AES, the fault number can only grow at each step. Indeed, for every block f, if a fault is stopped, then: $f(\texttt{'U'}, x)$ is certain, for a given input x. Now, this means that $f(\texttt{'0'}, x) = f(\texttt{'1'}, x)$, and this implies that f is not bijective. Therefore, differential attacks become difficult as the attacker observes an erased value, and cannot backtrack from the faulty ciphertext. The best case being when all the output bits are erased and thus no information that can be useful to generate the key is available.

As shown in Figure 4.23, the conversion of the dual-rail signals to single-rail turns a NULL into a `'0'`. This circuit makes use of both true and false signal halves, so as to prevent the CAD tool from simplifying half of the logic and balance the true and false networks. Therefore, if a fault occurs during the computation, it can be observed. This difference could be exploited by an attack, as done in the attack of Gilles Piret. However, the computed differential will not disclose any information about the last round key, since the XOR function used to mix it propagates a NULL.

All the considerations detailed regarding WDDL rely on the fact the gates are positive. Indeed, the gates will stick to zero unless valid values are produced. This is not true for delay insensitive gates which stay in a zero state and jam the computation. Notice that in WDDL the results are independent of the type of spacer used. It can be $NULL0 \doteq (0,0)$, $NULL1 \doteq (1,1)$, used as constants or interleaved alternatively or randomly.

4.3.4.2 Generalization to Arbitrary Fault Models

We consider two categories of faults:

1. **Asymmetric faults**, where bits can only be flipped from 1 to 0. This type of faults is typically encountered in WDDL circuits stressed by a global perturbation, such as under-voltage or over-clocking. Glitch attacks can lead to the same

symptom, because it manifests in adding a delay globally to all wires. Flashes of white light have been reported in [112, §12, page 163] to zero selectively the output of some operations. Equally, laser shots on SRAM-based FPGAs tend to favor $1 \rightarrow 0$ bit-flips over $0 \rightarrow 1$ [113]. Notice that in DPL with a (1, 1) spacer, the opposite transition occurs when trying non-invasive attacks. We do not detail this situation as it is the exact opposite of the 1 to 0 case.

2. **Symmetric faults**, where bits are susceptible of toggling in both directions. Laser shots can trigger both 1 to 0 and 0 to 1 transitions. This fault is thus semi-invasive, as opposed to the previous ones. Therefore, it models a more powerful attacker, at least able to chemically prepare the sample to attack.

In the context of asymmetric faults, DPL circuits are natively protected as such. In this respect, it is interesting to compare the pros and the cons of synchronous and asynchronous circuits. When exposed to under-voltage, asynchronous circuits will continue to work, down to a voltage value where the gates will not be supplied enough to produce a strong one. Below this threshold, errors of type "stuck at zero" will manifest, exactly as in the case of synchronous circuits. Overclocking is not an attack that applies to asynchronous circuits that are, by definition, clockless. However, we have noticed that this perturbation is ineffective exposing secrets. Therefore, a synchronous circuit will be less reliable in the presence of non-invasive faults, but as secure as an asynchronous circuit. A trade-off between the two approaches can be reached by considering synchronous circuits with jitter on the clock. The jitter can have a large variance, since even if it conducts to a setup time violation, the secrets remain safe. Therefore, with DPL, it is secure when used in addition with aggressive clock jitter.

If the attacker has the means to inject symmetric faults, then three types of protections must be considered:

1. When the fault induction is gentle, single bit flips is the most likely fault model. In this case, even if the fault is a 0 to 1 transition occurring during the evaluation stage, the only risk is to create a (1, 1), also called NULL1. However, in a dual way of the case study of the propagation of NULL0 values, we can show that the propagation of NULL1 consist in an erasure of the data, so that the syndrome does not convey any single bit of information about the faulty circuit internal state. DPL style thus forces the attacker to be less furtive.

2. With a more intense stress, the attacker will start to induce multiple faults with low multiplicity. In this case, a DPL gate can output completely false values. For instance, an AND gate for which the inputs are NULL0 and NULL1 evaluates to the correct value 0 (with respect to WDDL valid states), even if the two unfaulty inputs were both equal to 1. To protect the implementation against those attacks, additional detection hardware must be added so as to cross-check the computation. A little gain can however be obtained: As the DPL style is protected against single faults, a datapath of n bits can be checked with code words of only $n-1$ bits without risking to weaken the security level. A protection method at the technological level such as the one presented in [114] could be extended from SRAM points to DFFs and combinatorial gates. By using high-VT[1] P transistors (those that compute the `'1'`) and low-VT N transistors (those that compute the `'0'`), the designer could make the faults $1 \rightarrow 0$ much more likely than the opposite $0 \rightarrow 1$.

3. When the stress is very strong, then we expect the faults to be very frequent. Hence the recommendation to use physical captors spread on the chip surface.

4.3.5 WDDL w/o EPE

In this section we propose to improve the WDDL, in order to be more secure against multiple symmetrical faults. For this prupose, we introduce new varient of WDDL namely(WDDL w/o EE) which is a logic style dedicated to FPGAs that removes the EE without computing a rendezvous. Instead, each functional half gate receives the true and false inputs, and decides to output the VALID value only when all the inputs are VALID. This behavior can be achieved by a purely combinatorial gate. The detailed rationale behind the "WDDL w/o EE" style is the following:

- The gate outputs NULL{0,1} when the inputs are NULL{0,1} or transitional from this value.
- The gate outputs VALID only when all the inputs are VALID.
- In case of inconsistent values w.r.t. the DPL convention, the gate outputs an arbitrary NULL value.

[1] VT means here: "Voltage of Threshold".

This logic does not evaluate early by design, and propagates errors: if any input is stuck to NULL or if the input is out of specifications, then the output always remains to NULL too. In addition, this logic **does not generate glitches** even if the functionality is not positive, and **can be inverting**. Therefore, the synthesis is more optimized than for plain WDDL. For an AND function with inputs (a, b) , WDDL w/o EPE equivalent AND_T, AND_F is:

$$\begin{cases} F_t(a_t, a_f, b_t, b_f) &= a_t.b_t + a_t.a_f + b_t.b_f.a_f \,, \\ F_f(a_t, a_f, b_t, b_f) &= a_t.b_f + a_t.a_f + b_t.a_f + b_f.a_f \,. \end{cases}$$

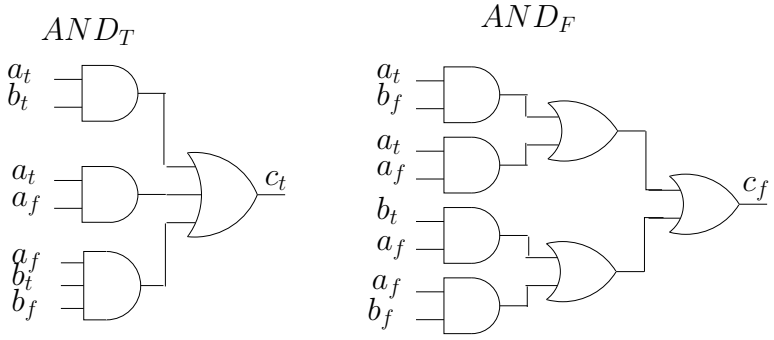


Figure 4.24: WDDL w/o AND gate

Architecture	CLB	Area Overhead
WDDL w EE	10937	NA
WDDL wo EE	13322	21%

Table 4.10: WDDL w/o area overhead

Once implemented the area overhead of the new version is 21% as shown in table 4.3.5.

