New Protection Strategies for Integrated Circuits

Jean-Michel Cioranesco

– PANTHÉON - SORBONNE –
UNIVERSITÉ PARIS 1

THÈSE DE DOCTORAT

Discipline : Informatique

présentée par

Jean-Michel CIORANESCO

Nouvelles Contre-Mesures pour la Protection de Circuit Intégrés

---

New Protection Strategies for Integrated Circuits

---

dirigée par David NACCACHE

Soutenue le 18 décembre 2014 devant le jury composé de :

M. David NACCACHE	Université Paris 1	directeur
M. Christof PAAR	RUB	
M. Pierre PARADINAS	CNAM	
M. Jean-Jacques QUISQUATER	UCL	rapporteur
M. Christophe ROSENBERGER	ENSICAEN	rapporteur
M. Camille SALINÉSI	Université Paris 1	
M. Marc WATIN-AUGOUARD	Général d'Armée	

Remerciements

Cela a été un grand honneur pour moi de travailler sous la direction de David Naccache. Je le remercie de m'avoir accordé sa confiance en acceptant la direction de cette thèse et me faisant découvrir le monde si passionnant de la cryptographie. Merci de m'avoir donné les clés pour comprendre quelques-uns de ses secrets. L'aide, les encouragements et les conseils que David n'a pas cessé de me prodiguer tout au long de mon travail de recherche ont été essentiels dans l'élaboration de cette thèse. Je lui exprime toute ma reconnaissance.

Je remercie chaleureusement Messieurs Jean-Jacques Quisquater et Christophe Rosenberger qui ont bien voulu accepter de rapporter sur ce travail.

Mes plus vifs remerciements s'adressent aussi à Monsieur Camille Salinési pour avoir accepté de présider le jury, ainsi qu'à Messieurs Christof Paar, Pierre Paradinas et le Général Marc Watin-Augouard qui me font l'honneur de participer au jury.

Je voudrais maintenant me tourner vers mes collègues et vers mes coauteurs. Merci à tous pour votre amitié et collaboration !

Toute ma reconnaissance, pour sa confiance et ses conseils, à Monsieur Yazid Sabeg sans qui ma vie professionnelle n'aurait pas pris l'orientation qu'elle a aujourd'hui.

Ma gratitude va aussi à Altis Semiconductor, fonderie française de semi-conducteurs, qui a permis de donner un caractère appliqué à mes recherches. Je remercie tout particulièrement l'équipe des doctorants, Roman Korkikian, Guilherme Ozari, Nicolas Champetier, Elhadi Amirouche, Rodrigo Portella et Simon Cogliani pour leur soutien. Je tiens à remercier Fabrice Nogueira et Fabien Lepape de l'équipe du failure analysis pour leurs conseils, ainsi que Laurent Dulau, Raymond Ribas et Raymond Fillipi pour nos échanges autour du design analogique. Une pensée également pour les managers d'Altis, Franck Morier, Brahim Belgacem, Arnaud Salomon, Gianmaria Mazzucheli et Pascal Louis. Je tiens également à adresser mes remerciements à l'équipe de Secure IC, dont Sylvain Guilley et Jean-Luc Danger, pour leur collaboration. Une mention particulière pour Hervé Chabanne et Vincent Despiegel de la société Morpho que je remercie pour nos échanges. Une pensée pour tous ceux que je n'ai pas pu nommer mais qui se reconnaîtront.

Mes vifs remerciements à la société Rambus et son département Cryptography Research, pour l'opportunité professionnelle qu'ils m'ont donnée. Je remercie Ron Black et Craig Hampel pour leur confiance.

Enfin, je voudrais remercier mes parents pour leur soutien sans failles, ma femme Chloé et mon fils Joseph à qui cette thèse est dédiée.

Résumé :

La sécurité matérielle, préoccupation primordiale de nos jours, concernait au départ certaines applications très spécifiques, essentiellement bancaires, grâce à l'avènement des cartes à puce. Mais aujourd'hui, les domaines d'application de la cryptographie embarquée sont très divers et de plus en plus nombreux. Ils vont des domaines très spécialisés comme par exemple, le domaine médical, à ceux grand public avec, comme cas particulier universellement connu, les téléphones intelligents. Ces derniers se retrouvent au croisement de toutes les applications personnelles, avec un besoin évident de confidentialité des données et également de sécurité d'accès des moyens de paiement.
Les attaques matérielles invasives (celles qui ont comme résultat des circuits modifiés, souvent de façon irrémédiable) ont fait de tous temps partie de l'environnement industriel. La recherche d'innovation associée à la compétition, a constamment poussé les industriels à analyser les produits de leurs concurrents et à en faire une retro-conception pour comprendre leur fonctionnement. Ce qui fut vrai pour la mécanique, l'est également pour l'électronique. La retro-conception de circuits électroniques permet d'identifier des solutions technologiques ou des conceptions de circuits spécifiques, afin de les implémenter.
Dans le cas de la sécurité embarquée et des algorithmes cryptographiques utilisés, le secret à récupérer est double. En premier lieu, trouver de quelle façon a-t-on conçu et implémenté un circuit spécifique à un algorithme cryptographique ; deuxièmement trouver les clés secrètes utilisées pour le chiffrement ou la signature. Il s'agit donc de comprendre une architecture mais également de récupérer des données. Ces clés secrètes sont le plus souvent gravées dans une mémoire non volatile, ce qui permet à un attaquant capable de sonder cette mémoire ou de l'analyser optiquement, de récupérer le secret, on parle alors d'*attaques invasives*. Une deuxième classe d'attaques permettant de récupérer le secret d'une puce électronique, est celle des *attaques par canaux cachés*. Ces attaques, afin de récupérer des secrets, utilisent l'analyse de certaines grandeurs physiques, telle que la consommation de courant par exemple.
Les attaques invasives (qu'on peut caractériser comme sophistiquées) requièrent non seulement des attaquants experts en semi-conducteurs mais aussi un matériel hautement spécialisé et (par conséquent) onéreux. Ces attaques sont généralement réservées aux laboratoires de fonderies de silicium ou aux centres de recherche qui possèdent les équipements nécessaires pour les réaliser. Les techniques employées pour cette classe d'attaques découlent de celles utilisées dans les laboratoires d'analyses de défauts et de tests des circuits intégrés. Le but premier de ces laboratoires est la mise en évidence de défauts dans les circuits lors de la production, le deuxième étant de tester leur bon fonctionnement et de les déboguer si nécessaire.
L'équipement des laboratoires d'analyses de défauts permet la mise à nu des circuits encapsulés sans les affecter, d'enlever la couche de passivation ou de certaines couches

de métaux pour l'analyse optique, ou même d'enlever ou d'ajouter certaines parties d'un circuit grâce à l'utilisation d'un FIB. Ces techniques, destinées à l'origine à des tests de qualité lors de la production de semi-conducteurs, ont été détournées de leur utilisation première et servent désormais aux attaques invasives. La première utilisation malveillante de ces outils est la retro-conception. Dans un univers technologique très compétitif, les différents acteurs d'un même marché analysent des produits concurrents afin de comprendre les solutions utilisées pour résoudre un problème donné et de détecter d'éventuelles violations des droits de brevet. Dès lors que les produits sécurisés sont apparus, l'intérêt des ingénieurs et observateurs pour la retro-conception a fortement augmenté. Prenons l'exemple des cartes téléphoniques à unités qui étaient la norme de la fin du siècle précédent. Les unités étaient représentées physiquement sur le circuit de la carte par des fusibles d'une échelle d'une centaine de micromètres. Au fur et à mesure de la conversation téléphonique, les fusibles étaient grillés en leur appliquant un courant nominal donné. La possibilité de modifier le circuit et de court-circuiter un fusible permettrait de créer une carte à utilisation illimitée. Bien heureusement, les outils nécessaires à un tel exploit sont réservés à des entités de grande ampleur. Mais la menace, bien réelle, continue de grandir avec la généralisation des cartes à puce.
Les laboratoires de tests, quant à eux, ont pour mission de déboguer les circuits sortant de la production. Les circuits peuvent être testés au niveau des tranches des semi-conducteurs, ils ne sont alors pas découpés et sont sondés directement. Les tests interviennent également après la mise en boitier où divers logiciels peuvent être téléchargés dans le circuit. Les techniques de tests peuvent être biaisées à des fins malveillantes puisqu'elles permettent de sonder un circuit sur des lignes de métaux qui ne sont pas destinées à communiquer vers l'extérieur. Ceci peut par exemple, permettre de sonder une ligne transportant les clés secrètes de l'algorithme de chiffrement. Un test de fabrication d'une puce requiert l'inclusion d'un circuit spécifique, la chaîne de scan, qui permet au testeur d'accéder à toutes les portes logiques du circuit afin de vérifier leur bon fonctionnement. Bien sûr, cette fonctionnalité constitue une menace pour la sécurité embarquée si elle n'est pas implémentée correctement. Un concepteur de circuit peut par exemple, concevoir un algorithme de chiffrement avec une clé secrète archivée dans une mémoire non volatile à laquelle seul l'algorithme pourrait accéder. Mais plus tard, dans l'intégration des différents blocs formant le circuit intégré, un autre ingénieur peut insérer une chaîne de scan par un logiciel, sans se soucier des contraintes de sécurité d'une partie du circuit, en rendant ainsi les clés secrètes accessibles par la chaîne de scan. De nombreuses attaques utilisant ce vecteur ont été rapportées. Il faut remarquer que l'inclusion d'une chaîne de scan dans un circuit sécurisé fait partie des problèmes les plus difficiles à résoudre. Une solution consiste notamment à utiliser un BIST pour scanner la partie sécurisée du circuit.
Contrairement au cas des attaques invasives, la mise en place d'attaques utilisant une chaîne de scan est facile à mettre en œuvre car généralement, elle ne nécessite pas de matériel très onéreux. Par ailleurs, très souvent les chaînes de scan sont désactivées

après les tests, au moyen de fusibles. Dans cette situation, la reconnexion de la chaîne de scan est rendue possible soit en appliquant du courant aux deux bouts du fusible, soit en utilisant une machine à faisceau d'ions focalisés (FIB) pour effectuer un dépôt de métal et reconnecter ainsi la partie brulée du fusible.
Les attaques par canaux cachés furent introduites à la fin du dernier siècle sous la forme d'analyse de la consommation de courant d'un circuit intégré et d'analyse temporelle de l'exécution d'un algorithme. Les attaques par canaux cachés font référence à l'utilisation de différentes grandeurs physiques comme source d'information. L'exemple de la consommation de courant illustre bien cette classe d'attaques. La technique actuelle des semi-conducteurs est caractérisée par le fait que les données qui ont été manipulées peuvent être corrélées avec la consommation de courant. Une observation de cette consommation permet donc d'obtenir certaines informations sur l'opération cryptographique qui a eu lieu. L'utilisation de la technique de filtrage analogique ou/et digitale, des techniques statistiques et des transformées de Fourier, rend l'analyse de la consommation de courant particulièrement efficace.
Un exemple « frappant », illustrant bien le concept de canal caché cette fois-ci dans le monde physique et non pas dans celui des circuits, est l'utilisation des sons émis par les différentes touches d'un clavier pour déchiffrer des frappes. Chaque touche a un son correspondant à une fréquence particulière. Si un attaquant peut enregistrer et transférer dans le domaine fréquentiel les sons émis par les touches d'un clavier en créant une bibliothèque de ces sons, il sera par la suite capable de lire ce qu'un utilisateur tape sur ce clavier !
Il existe différents canaux cachés dans un circuit intégré, exploitables avec plus ou moins d'efforts. Il est intéressant de mentionner que ces vulnérabilités peuvent concerner la couche matérielle dans le cas d'un circuit à application spécifique (ASIC) ou programmable (FPGA), mais également la couche logicielle embarquée. Ceci est vrai pour la consommation de courant, les émanations électromagnétiques, les injections de fautes, l'analyse des temps d'exécution et autres canaux cachés.
Les méthodes pour contrer les attaques invasives et les attaques par canaux cachés sont nombreuses et nécessitent différentes stratégies en fonction de la menace à contrer. On parle de contre-mesures pour désigner les actions défensives visant à empêcher l'utilisation malveillante d'un circuit imprimé.
Dans le cas des attaques invasives, les méthodes mises en place pour contrer un attaquant sont généralement coûteuses en surface de puce. Certaines solutions d'encapsulations sécurisées permettent de rendre les attaques plus difficiles à mettre en œuvre mais elles ne font que complexifier la tâche sans éliminer la vulnérabilité. Une autre technique est l'utilisation d'un morceau de circuit fictif visant à perturber l'attaquant dans son effort de retro-conception, en dissimulant les fonctions de la porte logique au niveau de la configuration. La solution la plus répandue pour lutter contre le sondage de certaines lignes logiques dans le circuit est l'utilisation d'un bouclier actif. Un bouclier est composé d'un capteur et d'un petit circuit logique ; le capteur est formé d'un

ou plusieurs fils en forme de serpentin s'étendant sur une ou plusieurs couches métalliques supérieures du circuit. Le circuit du bouclier actif détecte toute modification du capteur et est ainsi capable de déceler une éventuelle attaque invasive du circuit intégré en déclenchant un signal d'erreur.
Les contre-mesures concernant les attaques par canaux cachés sont très variées. Elles peuvent être redondantes lorsque l'origine physique de la fuite d'informations est la même, c'est le cas pour la consommation de courant et les émanations électromagnétiques. La stratégie choisie est bien sûr, la mise en place de moyens supprimant cette fuite. Dans le cas de la consommation de courant, les stratégies sont multiples, on peut bien évidemment recourir à l'utilisation d'un brouilleur (sorte de générateur de bruits) et de filtrer la consommation à l'aide d'une alimentation régulée interne à la puce, mais l'efficacité reste limitée. Une autre solution consiste en l'utilisation d'une porte logique balancée, qui consomme la même quantité de courant quelle que soit la donnée manipulée, cette contre-mesure engendre une forte augmentation de la surface du circuit. La meilleure solution reste toujours et encore, la suppression des vulnérabilités à un niveau algorithmique en utilisant par exemple, des techniques de masquage.

L'objectif de cette thèse est de proposer de nouvelles solutions pour protéger les circuits intégrés contre ces attaques physiques.
La première partie décrit les notions d'attaques par canaux cachés, d'attaques invasives et de retro-conception. Plusieurs exemples de ces types d'attaques ont pu être mis en œuvre pendant le travail de recherche de cette thèse, ils sont présentés en détail dans cette partie.
La deuxième partie est consacrée à des propositions de différentes contre-mesures pour contrer des attaques par canaux cachés ayant pour vecteur la consommation de courant.
La troisième partie est dédiée à la protection contre les attaques invasives en utilisant divers types de boucliers et capteurs. Nous conclurons ce manuscrit de thèse par la proposition d'un bouclier actif cryptographique inviolable ayant pour but premier de contrer le sondage, mais aussi celui de détecter l'injection de fautes et d'être immunisé contre les analyses par consommation de courant.

Descripteurs :

Attaques par canaux cachés, attaques invasives, attaques par fautes, analyse par consommation de courant, circuit intégré, bouclier actif, émanations électromagnétiques, chiffrement sécurisé, sécurité embarquée

Abstract :

Hardware security awareness was at first limited to very specific applications, mainly focused on the banking industry through the generalization of smart cards. Embedded security applications are now much more diverse, going from medical applications to general public applications. Products such as smartphones, which are at the center of all personal embedded applications, introduced an obvious need for data confidentiality and security in general.
Invasive attacks on hardware have always been part of the industrial scene. Constant research on innovation associated with high competition pushed circuit manufacturers to analyze competitors' products by reverse-engineering and to investigate their functioning. Reverse engineering enables circuit architects to understand technological or design solutions, and eventually to implement them.
In the case of embedded security and cryptography, there are two secrets to recover. First, there is the way designers implemented a specific cryptographic functions, and second, which secret keys are used for encryptions or signatures. There is both an architecture to understand and data to be retrieved. Secret keys are very often stored in a non-volatile memory, which allows an attacker to probe this memory or in some cases to read keys optically. These attacks are called *invasive attacks*. A second class of attacks is that of *side-channel attacks*. These ones are using physical manifestations of a crypto algorithm, such as power consumption, to retrieve a secret.
Invasive attacks form an advance class of attacks that requires a skilled attacker and very specialized and expensive hardware. These attacks are usually only performed by foundry failure analysis lab or research labs that have the tools to perform them. Recently, the business of counterfeiting chips became so profitable that some companies offer as service to reverse engineer and analyze integrated circuits, which is perfectly legal. Techniques used in this class of attacks come from foundries' failure analysis or test laboratory techniques. The goal of these foundries departments is to detect and understand the defects in circuits during manufacturing, then test and debug circuits afterwards.
Laboratory analysis equipments enable to depackage an integrated circuit without altering its functioning, to remove the passivation layer, to remove some metal layers for optical analysis or even to edit circuitry using FIB. These techniques were at first dedicated to the quality tests during manufacturing, but have been diverted to serve invasive attacks. The first malicious use of these tools was done for reverse-engineering. In a very competitive industry environment, different actors of a same market analyze competitive products to understand which solutions have been used to solve a given problem and detect possible patent violations. Since smart cards appeared, the interest in reverse engineering got really higher. Let us take the example of the French phone smart cards that were based on the concepts of credit units, physically represented on the card by fuses of a few hundred micrometer scales. While using the phone card, cre-

dits are consumed on the card using a fuse controller that burns fuses one by one. If an attacker would be able to short-circuit one of these fuses, the card would become an unlimited credit card as the fuse could not be burned. Fortunately, the tools needed for such an exploit are reserved for large entities, nevertheless the invasive attack threat exists and will keep growing with the smart card reign. Foundries' test laboratory's mission is to debug circuits exiting production. Circuits can be tested directly on wafers, dies have not been cut-out and they get directly probed on the wafer. Tests are also performed after dies are packaged. Techniques used during the test can be skewed to serve a malicious purpose since they allow an attacker to probe metal lines which are carrying secrets. For example, an attacker could probe a key management system (KMS) key delivery bus carrying keys for an encryption algorithm, which is usually composed of a single wire in case of serial bus, and could read the key out. The test phase of chip manufacturing requires the inclusion of a specific circuit dedicated for testing the correct functioning of logic gates. This functionality is of course, a threat to embedded security if not implemented carefully. A circuit designer can for example, design a cipher algorithm using a secret key stored on a non-volatile memory embedding several countermeasures, but having a DFT engineer inserting a scan-chain into his design without paying attention to security constraints. Then an attacker entering debug mode could read out the secret key. Several attacks using this vector have been documented, and integrating DFT in a secure circuit is a very hard problem to solve. One solution is to build a BIST at the design phase to scan the secure part of the design. On the contrary of invasive attacks, attacks using scan-chain are easier to implement since they do not usually require expensive material. Very often scan-chains are deactivated after test using dedicated fuses. In this case, reconnecting the scan-chain is possible either if a current can be applied on each side of the fuse, or by using a focused ion beam machine (FIB) to reconnect the burnt fuse.

Side-channel attacks were introduced at the end of the last century with power and timing analysis. Side-channel attacks refer to the use of physical parameters to deduce information on a system's architecture and the data being processed. Power analysis is the most studied examples of this class of attacks. The actual CMOS semiconductor technology presents the characteristic that the data which are handled can be correlated with power consumption. An observation of power consumption can reveal information on the cryptographic operation that took place and thus, threaten strong protocols and algorithms. The use of analog filtering, demodulation and digital signal processing combined with statistical methods renders power analysis a very efficient tool to break a cryptosystem.

A very good example of side-channel attacks is the use of sounds to analyze keyboard strokes. On a given keyboard, each stroke creates a different sound in the frequency domain. If an attacker can record sounds of each keystroke and can create a library of these sounds, then he is able to deduce what a user is typing by recording the keyboard sound. Different side-channels can be used on an integrated circuit, accessible

with more or less effort. It is interesting to note that these leakages concerns the hardware layer in the case of an ASIC or FPGA implementation as well as the software layer. This is true in the case of power analysis, electromagnetic emanation, fault injection, timing analysis and other side channels. Countermeasures design specific defensive actions aiming to prevent certain attacks on a chip. Many hardware countermeasures exist against invasive and side-channel attacks and they are very different depending on the threat they address.
In the case of invasive attacks, countermeasures are costly in the chip area. Some products use tempered packaging, which can make attacks more difficult but without eliminating the threat. Another technique is the use of fake circuitry to make reverse engineering harder and hiding logic gate functions at layout level. The most common protection against probing is the use of active shields. A shield is composed of a sensor and a small digital or analog circuit, the sensor is composed of one or several meander shaped wires spreading on one or several metal layers. The active shield circuit purpose is to detect any modification in the sensor and to trigger an error signal.
Side-channel hardware countermeasures are very diverse and can sometimes address different threats when the physical origin of the leak is shared as it is the case for power consumption and electromagnetic analysis. The chosen strategy is of course, to suppress the information leakage. In the case of power consumption, there are multiple strategies ; we can for example, use noise generators. Another solution consists in using balanced logic gates that consumed the same amount of power regardless of the data being processed, this countermeasure induces a significant increase in the chip's area. But the best solution remains to suppress leaks at an algorithmic level by using masking techniques, in this case designers need to pay attention how they handle the mask and the data.
The aim of this thesis is to propose new solutions in order to protect embedded circuits against some physical attacks described above. In a first part of the manuscript, we detail the techniques used to achieve side-channel, invasive attacks and reverse-engineering. I could implement several of these attacks during my thesis research, they will be detailed extensively. In the second part we propose different hardware countermeasures against side-channel attacks. The third part is dedicated to protection strategies against invasive attacks using active shielding and we conclude this work by proposing an innovative cryptographic shield which is faulty and dpa resistant.

Keywords :

Side-channel attacks, invasive attacks, differential power analysis, integrated circuit, active shield, embedded security, cryptography, fault attack

Abreviations

2D	2 Dimensions (circuit)
3D	3 Dimensions (circuit)
AES	Advanced Encryption standard
ASIC	Application Specific Integrated Circuit
BIST	Built In Self Test
CMOS	Complementary Metal Oxide Semiconductor
CFG	Clock Fault Generator
CPU	Central Processing Unit
CRT	Chinese Remainder Theorem
DFA	Differential Fault Attacks
DFT	Default and Fault Tolerance
DIY	Do It Yourself
DLL	Delay-Locked Loop
ECC	Error-Correcting Code
EDC	Error-Detecting Code
EMA	Electromagnetic Analysis
EMC	Electromagnetic Compatibility
FIB	Focused Ion Beam
FPGA	Field Programmable Gate Array
GCD	Great Common Divisor
GEC	General Error Counter
HD	Hamming Distance
HF	hydrofluoric Acid
HNO3	fuming Nitric Acid
HODPA	High Order Differential Power Analysis
HW	Hardware
HW	Hamming Weight
IC	Integrated Circuit
KMS	Key Management System
LASER	Light Amplification by Stimulated Emission of Radiation
LED	Light-Emitting Diode
LFSR	Linear Feedback Shift Register
LUT	Look-up table
Nd :YAG	Neodymium-doped Yttrium Aluminium Garnet
NMOS	N-channel Metal Oxide Semiconductor
NVM	Non Volatile Memory
PA	Power Analysis
PLL	Phase Locked Loop
PUF	Physical Unclonable Function

PMOS	P-channel Metal Oxide Semiconductor
PRNG	Pseudo Random Generator
RAM	Random Access Memories
RNG	Random Number Generator
RTL	Register Transfer Level
SCA	Side-Channel Attack
SEM	Scanning Electron Microscopy
SEU	Single Event Upset
SiP	System in Package
SRAM	Static Random Access Memory
SW	Software
TC	Target Circuit
PCB	Printed Circuit Board
PCC	Pearson Correlation Coefficient
UV	Ultra Violet

Table of contents

Introduction

Nowadays, hardware trust and security play a important role because integrated circuits (IC) are present in many critical infrastructures as financial, military, health, etc. Many cryptographic IPs are integrated to assure the security of ICs. But, these cryptographic IPs, in their turn, can be attacked.
Side-channel attacks (SCA) on an integrated circuit performing cryptographic operations are a hot topic of embedded security. These threats can be classified in three categories : non-invasive, semi-invasive and invasive attacks.

- Non-invasive attacks are characterized by the fact that the chip does not suffer any physical modification, its functionality remains intact. These attacks require very little hardware and are fairly inexpensive to achieve, therefore, they are a huge problem of IC security. Such an attack can take several forms. It can use the implantation of the algorithm to monitor the timing of certain executions (example timing). Another very common threat that we will develop extensively relates to power consumption and electromagnetic emanation observation. Other form of non-invasive attacks relates to injecting fault through the chips pads (input, clocks, power supply) by glitching the signals at precise timing. The goal is to perturb the functioning of the system by external means as for instance, peaks of tension in power supply or changes of temperature [8].

- In the case of semi invasive attacks, the chips packaging is removed on its surface but the silicon remains untouched. The attacker does not have any direct contact with the surface of the integrated circuit. In this class of attacks one can find techniques allowing reading the contents of a memory cell without direct probing [84]. Some attack scenarios by fault injection are also considered to be semi-invasive [90, 89]. Semi-invasive attacks require less competence and less investment in material than the invasive ones but they still require access to chemicals and advance knowledge on decapsulation. This scenario is also very common in electromagnetic analysis, where decapsulating the chip allows near field probe to capture emissions in a more localized manner. On the other hand, on a modern integrated chip, it is not obvious to locate the best place for performing an attack.