4.3.6 Analysis of the DFA Protection for DPL w/o EPE

Single bit faults are inefficient against DPL because they turn a VALID data into a NULL token, that propagates and leads to an unexploitable error since it hides the faulted value. This is the typical scenario described in paper [73]. Highly multiple

faults generate randomly a large quantity of NULL values along with some more unlikely but devastating bit-flips. However, as NULL values are systematically propagated, they proliferate very quickly after some combinatorial logic layers traversal. And as they have the nice property to contaminate VALID values, the risky coherent bit-flips (simultaneous $0 \xrightarrow{*} 1$ and $1 \xrightarrow{*} 0$ in one dual-rail couple), they jam their propagation hopefully before they reach the algorithm output. This absorption property is all the more efficient as the number of NULL generated by the multiple faults is high. Therefore, the only way to inject a poisonous fault is to stress the circuit sufficiently enough to have multiple faults, without nonetheless creating too many faults so as to leave a chance for them not to be absorbed during their percolation towards the outputs. But, hopefully, in this opportunity window of low stress (generation of 2, 3, or maximum 4 errors because of the high diffusion of cryptographic algorithms), efficient coding schemes can be used in supplement to the DPL w/o EE protection.

To be more accurate, we present a simple model that provides a convincing proof of our assertion. Let us consider a dual-rail circuit that is attacked with a perturbation that is focalized on $2n$ wires, and that has an intensity sufficient enough to cause $m \leq 2n$ simultaneous faults. We also make the optimistic hypothesis that the m faults are equidistributed over the $2n$ wires, and that the flips are truly symmetrical, *i.e.* it is as likely to flip to a 0 and to a 1. Those conditions modelize a worst case from the defense view point, because they foster coherent bit-flips susceptible to turn a VALID value into a VALID* one, by the mean of two antinomic flips on two wires pertaining to the same dual-rail couple. To further simplify the modelization, we also assume that the attacked block has a perfect diffusion: in practice, this is not exactly true for one round of an algorithm, but for at least two of them (and exactly two in the case of AES). Nevertheless, it helps us grasp more intuitively the idea of the proof without introducing overcomplicated considerations. Therefore, for a fault to successfully propagate through the round, no single NULL shall be generated. Otherwise, the NULL wave catches the fault, because of the perfect diffusion, as already depicted in Figure 4.16. The first constatation is that for VALID faults to be generated, m must be even. Indeed, they are generated by pairs. If, on the contrary, m is odd, then at least one NULL (bit-flip of one wire in a pair) is generated, leading to the VALID fault absorption. Then, a VALID fault is generated iff, given a unique fault, a second one occurs in the paired wire. For $m = 2$ faults, this happens with probability $1/(2n - 1)$. For more faults, the generation of solely paired faults consists in always pairing the remaining faults. Then, the probability to generate at least one VALID fault that survives

until the output is equal to:

$$p(2n,m) \doteq \begin{cases} \binom{n}{m/2} \Big/ \binom{2n}{m} & \text{if } m \text{ is even,} \\ 0 & \text{otherwise.} \end{cases}$$

This probability becomes very small starting from a multiplicity of 4 when m increases up to n[1]. This is to be contrasted with schemes involving a coding with error detection. They are basically able to detect:

- all the faults of multiplicity smaller than the error detection capability r[2], but
- only a ratio of $1 - 1/2^r$ faults for $m > r$.

The figure 4.25 compares the rate of successful faults injection depending on the multiplicity, for an $n = 8$ set of wires, respectively for the proposed scheme based on DPL w/o EE and for a classical integrity check with a linear code detecting $r = 2$ bits of error.

In fact this is the first time that a countermeasure against DFA proves efficient even in the context of a large number of faults. As a matter of fact, usual schemes, based on spatio-temporal or coding, can be defeated with high probability if the number of faults is greater than the detection capacity. Smartly enough, the implementations using DPL w/o EE take advantage of three properties that all contribute to destroy the VALID faults:

1. faults are very likely to alter only one wire in a pair, especially if the stress is badly localized, thus creating much more NULL tokens than wrong VALID pairs,
2. because of the protection against EE, NULL values win against VALID ones, hereby hiding in particular VALID fault propagation,
3. as the algorithms implement cryptography, they have a high diffusion, which helps the NULL values meet (and thus eat) the possibly faulted VALID values still alive.

[1] When m is too large, starting from n, the probability increases, because of the property: $p(2n,m) = p(2n,2n-m)$.

[2] Faults of multiplicity $m \leq r$ mutate a code word into a non-code word.

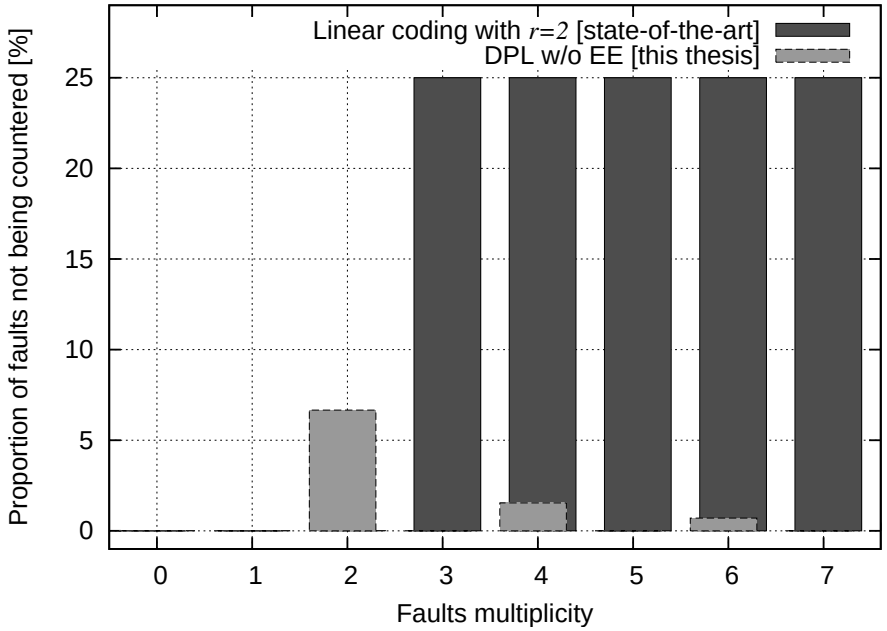


Figure 4.25: Probability that m faults injected on n wires be innocuous due to the protection conveyed by two different countermeasures: either a *detection* by an informational redundancy scheme or an *annihilation of the faulted data* by one or several VALID $\xrightarrow{*}$ NULL token transformations.

4.4 Conclusion

This chapter shows the state of the art of fault countermeasures as well as fault detection strategy.

A new strategy of fault protection based on resilience is introduced. The principle is based on the fact the injected faults are not exploitable as the information they contain about any secret is nullified. This property comes from the fact the circuit does not react to the fault as compared to a circuit protected with a detection structure which may confirm occurrence of the fault.

This chapter also has showed that fault resilience at logical level can take advantage of naturally redundant logic as the differential logic. Hence this logic type protects simultaneously against passive and active attacks. In this case it becomes a cheaper solution than detection based on codes. However the basic differential logic protects only against asymmetric faults, i.e. faults whose the effect is to nullify the targeted signal. An enhancement of this logic is to make it insensitive to the Early Propagation effect. Consequently this will also protect against asymmetric and symmetric faults.

An example of resilience at logical level is given with the WDDL logic. The experiences carried out on the AES platform implemented in WDDL clearly showed the efficiency of this logic to thwart setup time violation attacks, in addition to passive attacks like DPA. In order to improve WDDL we proposed and implemented new version called WDDL w/o EE which is free from early evaluation hence more robust against passive attacks and against multiple symmetric faults.

Chapter 5

Conclusion

In this thesis, different aspects of practical attack implementations: both "global" and "local", and their countermeasures, have been presented.

Chapter 1 outlines various aspects of cryptography and side channel attacks. It is specially focused on fault attacks and various Differential Fault Attack (DFA) proposed in the scientific literature. It also details some of the commonly used techniques to inject faults in a circuit.

In chapter 2, we present two different methods and their associated platforms to inject exploitable faults in cryptographic devices. The first method is a novel technique to inject global non-invasive faults based on the setup time violation. We demonstrated that this global method allows the injection of random faults in the desired circuit. In this method, we do not control the time and location of the occurrence of the faults. Nevertheless, when we create enough faults we are able to find required number of exploitable faults. Piret's DFA can be used to extract key from these exploitable faults. We have shown this attack can be carried out on both ASICs and FPGAs.

In the second method a laser beam has been used to inject surgical errors in a cryptographic circuit. Such faults are difficult to mount but have an obvious advantage in terms of control over time and location. Therefore we just need the exact number of faults as required by the DFA which need single byte fault. In our experiments, the target was a software implementation of AES on ATMEL ATMEGA128 microcon-

troller where we successfully introduced single byte faults in the penultimate round using an adequately triggered laser.

In chapter 3, we discussed DFA countermeasures. Over the years, the dominant strategy to counter faults has been their detection by spatial, temporal or information redundancy. We present in this thesis a new approach based on resilience. First of all, the resilience imposes no destruction of the secrets in case of a fault attack. In an implementation protected by fault-resilience, when a fault is injected successfully but has no consequence on the computation, the circuit shows no reaction towards fault as compared to a circuit protected with a detection-based scheme which may react and confirm occurrence of the fault. Even if the fault is injected during computation the attacker cannot exploit the faulted result to perform DFA.

Several concrete methods to implement resilient symmetrical encryption are proposed, amongst which a random mode of operation that is suitable for low-cost (without expensive module-level protections) embedded systems. When the designer can propose a hardware countermeasure, we suggest to use multi-valued or Dual-Rail Precharge Logic for protection. These logics simultaneously protect against observation and perturbation attacks, and are cheaper than detection based on codes.

As a perspective, we underline that an interesting topic to work on could be the "smart trigger". The principle of a smart trigger is to detect the starting point of an encryption cycle of the cryptographic circuit under test using its power consumption or electromagnetic emanations. From these side-channels, we can generate a signal automatically to trigger the laser gun in real-time. Thus it becomes possible to attack at any round without the need of a specific trigger signal which should not exist.

Another prospective could be to study the possibility and efficiency of injecting faults using electrostatics charges or electromagnetic pulses [115, 116]. In fact even if this method is less precise than laser attack, we can target small regions of the chip. Moreover, this method is non-invasive and the setup is cheaper than optical attacks as there is no specific and costly preparation stage.

Another interesting field of study which we can derive from this research is to implement a unified countermeasure against active and passive attacks which is independent from the used algorithms. As already shown that DPL are highly resilient against majority of the faults. DPL was initially introduced as a countermeasure against passive side channel attacks owing to its near-constant data independent power consumption. However some problems like early evaluation and technological bias reduces the efficiency of this countermeasure against the DPA but also the DFA as explained in chapter 3. We propose to implement a version of DPL which is free from its reported short-coming. Hence the proposed DPL would be a common and enhanced countermeasure against passive and active attacks.

While some new results can be expected in these fields, an upcoming topic could be to use fault attacks for reverse engineering secret cryptographic algorithms using "FIRE" attacks (Fault Injection for Reverse Engineering).

Publications

- ”Practical Setup Time Violation Attacks on AES” Nidhal Selmane, Sylvain Guilley et Jean-Luc Danger, **EDCC 2008**, The seventh European Dependable Computing Conference, Kaunas, Lithuania.
- ”Silicon-level solutions to counteract passive and active attacks” Sylvain Guilley, Laurent Sauvage, Jean-Luc Danger, Nidhal Selmane and Renaud Pacalet, **FDTC 2008**, 5th workshop on Fault Tolerance and Detection in Cryptography, IEEE-CS, Washington DC, USA.
- ”Fault Analysis Attack on an FPGA AES Implementation”, Nidhal Selmane, Farouk Khelil, Mohamed Hamdi, Sylvain Guilley and Jean-Luc Danger, **NTMS 2008**, Tangier, Morocco.
- ”Security Evaluation of Different AES Implementations Against Practical Setup Time Violation Attacks on FPGAs” Shivam Bhasin, Nidhal Selmane, Sylvain Guilley and Jean-Luc Danger, **HOST 2009** (Hardware Oriented Security and Trust) San Francisco, CA, USA.
- ”WDDL is Protected Against Setup Time Violation Attacks” Nidhal Selmane, Shivam Bhasin, Sylvain Guilley, Tarik Graba and Jean-Luc Danger, **FDTC 2009** (IEEE Fault Diagnosis and Tolerance in Cryptography), Lausanne, Switzerland.
- ”Combined SCA and DFA Countermeasures Integrable in a FPGA Design Flow” Shivam Bhasin, Jean-Luc Danger, Flament Florent, Tarik Graba, Sylvain Guilley, Yves Mathieu, Maxime Nassar, Laurent Sauvage and N. Selmane, **ReConFig 2009**, Cancun, Mexico.
- ”Fault Injection Resilience”,Sylvain Guilley, Laurent Sauvage, Jean-Luc Danger and N. Selmane, **FDTC 2010**, Santa Barbara, CA, USA.
- ”Countering Early Evaluation: An Approach Towards Robust Dual-Rail Precharge Logic” Shivam Bhasin,N. Selmane, Jean-Luc Danger and Sylvain Guilley, **WESS 2010**, Scottsdale AZ, USA.