- Invasive attacks are a less common attack scenario as they require very advanced knowledge and equipment on integrated circuits. An attack is invasive when the attacker has the possibility to modify physically the chip. Thus the attacker can gain direct access to the cryptographic system at various stage of its computation. An invasive attack begins usually by removing circuits packaging, technique known as « depackaging » (see Figure 7). The package serves as a protection for the IC and its bonding, allowing simple connections to the PCB. Very often one uses fuming nitric acid (HNO3) to dissolve the packaging without damaging the chip inside and hydrofluoric acid (HF). Then, thanks to a probing station, the attacker can have direct access to top layer metal lines of the circuit and observe directly the signals which transport the data. Recently advanced techniques based on focused ion beam (FIB) have been used to modify a circuitry [46]. Invasive attacks are generally very efficient but in return, they require not only a high competence of the attacker but also very expensive equipment [60, 89]. They also serve to clone chips which contain PUFs.

This thesis is organized in three chapters. In the first chapter we will review the state of the art in hardware attacks. In the second chapter we will then propose some HW DPA countermeasures. And in the third chapter we will propose invasive attacks countermeasures.

Part 1
State Of The Art

1 Hardware Attacks

1.1 Power and Electromagnetic Analysis

1.1.1 Power Analysis

1.1.1.1 Introduction

Differential Power Analysis (DPA) is a non-invasive attack that uses the biases power consumption of hardware devices to access keys and other secret information. Effective countermeasures are needed to protect keys and prevent attackers from using DPA for fraud, piracy, cloning, reverse engineering and espionage. In this paper we will present a protection circuit aiming using amplitude and time domain noise generation to hide the cryptographic algorithm signature. Power analysis was first introduced by P. Kocher [59], he discovered that the analysis of computers or microchip power consumption provide information about the operation they process. This is precisely one of the reasons why an hardware implementation of a cryptosystem might be vulnerable.

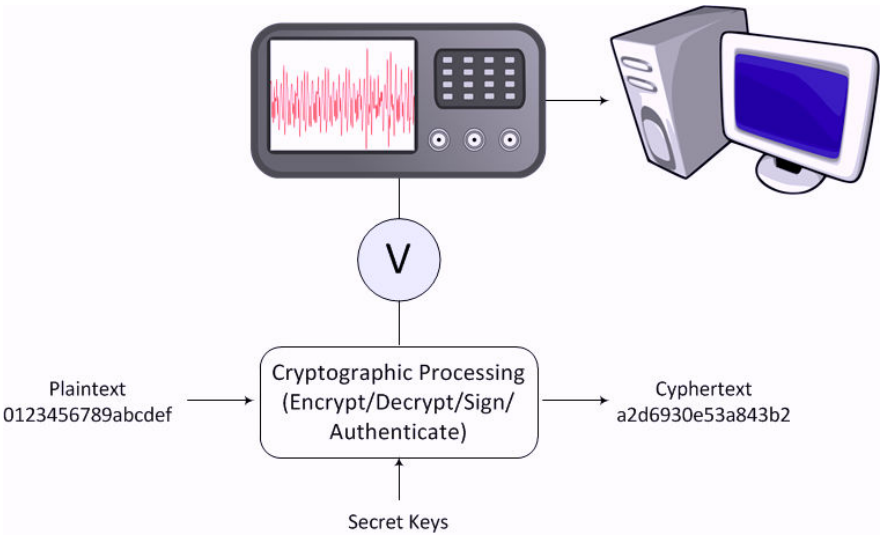


FIGURE 1.1 – schematic principle of power analysis

Side channel attacks have proven that a hardware implementation of a strong algorithm does not necessarily imply a secure cryptosystem, many studies [58, 59, 66] have shown that information leaks through power consumption or electromagnetic emanations. Transistors are voltage controlled gates, when a current is applied to the gate, current flows across the substrate and thecharge is delivered to the rest of the circuit. Arrangement of transistors is forming logic gates, and arrangement of gates forms a complex circuit ; transistors also serve to create registers which are the mean to store data in the circuit between operations. Whenever a register is changing state, switching from $1 \rightarrow 0$ or $0 \rightarrow 0$, it involves a certain power consumption.

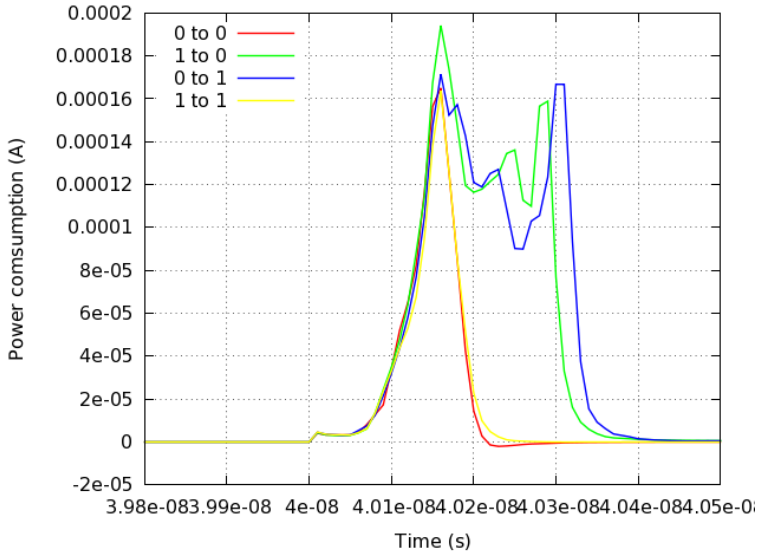


FIGURE 1.2 – Power consumption of a register for the four different possible switches

Figure 1.2 presents the differences in power consumption when a register is switching states. We simulate a one bit register with reset in a 130nm library, and we observe that the power consumption is different according to the value which is stored.
As registers switches are directly linked to the data manipulated by the cryptographic algorithm, an attacker can use statistical tools on the power signatures to retrieve the secret key of a cryptographic algorithm. The overall power consumption of the chips reflects the activity of all individual transistors, including registers, and is correlated to the computation being performed. In this paper we will investigate the idea of using secured registers in cryptosystems, I.E. registers that consume the same power regarding the data manipulated. Secured register would suppress the power leakage source and thwart Power Attacks.

1.1.1.2 Setting-up a DPA bench

A dpa platform can be set-up with regular lab equipment. For our setup we are using the following equipment :

- A computer for data acquisition and post-processing
- An FPGA board or a micro controller to load HW and SW designs to analyze. Development boards have usually their power supply pins easily accessible on the PCB, a common practice is to analyze the current flowing through a shunt resistor placed on the power supply. It is a better practice to measure power consumption after the power regulator as it is filtering part of the meaningful information. The board will be programmed by the computer through USB interface, but it also needs to receive encryption command. We achieved this using serial interface (RS232) and a UART interface in the FPGA.
- An oscilloscope with a sampling rate higher than the device operating frequency and trigger capability 1.3. In our setup, when encryption is launched on the FPGA, a trigger signal is issued in the RTL on one pin which is connected to the oscilloscope trigger input. This enables to start data acquisition at the same exact point of encryption, we thus avoid the need of aligning traces during post-processing. In device analysis, misalignment is a major issue which is overcome in our setup.
- A voltage probe that enable us to measure the current passing through the shunt resistor. (or an EM probe)

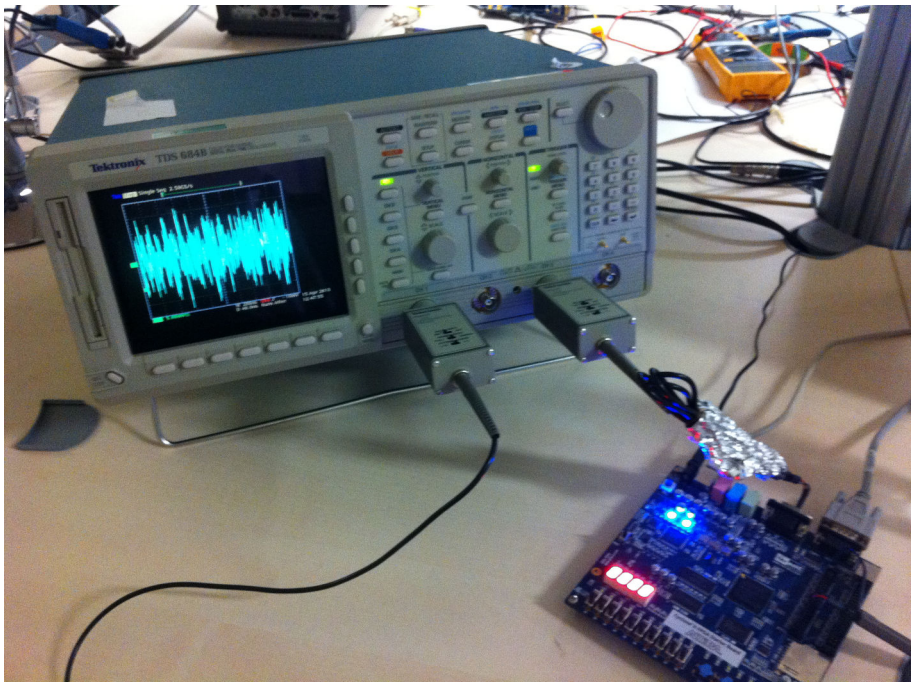


FIGURE 1.3 – DPA bench using an oscilloscope to acquire power traces from a HW cryptosystem running on FPGA

The DPA analysis is performed in two distinct steps :

- **Data Acquisition :** The bench computer is running an acquisition software. This software is composed of several part. First it contains the FPGA compiler and programmer that enables programming the HW design to the FPGA using USB and then the JTAG debug chain. This tool allows us to synthesize our design to the FPGA.
  The second part is composed of a C code which send an encryption request to the FPGA through RS232. For instance our AES design was receiving as input the 128 bit plain-text appended to the 128 bit key. Once the encryption is terminated, the design is sending back the cipher through this same RS-232. In the RTL design, we create a state for starting encryption which triggered a flag on a pin of the FPGA board. We connected this pin to the trigger input of an oscilloscope. When the oscilloscope receives an input trigger, he starts the acquisition of the measured signals for a defined number of sample. We adjusted the number of samples plus an offset to match the region of the encryption we want to capture. Once acquired the trace is dumped to the PC, in our case the oscilloscope was using a GPIB interface.
  This process achieves the acquisition of one sample composed of a plain-text, a key, a cipher-text and a power trace. This process is automated to be repeated again and again to achieve several hundred of thousands of encryptions.
- **Data Post Processing :** Once the captured a sufficient number of traces, this data needs to be post processed. The first thing to do is needed is to realign traces. Fortunately, due to our HW trigger, traces were sufficiently aligned to post-process them without realignment. Realignment is a complex process, first a reference trace is chosen and some characteristic points are selected, usually peaks of power related to an operation. According to these characteristic, traces are analyzed one by one, and offset so that the chosen points are aligned.
  After alignment, one needs to build a power model to correlate power with computed data. The principle of power modeling is to predict the power consumption given a key guess, for each trace correspond one value of predicted power. This power model is then correlated with actual power consumption. We can also sort traces according to a selected bit and a key guess.
  Correlation curve or dpa curve that will present a spike are of course very likely to correspond to a correct key bit or byte guess. The process is then iterated to retrieve the all keys.

1.1.1.3 Simple Power Analysis - Timing Analysis

Simple power Analysis is based on the sole observation of a cryptographic device power consumption. It can be performed using very little equipment, a simple oscilloscope

and voltage probe to monitor a chip's power consumption. It is a first approach on a black box attack to identify the type of algorithm or implementation we're dealing with by counting the number of rounds. On simpler algorithm SPA allows to disclose secret data by direct observation of power curves and identifying meaningful patterns. SPA involves understanding data being manipulated by the sole observation of power consumption over time.

For unprotected implementation, a single trace can be enough to get information on a cryptosystem. The famous example of RSA modular exponentiation [58] shows that a single power trace of an unprotected information can potentially reveal the secret key. This information can be done by observing the amplitude of the power consumption, or as in the RSA example it can be done by using the timing of an execution.

1.1.1.4 Differential Power Analysis

Differential Power Analysis as it was introduced is a rather simple but efficient way of showing a power leakage. The principle is to capture a certain amount of power traces from a given crypto operation using the same secret. DPA is traditionally operated on input bits, but there are many other ways power leakage can be proved. Let us take a given algorithm, we perform a certain amount of operations using the same secret, usually a secret key, and we capture the corresponding power traces. We then select one of the input bits of the algorithm and we separate the traces in two subsets according to the value of this input bit, "0" or "1". We then take the difference of the mean of the two subsets, and we obtain what is called a DPA trace. This trace is flat but presents some spikes where the selected input bit is used, this allows us to prove there is a leak and tells us where and how the input bit is used.

This principle allowed us to prove there is information leaked in the power trace, but it can also be used to retrieve the secret key. Let us use the same principle but with a key hypothesis. We take the example of AES 128bit. We have a set of traces T and a ciphertext C encrypted using the same unknown key K. Let us make an hypothesis on a key byte K_i, in the last round of AES the last operation is Sbox and then XOR Key, we call D the input of the Sbox, they verify the following equation :

$$C_i = Sbox(D_i) \oplus K_i.$$

We know the ciphertext C and we guessed the i-th byte of the key $_i$. We can then revert the equation and compute the corresponding intermediate byte D_i according to our key guess,

$$D_i = Sbox^{-1}(C_i \oplus K_j)$$

We select one bit in the byte i of the intermediate, and we separate traces in two subsets according to the values of this bit. We then trace the DPA curve, and if a spike is present,

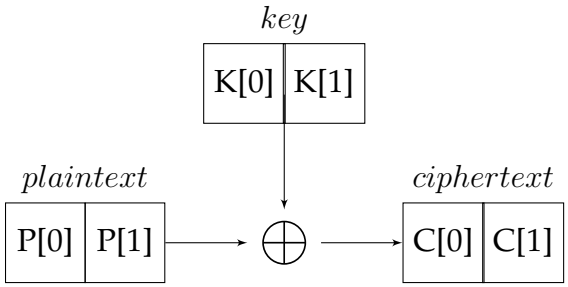


FIGURE 1.4 – Simple cipher example

there is a chance we guessed correctly the byte or parts of the bits of the key. With iteration, and guessing all bytes of the key, we can then retrieve the overall AES key.

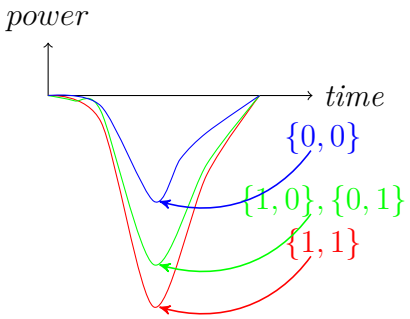
1.1.1.5 Correlation Power Analysis

Correlation Power Analysis is based on the same principle, but uses a statistical test to determine correlation between power and data. To understand how CPA works, it is important to spend some time on power modeling. Let us take the example of a simple padding encryption, the key is simply XORed to the text 1.4.

In sequential circuits, data intermediates and outputs are stored in registers. When data is written to $C[i]$ it will consume power proportionally to the data value. Depending on attacker's knowledge on the preceding state of the register, different power models are used. The most used power models are the Hamming Weight model and the Hamming Distance model. The Hamming Weight is very useful when the preceding states of the register are unknown. It is basically counting the number of ones in a set of registers. Figure 1.5 represents this power model applied to our simple XOR encryption example. On the contrary, when the preceding state of registers is known or can be guessed, we use the Hamming Distance, which computes the number of changing bits in a set of registers. An example of use of Hamming Distance to compute a power model is given in section 1.1.2.

We then apply this simple power model to the example given on Figure 1.4. We start with a key guess of $K = \{0, 0\}$, it is necessary to build a different power model for every key guess. We obtain the power model displayed on Figure 1.6.

Given a key guess, the Power Model we obtain predicts the power consumption of the encryption operation (Figure 1.4) according to the input plaintext. The goal is now to correlate the predicted power consumption with the real power consumption acquired through an oscilloscope. In some case when the preceding state of registers is known, we can use the Hamming Distance between the preceding state and the predicted state to mount a power model (cf. 1.1.2).



Plaintext	Cyphertext	Power Model
$\{0,0\}$	$\{1,0\}$	1ε
$\{1,0\}$	$\{1,1\}$	2ε
$\{0,1\}$	$\{0,0\}$	0ε
$\{1,1\}$	$\{0,1\}$	1ε

FIGURE 1.5 – Building a power model based on Hamming weight

#	Plaintext	Plaintext $\oplus$ K	Power Model
1	$\{0,0\}$	$\{0,0\}$	0
2	$\{0,1\}$	$\{0,1\}$	1
3	$\{1,0\}$	$\{1,0\}$	1
4	$\{1,1\}$	$\{1,1\}$	2

FIGURE 1.6 – Power model corresponding to key guess $K = \{0,0\}$

A single trace contains too much noise to be correlated as is, we need to use a batch of multiple encryptions to perform an efficient attack. Multiple statistical tools exist to correlate real power consumption with the predicted one, they are called distinguisher. We will introduce here the most used distinguisher which is the Pearson correlation coefficient. The following formula gives the Pearson correlation between two population x and y :

$$\rho_{xy} = \frac{cov(x,y)}{\sigma_{xx}.\sigma_{yy}} = \frac{E(xy) - E(x).E(y)}{\sigma_x.\sigma_y},$$

where cov is the covariance, σ_x is the standard deviation of x , μ_x is the mean of x , and E is the expectation.
If we apply the formula to a batch of n samples, we obtain

$$\rho_{xy} = \frac{n\sum_{i=1}^{n} x_i y_i - \sum_{i=1}^{n} x_i \sum_{i=1}^{n} y_i}{\sqrt{n\sum_{i=1}^{n} x_i^2 - (\sum_{i=1}^{n} x_i)^2}\sqrt{n\sum_{i=1}^{n} y_i^2 - (\sum_{i=1}^{n} y_i)^2}}$$

Let us apply this formula to our batch of samples, we acquired n traces of p points corresponding to n encryptions. We build a first vector V_1 composed of n elements, each element corresponds to the power model computed for the corresponding trace. We then build a second vector of size n*p, containing the n values of each point in time for all acquisitions. We then compute the Pearson coefficient for each point in time resulting on a Pearson vector of size n. The point of highest correlation among all traces indicates

the best candidate for a correct guess. Other distinguisher than Pearson coefficient are used to correlate data with power consumption. Statistical test as Mutual Information Analysis [41] taken from the information theory, or the Kolmogorov-Smirnov test has been used for power analysis. There is a constant debate in the research community on what distinguisher is the most efficient and this is mostly depending on the attacked implementation.

1.1.2 Example of Attack on AES

We implemented a basic 128bits AES in hardware, the code is following the FIPS-197 standard document [73]. The first clock is used to write $plaintext \oplus key$ to register RS, the next ten rounds are conventional AES rounds. In this implementation we use one 128 bits register noted RS, we note RS_i the register state at round i. It is the rewriting of this register that we will attack.

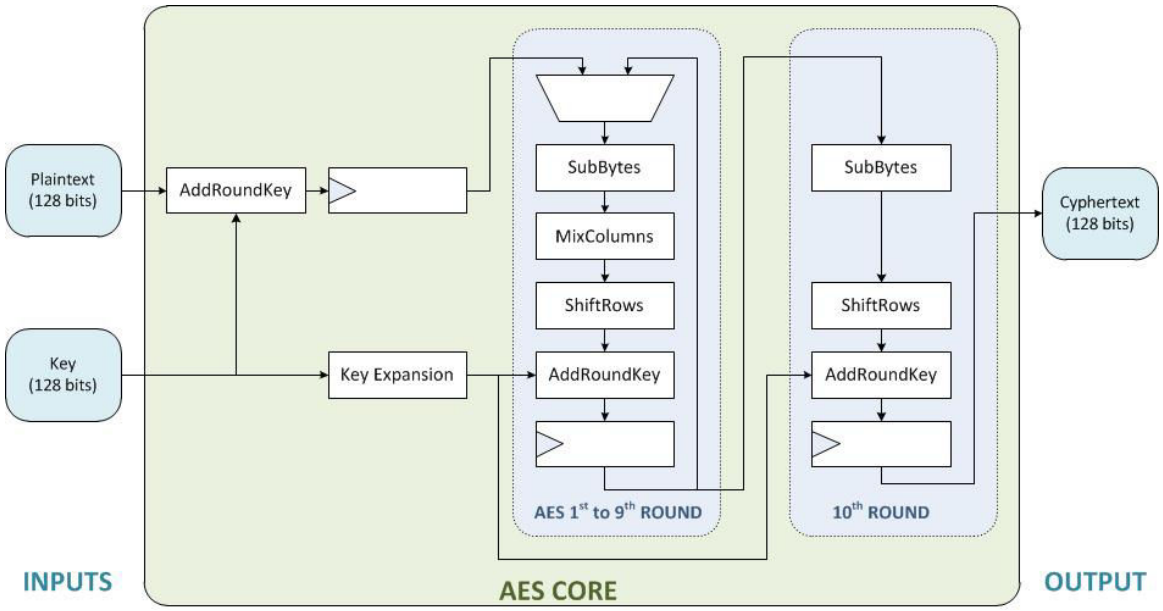
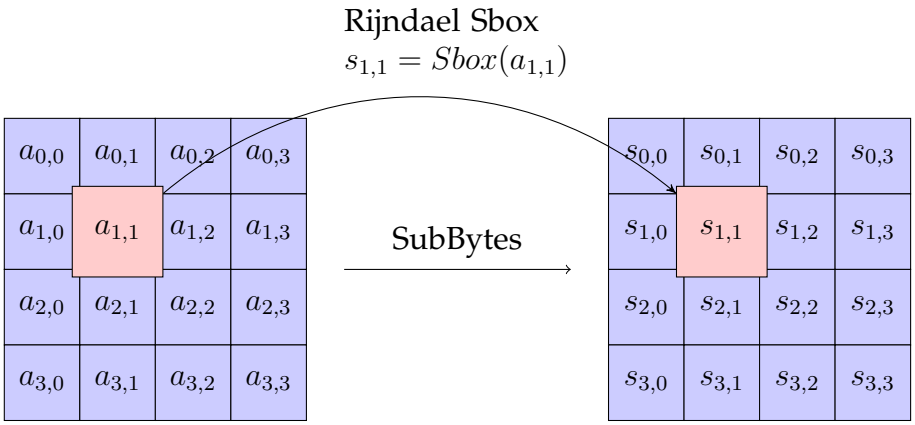
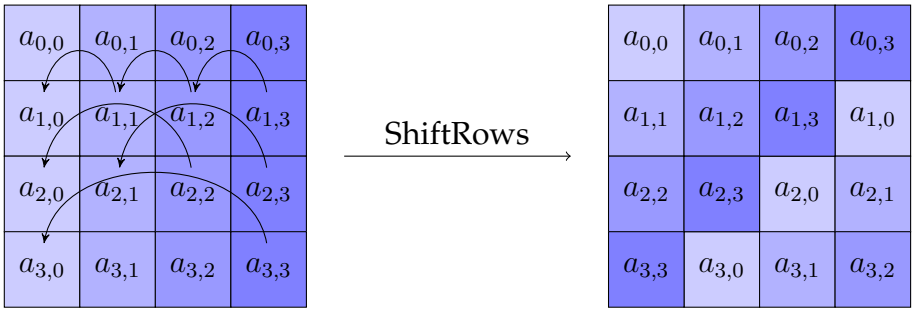
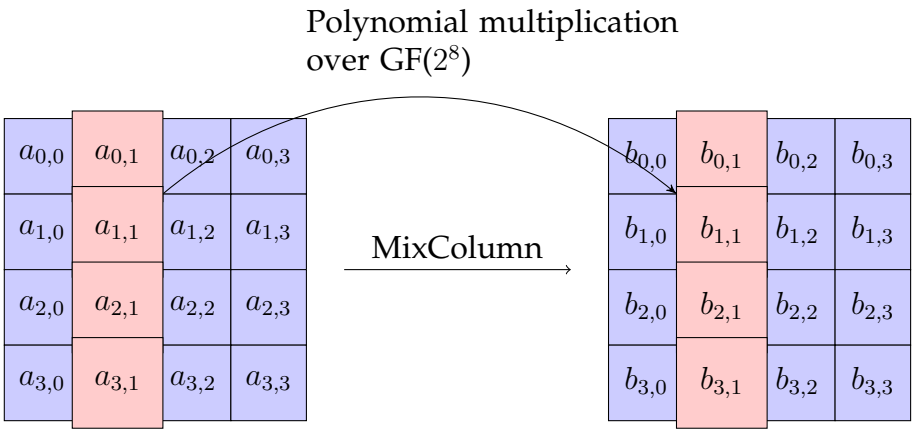


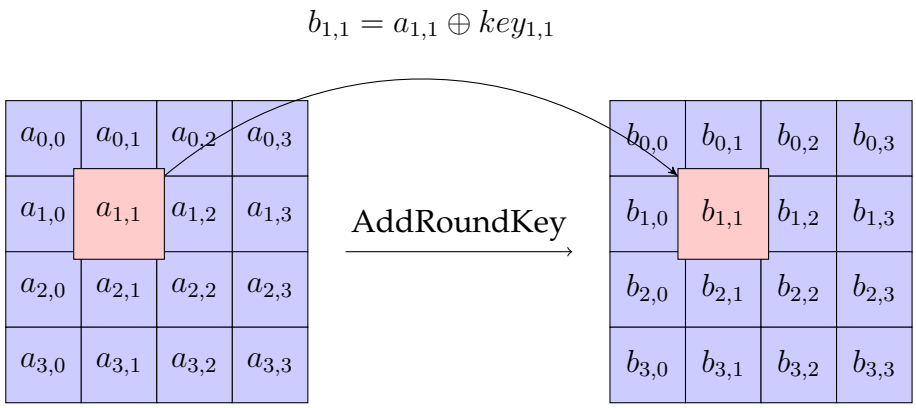
FIGURE 1.7 – Block Diagram of our fisrt AES version

The AES blocks are functions operating on arrays of 16 elements of one byte, the 128 bit input is split into words of 4 bytes being the four lines of the array. The SubByte function is a substitution of each element of the array using a look-up table, each byte in the array is updated using an 8-bit substitution box, the Rijndael S-box.