- "Security Evaluation of ASICs and FPGAs against Setup Time Violation Attacks", **IET Journals** Nidhal Selmane, Shivam Bhasin, Jean-Luc Danger and Sylvain Guilley.

Bibliography

[1] FEDERAL INFORMATION PROCESSING STANDARDS (FIPS) PUBLICATION. *Announcing the Advanced Encryption Standard (AES)*. Number 197. November 2001.

[2] SERGEI P. SKOROBOGATOV AND ROSS J. ANDERSON. **Optical Fault Induction Attacks** . **2523** of *LNCS*, pages 2–12. Springer, august 2002. CA USA.

[3] RAMESH KARRI, KAIJIE WU, PIYUSH MISHRA, AND YONGKOOK KIM. **Concurrent Error Detection Schemes for Fault Based Side-Channel Cryptanalysis of Symmetric Block Ciphers**. *IEEE Transactions on Computer-Aided Design*, **21**(12):1509–1517, december 2002.

[4] MARK KARPOVSKY, KONRAD J. KULIKOWSKI, AND ALEXANDER TAUBIN. **Robust Protection against Fault-Injection Attacks on Smart Cards Implementing the Advanced Encryption Standard**. *IEEE Transactions on Computer-Aided Design*, **21**(2), may 2004.

[5] P. KOCHER, J. JAFFE, AND B. JUN. **Differential Power Analysis**. In *CHES99, LNCS Springer*, **1666**, pages 388–397, 1999.

[6] J.-J. QUISQUATER AND D. SAMYDE. **ElectroMagnetic Analysis (EMA) Measures and Counter-Measures for Smart Cards**. In *e-SMART, LNCS Springer*, **140**, pages 200–210, 2001.

[7] JOHANNES BLÖMER AND JEAN-PIERRE SEIFERT. **Fault based cryptanalysis of the Advanced Encryption Standard**. In SPRINGER, editor, *Financial Cryptography*, **2742** of *LNCS*, pages 162–181, 2003.

[8] DAN BONEH, RICHARD A. DEMILLO, AND RICHARD J. LIPTON. **On the Importance of Eliminating Errors in Cryptographic Computations**. *Journal of Cryptology*, **14**(2):101–119, 2001.

[9] JOAN DAEMEN, MICHAËL PEETERS, GILLES VAN ASSCHE, AND VINCENT RIJMEN. **Nessie Proposal: NOEKEON**, 2000. `http://gro.noekeon.org/`.

[10] T. KALKER J.A BLOOM, I.J COX AND M.L. MILLER. **Copy protection for DVD video**. *Proceedings of the IEEE*, pages 1267–1276, July 1999.

[11] U.S. DEPARTMENT OF COMMERCE/NATIONAL INSTITUTE OF STANDARDS AND TECHNOLOGY. **"DATA ENCRYPTION STANDARD (DES)"**. *FIPS PUB 46-3*, October 1999.

[12] ELECTRONIC FOUNDATION. **"Cracking DES"**. *O'Reilly Media*, November 1998.

[13] A. SHAMIR R.L. RIVSET AND L.ADLEMAN. **A method for obtaining digital signatures and public-key cryptosystems**. *commun. ACM*, **21**:120–126, 1978.

[14] G.SEROUSSI I.F. BLAKE AND N.P. SMART. **Elliptic Curves in Cryptography**. *London Mathematical Society Lecture Notes Series. Combrige University Press*, **265**, 1999.

[15] PAUL C. KOCHER, JOSHUA JAFFE, AND BENJAMIN JUN. **Timing Attacks on Implementations of Diffie-Hellman, RSA, DSS, and Other Systems**. In *Proceedings of CRYPTO'96*, **1109** of *LNCS*, pages 104–113. Springer-Verlag, 1996. (PDF).

[16] PAUL C. KOCHER, JOSHUA JAFFE, AND BENJAMIN JUN. **Differential Power Analysis**. In *Proceedings of CRYPTO'99*, **1666** of *LNCS*, pages 388–397. Springer-Verlag, 1999.

[17] JEAN-JACQUES QUISQUATER AND DAVID SAMYDE. **ElectroMagnetic Analysis (EMA): Measures and Counter-Measures for Smard Cards**. In I. ATTALI AND T. P. JENSEN, editors, *Smart Card Programming and Security (E-smart 2001)*, **2140** of *LNCS*, pages 200–210. Springer-Verlag, September 2001. Nice, France. ISSN 0302-9743.

[18] HAGAI BAR-EL, HAMID CHOUKRI, DAVID NACCACHE, MICHAEL TUNSTAL, AND CLAIRE WHELAN. **The Sorcerer's Apprentice Guide to Fault Attacks**. *Proceedings of the IEEE*, **94**(2):370–382, 2006. DOI: 10.1109/JPROC.2005.862424.

[19] *Electronic signals and transmission protocols*. International Organization for Standardization, 2002.

[20] *Fault attacks on RSA CRT:Concrete results and prcatical countermeasures*. Springer, 2002.

[21] *Cryptographic smart cards*, **03**, 1996.

[22] *Design principles for tamper-resistant smartcard processors*, 1999.

[23] MICHEL AGOYAN, JEAN-MAX DUTERTRE, DAVID NACCACHE, BRUNO ROBISSON, AND ASSIA TRIA. **When Clocks Fail: On Critical Paths and Clock Faults**. In *CARDIS*, pages 182–193, 2010.

[24] *Robust protection against fault-injection attacks on smart cards implementing the advenced encryption stadard*, 2004.

[25] *Faults and side-channel attacks on pairing based cryptography*, 2004.

[26] MICHEL AGOYAN, JEAN-MAX DUTERTRE, AMIR-PASHA MIRBAHA, DAVID NACCACHE, ANNE-LISE RIBOTTA, AND ASSIA TRIA. **How to flip a bit?** *On-Line Testing Symposium, IEEE International*, **0**:235–239, 2010.

[27] JEAN-JACQUES QUISQUATER AND DAVID SAMYDE. **Eddy current for Magnetic Analysis with Active Sensor**. In *Esmart 2002, Nice, France*, **9**, 2002.

[28] MARTIN OTTO. **fault attacks and countermeasures**, December 2004. Dissertation, Paderborn University .

[29] CHRISTOPHE GIRAUD. **DFA on AES**. In SPRINGER, editor, *Advanced Encryption Standard (AES) 4th international conference, LNCS springer*, **3373** of *LNCS*, pages 27–41, May 2005. Bonn, Germany.

[30] GILLES PIRET AND JEAN-JACQUES QUISQUATER. **A Differential Fault Attack Technique against SPN Structures, with Application to the AES and KHAZAD**. In *CHES*, **2779** of *LNCS*, pages 77–88. Springer, September 2003. Cologne, Germany.

[31] CHIEN-NING CHEN AND SUNG-MING YEN. **Differential fault analysis on AES key schedule and some countermeasures**. In SPRINGER, editor, *Information Security and Privacy*, **2727** of *LNCS*, pages 118–129, 2003.

[32] P. DUSART, G. LETOURNEUX, AND O. VIVOLO. **Differential Fault Analysis on A.E.S**. In SPRINGER, editor, *Applied Cryptography and Network Security*, **2846** of *LNCS*, pages 293–306, October 2003. Kunming, China.

[33] SHALMANI MOHAMMAD MORADI AMIR AND SALMASIZADEH MAHMOUD. **A Generalized Method of Differential Fault Attack Against AES Cryptosystem**. **4249**:91–100, 2006.

[34] MICHAEL TUNSTALL AND DEBDEEP MUKHOPADHYAY. **Differential Fault Analysis of the Advanced Encryption Standard using a Single Fault**. Report 2009/575, 2009. `http://eprint.iacr.org/2009/575`.

[35] BRUNO ROBISSON AND PASCAL MANET. **Differential Behavioral Analysis**. In *CHES*, **4727** of *LNCS*, pages 413–426. Springer, September 10-13 2007. Vienna, Austria.

[36] YANG LI, KAZUO SAKIYAMA, SHIGETO GOMISAWA, TOSHINORI FUKUNAGA, JUNKO TAKAHASHI, AND KAZUO OHTA. **Fault Sensitivity Analysis**. In *CHES*, **6225** of *Lecture Notes in Computer Science*, pages 320–334. Springer, August 17-20 2010. Santa Barbara, CA, USA.

[37] JUNKO TAKAHASHI, TOSHINORI FUKUNAGA, AND KIMIHIRO YAMAKOSHI. **DFA Mechanism on the AES Key Schedule**. In IEEE COMPUTER SOCIETY, editor, *FDTC 2007 Workshop*, pages 62–74, September 2007. Vienna, Austria.

[38] NEIL H.E. WESTE AND DAVID HARRIS. *CMOS VLSI Design: A Circuits and Systems Perspective*. Addison Wesley, 2004. 3 edition (May 11, 2004), ISBN: 0321149017.

[39] **"Circuits Multi-Projets" (alias CMP, <cmp@imag.fr>) Annual Report 2005**. `http://cmp.imag.fr/aboutus/gallery/details.php?id_circ=63&y=2005`.

[40] SUMIO MORIOK AND AKASHI SATOH. **An Optimized S-Box Circuit Architecture for Low Power AES Design**. *Lecture Notes in Computer Science*, **2523/2003**(6):271–295, 2003.

[41] L. NAVINER JL DANGER AND G. DUC. *Application Specific Integrated Circuits*. august 2009. `http://sen.enst.fr/filemanager/active?fid=501`.

[42] JULIEN FRANCQ AND OLIVIER FAURAX. **Security of several AES Implementations against Delay Faults**. In *Proceedings of the 12th Nordic Workshop on Secure IT Systems (NordSec 2007)*, October 2007. Reykjavík, Iceland.

[43] SAAR DRIMER, TIM GÜNEYSU, AND CHRISTOF PAAR. **DSPs, BRAMs and a Pinch of Logic: New Recipes for the AES on FPGAs**. In *IEEE Symposium on Field-Programmable Custom Computing Machines (FCCM)*, pages 99–108. IEEE, 14-15 Apr 2008. Stanford, Palo Alto, CA.

[44] ALESSANDRO BARENGHI, GUIDO BERTONI LUCA BREVEGLIERI, MAURO PELLICIOLI, AND GERARDO PELOSI. **Low Voltage Fault Attacks to AES**. In *HOST (Hardware Oriented Security and Trust)*. IEEE Computer Society, June 13-14 2010. Anaheim Convention Center, CA, USA.

[45] *14th IEEE International On-Line Testing Symposium (IOLTS 2008), 7-9 July 2008, Rhodes, Greece*. IEEE, Jul. 2008.

[46] GAETAN CANIVET, P. MAISTN, RÉGIS LEVEUGLE, FRÉDÉRIC VALETTE, JESSY CLÉDIÈRE, AND MARC RENAUDIN. **Robustness evaluation and improvements under laser-based fault attacks of an AES crypto-processor implemented on a SRAM-based FPGA**. In *European Test Symposium*, page 251, 2010.

[47] TAL MALKIN, FRANÇOIS-XAVIER STANDAERT, AND MOTI YUNG. **A Comparative Cost/Security Analysis of Fault Attack Countermeasures**. In *FDTC*, **4236** of *Lecture Notes in Computer Science*, pages 159–172. Springer, October 10 2006.

[48] SUNG-MING YEN AND MARC JOYE. **Checking Before Output May Not Be Enough Against Fault-Based Cryptanalysis**. *IEEE Trans. Computers*, **49**(9):967–970, 2000. DOI: 10.1109/12.869328.

[49] PAUL C. KOCHER. **Leak-resistant cryptographic indexed key update**, March 25 2003. United States Patent 6,539,092 filed on July 2nd, 1999 at San Francisco, CA, USA.

[50] **Workshop on "Provable Security against Physical Attacks"**, February 10-19 2010. Amsterdam, Netherlands. `http://www.lorentzcenter.nl/lc/web/2010/383/program.php3?wsid=383`.

[51] YANG LI, SHIGETO GOMISAWA, KAZUO SAKIYAMA, AND KAZUO OHTA. **An Information Theoretic Perspective on the Differential Fault Analysis against AES**. Cryptology ePrint Archive, Report 2010/032, 2010. `http://eprint.iacr.org/`.

[52] GUIDO BERTONI, LUCA BREVEGLIERI, ISRAEL KOREN, AND PAOLO MAISTRI. **An Efficient Hardware-Based Fault Diagnosis Scheme for AES**. *proceedings of DFT*, **52**:130–138, 2004.

[53] G. BERTONI, L. BREVEGLIERI, I. KOREN, P. MAISTRI, AND V. PIURI. **Error Analysis and Detection Procedures for a Hardware Implementation of the Advanced Encryption Standard**. *IEEE Transactions on Computer-Aided Design*, **52**(4), April 2003.

[54] CHIH-HSU YEN AND BING-FEI WU. **Simple error detection methods for hardware implementation of Advanced Encryption Standard**. *IEEE transactions on computers*, **55**(6):720–731, june 2006.

[55] MARK KARPOVSKY, KONRAD J. KULIKOWSKI, AND ALEXANDER TAUBIN. **Differential Fault Analysis Attack Resistant Architectures For The Advanced Encryption Standard**. *DSN 2004*, (9), August 2004.

[56] P. MAISTRI AND R. LEVEUGLE. **Double-Data-Rate Computation as a Countermeasure against Fault Analysis**. **57**, pages 1528 –1539, nov. 2008.