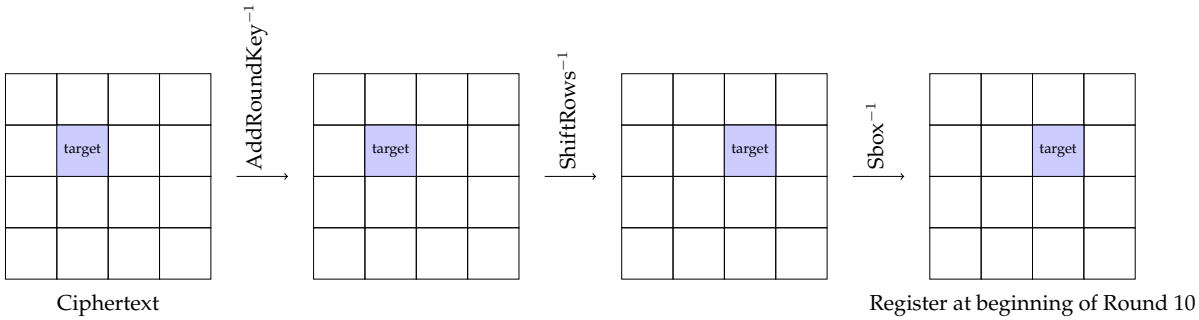

Rijndael Sbox
$s_{1,1} = Sbox(a_{1,1})$
SubBytes


ShiftRows


Polynomial multiplication
over GF(2^8)
MixColumn



We will here present an attack on the last round of AES, our primary target for the attack is the register RS. At the last round, RS gets rewritten from RS_10 to ciphertext. This operation consumes power and some information is leaked through that power. We will first build a basic power model on a single key byte.



We are building our power model on the operation $O(RS_{10} \leftarrow C)$ which consumes a power $P \propto HD(RS_{10}, C)$.
Or $HD(RS_{10}, C) = HW(RS_{10} \oplus C)$
So we choose one byte of the key that we guess among the 2^8 possibilities, and we build a power model for this guess. We then correlate the power model we have created with the power traces of the corresponding encryptions, we compute the Pearson correlation coefficient for each point of the graph and we look for a correlation on the 10_{th} round. Most of the correlation curves are flat or do not present any significant spike, but for one key guess we find a correlation indicating a correct key guess (see Figure 1.8).

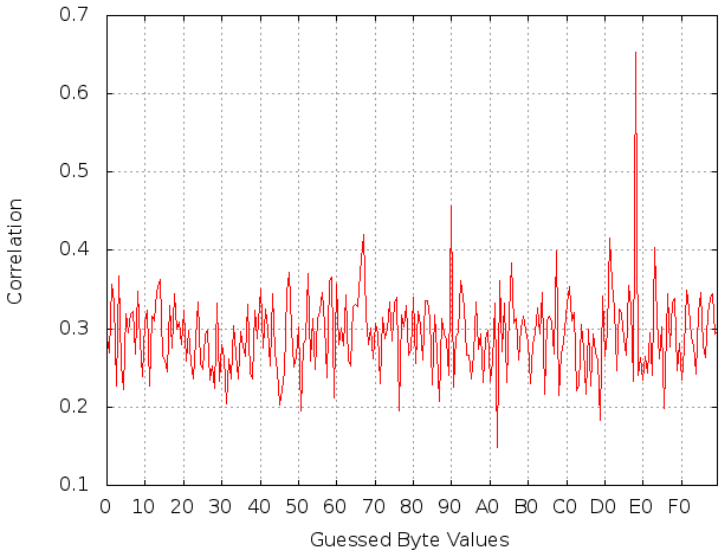


FIGURE 1.8 – Highest correlation point according to the different key guesses for a batch of 200 traces

For this unprotected implementation, only 200 traces were needed to have a serious key candidate among all possibilities. Figure 1.9 represents the correlation of this correct key guess over time, we can see the moment in time where the correlation happens, and as expected, it is happening during the tenth round of AES.

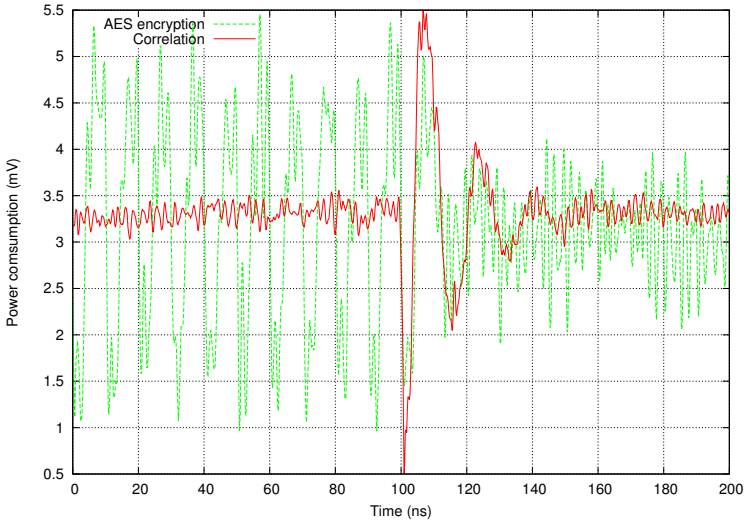


FIGURE 1.9 – AES power trace versus time (in green) and correlation versus time (in red) for a batch of 5000 traces

1.1.2.1 Correlation Frequency Analysis

Analyzing a power trace in the power domain presents the drawback of capturing the entire spectrum of frequencies, which contains several undesired signals. To realize DPA, we actually do not need to analyze the entire spectrum of frequencies, but only the frequencies that are carrying meaningful operation. Using Fast Fourier Transform we can convert power traces from power domain to the frequency domain. Once in the frequency domain, unwanted frequencies can be filtered. In synchronous designs, clocks are re-written at every positive clock edge, thus the clock frequency is carrying the register signal. After removing unwanted frequencies, we can then flip back from frequency to the power domain by using FFT and perform correlation of the cleaned power traces. We will obtain a much better attack result after operating this post-processing.

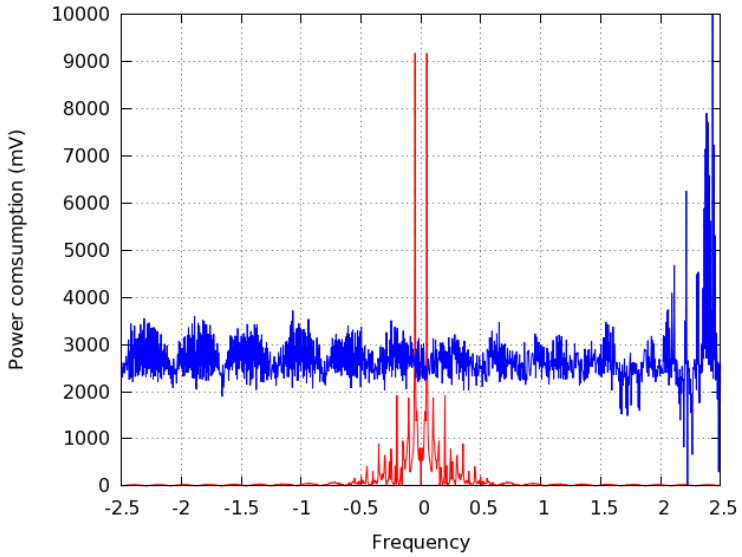


FIGURE 1.10 – AES power trace in frequency domain (in red) and correlation versus time (in blue) for a batch of 1000 traces

1.1.2.2 High Order Power Analysis

High order power analysis is targeted towards protecting implementation by using masking techniques. The statistical techniques used are similar to the one we described, but HODPA implies more post-processing of the traces and so requires more time to be achieved. We refer the reader to [76] for further information on this technique.

1.1.3 Electromagnetic Analysis

The principle of electromagnetic analysis is the same as power analysis. Both attacks are using the same post processing techniques, the difference lies on the source of the leakage. Current and the electromagnetic field are linked through the Maxwell's equation ; when a current flows, it creates a proportional local electromagnetic field.
Instead of measuring power consumption of the target circuit through a shunt resistor, we will measure local electromagnetic variations using an EM probe. Professional probes exist on the market, they were at first intended to help Electromagnetic Compatibility (EMC) by measuring local magnetic field emission. Those professional EMC probes are a perfect match for recording power traces, but anyone can build homemade probes by using a little coil, a filter and an amplifier. In Figure 1.11 one can see a very small coil made of insulated copper wire, that we rolled around a toothpick. The results obtained with this DIY setup were pretty decent.

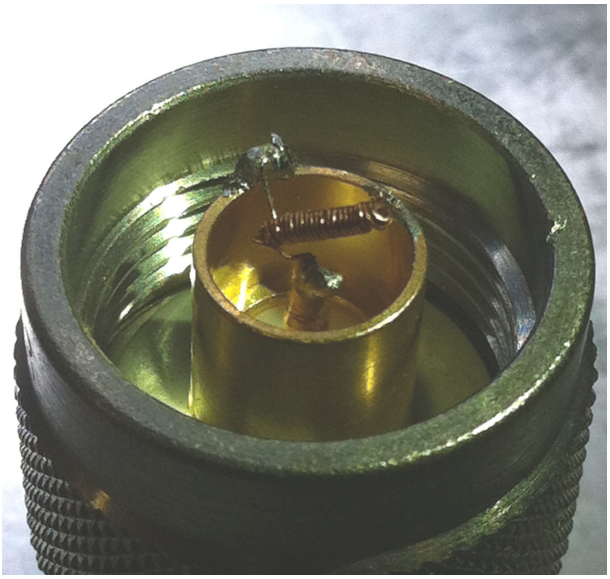


FIGURE 1.11 – 0.5cm wide coil that serves as an local EM probe

1.2 Fault Attacks

The first observation of induced faults in an electronic system was accidental. It was noticed that radioactive particles produced by elements, which were naturally present in chips' packaging [67], induced errors in their functioning. More precisely, the residuals of Uranium-235, Uranium 238 and of Thorium-230 which are present in packaging, decay in Plumb-206 by radiating alpha particles. It was observed that these particles create

charges that cause bits flipping in the memory. Even if the particles are present at low rates of two or three parts by a million, these tiny quantities are sufficient to affect chip's behavior. It happened that the research and the simulation of the effect of cosmic rays on the semi-conductors started at the same time [106]. Cosmic rays are very weak at ground level because of the protective effect of the terrestrial atmosphere but their effect increases in the upper atmosphere and in space. This is worsened by the fact that the error probability in a system highly increases with the number of RAM cells. This stimulated research in this field by entities as NASA or Boeing. The origin of this research on fault tolerance was related to counter electronic circuits' weaknesses in face of charged particles.

Significant efforts in engineering have been made in order to harden electronic devices supposed to work in hostile environments. They have been done mainly by using simulations allowing to design circuits while studying the effects of random faults. Several methods for injecting faults in hardware were discovered and experimented. All these faults cause similar behavior for the affected chips. An example that we will detail later on is the use of lasers in order to imitate the effect of charged particles on a chip and flip bits in a much localized manner [44].

Fault injection is an active attack, it modifies the behavior of a chip, either temporarily or permanently. If an attacker is able to modify the contents of a memory space or inject a fault during a cryptographic algorithm execution, some errors in the computation will almost certainly occur. If a final erroneous result, depending on the value of the secret, is furnished to the attacker, he might obtain some information from this result by comparing it with a correct result for the same operation.

This class of attacks represents a real threat for the embedded devices and often commercial products suffered frauds subsequent to fault attacks.

There are many ways to produce faults on an electrical circuit [8]. The malicious exploitation of such faults injection is also various : exploitation going from a simple alteration of the number of rounds in a symmetric cryptographic algorithm execution [29], to differential fault attacks (DFA) which are mathematical in nature. We will now detail very briefly some of the most used attack methods against secure devices which require protections.

In a malicious context, faults can be caused by a large range of intrusive and non-intrusive methods such as lasers, electromagnetic perturbations, voltage variations, clock glitches, or temperature modification. In this section, we will describe the most common fault injection techniques.

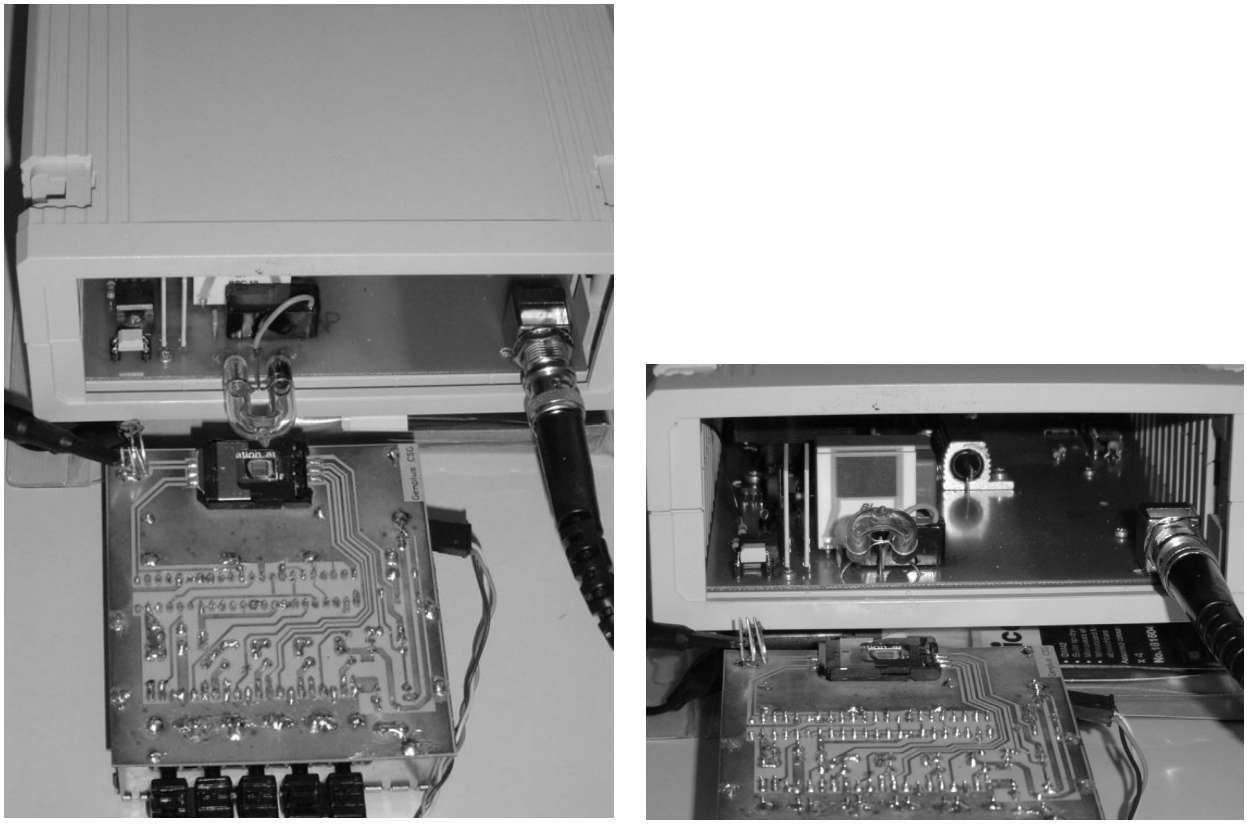


FIGURE 1.12 – Attack Bench of a white light attack. Notice the presence of a simple flash of an commercial camera. (photo courtesy of Gemplus)

1.2.1 Simple luminous radiations

In 2002, Skorobogatov [90] proposed to use a concentrated light ray for injecting faults. A simple camera flash was used as a white light source. Attackers employed afterwards a microscope and aluminum foil for concentrating light rays. The attack allows modifying chosen bits in SRAM memory contents. The chip has to be decapsulated in order to let the light rays reach the targeted zone. Nevertheless, this attack is very powerful since the attacker has control over targeted bits in the memory. All the electronic circuits are sensitive to photoelectric effects. The current induced by photons can provoke faults if the circuit is exposed to an intense light during a short period of time. This is a low-cost way for injecting faults [91].

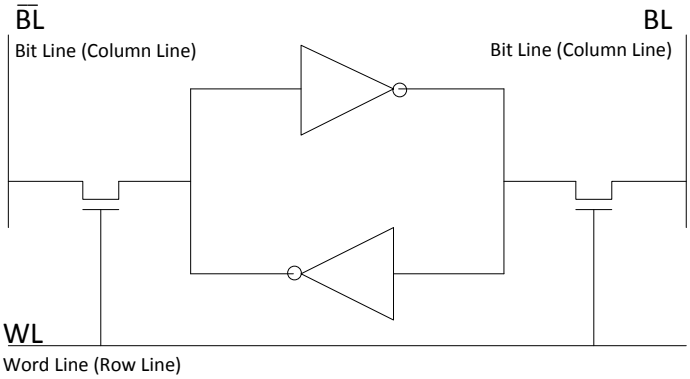


FIGURE 1.13 – Typical architecture of a SRAM cell

1.2.2 Laser (Light Amplification by Stimulated Emission of Radiation)

1.2.2.1 Generalities

A laser is an electromagnetic radiation in the visible or invisible domain. The laser light is monochromatic, unidirectional, coherent and artificial. A laser beam can be very narrow (only some micrometers wide). The beam can pass through various obstacles before impacting a target for a very short period of time.

It is well-known that lasers alter circuits' functioning. The nowadays foundry technology node is in the nanometer range. This very narrow beam added to the temporal precision of lasers, makes them a particularly strong attack vector.

The photoelectric effect of laser on the silicon can be resumed as follows : the exposure of SRAM (Static Random Access Memory) to a laser causes bits to flip [91, 34, 8, 26], phenomenon known under the name of Single Event Upset (SEU). By adjusting the beam energy below the destruction threshold of the circuit, the target can perturb the chip's functioning without inducing permanent faults.

A conventional SRAM bit cell (see Figure 1.13) is composed by two inverters. Each cell has two additional transistors which control the access to the content of the cell during the reading and writing operations. Each inverter is composed of two transistors, so a basic SRAM cell is composed of six MOS.

Every state of the four transistors corresponds to a register value. By construction, the cell admits only two stable states, 1 or 0. In each stable state two transistors are ON whereas the two others are OFF.

If a laser beam reaches the drain junction of the PN junction of the blocked transistor 1.14, the charge induced by the beam in the junction creates a temporary current that

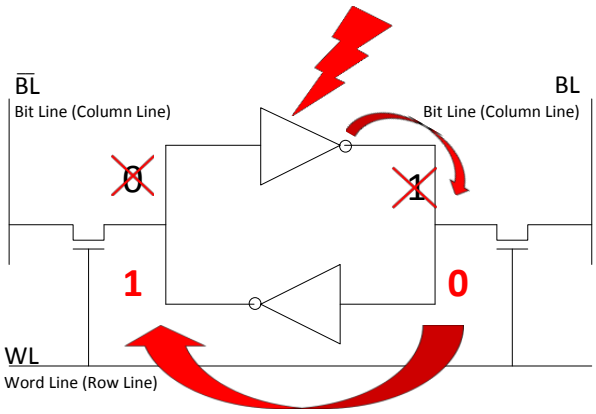


FIGURE 1.14 – SRAM cell state shifting due to a laser hit

makes the transistor flowing and inverts the logical output value of the inverter [34, 8].

From an adversary point of view, another advantage of laser fault injection is the reproducibility. Identical faults can be reproduced by automating the laser trigger and parameters and having control over the target.

In a laser attack, the attacker usually controls the diameter of the beam, its wavelength, the quantity of the emitted energy, the impact coordinates (i.e., area of the chip where we want to induce faults) as well as the exposure duration. Sometimes, the attacker can also control the timing of the attack (laser trigger synchronization with a precise clock cycle), the target clock frequency, Vcc and the target temperature.

Finally, the laser can be targeted from the top of the die but also from the backside, where some more localized faults can be induced in the substrate. It is important to notice that the two faces of the chip have very different behaviors once exposed to a laser beam,

- Front side attacks are particularly appropriate for green lasers ($\simeq$532 nm wavelength). The optical visibility of a target's logical block makes front side attacks much easier than the backside. Light reflection and diffraction of top-metal lines make it hard to aim a chip with a good precision. Moreover IC contains multiple metal layers depending on the technology and the transistor size keeps on shrinking. As a consequence, it becomes harder and harder to shoot precisely at a targeted area of an IC.
- Since a laser has to cross a thick silicon substrate from the back of a chip, backside attacks are very effective using infra-red lasers ($\simeq$1064 nm wavelength). The lack of visibility renders the positioning harder, but on the other hand backside attacks allow getting rid of the reflection on top-layer metal lines.

To put it in a nutshell, lasers can induce a large variety of faults from transient to permanent modifications [40, 80]. Faults induced by lasers are similar to the ones produced by white light, but with the advantage of precision, a laser beam allowing to aim at much smaller areas of a target.

1.2.2.2 Mounting a Laser Fault injection bench

Our goal here was to mount a LASER fault injection bench using an equipment we scavenged from the failure analysis laboratory. Table 1.15 provides the list of the equipment we used to mount our fault attack bench.

Instrument	Model	Manufacturer
LASER Nd :YAG	L-211-G1	HOYA Candeo Optronics Corporation
LASER pulse controller	L-211-P1	HOYA Candeo Optronics Corporation
XYZ Table	CPC-3DN	CHUO Corporation Precision Industrial
CCD Camera	DXC-101P	Sony Corporation
Microscope	BH2-UMA	Olympus Corporation

FIGURE 1.15 – List of equipment forming the fault attack bench

Once the all equipment has been assembled, we needed to create the proper software to control the XYZ table and trigger the LASER shooting. The XYZ Table has a GPIB interface (IEEE-488.2), that we connected to the computer by using NI GPIB-USB-HS (USB controller for GPIB interface). After installing the proper GPIB libraries on Linux and understanding the Japanese command manual of the CHUO XY table, the position of where the laser could shoot on the chip could be controlled by using a C++ SW.

The LASER we are using in the fault injection bench is a pulsed Neodymium-doped Yttrium Aluminium Garnet LASER. Pulse duration is 8ns and the LASER emits light with a wavelength of 1064nm in the infrared spectrum. A shutter mechanism allows to increase or reduce the area we are shooting at, we are always careful not to close the shutter completely to avoid diffraction effect. The LASER power supply allows to control the supply voltage in the range of 300V to 900V, enabling to control the pulse energy. The LASER shoot is manually controlled via a trigger that we needed to automate. Our first attacked device is a Texas Instrument MSP430g2231 μcontroller with a socket mounted development board [48] that allows to easily decapsulate and replace damaged chips, we decided to use this same μcontroller to trigger the LASER while it would launch cryptographic computations. We created a simple circuit using an electrical relay 1.16 that on one side, we soldered to a pin of the MSP430 development board and on the other side, we connected to the LASER power supply, L-211-P1, trigger in-

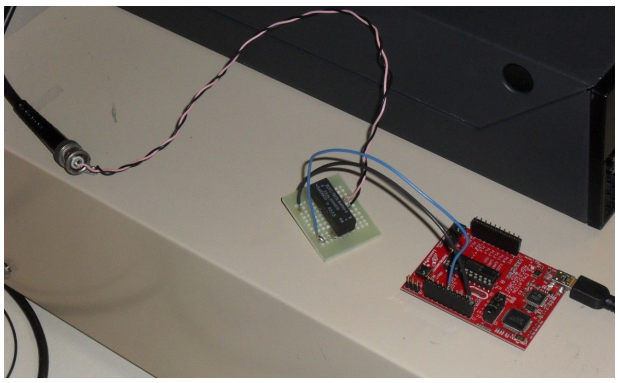


FIGURE 1.16 – Simple relay circuit automating the LASER shooting from the MSP430 development board

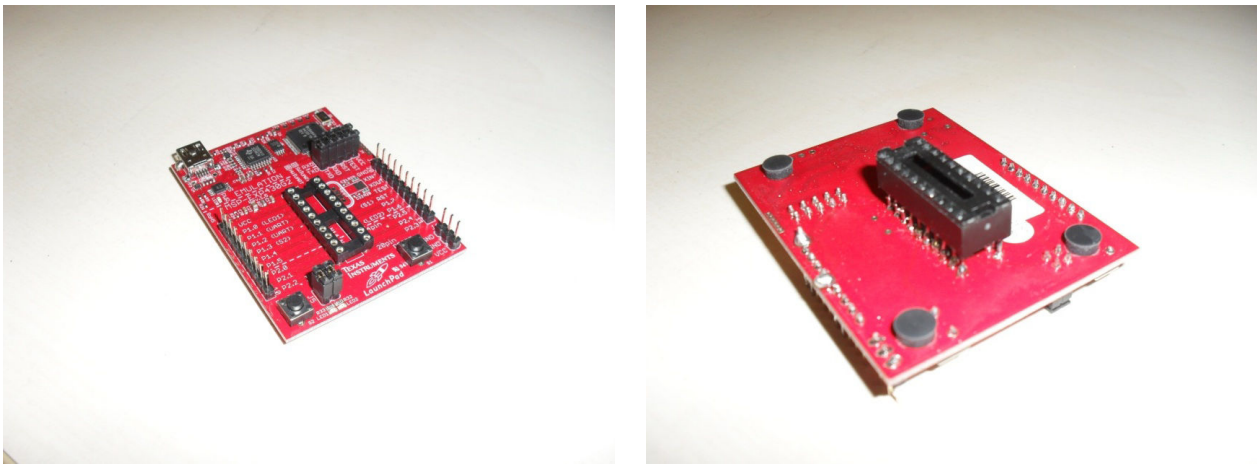


FIGURE 1.17 – We prepared two development board, one for frontside attack, one for backside attack.

put. The L-211-P1 is launching a LASER pulse when the voltage on the trigger input is raised to 3,5V. It seems the shoot is triggered on the falling edge.

At the beginning we were aiming using an optical sight but the precision was low, the area shot was not corresponding to the targeted area of the chip. The LASER offers the possibility of using a white light input for accurate targeting 1.18, so we employed a HOYA-SCHOTT HL100R for precisely targeting a die area without the use of an optical sight. The visualization of the die on camera can be done using a CRT monitor, but we got better results by using a digital video acquisition card. This capability has been very useful when targeting on chip-memory.

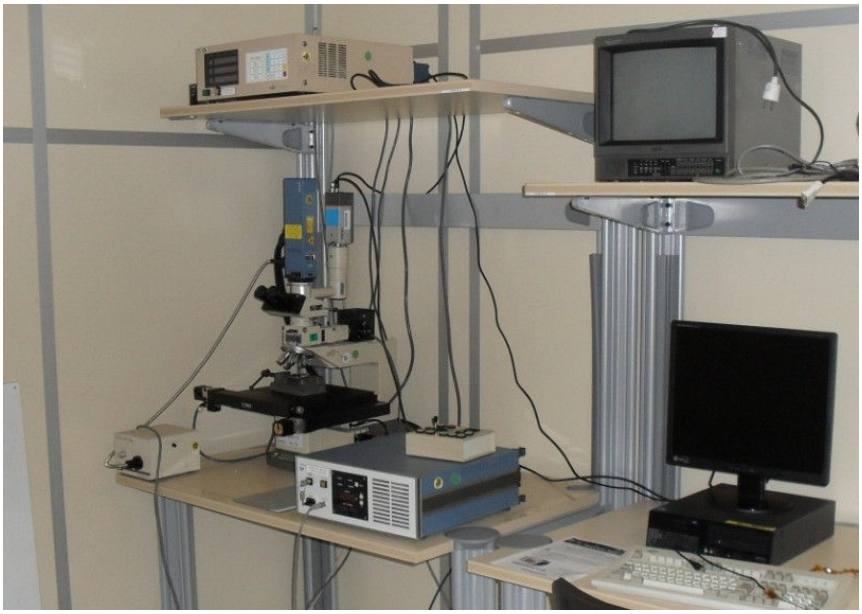


FIGURE 1.18 – Overall view of the complete fault injection bench

1.2.2.3 Implementation of an attack

We will now describe how we used for successfully injecting faults in the targeted μcontroller and the Bellcore attack we mounted. In order to successfully mount the LASER fault injection attack, we could not just shoot randomly in the chip. We needed to do some reverse engineering in order to decapsulate the chip and understand where the on-chip memory is located. The decapsulation process and the optical localization of the SRAM on the MSP μcontroller is described in section 1.3.1.

Depending on the energy carried by the LASER, we distinguish three different cases :

- The energy is too important and the memory cell is destroyed. The result is a permanent fault.
- The energy is high enough to induce gate transition but low enough not to destroy them. We achieved a transient fault.
- The energy is too low to flip a gate, no fault is created.

Once we located the memory on the μcontroller, we started by researching which voltage would give us a transient fault. To achieve that purpose, we wrote a simple SW that filled the memory with a integer table [i,j], where i is the index of the table and j is the value we write to it, and we checked after every LASER shoot if the memory content has been modified. As the voltage control is changed manually between 300 and 900 V, we blinked the green LED if no fault is detected and the red LED if a fault is detected. Using this method we were able to identify the threshold voltage for inducing transient

fault for front-side LASER shooting, which is 567V. It is interesting to notice that several shooting attempts at the same location in the SRAM with slightly different voltages will produce different faults. Table 1.19 gives results of three different attempts, the values $[\,i,\,j\,]$ in the table gives the index of the memory cell and the value it contains.

Once we could successfully inject a fault in the μcontroller's SRAM, we mounted a simple attack on a cryptographic algorithm, the Bellcore attack [22]. This attack is based on the RSA signature, attacking the part of the scheme by using the Chinese remainder theorem (CRT).

Let us recall the RSA textbook signature.

- Generation of keys
  - p and q two prime numbers sufficiently big,
  - let $n = p\,q\Big(\varphi(n) = (p-1)(q-1)\Big)$,
  - let e be a prime number with $\varphi(n)$ and d such that $ed = 1(\text{mod }\varphi(n))$,
  - private key= $(p,\,q,\,d)$ and public key =$(n,\,e)$,
- Signature
  - Compute $s \equiv m^d\,[n]$,
- Verification
  - recover $m \equiv s^e\,[n]$.

Let now recall the Chinese remainder theorem.

Theorem *Suppose $n_1, n_2, \ldots, n_k$ are positive integers that are pairwise coprime. Then one has the following isomorphism between a ring and the direct product :*

$$\frac{Z}{n_1 n_2 \ldots n_k Z} \simeq \frac{Z}{n_1 Z} \times \ldots \times \frac{Z}{n_k Z}.$$

The isomorphism is given by

$$a\,[n_1 n_2 \ldots n_k] \longmapsto \Big(a[n_1], \ldots, a[n_k]\Big).$$

The following steps are done in the **signature** process for the RSA-CRT :

- let $d_p \equiv d\,[\varphi(p)]$ and $d_q \equiv d\,[\varphi(q)]$,
- let $s_p \equiv m^{d_p}\,[p]$ and $s_q \equiv m^{d_q}\,[q]$,
- let p^{-1} mod q be the modular inverse of p modulo q,

SRAM Initialization	1st attempt (567V)	2nd attempt (567V)	3rd attempt (568V)
[0 :0]	[0 :6]	[0 : 6]	[0 : 6]
[1 :1]	[1 : 6]	[1 : 6]	[1 : 6]
[2 :2]	[2 : 6]	[2 : 6]	[2 : 6]
[3 :3]	[3 : 6]	[3 : 6]	[3 : 6]
[4 :4]	[4 : 6]	[4 : 6]	[4 : 6]
[5 :5]	[5 : 6]	[5 : 6]	[5 : 6]
[6 :6]	[6 : 6]	[6 : 6]	[6 : 6]
[7 :7]	[7 : 6]	[7 : 6]	[7 : 6]
[8 :8]	[8 : 14]	[8 : 14]	[8 : 12]
[9 :9]	[9 : 14]	[9 : 14]	[9 : 12]
[10 :10]	[10 : 14]	[10 : 14]	[10 : 12]
[11 :11]	[11 : 14]	[11 : 14]	[11 : 12]
[12 :12]	[12 : 14]	[12 : 14]	[12 : 12]
[13 :13]	[13 : 14]	[13 : 14]	[13 : 12]
[14 :14]	[14 : 14]	[14 : 14]	[14 : 12]
[15 :15]	[15 : 14]	[15 : 14]	[15 : 12]
[16 :16]	[16 : 22]	[16 : 22]	[16 : 18]
[17 :17]	[17 : 22]	[17 : 22]	[17 : 18]
[18 :18]	[18 : 22]	[18 : 22]	[18 : 18]
[19 :19]	[19 : 22]	[19 : 22]	[19 : 18]
[20 :20]	[20 : 22]	[20 : 22]	[20 : 18]
[21 :21]	[21 : 22]	[21 : 22]	[21 : 18]
[22 :22]	[22 : 22]	[22 : 22]	[22 : 18]
[23 :23]	[23 : 22]	[23 : 22]	[23 : 18]
[24 :24]	[24 : 30]	[24 : 30]	[24 : 26]
[25 :25]	[25 : 30]	[25 : 30]	[25 : 26]
[26 :26]	[26 : 30]	[26 : 30]	[26 : 26]
[27 :27]	[27 : 30]	[27 : 30]	[27 : 26]
[28 :28]	[28 : 30]	[28 : 30]	[28 : 26]
[29 :29]	[29 : 30]	[29 : 30]	[29 : 26]
[30 :30]	[30 : 30]	[30 : 30]	[30 : 26]
[31 :31]	[31 : 30]	[31 : 30]	[31 : 26]
[32 :32]	[32 : 32]	[32 : 32]	[32 : 32]
[33 :33]	[33 : 33]	[33 : 33]	[33 : 33]

FIGURE 1.19 – Results of the fault injection of the table stored in SRAM of the MSP μcontroller

- using the Chinese remainders theorem the signature s is computed as follows :

$$s = s_p + p\Big(s_q - s_p\Big)\Big(p^{-1} \bmod q\Big),$$

- for the proof, observe that making a modulo p at each side, one has
  - $s \bmod p = s_p \bmod p + p\Big(s_q - s_p\Big)\Big(p^{-1} \bmod q\Big) \bmod p,$
  - $s \bmod p = s_p \bmod p + p \star T \bmod p,$
  - $s \bmod p = s_p \bmod p,$
  - $s \bmod p = s_p,$
- and making a modulo q at each side, one obtains
  - $s \bmod q = s_p \bmod q + p\Big(s_q - s_p\Big)\Big(p^{-1} \bmod q\Big) \bmod q,$
  - $s \bmod q = s_p \bmod q + p\Big(s_q - s_p\Big)\Big(p^{-1} \bmod q\Big),$
  - $s \bmod q = s_p \bmod q + \Big(s_q - s_p\Big) \star 1 \bmod q,$
  - $s \bmod q = s_q \bmod q,$
  - $s \bmod q = s_q.$

The Bellcore attack works in the following way. One knows that the exact execution of the RSA-CRT algorithm gives the value

$$s = s_p + p\Big(s_q - s_p\Big)\Big(p^{-1} \bmod q\Big).$$

Principle. The Bellcore attack is based on the introduction of a fault in the computation of

$$s \equiv m^{d_q}[q].$$

Therefore

$$s' = s_p + p\Big(s'_q - s_p\Big)\Big(p^{-1} \bmod q\Big),$$

with

$$s' = m^{d_p}[p] \quad \text{and} \quad s' \neq m^{d_q}[q].$$

Finally, one can find with just one fault the prime number p,

$$s - s' \equiv 0\,[p] \quad \text{and} \quad s - s' \neq 0 \bmod q \Longrightarrow \gcd\,(s - s', pq) = p.$$

We ran RSA-CRT on the integer table stored in SRAM. Using the protocol we described, we started to inject a transient fault in the SRAM but we could not achieve the proper timing to get the fault in the cell just before it got used. Nevertheless, we could successfully inject a permanent fault in the signature, which lead us to recover the secret exponent p.
We stopped here the development of this bench, the continuation of this task would have been to modify the fault injection timing to increase its randomness and run the test during several hours to get a successful hit. We also could have continued researching on other attacks as the Piret-Quisquater on AES-128-ECB [79], and get an socket-mounted FPGA board to start testing the fault resistance of the designs we implemented.

1.2.3 Voltage Glitch

The power source of a smart card is externally driven and can be easily manipulated by an attacker ; he can control the external voltage source plugging in the voltage pin. Big current variations in brief intervals of time (impulse or glitch) can be sent to the card. These current spikes produce either memory errors, or changes in the code's execution, jumping from an instruction to another. This last property may allow modifications of a loop counter or a conditional jump in the code. There are many descriptions in the literature of fault attacks using of this technique, we refer the reader to [20, 16, 21, 7, 17, 105]. Voltage variations during code's execution can corrupt the data, corrupt the executed instruction or provoke a jump of some instructions.

1.2.4 Clock Glitch

Let us take the example of a smart card. As for the power source, smart cards receive their clock signals externally. The card reader provides the clock signal for the smart card, so that the attacker has the possibility to alter it. Smart cards often work at 3,57 MHz frequency. Rapid frequency variations can be sent to the chip in order to perturb its functioning. These perturbations are similar to those caused by voltage spikes. This method, like the preceding one, does not require advanced hardware to be implemented, and therefore is widely used in practice [6, 15, 78].

Clock glitches can

- Either create errors in memory read accesses. The circuit tries to read a value from the bus before the memory had time to stabilize RAM flip-flops output.
- Or can provoke an instruction jump causing the circuit to perform instruction n+1 before finishing instruction n.

1.2.5 Clock Errors

In a clock fault injection attack, an adversary can have access to the clock of the target. The clock faults violate the circuit's synchronism, a basic hypothesis underlying the functioning of traditional IC. Most ICs perform computations by treating data outputted by the combinatorial logic blocks separated by D-flip-flops register barriers which share the same clock (see Figure 1.20).

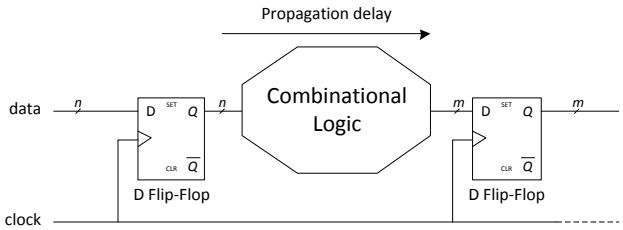


FIGURE 1.20 – Synchronous representation of digital circuits

Usually data are stored by clocked registers. Computed data travel among registers and undergo modifications by intermediate combinatorial blocks. The time required in order to propagate data through combinatorial logic is called « propagation delay ».

The propagation delay, as well a second delay, inherent to the use of D-flip-flops, called« set-up time », defines the circuit's maximal operational frequency (or the nominal circuit period). Indeed, in order to guarantee a normal functioning of the circuit, the clock period has to be strictly greater than the maximal propagation delay of the circuit (this maximal propagation delay is called « critical path ») plus registers set-up time, namely

$$t_{clock} > t_{critical} + t_{set-up}$$

Actually, each bit of data stored in a register is the result of computation depending on the combinatorial computation function of several input bits of data. The time between the output of the previous registers and the input of next registers determines the delay. This delay will depend on performed logical operations, as well as the propagation time which varies with temperature and voltage source.

Overclocking consists in reducing the clock period (or in other words increasing the clock frequency). If set-up delays time is not respected, D flip-flop's input signal might not have enough time to reach the flip-flop. This leads to a potential set-up of false data at the entry of the flip-flop. Several authors used overclocking as a method for fault injection [43, 88].

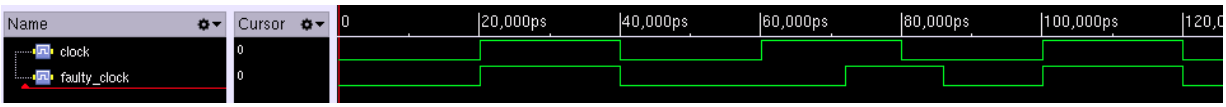


FIGURE 1.21 – Normal clock signal ($clock$) and a perturbed clock signal ($faulty\_clock$)

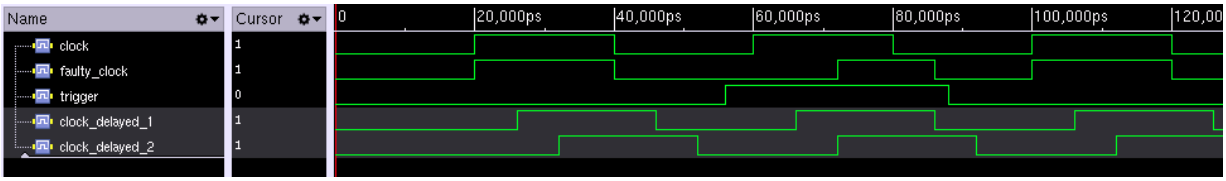


FIGURE 1.22 – Generation of a $faulty\_clock$

A reduced clock period can also potentially affect logical paths whose propagation time is greater than the clock period minus the set-up time. From the attacker's point of view, the possibility of controlling in a precise way the clock period is crucial in order to introduce controlled faults. It is worth mentioning that changes in temperature and voltage variations can also be used for such fine-tuned controls.

Faults attacks consist in injecting faults at very precise moments. In order to avoid injecting faults continuously, glitch timing has to be brief and discreet. Fault injections examples that we will describe later on carry these two properties.

To inject controlled fault using clock variations, the attacker does control two parameters : the precise moment when a fault occurs and the clock glitch duration. This time period has to be controlled with high resolution, typically several dozens of picoseconds.

Figure 1.21 shows a normal clock signal ($clock$) and a perturbed clock signal whose aim is to provoke faults $faulty\_clock$.

The two curves differ only within 20ns and 30ns. During this interval of time, the $faulty\_clock$ period is reduced by Δns. This time reduction of Δ causes a fault due to violation of flip-flop set-up time. Let mention that reducing clock cycle before the low state has no harmful effects on the computations carried down. This shows that the perturbing clock at very precise and narrow period of time is a key to a successful fault injection.

Generating a $faulty\_clock$ with enough precision is a hard task. To do it, generally it is recommended to use a Delay Locked Loop (DLL) from recent FPGA families. Two delayed clocks ($clock\_delayed\_i$) can be generated from $clock$. The delayed signal $clock\_delayed\_i$, delayed by Δi from $clock$, are programmable. Then $faulty\_clock$ is obtained by switching between the two $clock\_delayed\_i$ signal using a trigger. Figure 1.22 describes this procedure.

If $clock\_delayed\_2$ is delayed with $\Delta 2$ units of time, then $clock\_delayed\_1$ has to be delayed with $\Delta 1 = \Delta 2/2$ in order to preserve a cyclical rate of 50% during the narrow

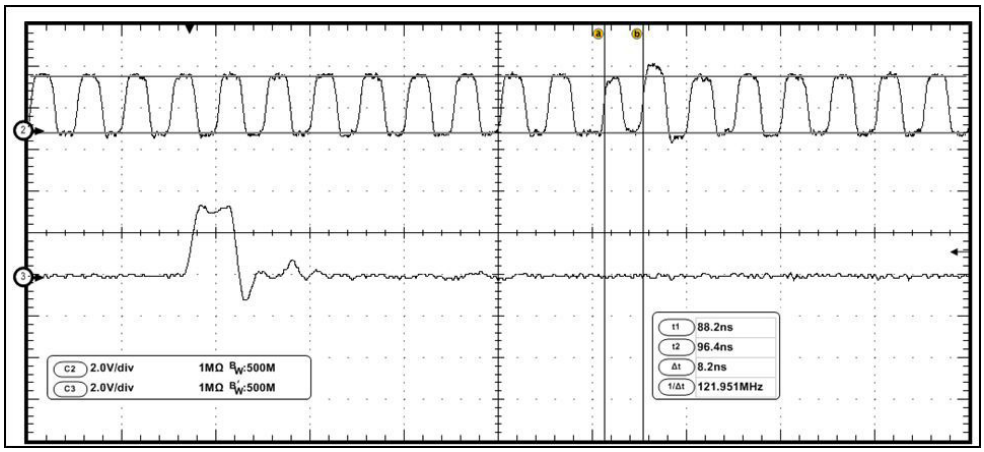


FIGURE 1.23 – $faultyclock$ (high signal) and AES_start (low signal) (photo courtesy [2])

fault interval of $faulty\_clock$.

$faultyclock$ is made by combining the rising edge of $clock\_delayed\_2$ with the falling edge of $clock\_delayed\_1$. The switch timing between the two signals is controlled using a trigger signal which positions the clock perturbation in time. $faulty\_clock$ resolution can be controlled (35ps in the example given here) depending on Δt, which the minimal elementary delay that a DLL is able to perform.

Here the period of $clock$ (10ns) was reduced at $[t_1, t_2] = 49\Delta t \approx 8.2ns$ in $faultyclock$. The high level of the low signal indicates the beginning of the attacked operation (an AES here).

Today, any commercial FPGA development platform is fairly inexpensive and range this attack in the « DIY in your garage » type of attacks as DPA.

The attack platform (Figure 1.24) is composed of a target circuit (TC), a clock faults generator (CFG) and a PC.

Algorithm running on the TC is targeted in the attack (our example present an AES executed on FPGA). The TC's clock is supplied by CFG. The secret key, the plaintext and the starting signal of the encryption (TRIGGER) arrives to TC from a PC via a serial port (RS232). After the encryption is completed, the result, called ciphertext, is read via the serial port.

The TC furnishes a TRIGGER signal to the CFG in order to indicate the precise beginning of encryption. This information, absent in reality, is added here facilitate the positioning in time the faulty period. If the chip is badly protected against side-channel attacks, a power analysis can help identifying the beginning of encryption.

The CFG generates a continuous clock of 100 MHz which synchronize the TC. When TRIGGER is high, it indicates that encryption starts and a decrementing counter is turned on. When this counter reaches zero, a faulty period is produced. The serial port

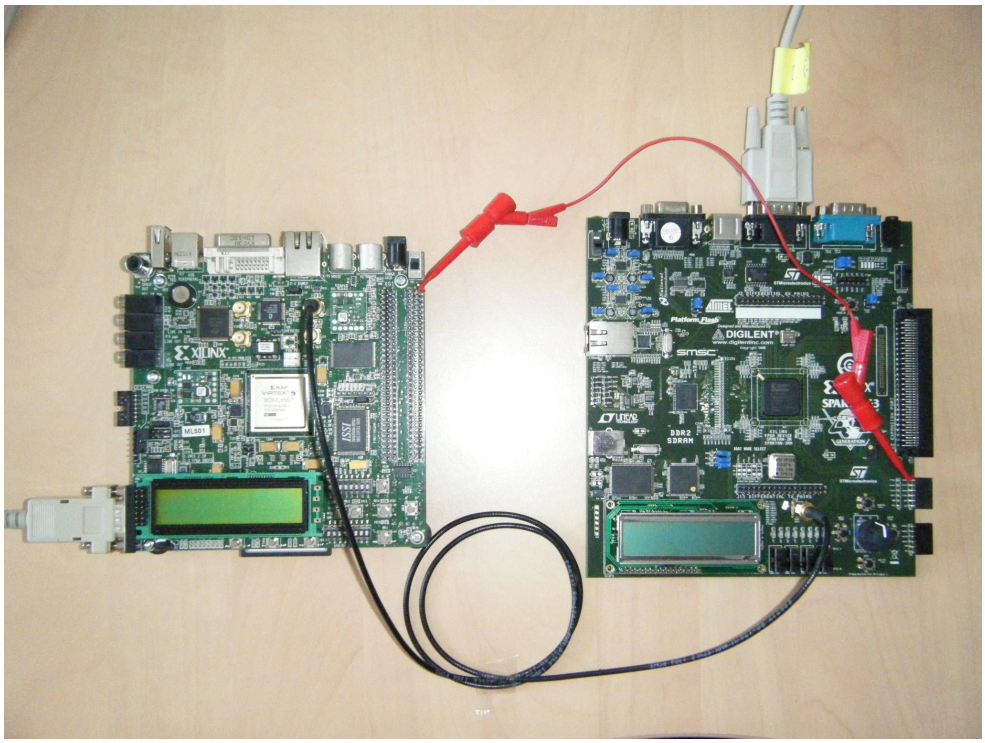


FIGURE 1.24 – External FPGA board feeding the faulty clock to the main FPGA board running cryptographic algorithms (photo courtesy [2])

between CFG and PC is used to control the exact Δ value for which the faulty period will be introduced.

A way to overcome this kind of attack in a secure element is to implement an internal clock in the SoC. Thus it is impossible for an attacker to hijack the clock at a PCB level. To modify the clock an attacker needs to perform an invasive attack in the SoC.

1.2.6 Temperature

In their component specifications, electronic circuits' manufacturers define an operating temperatures range in which their circuit works correctly. An attacker might want to go out from the normal temperatures range to see if the circuit will behave abnormally. This might be done using an alcohol cooler.

The aim of this attack is double : provoke a discharge of RAM cells whose sensitivity to heat is known, and exploit the fact that in the majority of non-volatile memories(NVM) the operating range differs according to which operation is carried out. For instance, on a same chip, one can have a write operation in a range P1, and an erase operation in a range P2, which doesn't overlap P1. By controlling temperature, it is sometimes

possible to inhibit one operation without inhibiting the other one, and doing so to put the chip in a state that it was not designed for.

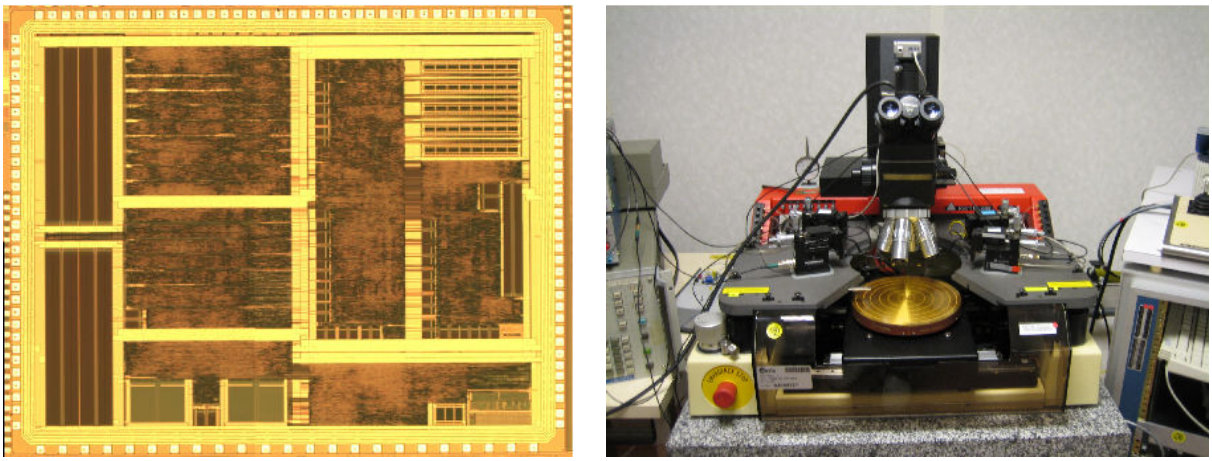
1.2.7 Probing Fault Attacks

A probe station is a laboratory device that allows acquiring hardware signals from internal nodes of a chip. They originated from the test department of production in a foundry where chips are tested on wafers. Some needles are used to retrieve signals and other are used to power-up the chip, thus a station allows to inject perturbed signals in a target circuit.