[57] ROSS ANDERSON AND MARKUS KUHN. **Tamper Resistance – a Cautionary Note**. In *In Proceedings of the Second USENIX Workshop ON Electronic Commerce*, pages 1–11, 1996.

[58] SUNG-MING YEN, SEUNGJOO KIM, SEONGAN LIM, AND SANG-JAE MOON. **RSA Speedup with Chinese Remainder Theorem Immune against Hardware Fault Cryptanalysis**. *IEEE Trans. Computers*, **52**(4):461–472, 2003. DOI: 10.1109/TC.2003.1190587.

[59] DAN BONEH, RICHARD A. DEMILLO, AND RICHARD J. LIPTON. **On the Importance of Checking Cryptographic Protocols for Faults**. In *Proceedings of Eurocrypt'97*, **1233** of *LNCS*, pages 37–51. Springer, May 11-15 1997. Konstanz, Germany.

[60] ROSARIO GENNARO, ANNA LYSYANSKAYA, TAL MALKIN, SILVIO MICALI, AND TAL RABIN. **Algorithmic Tamper-Proof (ATP) Security: Theoretical Foundations for Security against Hardware Tampering**. In *TCC*, **2951** of *Lecture Notes in Computer Science*, pages 258–277. Springer, February 19-21 2004. Cambridge, MA, USA.

[61] ARNAUD BOSCHER, HELENA HANDSCHUH, AND ELENA TRICHINA. **Blinded Fault Resistant Exponentiation Revisited**. In *FDTC*, pages 3–9. IEEE Computer Society, September 6 2009. Lausanne, Switzerland.

[62] ARNAUD BOSCHER, ROBERT NACIRI, AND EMMANUEL PROUFF. **CRT RSA Algorithm Protected Against Fault Attacks**. In *WISTP*, **4462** of *LNCS*, pages 229–243. Springer, May 9-11 2007. Heraklion, Crete, Greece.

[63] ELI BIHAM AND ADI SHAMIR. **Differential Fault Analysis of Secret Key Cryptosystems**. In *CRYPTO*, **1294** of *LNCS*, pages 513–525. Springer, August 1997. Santa Barbara, CA, USA.

[64] PASCAL PAILLIER. **Public-Key Cryptosystems Based on Composite Degree Residuosity Classes**. In *EUROCRYPT*, **1592** of *Lecture Notes in Computer Science*, pages 223–238. Springer, May 2-6 1999. Prague, Czech Republic.

[65] ROBERT P. MCEVOY, MICHAEL TUNSTALL, CLAIRE WHELAN, COLIN C. MURPHY, AND WILLIAM P. MARNANE. **All-or-Nothing Transforms as a Countermeasure to Differential Side-Channel Analysis**. Cryptology ePrint Archive, Report 2009/185, April 30 2009. `http://eprint.iacr.org/2009/185`.

[66] ROBERT P. MCEVOY, MICHAEL TUNSTALL, CLAIRE WHELAN, COLIN C. MURPHY, AND WILLIAM P. MARNANE. **A differential side-channel analysis countermeasure**. European Patent Application (EP 2148462 A1), filled in 27.01.2010.

[67] MARCEL MEDWED, FRANÇOIS-XAVIER STANDAERT, JOHANN GROSSSCHÄDL, AND FRANCESCO REGAZZONI. **Fresh Re-Keying: Security against Side-Channel and Fault Attacks for Low-Cost Devices**. In *AFRICACRYPT*, **6055** of *LNCS*, pages 279–296. Springer, May 03-06 2010. Stellenbosch, South Africa. DOI: 10.1007/978-3-642-12678-9_17.

[68] JEAN-SÉBASTIEN CORON AND AVRADIP MANDAL. **PSS Is Secure against Random Fault Attacks**. In *ASIACRYPT*, **5912** of *LNCS*, pages 653–666. Springer, December 6-10 2009. Tokyo, Japan.

[69] SHIVAM BHASIN, TAOUFIK CHOUTA, GUILLAUME DUC, JEAN-LUC DANGER, AZIZ EL AABID, FLORENT FLAMENT, PHILIPPE HOOGVORST, TARIK GRABA, SYLVAIN GUILLEY, HOUSSEM MAGHR'EBI, OLIVIER MEYNARD, MAXIME NASSAR, RENAUD PACALET,

LAURENT SAUVAGE, NIDHAL SELMANE, AND YOUSSEF SOUISSI. **Combined countermeasures against perturbation & observation attacks**. In *PASTIS (PAca Security Trends In embedded Security)*, Gardanne (École des Mines de Saint-Étienne), France, June 16-17 2010. http://www.secure-ic.com/PDF/pastis_2010.pdf.

[70] YUICHI BABA, ATSUSHI MIYAMOTO, NAOFUMI HOMMA, AND TAKAFUMI AOKI. **Multiple-Valued Constant-Power Adder for Cryptographic Processors**. In *ISMVL*, pages 239–244. IEEE Computer Society, May 21-23 2009. Naha, Okinawaw, Japan.

[71] SIMON MOORE, ROBERT MULLINS, PAUL CUNNINGHAM, ROSS ANDERSON, AND GEORGE TAYLOR. **Improving smart card security using self-timed circuits**. In *ASYNC (Asynchronous Circuits and Systems)*, pages 211– 218, April 2002. ISSN: 1522-8681, ISBN: 0-7695-1540-1j INSPEC Accession Number: 7321683.

[72] SIMON W. MOORE, ROSS J. ANDERSON, ROBERT D. MULLINS, GEORGE S. TAYLOR, AND JACQUES J. A. FOURNIER. **Balanced self-checking asynchronous logic for smart card applications**. *Microprocessors and Microsystems*, **27**(9):421–430, 2003.

[73] NIDHAL SELMANE, SHIVAM BHASIN, SYLVAIN GUILLEY, TARIK GRABA, AND JEAN-LUC DANGER. **WDDL is Protected Against Setup Time Violation Attacks**. In *FDTC*, pages 73–83. IEEE Computer Society, September 6th 2009. In conjunction with CHES'09, Lausanne, Switzerland. DOI: 10.1109/FDTC.2009.40; Online version: http://hal.archives-ouvertes.fr/hal-00410135/en/.

[74] SHIVAM BHASIN, JEAN-LUC DANGER, FLORENT FLAMENT, TARIK GRABA, SYLVAIN GUILLEY, YVES MATHIEU, MAXIME NASSAR, LAURENT SAUVAGE, AND NIDHAL SELMANE. **Combined SCA and DFA Countermeasures Integrable in a FPGA Design Flow**. In *ReConFig*, pages 213–218. IEEE Computer Society, December 9–11 2009. Cancún, Quintana Roo, México, DOI: 10.1109/ReConFig.2009.50, http://hal.archives-ouvertes.fr/hal-00411843/en/.

[75] YANNICK MONNET, MARC RENAUDIN, RÉGIS LEVEUGLE, CHRISTOPHE CLAVIER, AND PASCAL MOITREL. **Case Study of a Fault Attack on Asynchronous DES Crypto-Processors**. In *FDTC*, **4236** of *Lecture Notes in Computer Science*, pages 88–97. Springer, October 10 2006. Yokohama, Japan.