These operations are done by positioning thin needles (or probes) on the surface of the chip, either automatically, or with the help of automated manipulating tools activated by an operator. If the probed surface of the chip is stimulated electrically, the signal carried by the needle is amplified and acquired. The device shown in Figure 1.25 can be programmed to explore automatically a surface XY (surface of $200 \times 200mm$) with a resolution of $0.5\mu m$ and a precision of $\pm 1.5\mu m$. Most of probe stations are able to perform several simultaneous probing by using multi-probe cards.

Simultaneously, the device can be used to modify signals inside the target circuit. This is done by coupling the station to a FPGA (e.g. Xilinx 550) programmed for injecting pulses with different lengths of time at particular moments via the probe needles.

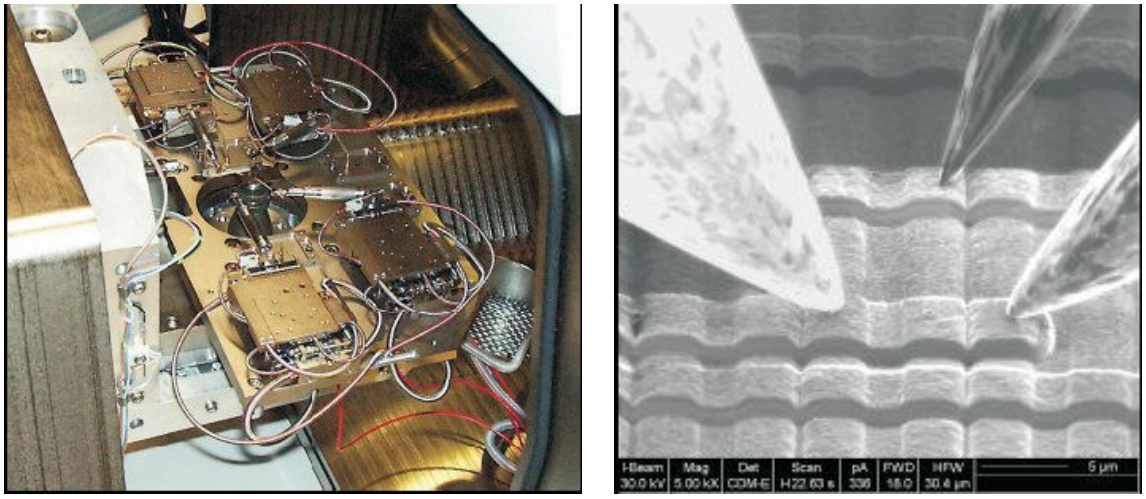
An optical probing station (Figure 1.25) allow an attacker to feed and collect signals from a target chip through its input and output pins. It enables an attacker to implement the fault injections we described in section 1.2.



(a) (b)

FIGURE 1.25 – A decapsulated chip with an exposed pad ring (a) and an optical probing station (b).

To probe directly metal lines (Figure 1.26b), an attacker must use more sophisticated tools. Using a probing station installed in a SEM (Figure 1.26a), it is possible to probe apparent metal lines. To probe hidden metal lines, a FIB must be used to remove layers and create new connections.



(a) (b)

FIGURE 1.26 – A probing station mounted in a SEM and a tilted SEM view of circuit lines being probed.

1.2.8 Electromagnetic Perturbations

Electromagnetic perturbations are experimented today in some laboratories. Given the relatively important dispersion of electromagnetic radiations, this technique is rarely used or sometimes used in an ad-hoc manner. For instance, by bringing closer an oscillating circuit to a random number generator based on an oscillating loop, one can impose an oscillating frequency to the generator by a simple electromagnetic coupling effect.

Nonetheless recent significant advances in EM probe design allow injecting faults in a very local manner.

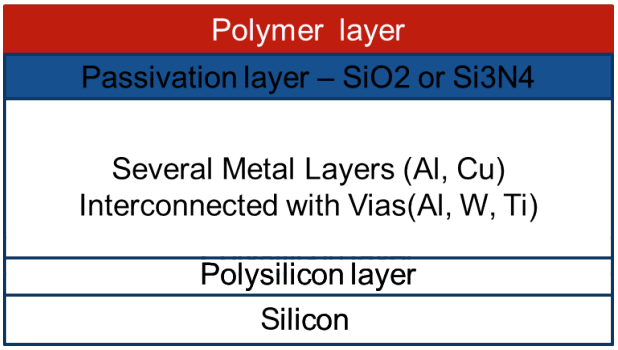


FIGURE 1.27 – Different Layers of a Silicon Chip

1.3 Invasive Attacks

1.3.1 Reverse Engineering

A chip is composed of several layers as described on Figure /reflayers (from the topmost to the lowest) :

- a polymer layer that is protecting the passivation layer from packaging component or glue
- the passivation layer usually made of silicon oxide (SiO_2), or eventually silicon nitride Si_3N_4. This layer protects the circuit from impurities after manufacturing in the clean room.
- N metal layers with interconnections (vias) between layers. We refer the topmost metal layer as "top-layer", he is usually carrying signals to the pad-ring but depending on the technology and the application, the top-layer can have different use. In a security chip, the top-layers will be used to create an active shield, usually in copper since it allows a smaller etching. In an RF chip the top-layer will usually be aluminum, with a much thicker etch to create antennas and other MEMS device.
- The polysilicon layer, which forms transistor's gates.
- And finally the silicon substrate with some diffusion layers.

The die is encapsulated in a plastic packaging, and the different in/out are wired using gold wire and recently copper wire in cheaper packaging to diminish the cost. We will point out that copper wired pad-rings are harder to decapsulate on from topside without altering any connection.

The decapsulation of a die can be achieved in several manners :

- Use of a miniature milling machine on the back-side
- Use of a computer-controlled micro-polishing machine on the back-side
- Use of chemicals on the front-side

FIGURE 1.28 – Fuming Nitric Acid (HNO_3) and Hydrofluoric Acid (HF)

- Use of an automated decapsulation machine

In order to reveal the die, the plastic covering it, has to be removed but without altering the chip bonding. We are using heated fuming nitric (Figure 1.28) acid at 60 degrees Celsius to attack the packaging. We rinse the chip every minute and iterate the operation until the die appears. We then clean the chip with acetone and leave the cheap in an ultrasonic bath of acetone that removed all plastic residues. We got very good results with this technique, but the bonding was very often damaged due to a long exposition to acid. One of the chips we decapsulated had a copper bonding, and made the operation very complicate since the bonding was getting damaged during decapsulation.

Fortunately, some machines exist on the market that allows to decapsulate precisely a chip depending on its packaging (Figure 1.29). These machines are also using fuming nitric acid, as manual decapsulation, but enable a very precise control over the time of exposure and the temperature of the acid. They decapsulate any kind of packaging, using different masks for the acid exposure, and the results are stunning. Figure 1.30 represents a Texas Instrument MSP4302231 microprocessor decapsulated with this machine to perform fault injection on embedded software.

After decapsulating the chip, the surface is rather dirty, with lots of plastic leftovers and other residues. An ultrasonic bath (Figure 1.31) in isopropyl helps removing all imperfections, and offers a clean surface for microscope optical observation.

FIGURE 1.29 – Hamamatsu PA103 Automated Decapsulation Machine

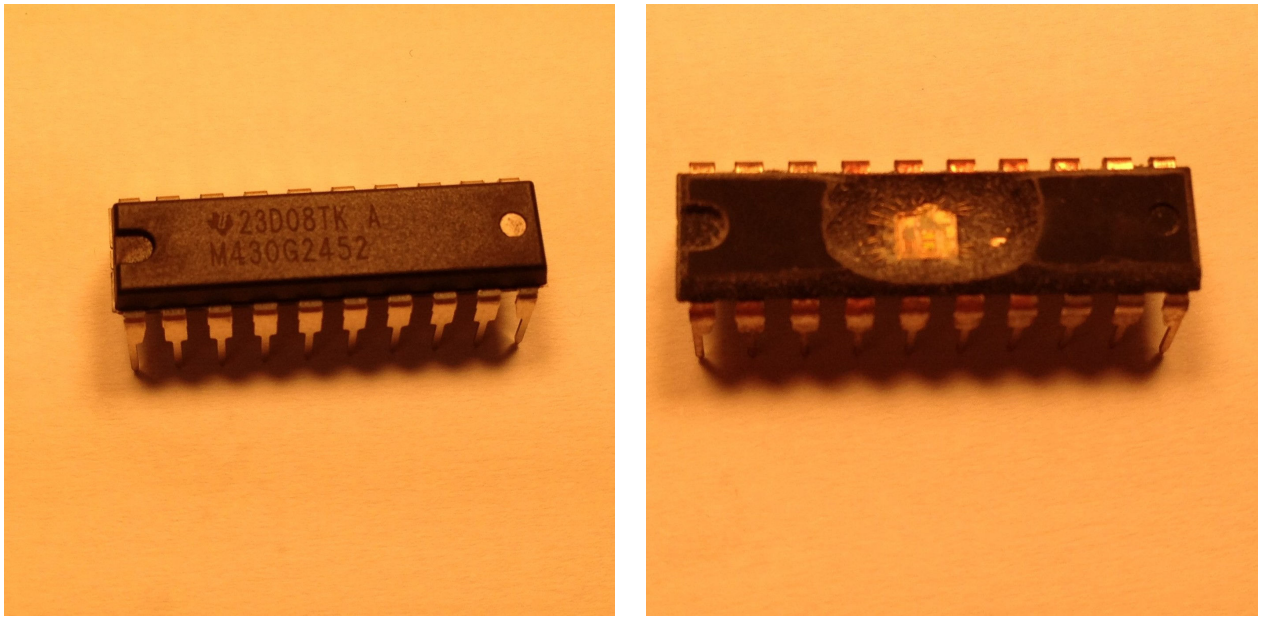
(a)

(b)

FIGURE 1.31 – TI MSP4302231 in an ultrasonic bath after depacking (a) and close-up (b)

At this point somebody analyzing a circuit will be willing to capture sections of the die layers after layers. The top-layer is usually helpful to understand the overall layout of the chip. It allows to distinguish memories, logic blocks and analog blocks. To understand the architecture at a lower level of detail, we need to observe the die layer after layers.

This process can be done mechanically or chemically. We can use a polisher (Figure 1.32) to remove the layers little by little but the result is usually not good as it is very hard to maintain the sample perfectly horizontal.



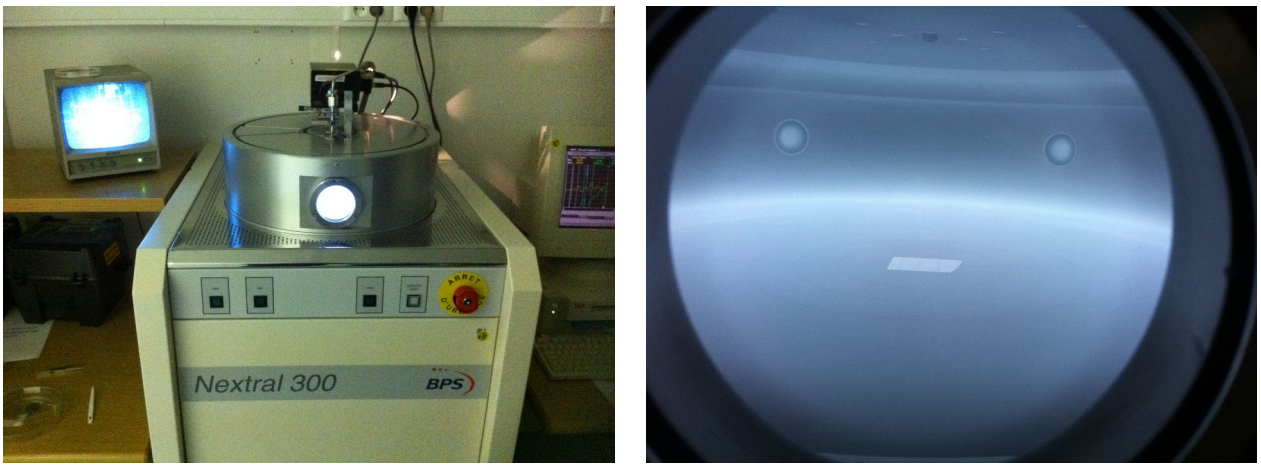
(a) (b)

FIGURE 1.30 – TI MSP4302231 microprocessor (a) and the same chip decapsulated ready for fault injection(b)

(a) (b)

FIGURE 1.32 – A Mecapol polishing machine (a) and an attempt of mechanically remove the active shield of a SMART CARD die(b)

Using a plasma etching machine (Figure 1.33), we can remove layers in a much more controlled way.



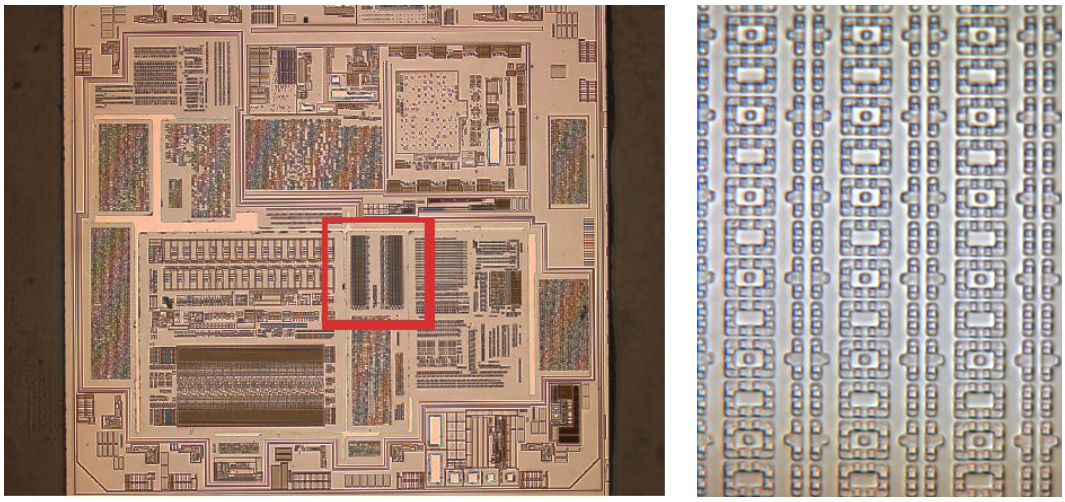
(a) (b)

FIGURE 1.33 – A plasma etching machine (a) and a sample being processed(b)

Finally, a bath of hydrofluoric acid will remove all layers, leaving the silicon substrate with the different implants visible.

1.3.2 Optical Analysis

After decapsulation, the die is exposed and can be observed. In order to successfully inject faults in a circuit, one can scan a die surface using a XY table, searching for an injection point. A more efficient way of successfully injecting faults, is to spatially resolve the fault location so that only the right timing is to be determined. Figure 1.34 is the polysilicon layer of a Texas Instrument MSP4302231 microprocessor. This optical analysis provides us a better understanding of the transistor configuration, and we can easily infer where the SRAM is located on the die. Some companies based their business on optically reverse-engineer chips, offering their customers to recreate the netlist of a given product. This activity that help counterfeiting is not illegal, and automated tools are identifying logic cells to recreate netlist for their user. These tools are very helpful when the security is based on some secret logic, and they are somehow the motivation for the rise of physically unclonable functions (PUF). We also refer the reader to research on photon optical emissions, which is another failure analysis technique which has been tuned to be used for attacks, where we can literally observe bits in a circuit.



(a) (b)

FIGURE 1.34 – TI MSP4302231 polysilicon layer (a) and a zoom on SRAM cells(b)

1.3.3 FIB Circuit Editing

Focused Ion Beam (FIB) is a machine used in failure analysis to analyze and modify the faulty circuit. FIB enables to dig trench in a device to observe the connections between layers at a nanometer scale (Figure 1.35), but also to deposit metal layer to create new connections. On the contrary of SEM, FIB is destructive and can damage the circuit.

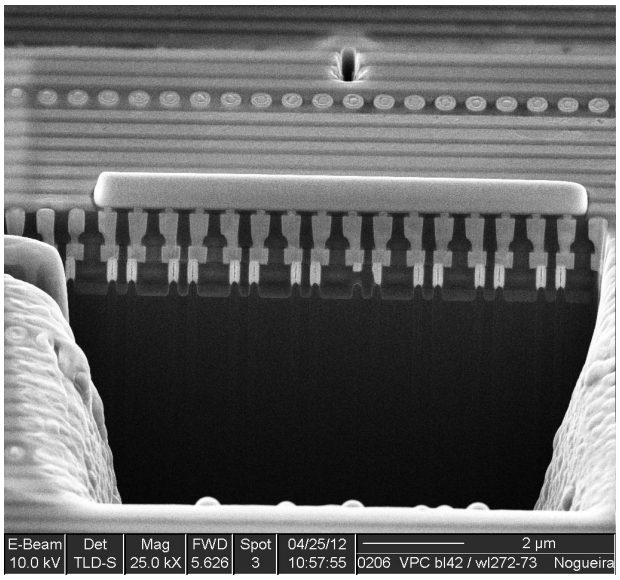


FIGURE 1.35 – Faulty device under FIB analysis where a connection problem is identified

FIB became of great interest regarding attacks as they enable to modify circuitry. A very well known use of FIB regarding invasive attacks relates to fuses. In security chips, fuses are used to program keys in the fab or to deactivate scan chains. They are very costly in terms of area and require the use of a fuse controller block, to prevent the fuses from being over burnt. Some attacks based on re-burning the same fuse multiple times successfully re-open fuse path. FIB allows an attacker to bypass the fuse controller protection, and re-open fuses. This is especially achievable since fuses are based on the large technology node, usually 130nm and higher, and are very large instance easily located on a die. If such a fuse can be reconnected (Figure 1.36), it would reopen the scan chain access that might let the user read or program keys and other secrets.

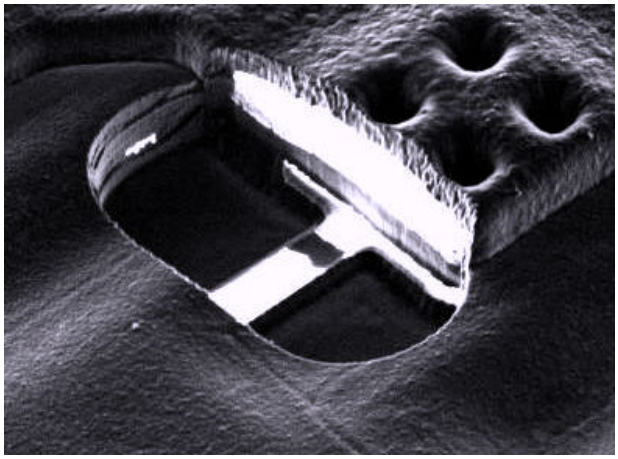


FIGURE 1.36 – Blown polysilicon fuse reconnected using FIB (Photoś courtesy of O.Kömmerling)

2 Hardware Security Sensors and Countermeasures

In this Chapter, we present a full range of hardware security sensors. Some part of the sensor might be implemented in software but we will only consider sensors and countermeasures that rely on a hardware piece. To each sensor is associated a protective action taken whenever the sensor is triggered.

2.1 Security Error Management

The state of each sensor can be checked by software verifications (the operating system is monitoring each sensor state) or by a wired piece of logic (if the alarm triggers, the chip enters automatically in a defined secure state). In case of software verification, the state of hardware sensors should be checked by the operating system before any critical operation. In case of wired sensors monitoring, one should make sure that the operating system is programmed to react to every alarm coming from the security sensors.

2.1.1 Global Errors

An important part of the sensor hardware implementation relates to error monitoring ; it is common to have a counter at each sensor level and a global counter to monitor errors. In a secure system, every error coming from a security sensor will generate an interrupt, so in some case one might want to increment a counter to allow a certain amount of normal functioning errors and set a threshold to trigger a global error.

2.1.2 Integrity Errors

In a secure system memory is usually split into different domains which have different security levels regarding their integrity and protection requirements :

- non-protected memories containing data whose alteration while only affect non sensitive information. No specific protection is required for such use,
- sensitive memory containing sensitive user information that we do not want to be modified. A typical checksum is sufficient at this level of integrity requirement,
- critical memory contains critical information for the system. It can range from configuration information, secure boot code, cryptographic keys, credit-card number and such. This type of memory will be protected with cryptographic hashing or redundancy and continuous hardware verification.

2.1.3 Protective actions

Depending on the global error counter value and on the type of alarm that triggered errors, one or several protective actions can be taken from the following list :

- *Passage to an infinite loop.* The component will pass in an infinite loop until the next power-on.
- *Erasement of the RAM.* The chip will erase all the contents of the RAM by activating a hardware port provided to this end.
- *Temporizing.* The chip will write an error flag in Flash at T=1. Afterwards, it will start a counting of a number T of cycles. When the counter reaches T, the chip will write the flag in Flash at 0 and will go back to normal modes of operation.
- *Erasement of the non-volatile memory.* This action consists of deleting the whole of the contents of the Flash to protect secret content from being retrieved such as cryptographic keys.
- *Suicide.* This action consists of placing a “Suicide” bit in NVM and launching a RAM and Flash erasement.
- *Deactivating.* This operation consists of positioning a deactivation flag at 1, so no operation is anymore allowed.

2.2 Side-Channel Countermeasures Taxonomy

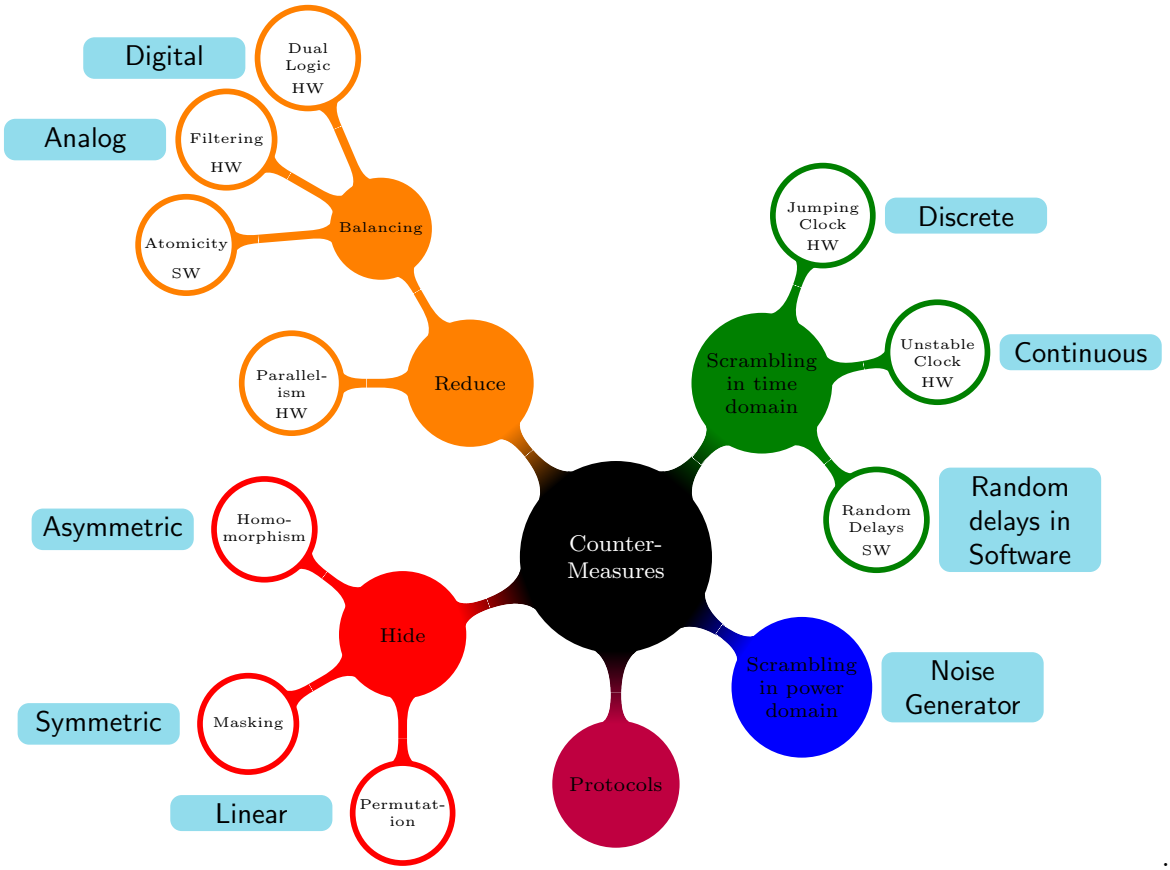


FIGURE 2.1 – Taxonomy of Side Channel Attacks Countermeasures

Since 1998, when the notion of attacking a crypto algorithm based on power analysis was introduced, chip manufacturers introduced several security mechanisms to prevent correlation of SMART CARDS operations and power consumption.

A first way to decorrelate these two variables that we will investigate later on, consists in introducing temporal delays in the code execution. So, replaying the same operation will not produce identical power consumption. The number of clock cycles taken to complete a computation will be different (alternatively, some implementations will change the clock frequency irregularly in order to create this temporal delay).

A second way of hardening a chip against attacks is to use power jammers which add random noise to the power consumption.

All these mechanisms will make the synchronization of a fault injection trigger very

hard. Obviously, the smaller is the number of cycles which have to be perturbed, the bigger will be the benefits of side-channel countermeasures to prevent fault injections.

Thus, activation of specific peripherals, such as cryptographic coprocessors or charge pumps is hard to dissimulate. Also, one will take care that such events could not serve as a reference point for triggering fault injections.

There are eleven main categories of countermeasures against side-channel attacks. Each countermeasure has different protection feature and impact the overall system performance differently (increasing area, power, throughput, frequency, a.s.o).

2.2.1 Power Scrambling

Power scrambling in time domain consists in interrupting and restarting the protected algorithm in order to spread the power leakage randomly in time. Thus when acquiring a batch of power traces, an attacker cannot correlate power traces as is. He needs first to pre-process traces to realign the different part of the attacked operation among the batch of traces. This countermeasure is usually implemented as follows :

- A dedicated hardware block is randomly sending pulses to the CPU in the case of a SW implantation or to the hardware crypto module.
- The internal clock is presenting instabilities in frequency or duty cycle.
- A dedicated process will control the computation output timing by inserting random delays.

2.2.2 Balancing

Balancing power consumption consists in physically duplicating signals or part of a circuit in order to equilibrate the overall circuit power consumption and ensure the same amount of registers are changing states for every operation. Balancing power consumption is typically implemented as follows :

- Physical signals are encoded to carry the hamming weight and balance register transition (01 represents a 0 and 10 represents a 1). Encoding signals in this manner also brings the benefit of being fault tolerant, if a signal takes the value 11 or 00, one can design the HW in order to propagate that signal and trigger an error signal.
- The HW core or SW is representing and handling some sensitive data in a redundant format.
- In HW part of the algorithm may be implemented twice in order to present the same input capacitance to the duplicated registers.

2.2.3 Dummy Cycles

Dummy cycles are a sophisticated power scrambling technique in time domain where operations or instructions are randomly executed instead of simply delaying them.

- The device is executing automatically fake operations or opcode during pauses.
- Dummy cycle is executed randomly in the code to confuse the attacker.

2.2.4 Parallelism

Parallelism consists in executing simultaneously several operations which are meaningful for the ongoing operation to make side channel analysis harder. For instance, if the crypto algorithm is using look-up tables (LUT), we will access different LUT simultaneously rather than sequentially (if the algorithm allows it). Parallelism is usually only implemented in hardware.

2.2.5 Filtering

Filtering consists in averaging or concentrate power consumption over a defined window of time in order to make instantaneous power consumption measurements harder. Filtering is usually achieved using embedded capacitor or filters which are blocking power consumption on high frequency (corresponding to the system frequency where gates are switching).

2.2.6 Power Jamming

Power jamming consists in adding random noise to the device power consumption. Power jamming is usually implemented in the following manner :

- Connect (in serial or parallel) to the circuit or the CPU a block consuming power randomly.
- Pre-charge randomly buses and lines before use.
- Randomly activate different part of the chip (co-processor, charge pump, external I/Os, etc).

2.2.7 Protocol

Protocol solutions can be implemented to limit power leakage or make it unusable. This class of countermeasures consists usually in :

- Limiting the number of encryption done with a given key. A counter or an external module can revoke a key and load a new one periodically.
- Include a random field in every cipher or signature.
- Update keys continuously.

2.2.8 Atomicity

Atomicity consist in avoiding conditional jumps in a code. Atomicity is usually only applicable to software. It consists in coding the algorithm so that the different controls and operations taken are independent from the data being processed. Most of symmetric ciphers and hashing algorithms are naturally respecting this principle, but he famous example of RSA modular exponentiation shows its importance.

2.2.9 Homomorphism

Homomorphism consists in using arithmetical properties of some public key cryptographic algorithm in order to compute their results in a manner chosen among several ways of computing the same output. For instance, the RSA algorithm consists in computing the quantity $c = m^d$. It is possible to generate a nonce r and compute$c = m^r m^{d-r}$ thus hiding the use of number d.

2.2.10 Permutation

Permutation consists in randomly rearrange different subprocesses which are not interdependent, so that different acquired power traces of the exact same computation will differ.

2.2.11 Masking

Masking is an algorithm level countermeasure. It consists in rewriting an algorithm, usually a LUT like an Sbox, in a random manner before running the code. Masking is heavily used to protect LUT based algorithm as DES or AES.

2.3 Protection against faults

2.3.1 Fault attacks taxonomy

Faults attacks can be subdivided into different classes according to several criteria. For each parameter, we will describe the attacker scenarios from the weaker to the most severe one.

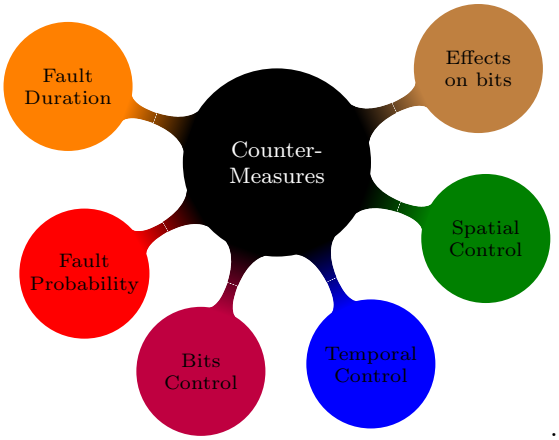


FIGURE 2.2 – Fault Attack Taxonomy

2.3.1.1 Spatial control

The attacker must have certain control on the position where faults occur :

- the attacker can modify only a given datum,
- the attacker can modify only precise bits of a variable,
- the attacker has no control over which variable will be modified.

2.3.1.2 Temporal control

This control of the fault injection is a crucial parameter, since

- either the attacker can choose the exact moment when a fault will be injected regarding the target circuit algorithm time-line, this fault can affect a block of different operations,
- or the attacker does not have any temporal control.

2.3.1.3 Number of bits

The number of bits an attacker can modify determines the fault injection efficiency, depending on the ability of the attacker to modify

- a single bit,
- a single byte,
- or an unknown number of bits in a variable.

2.3.1.4 Fault probability

Injecting a fault does not necessarily have an effect on the circuits behavior, or at least not the effect expected by the attacker. To assess the dangerousness of fault attack, one has to consider the success probability of a fault.

For instance, suppose that the attack requires fault injection in a precise portion of the memory but the attacker is only able to inject faults in random places. Obviously, this will affect the success probability of the attack.

2.3.1.5 Fault duration

A fault can affect a circuit for more or less time. One considers three types of faults according their duration : transient, permanent or destructive.

- A *transient fault* has a very short effect in time before that the circuit recovers its normal behavior. Generally, this interval of time is very short. Thus, if a transient fault is injected in a variable during computation, at the next operation time, the variable will recover its original value.

- A *permanent fault* affects a variable until the algorithm finishes, or it is explicitly crushed by another value being used.

- A *destructive fault* destroys definitively a part of the circuit by fixing some precise bits to a given value.

2.3.1.6 Effect on bits

A fault can affect a set of bits in the memory in various ways as follows :

- bits can be fixed at an arbitrary value thanks to a destructive or permanent fault

- bits can take an opposite value, i.e., if a bit has the value 1, then the fault will transform it into 0 (and conversely too)
- bits can take random values, then an attacker has no control on the variable value after the fault.

2.3.2 Protection against permanent fault

One calls « permanent faults », faults on an attacked chip which are irreversible. They can affect either memories (NVM or RAM cell blocked permanently at a constant) or logical functions. In both cases, countermeasures allow the detection of an alteration.

- UV detectors prevent exposing NVM (EEPROM or Flash) to an anomalous quantity of UV radiations and react as soon as an exposition threshold is reached. Let mention that the X-rays and ion-beams can serve as faults sources (however this is less common). This operating mode has the advantage of allowing injecting faults without depacking the chip. We refer the reader to [31] for more details.
- Error-correcting code (ECC) mechanisms and/or error-detecting codes (EDC or checksum) detect the moment when a value contained in the memory changes its initial (correct) state. According to the implementation and to the number of bits used for ECC or checksum, some faults could be detected or even corrected.

2.3.3 Protection against transient faults

Transient faults are induced only for a limited period of time in a given process. Since these faults can occur at any moment anywhere in the target circuit, it is rather hard for circuit designers to develop efficient universal countermeasures.

Therefore, instead of detecting faults themselves, designers develop methods enabling to detect effects of faults or extreme physical conditions liable to provoke these faults.

2.3.4 Protection against frequency or voltage manipulations

The cheapest way to provoke faults in a secure chip consists in manipulating input signals. Usually, input, output and reset to zero signals are designed in a robust way. Thus, attackers will most often try to inject perturbations using the chips power supply or via an external clock.

2.3.4.1 Protection against frequency manipulations

This threat has to be taken into account in any chip using an external clock, the best countermeasure being the use of internal clock.

In case of an external clock, high frequency faults detectors and low frequency ones (against step by step clock sequencing) will be implemented to detect clocks frequencies outside of the design range. A specific detector can be added for detecting anomalous pulses on the clock signal (clock glitches), as well as cyclical reports not in accordance with the normal conditions of functioning.

2.3.4.2 Protection against voltage manipulations

Manipulating the power supply is a very common way to perturb a chips functioning. To prevent faults induced via Vcc and Vss voltage modifications, a pulse detector can be added to the chip.

Moreover, if the chip is supplied by a voltage regulator (which is the case in all technologies under $0.35\mu m$) a detector for internal voltage manipulations (abnormally high, low or pulse) should be implemented.

2.3.4.3 Protections against light attacks

All ASICs are sensitive to light at different degrees. This is why an attacker can induce faults into a chip by injecting a sufficient quantity of energy using lasers or even a white light source (e.g. camera flashed triggered by an FPGA).

- *Metal plate shielding*

  A first protection against light exposure consists in covering sensitive parts of the circuit by a metallic shield.

  Note that some metal shields are not opaque to light. Therefore, experiments on the required metal thickness are needed to validate this countermeasure.

- *Individual light detectors*

  It is also possible to implement light detectors directly embedded in silicon by designing some transistors of the circuit in such a manner that light exposure causes them to cease functioning.

- *Distributed light detectors*

  In a distributed system, light detectors are spread in the circuit in order to avoid local light attacks (as lasers). Generally, if an attacker performs reverse-engineering

on the chip he will identify those detectors with the help of a simple optical microscope, unless a specific design method is used.

2.3.4.4 Generic faults countermeasures

- *Dual logic*

  As mentioned in section 5, a countermeasure method preventing at the same time the side-channels attacks and fault attacks consist in designing some parts of the circuit in dual logic. This logical coding mode obliges the attacker to manipulate two signals at once ; the back draw is that it is doubling the area and power of the circuit protected.

- *Redundant logic*

  Another method of protection against fault injections consists in designing the circuit in a redundant way. This technique is usually applied only to sensitive parts of a circuit since it implies increasing the chips area. You then have a voting system between the different redundant outputs, usually the best out of three. This method is commonly in critical systems as an aircraft.

  Redundancy is useful for sensitive logical functions as well as RAM cells or registers (as for example, the pointer stacks).

- *Memory protection*

  In order to protect the memory against faults, it is possible to perform systematic reads after any writes to verify that the operation took place. This can be done through both SW and HW.

  There are several options in order to protect read operations. A first option consists in adding redundancy to the operation as, for instance, doubling it. The result of the two reads is compared to be sure that there were no faults. In the case of RAM, two real reads will be distributed on four read operations ; two of them will address one decoy data. So the attacker will not have any certainty which one was the retrieved data, even if he succeeds to corrupt two reads over four.

  Another solution to protect a chip against faults is the use of canaries ; one could permanently read the canaries addresses in which test data have been placed. The corruption of a read test datum will serve as a fault detector.

- *Detection of illegal opcodes*

  Each CPU core is designed with an instruction set (opcodes). A CPU has the possibility to detect illegal opcodes. Thus a chip will provoke an error (augmentation of a GEC) if the execution of an illegal opcode is attempted.

2.4 Protection Against Invasive Attacks

2.4.1 Impacting spatial localization

Attacking a secure circuit by using a simple light beam requires very precise spatial localization of the targeted function or, at least, a control of the position of the beam with respect to the different blocks forming the chip.

There are different strategies of countermeasures against direct observation of the chip (using an optical microscope) in order to make difficult such a spatial localization. Let us mention in particular, the use of glue logic [1], or of shielding making an identification of specific zone more complex. Shields on top metal layers hide the buried layers and render the navigation on different layer depths very difficult (different layers can be observed optically by focusing a microscope to different depth). Such a shield also serves as a function of anti-probing device since it prevents an attacker to reach metal lines located underneath. Part of this thesis work will be focused on active shielding techniques on secure ICs.

In recent semi-conductors technologies, chemical etching or infrared observation of the backside of the chip is necessary to localize basics functional blocks of the chip.

2.4.2 Active shield mechanisms

Active shields are implemented in many security chips requiring temper resistance against invasive attacks such as probing. Most basic active shields are composed of a metal wire in a meander shape, covering the top surface of the chip, and a control circuit which permanently verifies that the wire is not modified or disconnected.

An active shield protects a chip against physical probing and the most of fault attacks. Indeed, fault injection attempts will cause perceptible modification of the electrical response of the wire and will be detected by the protected circuit. We refer the reader to section 6 and 7 to a detail analysis showing how active shield can be implemented, and on novel implementations of active shields using an cryptographic algorithm to detect modification.

2.4.3 Silicon sensitivity

Transistors sizes are in constant diminution, which entails the increase of circuits sensitivity with respect to the faults. Indeed the number of electrical charges representing a

1. Technique allowing to mix different logical functions in order to route them with standard cells.

bit is also decreasing. Consequently, a low level perturbation will be sufficient to flip a bit. This forces hardware designers to implement fault detections and correction algorithms on full scale.

However, if a technology is too sensitive, it becomes harder and harder to choose specific transistors or to find appropriate operating conditions to inject faults. This is property is interesting and deserves to be studied and optimized during circuit conceptions.

Part 2
DPA and EMA Countermeasures

3 Buying AES design resistance with speed and energy

3.1 Introduction

The Advanced Encryption Standard (AES) algorithm, also known as Rijndael, is a widely used block-cipher standardized by NIST in 2001 [73]. Compared with its predecessor DES [3], the AES features longer keys, larger plaintexts and more involved basic binary transformations [12].

Despite the fact that AES is mathematically safer than the DES, straightforward AES *implementations* are not necessarily secure and several authors [59, 58, 66] have exhibited ways of exploring information that leaks from AES implementations. Such leakage is typically power consumption, electromagnetic emanations or the time required to process data. Additional constraints such as fault-resistance, chip technology, performance, area, power consumption, and even patent compliance further complicate the design of real-life AES coprocessors.

This article addresses resistance against two physical threats : power and fault attacks. The proposed AES architecture leverages the algorithm's structure to create low-cost protections against these attacks. This allows very flexible runtime configurability without significantly affecting performance.

The remaining of the paper is organized as follows : Section 3.2 recalls the AES' main features and proposes an architecture for implementing it. Section 3.3 explains how to add power scrambling and fault detection to the proposed implementation. The result is a chip design allowing 29 different software-controlled runtime configurations. Section 3.4 introduces an idea of reducing the memory required to store *state keys* in the decryption mode. Section 3.5 compares simulation and synthesis results between an unprotected AES and our protected implementations. While Section 3.6 concludes this implementation study.

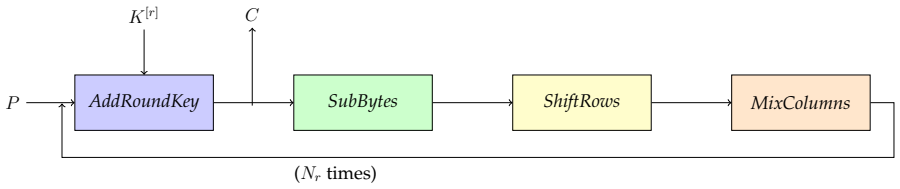


FIGURE 3.1 – AES Encryption Flowchart.

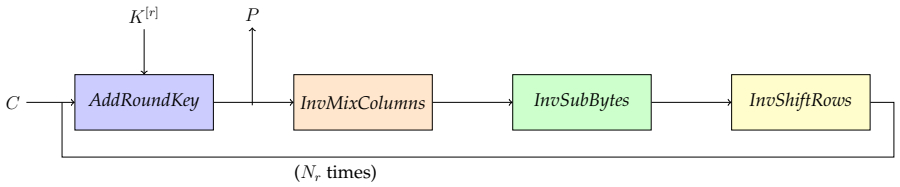


FIGURE 3.2 – AES Decryption Flowchart.

3.2 The Proposed AES Design

The AES is a symmetric iterative block-cipher that processes 128-bit blocks and supports keys of 128, 192 or 256 bits [73]. Key length is denoted by $N_k = 4, 6,$ or 8, and reflects the number of 32-bit words in the key. At start, the 128-bit plaintext P is split into a 4×4 matrix S of 16 bytes called *state*. The *state* goes through a number of rounds to become the ciphertext C.

The number of rounds N_r is a function of N_k. Possible $\{N_r, N_k\}$ combinations are $\{10, 4\}$, $\{12, 6\}$ and $\{14, 8\}$. A particular round $1 \leq r \leq N_r$ takes as input a 128-bit *state* $S^{[r]}$ and a 128-bit *round key* $K^{[r]}$ and outputs a 128-bit *state* $S^{[r+1]}$. This is done by successively applying four transformations called *SubBytes*, *ShiftRows*, *MixColumns* and *AddRoundKey*.

The AES encryption starts with an initial *AddRoundKey* transformation followed by N_r rounds consisting of four transformations, in the following order : *SubBytes*, *ShiftRows*, *MixColumns* and *AddRoundKey*. *MixColumns* is skipped in the final round ($r = N_r$). If during the last round *MixColumns* is by-passed, we can look upon the AES as the 4-block iterative structure shown in Figure 3.1.

Decryption has a similar structure in which the order of transforms is reversed (Figure 3.2) and where inverse transformations are used (Note that *AddRoundKey* is idempotent). In both designs, a register barrier at the end of each transformation block is used to save intermediate results. Therefore the intermediate information that eventually yields $S^{[r]}$ is saved four times during each AES round. It takes $4N_r + 1$ clock cycles to encrypt (or decrypt) a data block using this design.

Figure 3.1 and Figure 3.2 show that during each clock cycle, only one block of the chain actually computes the *state*, while the other three blocks are processing useless data.

This is potentially risky, as the three concerned blocks "chew" computationally useless data related to P (or C) and $K^{[r]}$ and thereby expose the design to unnecessary side-channel attacks. This computation is shown in Figure 3.3 where red arrows represent the path of usefully active combinatorial logic.

3.3 Energy and Security

3.3.1 Power Analysis

We assume that the reader is familiar with the power [66] and fault [54] attacks that we do not recall here.

To benchmark our design the AES was implemented on FPGA. Power was measured at 1GS/s sampling rate with 250MHz bandwidth using PicoScope 3407A oscilloscope. To guarantee the identical conditions every new plaintext was given to the FPGA at the same clock after the reset.

We performed a Correlation Power Attack (CPA) on the first AES Sbox output since Sbox operation is generally considered as the most power gluttonous. Our power model was based on the number of flipped register's bits in the Sbox module when the initial register's barrier R_0 is rewritten with the Sbox output as follows :

$$\text{HD}(Sbox[P \oplus K_0], R_0) = \text{HW}(Sbox[P \oplus K_0] \oplus R_0) \tag{3.1}$$

where R_0 is the previous register's state ; P is a given plaintext ; K_0 is the AES master key.

The value R_0 was assumed to be constant since all the encryptions were performed at the same clock after the reset. When R_0 could not be computed then all possible 256 values were tried. Pearson correlation coefficient was used to link the model and the genuine consumed power.

The following section presents a reference evaluation of the unprotected AES implementation showing its vulnerability compared to two (LFSR and tri-state buffers) side-channel countermeasures introduced later.

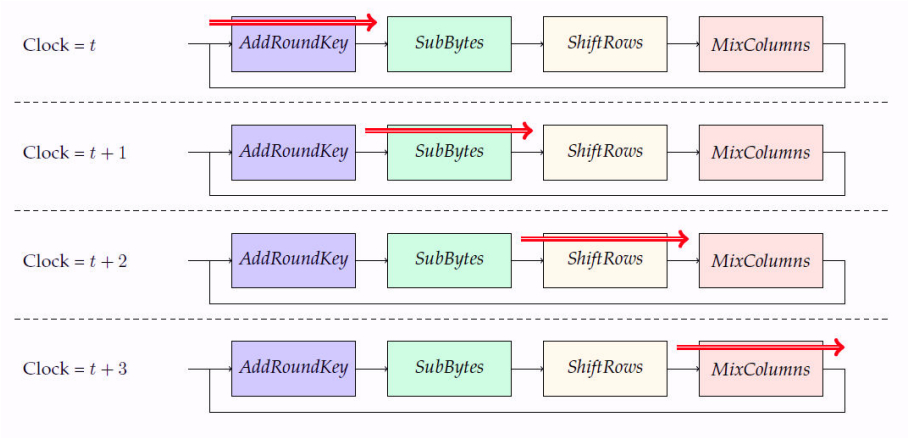


FIGURE 3.3 – Flow of Computation in Time.

3.3.2 Power Scrambling

It is a natural idea to shut down unnecessarily active blocks. To do so, each block receives a new 1-bit input named *ready* activating the block when $ready = 1$. If $ready = 0$, the block's pull-up resistors are disconnected using a tri-state buffer connected to the power source. This saves power and also prevents the circuit from leaking "unnecessary" side-channel information.

Logically the pipeline architecture that we have just described has to be less vulnerable against First Order DPA attacks. Its four register barriers introduce additional noise, so we expect that the correlation shall be at least smaller that for the AES design with one round per clock computation.

To asses the security of each proposed design, we will compare an incorrect key byte correlation to a correct key byte correlation. Figure 3.4 shows these two coefficients. As expected, the correct key is correlated to the power traces, however even for 500,000 traces Pearson correlation coefficient is smaller than 0.015. Anyway, this implementation is vulnerable.

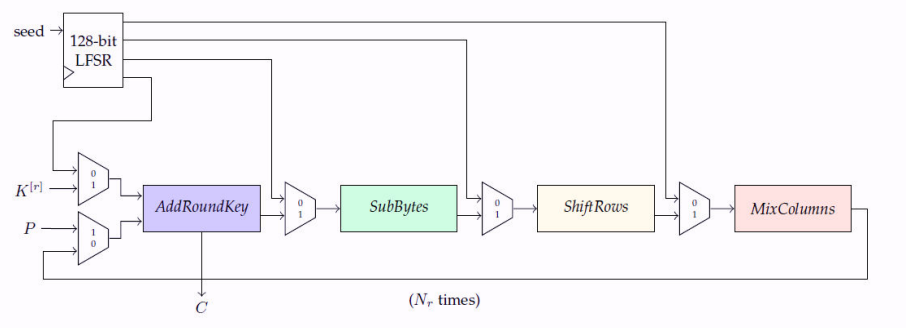


FIGURE 3.5 – Power Scrambling with a PRNG.

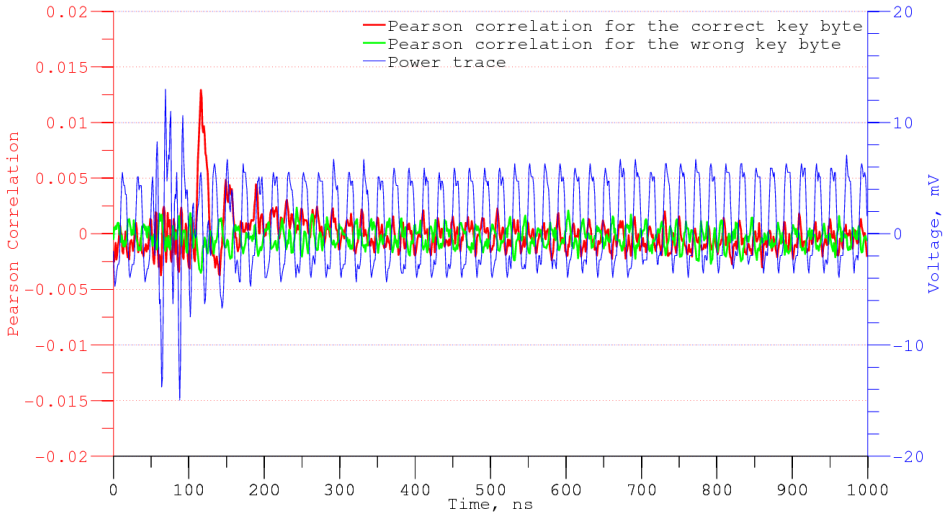


FIGURE 3.4 – Unprotected implementation : Pearson correlation value of a correct (red) and an incorrect (green) key byte guess. 500,000 power traces.

To exploit the unused blocks to hide the device's power signature even better we propose two modifications. The first consists in injecting (pseudo) random data into the unused blocks, making them process that random data. Subsequently, three of the four blocks will consume power in an unpredictable manner. Note that because we use the exact same gates to compute and to generate noise, the expected spectral and amplitude characteristics of the generated noise should mask the leakage quite well. Although any random generator may be used as a noise source, we performed our experiments by using a 128-bit LFSR. An LFSR is purely coded in digital HDL, making tests easier to implement.

Figure 3.5 shows that a multiplexer controlled by the *ready* signal selects either the useful intermediate *state* information or the pseudo-random LFSR output. For the *AddRoundKey* block, LFSR data replace the key. Therefore when *AddRoundKey*'s *ready* $= 0$,

pseudo-random data (unrelated to the key) are xored with the *state* coming from the previous block (*MixColumns* if encrypting, *InvShiftRows* if decrypting). For the other blocks, the pseudo-random data replaces the *state* when $ready = 0$.