[76] STEFAN MANGARD, ELISABETH OSWALD, AND THOMAS POPP. *Power Analysis Attacks: Revealing the Secrets of Smart Cards*. Springer, December 2006. ISBN 0-387-30857-1, http://www.dpabook.org/.

[77] DANIL SOKOLOV, JULIAN MURPHY, ALEXANDER BYSTROV, AND ALEX YAKOVLEV. **Design and Analysis of Dual-Rail Circuits for Security Applications**. *IEEE Trans. Comput.*, **54**(4):449–460, 2005.

[78] JEAN-LUC DANGER, SYLVAIN GUILLEY, SHIVAM BHASIN, AND MAXIME NASSAR. **Overview of Dual Rail with Precharge Logic Styles to Thwart Implementation-Level Attacks on Hardware Cryptoprocessors, — *New Attacks and Improved Counter-Measures* —**. In *SCS*, IEEE, pages 1–8, November 6–8 2009. Jerba, Tunisia. Complete version online: http://hal.archives-ouvertes.fr/hal-00431261/en/. DOI: 10.1109/ICSCS.2009.5412599.

[79] CHRISTOPHE GIRAUD AND HUGUES THIEBEAULD. **A Survey on Fault Attacks**. In KLUWER, editor, *CARDIS*, pages 159–176, 2004. Toulouse, France.

[80] DAISUKE SUZUKI AND MINORU SAEKI. **Security Evaluation of DPA Countermeasures Using Dual-Rail Pre-charge Logic Style**. In *CHES*, **4249** of *LNCS*, pages 255–269. Springer, 2006. Yokohama, Japan. http://dx.doi.org/10.1007/11894063_21.

[81] KONRAD J. KULIKOWSKI, MARK G. KARPOVSKY, AND ALEXANDER TAUBIN. **Power Attacks on Secure Hardware Based on Early Propagation of Data**. In *IOLTS*, pages 131–138. IEEE Computer Society, 2006. Como, Italy.

[82] KRIS TIRI AND INGRID VERBAUWHEDE. **A Logic Level Design Methodology for a Secure DPA Resistant ASIC or FPGA Implementation**. In *DATE'04*, pages 246–251. IEEE Computer Society, February 2004. Paris, France. DOI: 10.1109/DATE.2004.1268856.

[83] YUVAL ISHAI, MANOJ PRABHAKARAN, AMIT SAHAI, AND DAVID WAGNER. **Private Circuits II: Keeping Secrets in Tamperable Circuits**. In *EUROCRYPT*, **4004** of *Lecture Notes in Computer Science*, pages 308–327. Springer, May 28 – June 1 2006. St. Petersburg, Russia.

[84] RÉGIS LEVEUGLE. **Early Analysis of Fault-based Attack Effects in Secure Circuits**. *IEEE Trans. Computers*, **56**(10):1431–1434, 2007.

[85] FRANCESCO REGAZZONI, THOMAS EISENBARTH, JOHANN GROSSSCHÄDL, LUCA BREVEGLIERI, PAOLO IENNE, ISRAEL KOREN, AND CHRISTOF PAAR. **Power Attacks Resistance of Cryptographic S-Boxes with Added Error Detection Circuits**. In *DFT*, pages 508–516. IEEE Computer Society, September 26-28 2007. Rome, Italy.

[86] VINCENT MAINGOT AND RÉGIS LEVEUGLE. **Influence of error detecting or correcting codes on the sensitivity to DPA of an AES S-box**. In *SCS*, IEEE, pages 1–5, November 6–8 2009. Jerba, Tunisia. DOI: 10.1109/ICSCS.2009.5412600.

[87] MARK G. KARPOVSKY, KONRAD J. KULIKOWSKI, AND ALEXANDER TAUBIN. **Robust Protection against Fault Injection Attacks on Smart Cards Implementing the Advanced Encryption Standard**. In *DSN*, pages 93–101. IEEE Computer Society, June 28 – July 01 2004. Florence, Italy.

[88] KRIS TIRI, DAVID HWANG, ALIREZA HODJAT, BO-CHENG LAI, SHENGLIN YANG, PATRICK SCHAUMONT, AND INGRID VERBAUWHEDE. **A side-channel leakage free coprocessor IC in 0.18 μm CMOS for Embedded AES-based Cryptographic and Biometric Processing**. In *DAC*, pages 222–227. ACM, June 13-17 2005. San Diego, CA, USA.

[89] THOMAS POPP, MARIO KIRSCHBAUM, THOMAS ZEFFERER, AND STEFAN MANGARD. **Evaluation of the Masked Logic Style MDPL on a Prototype Chip**. In *CHES*, **4727** of *LNCS*, pages 81–94. Springer, Sept 2007. Vienna, Austria.

[90] RAFAEL SOARES, NEY CALAZANS, VICTOR LOMNÉ, PHILIPPE MAURINE, LIONEL TORRES, AND MICHEL ROBERT. **Evaluating the robustness of secure triple track logic through prototyping**. In *SBCCI'08: Proceedings of the 21st annual symposium on Integrated circuits and system design*, pages 193–198, New York, NY, USA, September 1-4 2008. ACM.

[91] RAFAEL SOARES, NEY CALAZANS, VICTOR LOMNE, THOMAS ORDAS, PHILIPPE MAURINE, LIONEL TORRES, AND MICHEL ROBERT. **Evaluation on FPGA of Triple Rail Logic Robustness against DPA and DEMA**. In *DATE, track A4 (*Secure embedded implementations*)*, pages 634–639. IEEE, April 20–24 2009. Nice, France.

[92] SYLVAIN GUILLEY, PHILIPPE HOOGVORST, YVES MATHIEU, RENAUD PACALET, AND JEAN PROVOST. **CMOS Structures Suitable for Secured Hardware**. In *DATE'04 – Volume 2*, pages 1414–1415. IEEE Computer Society, February 2004. Paris, France. DOI: 10.1109/DATE.2004.1269113.

[93] SYLVAIN GUILLEY, SUMANTA CHAUDHURI, LAURENT SAUVAGE, PHILIPPE HOOGVORST, RENAUD PACALET, AND GUIDO MARCO BERTONI. **Security Evaluation of WDDL and SecLib Countermeasures against Power Attacks**. *IEEE Transactions on Computers*, **57**(11):1482–1497, nov 2008.

[94] SYLVAIN GUILLEY, FLORENT FLAMENT, RENAUD PACALET, PHILIPPE HOOGVORST, AND YVES MATHIEU. **Security Evaluation of a Balanced Quasi-Delay Insensitive Library**. In *DCIS*, Grenoble, France, nov 2008. IEEE. 6 pages, Session 5D – Reliable and Secure Architectures, ISBN: 978-2-84813-124-5, full text in HAL: `http://hal.archives-ouvertes.fr/hal-00283405/en/`.