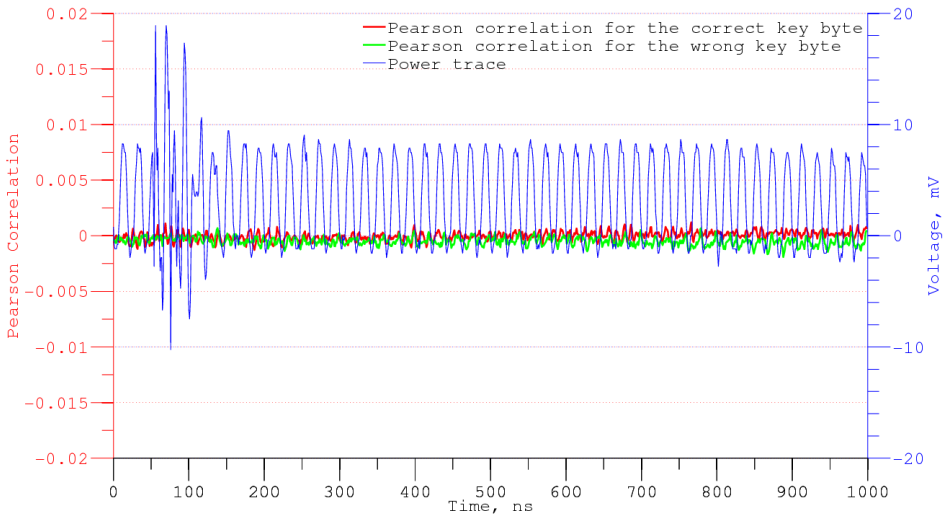


FIGURE 3.6 – LFSR implementation : Pearson correlation value of a correct (red) and an incorrect (green) key byte guess. 1,200,000 power traces.

Attacks performed on this implementation revealed that this countermeasure increases key lifetime. Figure 3.6 is the equivalent of Figure 3.4 for the protected implementation using an LFSR. The correct key correlation can not be distinguished from the incorrect key correlation even with 1,200,000 traces. However, we assume that this implementation still might be vulnerable if more traces are acquired or if Second Order DPA is applied.

Real-life implementations must use true random generators. Indeed, if a deterministic PRNG seed is used the noise component in all encryptions becomes constant and cancels-out when computing differential power curves.

A second design option interleaves tri-state buffers between blocks to hide power consumption. By shutting down the three useless blocks, we create a scrambled power trace where one block computes meaningful data while the other three "process" high impedance inputs, which means that these blocks "compute" the leakage current coming from their inputs.

As illustrated in Figure 3.7, the input signal $ready_i$ determines which blocks are tri-stated and which block is computing the AES *state*. In other words, the $ready_i$ signal "jumps" from one block to the next, so that only one block is computing while the other three are scrambling the power consumption. Although this solution has a smaller overhead in

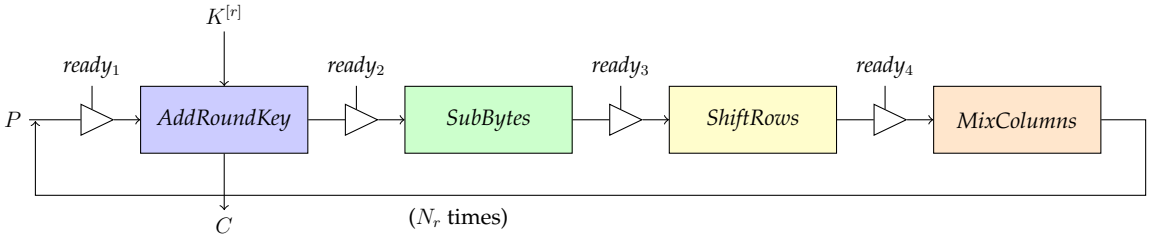


FIGURE 3.7 – Power Scrambling with Tri-State Buffers.

terms of area (as it does not require random number generation) tri-state buffers tend to be slow. Furthermore, the target environment (FPGA or IC digital library) must offer tri-state cells.

The experimental results we obtained on FPGA were surprising, we couldn't attack the design with 800,000 power traces. The correlations shown in Figure 3.8 do not allow to visually distinguish the correct key from a wrong guess. As before we assume that this implementation can be still attackable if more power traces are acquired or if Second Order DPA is applied.

A full study of this solution would require an ASIC implementation with real tri-state buffers, as an FPGA emulates these buffers and may turn out to be resistant because of an undesired CLB mapping side effects.

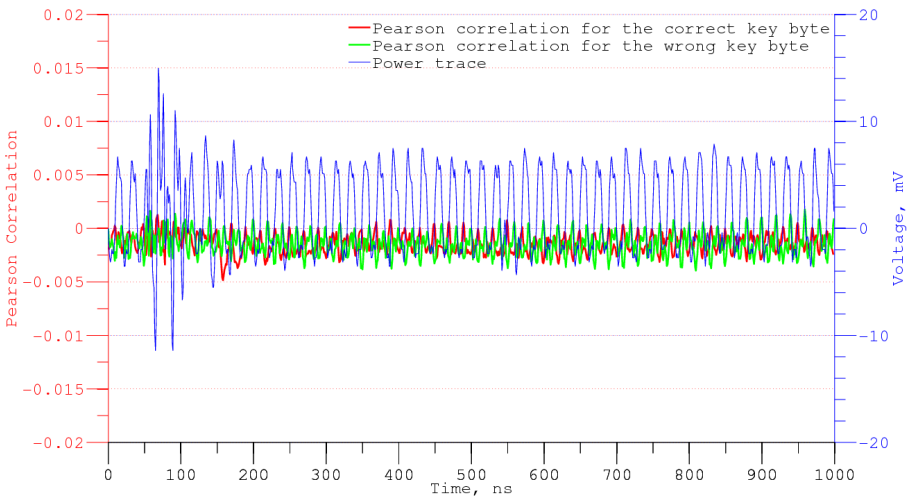


FIGURE 3.8 – Tri-state buffers implementation : Pearson correlation value of the correct key byte (green) and a wrong key byte guess (red). 800,000 power traces.

3.3.3 Transient Fault Detection

We will now use idle blocks to check for transient faults. Each block in the chain can "stutter" during two consecutive clock cycles to recompute and check its own calculation. For instance, as shown in Figure 3.9, at clock t, a given block B_i receives a *ready*$_i$ signal, computes the *state* and saves it in the register barrier R_i. At clock $t+1$, the result enters the next block $B_{i+1 \bmod 4}$ which is now working, while B_i reverts to checking, *i.e.*, B_i recomputes the same output as at clock t and compares it to the saved B_i value. This process is repeated for the other blocks in the chain. If any transient fault happens to cause a wrong result at the output of any block, the error will be detected within one clock cycle.

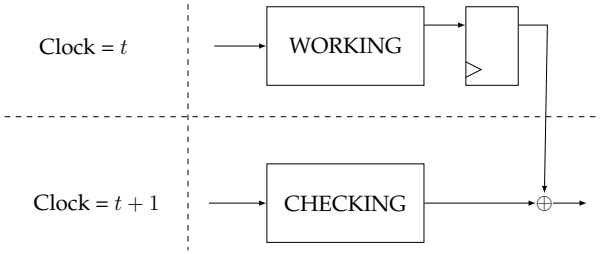


FIGURE 3.9 – Transient Fault Detection Scheme for AES.

3.3.4 Permanent Fault Detection

The AES structure of Section 3.2 also allows us to use one block of the chain to compute a pre-determined plaintext or ciphertext. The encryption (or decryption) of a chosen input (*e.g.* the all-zero input Z) is pre-computed once for all and hardwired (let $W = \text{AES}(Z)$ denote this value). While the system processes the actual input through one block (out of four) during any given clock cycle, another block is dedicated to recompute W. One clock after the actual C emerges, AES(Z) can be compared to the hardwired reference value W. If $W \neq \text{AES}(Z)$, a transient or a permanent fault occurred.

In this scenario, the system starts by computing AES(Z) in the first clock cycle, followed by the actual computation of C. This allows the implementation to check up all the blocks during the execution and make sure that no permanent fault occurred. In the last clock cycle, while C is being processed in the last block, the correctness of AES(Z) is compared with the hardwired value before outputting C.

In Figure 3.10, the red arrows represent data flow through the transformation blocks. After the initial clock cycle, the first block starts computing C. The WORKING blocks represent the calculation of C. The CHECKING blocks represent the calculation of AES(Z).

While AES(Z) will be calculated in $4N_r + 1$ clock cycles, C will be calculated in $4N_r + 2$ cycles. If the fault needs to be caught earlier, the solution described in [13] can be adapted. Yet another option consists in comparing intermediate Z encryption results (*i.e.* intermediate *state* values) to hardwired ones. Note that our design differs from [13] where a the decryption block is used for checking the encryption's correctness [12].

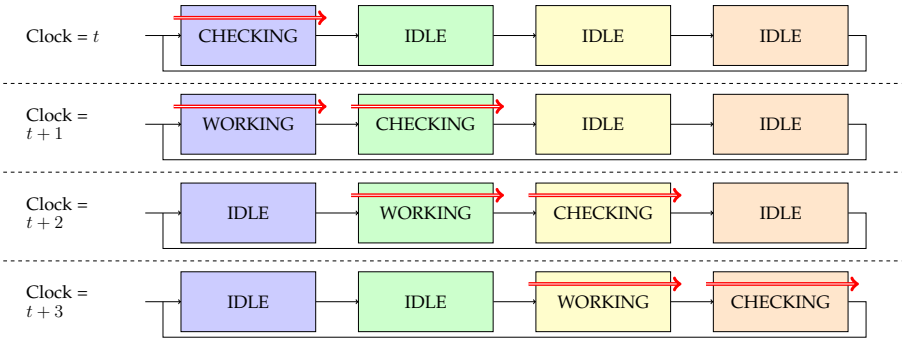


FIGURE 3.10 – Permanent Fault Detection Scheme for AES.

3.3.5 Runtime Configurability

The proposed AES architecture is a 4-stage pipeline where each stage can be used independently of the others. As already noted, blocks can perform five different tasks :

- Compute a meaningful state ;
- Be in idle state to save energy ;
- Scramble power consumption ;
- Check for transient faults by recomputing previous calculation ;
- Check for permanent faults by computing a known input.

To explore all possible combinations, we proceed as follows : first, we generate all $5^4 = 625$ combinations (5 operations for 4 transformation blocks). We can consider a subset of these combinations if we work with 4 operations only, and remember that each E entry represents two actual options (tri-state or idle). This reduces the number of combinations to $4^4 = 256$. We eliminate all configurations that are circular permutations of others, *i.e.* already counted configurations shifted in time. We also eliminate the meaningless configurations in which there does not exist even one block computing. All configurations having more than one permanent fault protection block at a time, are removed as they do not add any extra protection. Finally, we eliminate the cases where a transient fault checking is not preceded by a computing block or by a permanent fault verification.

Table 3.1 shows that the design can perform 29 different task combinations, where *C* stands for computing, *E* stands for energy (power scrambling, idleness or any combina-

TABLE 3.1 – 29 Possible Configurations.

	Block 1	Block 2	Block 3	Block 4
	C	C	C	C
	C	C	C	E
	C	C	C	T
’	C	C	C	P
	C	C	E	E
	C	C	E	T
	C	C	E	P
	C	C	T	T
	C	C	T	P
	C	C	P	E
	C	C	P	T
	C	E	C	E
	C	E	C	T
	C	E	C	P
	C	E	E	E
	C	E	E	T
	C	E	E	P
	C	E	T	T
⋆	C	E	T	P
	C	E	P	E
⋆	C	E	P	T
	C	T	C	P
	C	T	T	T
	C	T	T	P
⋆	C	T	P	E
	C	T	P	T
⋆	C	P	E	E
	C	P	E	T
	C	P	T	T

TABLE 3.2 – Number of Configurations.

C	E	P	T	Configurations
4				1
3	1			1
1	3			1
3		1		1
3			1	1
1			3	1
2			2	1
1	1		2	1
1		2	1	1
2	2			2
1		1	2	2
2	1	1		3
1	2	1		3
1		1	2	3
2		1	1	3
1	1	1	1	4

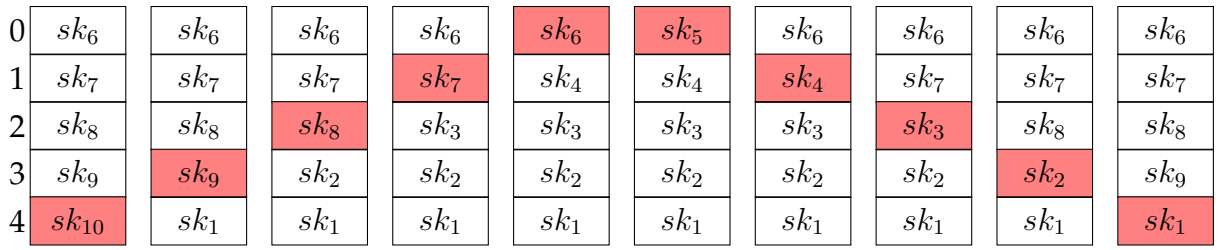


FIGURE 3.11 – Memory Halving for AES Decryption When $N_r = 10$.

tion of these two if there are more than two *Es* in the considered configuration), *T* stands for transient fault checking and *P* stands for permanent fault checking. These options can be activated *during runtime* according to the system's constraints such as power consumption or speed. If there are no specific requirements, we recommend any of the four best configurations protecting against all attacks at once. These are singled-out in Table 3.1 by a $\star$.

Table 3.2 shows the number of configurations per protection goal. Note that for a given protection goal, different configurations can be alternated between executions without any performance loss.

3.4 Halving the Memory Required for AES Decryption

As we have seen, it takes $4N_r + 1$ clock cycles to encrypt or decrypt an input. The first block of the chain, *AddRoundKey* xors the *state* with the *subkey*. Therefore, the *key expansion* block is designed to deliver a new 32-bit *subkey* chunk at each clock cycle.

When decrypting, the AES uses *subkeys* in the reverse order, so all *subkeys* need to be expanded and stored in memory before decryption starts. For that, decryption requires a $128N_r$-bit buffer. These $128N_r$ bits are stored in a register having N_r records of 128 bit each. Nevertheless, it is possible to halve the number of records by using the following idea : let sk_{N_r} be the *subkey* required at round N_r. All *subkeys* are computed but only the last $N_r/2$ *subkeys* are stored in memory. After the first 4 clock cycles, *AddRoundKey* block uses sk_{N_r} (the first *AddRoundKey* uses the initial *key* sk_0 which we assume to be already recorded). After 4 more cycles, sk_1 is saved in the record previously occupied by sk_{N_r}. The buffer continues to be used in such a way that each previously used (*i.e.* read) *subkey* is replaced by a new *subkey* of rank smaller than $N_r/2$. By the time that AES decryption requires $sk_{N_r/2}$, the *subkeys* sk_1 to $sk_{N_r/2-1}$ would have already been replaced *subkeys* sk_{N_r} to $sk_{N_r/2}$.

As shown in Figure 3.11, only 5 records are required when $N_r = 10$. Analogously, $\{6, 7\}$ records are required for $N_r = \{12, 14\}$. The red positions are *subkeys* being used at each

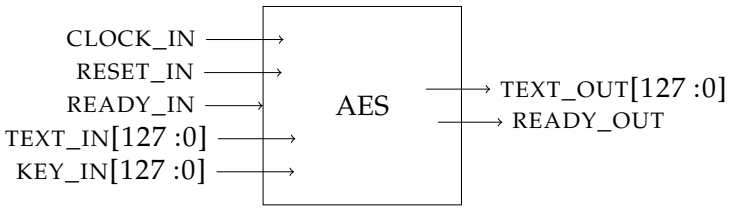


FIGURE 3.12 – AES Design's Inputs and Outputs.

AddRoundKey operation, from left to right. Note that we assume that the initial *key* sk_0 is known and does not need to be stored.

The algorithm is formally defined as follows : Create a buffer of $N_r/2$ records denoted $r[0], \ldots, r[N_r/2 - 1]$. Place in each $r[i]$ the *subkey* $sk_{i+1+N_r/2}$.

Define the function f as

$$f(i) = \frac{|2i - N_r - 1| - 1}{2}$$

When sk_i is needed, fetch it from $r[f(i)]$. After this fetch operation update the record $r[f(i)]$ by writing into it sk_{N_r-i+1}.

3.5 Implementation Results

A 128-bit datapath AES encryption core was coded and tested in Verilog and compiled using Cadence *irun* tool. Cadence *RTL Compiler* was used to map the design into a 45nm *FreePDK* open cell digital library. Figure 3.12 represents the inputs and outputs of the AES core. The module contains a general clock signal called *CLOCK_IN*, an asynchronous low-edge reset called *RESET_IN* and a *READY_IN* signal that flags the beginning of a new encryption. Plaintext is fed into the device *via* the 128-bit bus *TEXT_IN*, while the 128-bit key is fed to the system through the input called *KEY_IN*. The module outputs two signals : *TEXT_OUT*, which contains the resulting plaintext and *READY_OUT*, that represents a valid output.

Table 3.3 compares an unprotected AES core to the countermeasures described in this paper. The increase in terms of area is $\sim 6\%$ for the LFSR implementation and $\sim 4\%$ for the tri-state design. The LFSR implementation showed almost no increase in terms of power consumption. Since tri-state buffers shut down three out of four blocks per clock, we expect a reduction in the power consumption. The tri-state design saves roughly 20% of power compared to the unprotected AES. As tri-state buffers tend to be slower, this

TABLE 3.3 – Unprotected AES, LFSR and Tri-State Buffer Designs Synthesized to the 45nm *FreePDK* Open Cell Library.

	Unprotected	**LFSR**	**Tri-state**
Area (μm^2)	61,581	65,194	64,243
Number of cells	10,643	11,035	11,162
sequential	783	911	787
inverters	1,483	1,614	1,493
logic	8,375	8,506	8,368
buffers	2	4	2
tri-state buffers	0	0	512
Total power (mW)	2.10	2.16	1.68
leakage power	1.20	1.28	1.26
dynamic power	0.89	0.87	0.41
Timing (ps)	645	645	806
Frequency (GHz)	1.55	1.55	1.24
Throughput ($Gbit/s$)	4.84	4.84	3.87

TABLE 3.4 – Spartan3E-500 Utilization Summary Report.

	Unprotected	**LFSR**	**Tri-state**
Number of Occupied Slices	1,994	2,290	2,296
Number of Flip Flops	1,142	1,270	1,146
Number of LUTs	3,521	4,106	4,031
Timing (ns)	10.789	10.714	11.580
Frequency (MHz)	92.68	93.33	86.35
Throughput ($Mbit/s$)	289.3	291.3	269.6

design lost 20% in terms of clock frequency and throughput, while the LFSR version showed no speed loss, as expected.

Table 3.4 shows the three designs benchmarks in FPGA. They were coded in Verilog and synthesized to the Spartan3E-500 board using the Xilinx ISE 14.7 tool. LFSR and tri-state designs showed an area overhead of $\sim 15\%$ compared to the unprotected AES implementation. In terms of performance, LFSR design showed no loss, while the tri-state core lost $\sim 7\%$.

3.6 Conclusion

We described an unprotected AES implementation sliced in four clock cycles per round. Making use of this approach, we built on top of the unprotected core two power scrambling ideas to thwart side-channel attacks, such as CPA. We demonstrated how the design can also prevent fault injection by recomputing its internal *state* values or by compromising one out of four blocks at each clock to compute the encryption of a known plaintext. We then exhibited simulation results and showed the comparison of the unprotected against the protected cores. The results confirm that the overhead in terms of area, power and performance is small, making this countermeasure attractive.

Moreover, the proposed AES architecture provides different options to tune the design into the user's need. Among 29 different configurations, examples include : to make the proposed AES a 4-stage pipeline (*i.e.*, compute four different plaintexts per execution), to use three blocks to generate noise against power attacks, or to use one inactive block in the chain to recompute for encryption correctness. In addition to the proposed AES implementation, we presented a simple scheme to halve the number of memory positions required for storing *subkeys* when AES is performing decryption.

4 A Low-Cost Noise Generator

4.1 Proposed Design

The countermeasure is based on the use of asynchronous noise generation, basically we will be using a generic circuit to create random noise and blur the encryption's power consumption. This circuit can be formed of several LFSR initiated with random values, but our idea is not limited to a specific design. We aim to protect an encryption circuit running on a clock frequency of F_1, to do so we use at least one second clock F_2 so that F_1 and F_2 are co-primes (PGCD (F_1, F2)=1).

We then use a PLL to generate several sub-clocks $F_{1,i}$ so that $F_{1,i}$ runs at a higher multiple frequency than F_2, in addition, $F_{1,i}$ and F_1 are chosen as co-primes. We use these i clocks $F_{1,i}$ to feed i protection circuits containing several registers that are re-written at each clock, we thus create an asynchronous random noise. The integrated circuit should have a verification circuit that checks that the clock frequencies are not modified to ensure that an attacker is not bypassing the countermeasure by disabling or giving a wrong value to clock F_2.

The simplest circuit to create random noise are LFSR, Linear Feedback Shift Register (Figure 4.1). An LFSR is a basic Pseudo Random Number Generator, PRNG, composed of an internal state updated with a linear boolean function. LFSR are very efficient in hardware as they can operate at very high frequencies and are used as primitives for many stream ciphers.

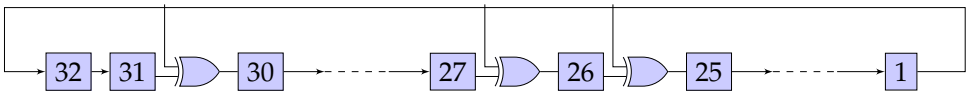


FIGURE 4.1 – 32 bits Galois LFSR

Let $S_1, ..., S_L \in F_2$ be the L bits of the initial states, L being the length of the LFSR.
Let $c_1, ..., c_L \in F_2$ be the feedback coefficients generating the output sequence by the following binary recursion :

$$S_{L+i} = c_1.S_i + ... + C_L.S_{L+i-1}$$

The output sequence is clearly ultimately periodic with a period lower than 2^L. So the security of a single LFSR is rather low as it is possible to recover the internal state via the knowledge of L consecutive outputs, there is a safety-period after which the LFSR has to be stopped and re-seeded.
This is why we need to use a combination of several LFSR. Moreover, to ensure the randomness of the noise, we need to use a pseudo RNG to provide the initial seed for the startup position of the register along with the control sequence of the register (see Figure 4.5). Basically, we will start to access one of the registers in a random position and we will switch randomly at a random time between the register starting from a random position. The goal is to create a purely non deterministic system that will be immune from library attacks.
At each new encryption or safety-period the LFSRs are re-seeded by the PRNG. There is no need for a high entropy RNG, just a non deterministic, non Gaussian one will be sufficient. Depending on the size of the LFSR we may want to stop them running after encryption to save energy. We will be testing this solution with Galois LFSRs, but many other kinds of LFSR exist such as Fibonacci or non-linear ones that could fulfill this role as well.

4.2 FPGA implementation

To prove our idea we will implement the countermeasures on an FPGA, the ciphering algorithm we use is an unprotected implementation of a 128 bits AES [73] (see Figure 4.2), the throughput of the design is 128 bits which means that it is performing one round of AES at every clock. We implemented the design on an Altera Cyclone II stater board [5], the AES is running on the main clock of $F_1 = 50MHz$.

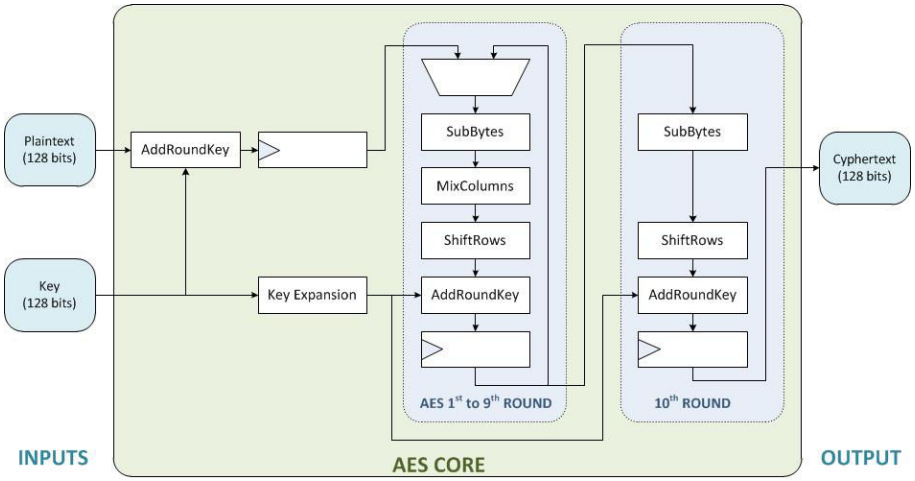


FIGURE 4.2 – Block schematic of the implemented AES

The implemented AES is unprotected and is very vulnerable to DPA, the secret key can be extracted with only few hundreds encryptions. We realize a ciphertext-based attack on the last round of encryption and we correlate power model and traces using Pearson correlation coefficient.

Figure 4.3 shows in red the power consumption of the FPGA, we can clearly distinguish the ten rounds of the encryption. The absolute value of the Pearson correlation coefficient is shown in blue, it is correlating at the tenth round when registers are rewritten. As we are using the correct key guess, the correlation is very high. This maximum of the Pearson coefficient in time gives the point where the leak occurs.