[95] SYLVAIN GUILLEY, LAURENT SAUVAGE, FLORENT FLAMENT, PHILIPPE HOOGVORST, AND RENAUD PACALET. **Evaluation of Power-Constant Dual-Rail Logics Counter-Measures against DPA with Design-Time Security Metrics**. *IEEE Transactions on Computers*, **9**(59):1250–1263, September 2010. DOI: 10.1109/TC.2010.104.

[96] MAXIME NASSAR, SHIVAM BHASIN, JEAN-LUC DANGER, GUILLAUME DUC, AND SYLVAIN GUILLEY. **BCDL: A high performance balanced DPL with global precharge and without early-evaluation**. In *DATE'10*, pages 849–854. IEEE Computer Society, March 8-12 2010. Dresden, Germany.

[97] JEAN-LUC DANGER AND SYLVAIN GUILLEY. **Circuit de cryptographie programmable – Logique BCDL (Balanced Cell-based Differential Logic)**, 25 Mars 2008. Brevet Français FR08/51904, assigné à l'Institut TELECOM; WO/2009/118264.

[98] SYLVAIN GUILLEY, SUMANTA CHAUDHURI, LAURENT SAUVAGE, JEAN-LUC DANGER, TAHA BEYROUTHY, AND LAURENT FESQUET. **Updates on the Potential of Clock-Less Logics to Strengthen Cryptographic Circuits against Side-Channel Attacks**. In *ICECS*, IEEE, pages 351–354, December 13–16 2009. Medina, Yasmine Hammamet, Tunisia. DOI: 10.1109/ICECS.2009.5411008.

[99] ZHONGCHUAN C. YU, STEPHEN B. FURBER, AND LUIS A. PLANA. **An Investigation into the Security of Self-Timed Circuits**. In *ASYNC*, pages 206–215. IEEE Computer Society, May 12-16 2003. Vancouver, BC, Canada.

[100] **3rd Generation Smart Card Project, G3Card; European project under grant IST-1999-13515**. Website: `http://www.g3card.org/`.

[101] ZHIMIN CHEN AND YUJIE ZHOU. **Dual-Rail Random Switching Logic: A Countermeasure to Reduce Side Channel Leakage**. In *CHES*, **4249** of *LNCS*, pages 242–254. Springer, 2006. Yokohama, Japan, `http://dx.doi.org/10.1007/11894063_20`.

[102] ROBERT P. MCEVOY, COLIN C. MURPHY, WILLIAM P. MARNANE, AND MICHAEL TUNSTALL. **Isolated WDDL: A Hiding Countermeasure for Differential Power Analysis on FPGAs**. *ACM Trans. Reconfigurable Technol. Syst. (TRETS)*, **2**(1):1–23, 2009.

[103] THE "XILINX TMR TOOL". FEATURES DESCRIPTION AT THIS WEB PAGE:. **`http://www.xilinx.com/ise/optional_prod/tmrtool.htm`**.

[104] AMIR MORADI, THOMAS EISENBARTH, AXEL POSCHMANN, CARSTEN ROLFES, CHRISTOF PAAR, MOHAMMAD T. MANZURI SHALMANI, AND MAHMOUD SALMASIZADEH. **Information Leakage of Flip-Flops in DPA-Resistant Logic Styles**. Cryptology ePrint Archive, Report 2008/188, 2008. `http://eprint.iacr.org/`.

[105] NIST FIPS (FEDERAL INFORMATION PROCESSING STANDARDS) PUBLICATION 140-2. **Security Requirements for Cryptographic Modules**. page 69, May 25 2001.

[106] NIST FIPS (FEDERAL INFORMATION PROCESSING STANDARDS) PUBLICATION 140-3. **Security Requirements for Cryptographic Modules (Draft, Revised)**. page 63, 09/11 2009.

[107] **Common Criteria (ISO/IEC 15408)**. `http://www.commoncriteriaportal.org/`.

[108] STEFAN MANGARD, NORBERT PRAMSTALLER, AND ELISABETH OSWALD. **Successfully Attacking Masked AES Hardware Implementations**. In LNCS, editor, *Proceedings of CHES'05*, **3659** of *LNCS*, pages 157–171. Springer, August 29 – September 1 2005. Edinburgh, Scotland, UK.

[109] SYLVAIN GUILLEY, LAURENT SAUVAGE, JEAN-LUC DANGER, TARIK GRABA, AND YVES MATHIEU. **Evaluation of Power-Constant Dual-Rail Logic as a Protection of Cryptographic Applications in FPGAs**. In *SSIRI*, pages 16–23, Yokohama, Japan, jul 2008. IEEE Computer Society. DOI: 10.1109/SSIRI.2008.31, `http://hal.archives-ouvertes.fr/hal-00259153/en/`.

[110] NIDHAL SELMANE, SYLVAIN GUILLEY, AND JEAN-LUC DANGER. **Setup Time Violation Attacks on AES**. In *EDCC, The seventh European Dependable Computing Conference*, pages 91–96, Kaunas, Lithuania, may 7-9 2008. ISBN: 978-0-7695-3138-0, DOI: 10.1109/EDCC-7.2008.11.

[111] FAROUK KHELIL, MOHAMED HAMDI, SYLVAIN GUILLEY, JEAN-LUC DANGER, AND NIDHAL SELMANE. **Fault Analysis Attack on an FPGA AES Implementation**. In *NTMS*, pages 1–5, Tangier, Morocco, nov 2008. IEEE. DOI: 10.1109/NTMS.2008.ECP.45.

[112] CHRISTOPHE CLAVIER. *De la Sécurité des Cryptosystèmes Embarqués*. PhD thesis, (french). Université de Versailles Saint-Quentin-en-Yvelines, November 23 2007.

[113] VINCENT MAINGOT, JEAN-BAPTISTE FERRON, RÉGIS LEVEUGLE, VINCENT POUGET, AND ALEXANDRE DOUIN. **Configuration errors analysis in SRAM-based FPGAs: software tool and practical results**. *Microelectronics Reliability*, **47**(9-11):1836–1840, 2007.

[114] G. TORRENS, B. ALORDA, S. BARCELÓ, J. L. ROSSELLÓ, S. BOTA, AND J. SEGURA. **An SRAM SEU Hardening Technique for Multi-Vt Nanometric CMOS Technologies**. In *DCIS*, November 12–14 2008. ISBN: 978-2-84813-124-5, Grenoble, France.

[115] JEAN-JACQUES QUISQUATER AND DAVID SAMYDE. **Radio Frequency Attacks**. In HENK C. A. VAN TILBORG, editor, *Encyclopedia of Cryptography and Security*. Springer, 2005.

[116] FABIAN VARGAS, D. L. CAVALCANTE, E. GATTI, DÁRCIO PRESTES, AND D. LUPI. **On the Proposition of an EMI-Based Fault Injection Approach**. In *IOLTS*, pages 207–208. IEEE Computer Society, July 6-8 2005. Saint Raphaël, France.