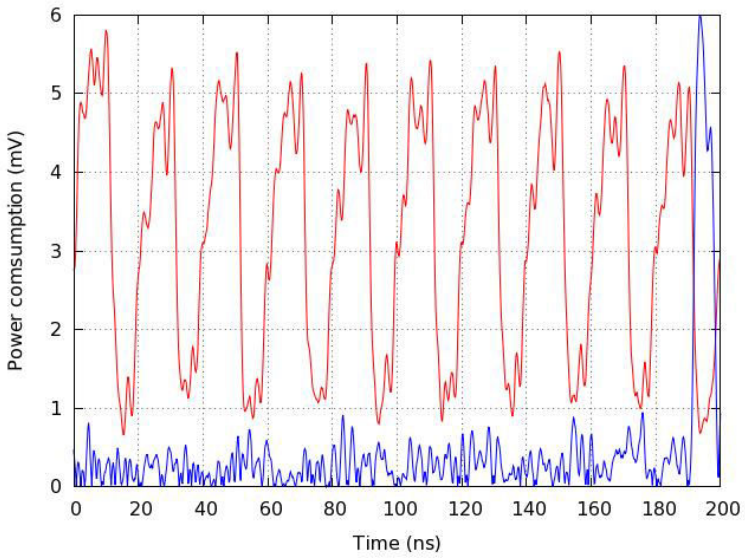


FIGURE 4.3 – DPA correlating one correct byte of the key for 1000 encryptions

To implement the countermeasure, we integrated three Galois LFSRs in our design. The three LFSRs have different lengths, they are seeded with random values and never stop running even between different encryptions. So it is a kind of worst-case implementation as the ideal solution would be re-initiated for each new encryption.
We used the second clock of the board running at $F_2 = 27MHz$ in combination with three PLLs to generate different frequencies. We verified that $PGCD(F_1, F_2) = 1$, this insure that the two clocks feeding the aes and the noise generator will be completely desynchronized. The noise generator is using three different LFSRs, which have different sizes and are all connected to different PLL, they are implemented as described in Table 4.4.

LFSR	Size	Taps Positions	PLL Frequency	Phase Offset
$Galois\ n_1$	32-*bit*	$(32, 30, 26, 25)$	$27 * 5 = 135Mhz$	$0\,\text{deg}$
$Galois\ n_2$	48-*bit*	$(48, 44, 41, 39)$	$27 * 6 = 162Mhz$	$45\,\text{deg}$
$Galois\ n_3$	64-*bit*	$(64, 63, 61, 60)$	$27 * 15 = 405Mhz$	$67.5\,\text{deg}$

FIGURE 4.4 – Characteristics of the three implemented LFSRs

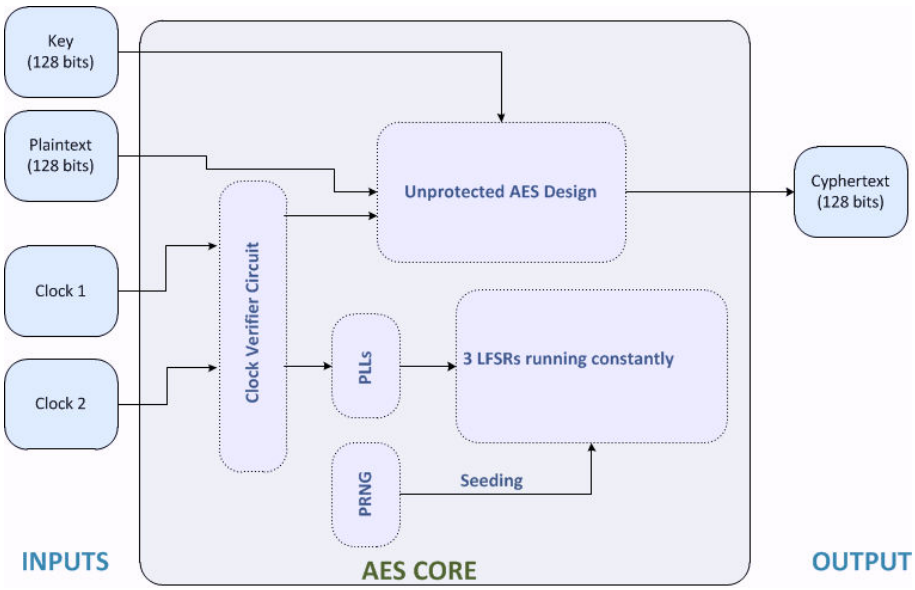


FIGURE 4.5 – Block schematic of the countermeasure

We performed a comparative DPA attack guessing one byte of the key on an unprotected version and the one carrying our countermeasure (Figure 4.6). The countermeasure provides a fairly good level of protection against first order DPA.

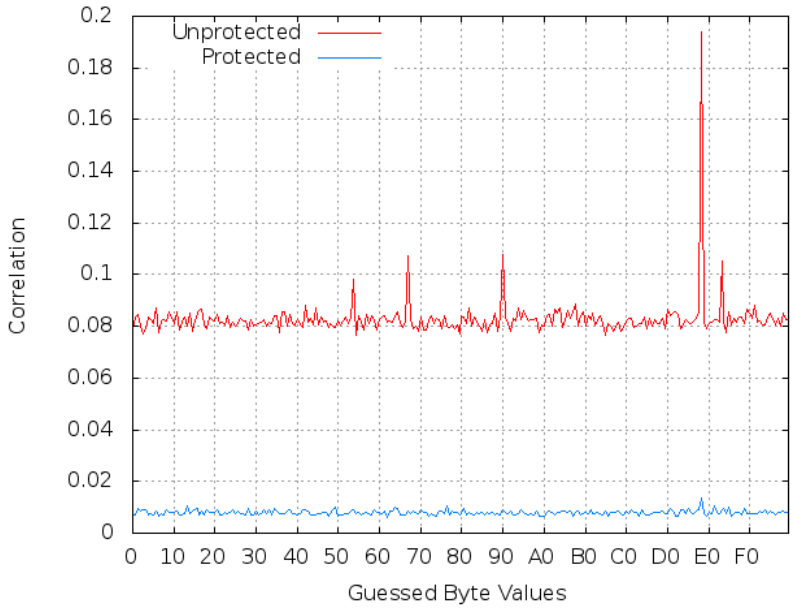


FIGURE 4.6 – DPA unprotected AES for 200000 encryptions

Electromagnetic analysis (EMA) allows an attacker to perform a more targeted attack by measuring a near field emission. In the FPGA floor planning the AES and the LFSRs are physically in different regions of the chip which allows us to extract more information by placing the electromagnetic probe on top of the encryption circuit. This concept proved to be right in reality as we were able to guess the correct byte of the key much faster by using EMA rather that DPA (Figure 4.7). To prevent this flaw we can take a special care of the floor planning during the synthesis to intricate the LFSRs within the encryption design.

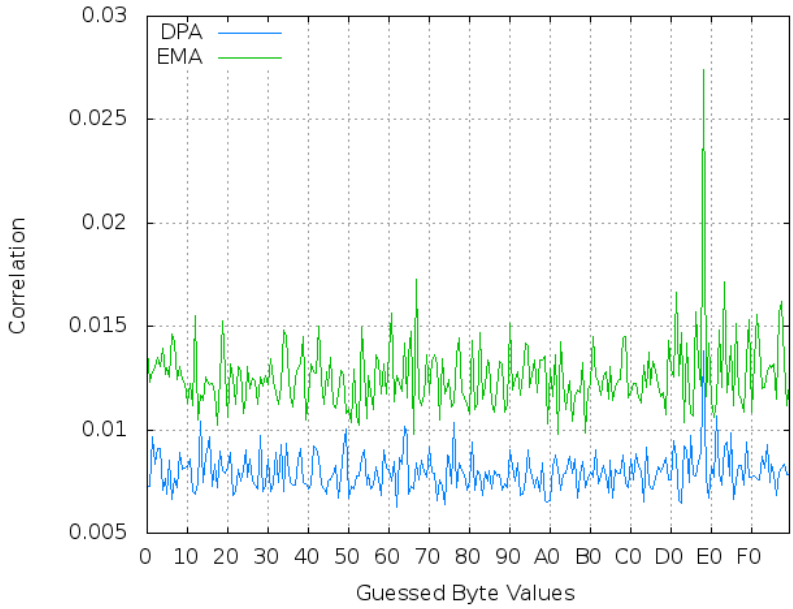


FIGURE 4.7 – DPA versus EMA correlation on the hardened design for 200000 encryptions

Another limitation of such countermeasure is the filtering in the frequency domain. To prevent this we want to choose two frequencies F_1 and F_2 carefully.
If $g = gcd(F_1, F_2)$, we want to choose $f_1 = F_1/g$ and $f_2 = F_2/g$ as close as possible, so that the filtering of unwanted frequencies will be hard to perform without loosing part of the meaningful information.

5 Antagonist registers to reduce data leakage

5.1 Principle of antagonist register

The idea is to compensate the register switch power leakage by creating a couple of antagonist registers, R_1 and R_2. R_1 is used for the output value of the algorithm and R_2 is storing $R_2 = \overline{R_1}$.

The power consumption for a register depends on switching its state during clock edges, so cutting this dependence prevents the register to leak information about storing value by power consumption analysis. Since it is not possible to avoid all state switching on a register, the solution is to make the register switch its state during all clock edges even when the stored value keeps the same.
One register can not keep and switch its value at the same time, hence it must be implemented as a pair of flip-flops, one is the main flip-flop responsible for storing the value and the other is the antagonist. The principle of antagonist is to provide a state switch power consumption when the main one does not switch its state.

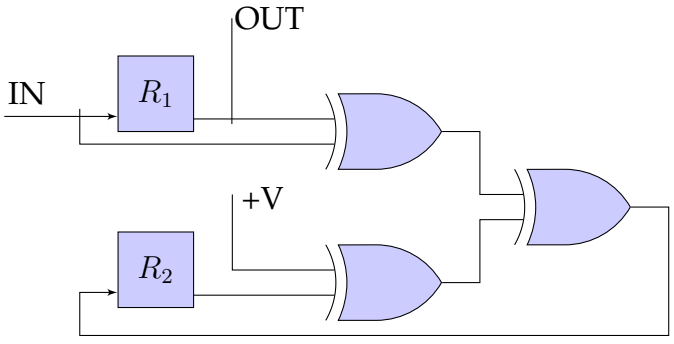


FIGURE 5.1 – Antagonist Registers

IN	R_1	R_2	OUT	Switch
0	0	-	0	R_2
0	1	-	0	R_1
1	0	-	1	R_1
1	1	-	1	R_2

TABLE 5.1 – Truth Table of the first Design

Figure 5.1 presents the schematic circuit for an antagonist register. The main flip-flop is connected to circuit input and output, it stores the input signal and drives the output signal, switching its state during a clock edge when the input switches. Also, an XOR gate is connected to input and output to detect if the main flip-flop switches, enabling and disabling the antagonist. When the flip-flop input and output are the same (0 xor 0, 1 xor 1), the XOR gate output is zero.
The antagonist consist of another flip-flop with inverted feedback that makes it to switch its state during a clock edge when enabled. If the main flip-flop does not switch during a clock edge, the antagonist is enabled to switch. Both flip-flops must have the same load on the output to generate the same power consumption when switching, that is the reason for using just XOR gates to create a balanced circuit.

5.2 FPGA Implementation

As a case study, we chose to test the antagonist registers on the Advanced Encryption Standard [73] (i.e. AES). We will then attack our unprotected implementation & protected implementation with the first order DPA. We hope to see a great improvement in the security meaning that a much greater number of encryption will be necessary to retrieve the secret key.

Figure 1.7 presents the structure of our AES hardware implementation, we will run our experimentation on an Altera cyclone II FPGA board [5]. The register R_i rewrites itself after each round to store the new current value until all rounds are completed. The protected implementation (Figure 5.2) presents the same structure except that registers are replaced by the antagonist registers.

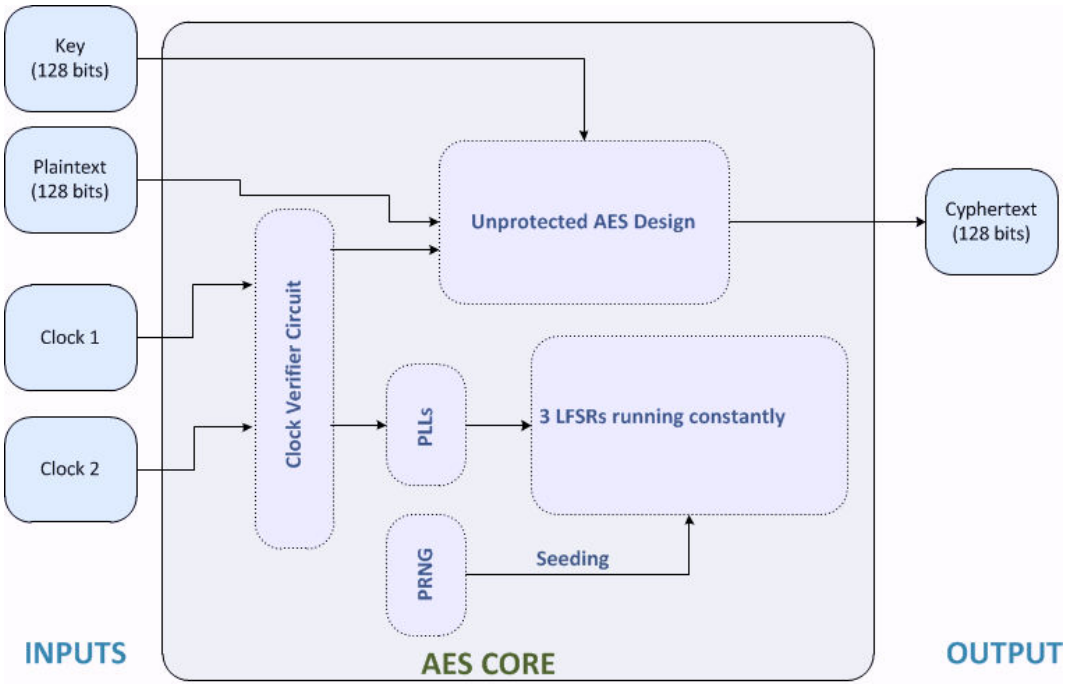
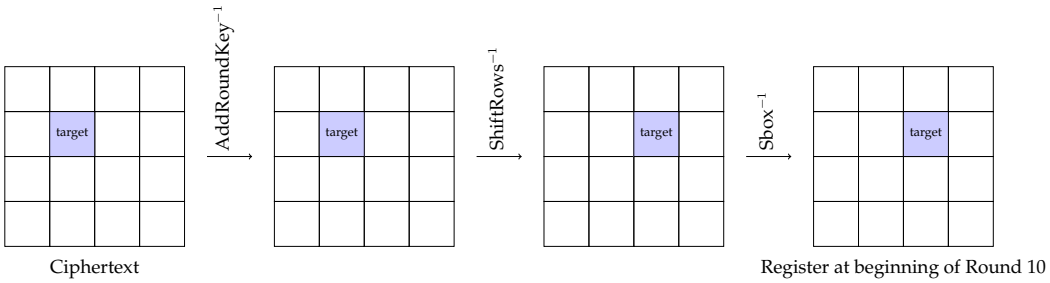


FIGURE 5.2 – Structure of our FPGA Implementation

5.3 Experimental Results

To assess the efficiency of our protected registers, we will perform a first order DPA on the unprotected VS protected implementations. We will perform a Correlation Power Attack (I.E. CPA) on the last round of AES. Our primary target for the attack is the register RS, at the last round it gets rewritten from RS_{10} to ciphertext. This operation consumes power and some information is leaked through that power. We will first build a basic power model on only one byte of the key.



We are building our power model on the operation $O(RS_{10} \leftarrow C)$ which consumes a power $P \propto HD(RS_{10}, C)$.

Or $HD(RS_{10}, C) = HW(RS_{10} \oplus C)$
So we choose one byte of the key that we guess among the 2^8 possibilities, and we build a power model for this guess. We then correlate the power model we have created with the power traces of the corresponding encryptions, we compute the Pearson correlation coefficient for each point of the graph and we look for a correlation on the 10_{th} round.

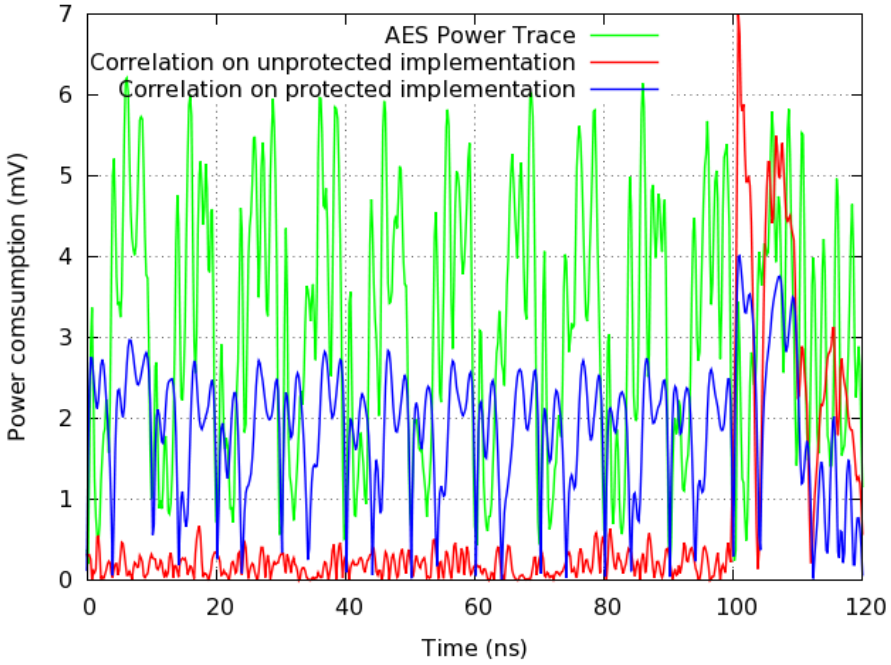


FIGURE 5.3 – Correlation Power Attack (CPA) on antagonist register versus unprotected register using 5000 traces

Figure 5.3 presents a correlation power attack using a correct byte of the key to build a power model. For the unprotected register, the correlation is very sharp, showing the exact moment the register is rewritten. The antagonist register in the contrary brings a much smaller correlation, and the correlation is spread over all rounds. The balanced registers are hiding the manipulated data, and it is almost like the registers are using the same power consumption for every possible set of transitions. Unfortunately, the output load of the antagonist register is not really the same as the "normal register" , so the consumption is a little unbalanced which causes the register to leak some information. But the solution brings a significant improvement in terms of security.

Figure 5.4 represents a key guess of all possible key bytes using 100 000 traces. We kept the highest correlation value for each possible byte and compare them. On the unprotected implementation we get a pike at 0,39 whereas the antagonist registers give a correlation at 0,22. We can thus deduce that the antagonist diminishes significantly the power leakage without modifying the throughput of the device.

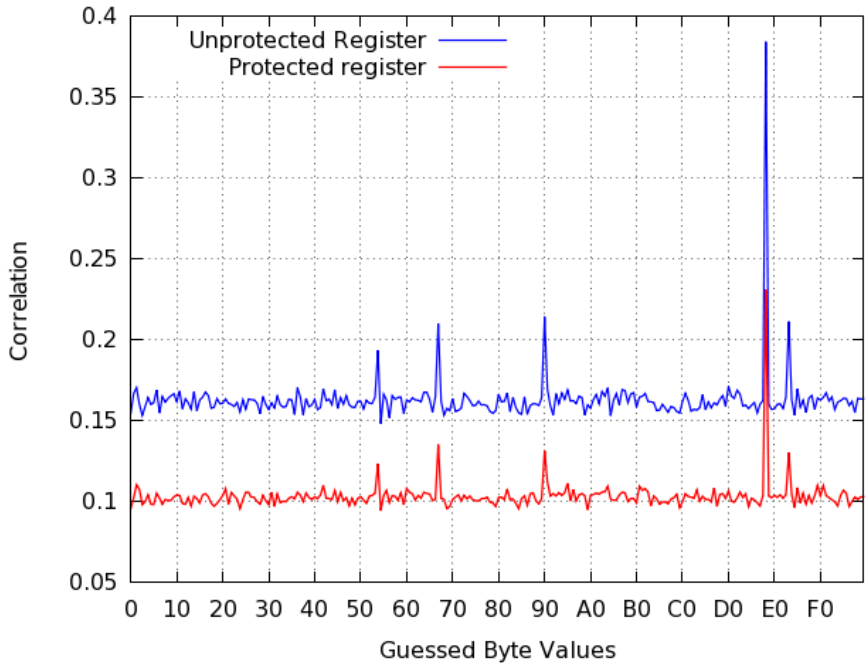


FIGURE 5.4 – CPA guessing key bytes on antagonist registers versus unprotected implementation using 100000 traces

Part 3
Protections Against Invasive Attacks

6 The Sandwich Capacitors Shield

6.1 Introduction

Invasive attacks are a major threat to IC security. These attacks allow an opponent to extract secret information from a chip without destroying it. Digital shielding is widely used in the smart-card industry to prevent invasive attacks. Digital shielding consists in using top-metal layer wires carrying logic signals to protect an IC. Attackers have been able to defeat such protections by using laser cutting, micro-probing and focused ion beams (FIB) [95]. Amongst the other solutions devised against such attacks, a number of authors explored the use of capacitance measurements [61, 51].
This paper presents a circuit protection mechanism using capacitive sensors to detect chip modifications. This countermeasure makes the IC tamper resistant to invasive physical attacks attempted from above. When the protected circuit senses the intrusive attack it reverts into an emergency mode and takes protective steps such as the erasure of sensitive data.

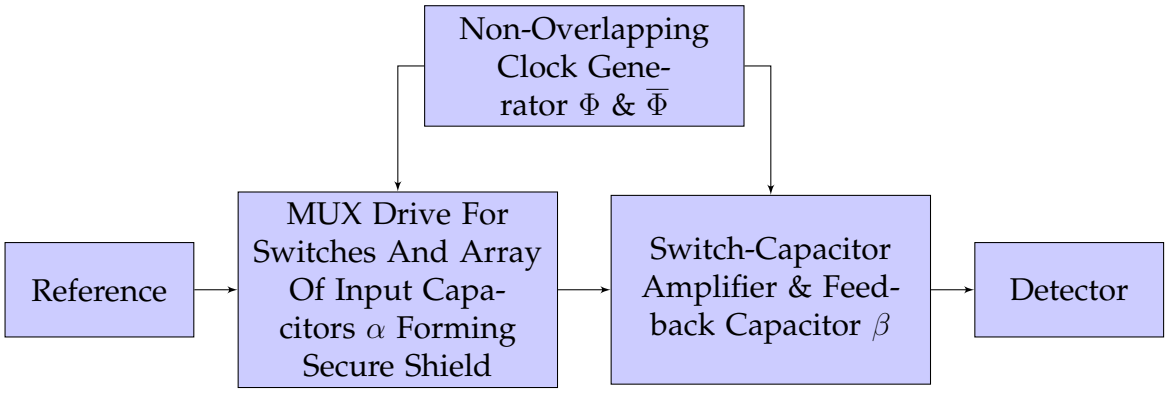


FIGURE 6.1 – Functionnal block schematic of the countermeasure

6.2 Description of the shield

The sandwich capacitor's grid spread on the IC's top layer covering the protected circuit underneath. The circuit measures the capacitance values along the grid and detects changes caused by invasive attacks. Figure 6.1 represents a possible implementation of the countermeasure using solely capacitive sensors, while Figure 6.2 proposes an extension using complementary sensors and detectors to resist a wider range of attacks.

It uses the capacitor measurement along with thermal monitoring to address temperature attacks and and light detection to counter laser fault injection.

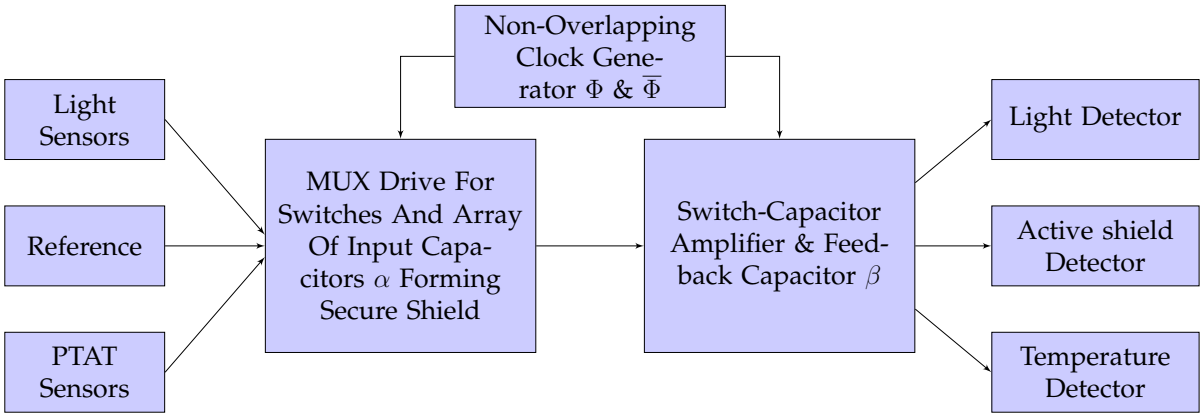


FIGURE 6.2 – Shielding with other sensors

We use a switched-capacitor circuit to compare the measured capacitor to a reference capacitor [1]. As all capacitors on the grid are multiplexed, we only need a unique detection circuit for the sensor evaluation.

6.2.1 Sandwich Capacitor's Grid

Metal to metal fringe capacitors (Figure 6.8) are known in the industry under many names such as VPP, for vertical parallel plate capacitors, MOM, for metal on metal capacitor, they are also sometimes referred to as sandwich capacitors.
This type of capacitor presents major advantages for our application ; they can be implemented in one metal layer and cover a specified area efficiently. Moreover, due to their fingers shape, each small modification on this capacitor will affect its value dramatically (cf. Section 6.3.1). To avoid a trivial bypass of the countermeasure by feeding the proper capacitance value to the switched-capacitor circuit accessible from top-layer

1. We refer to reference capacitor as the feedback capacitor in the switched-capacitor circuit.

metal lines, Figure 6.3 proposes a layout implementation of a two layer fringe capacitor with a single finger for each plate A and B.

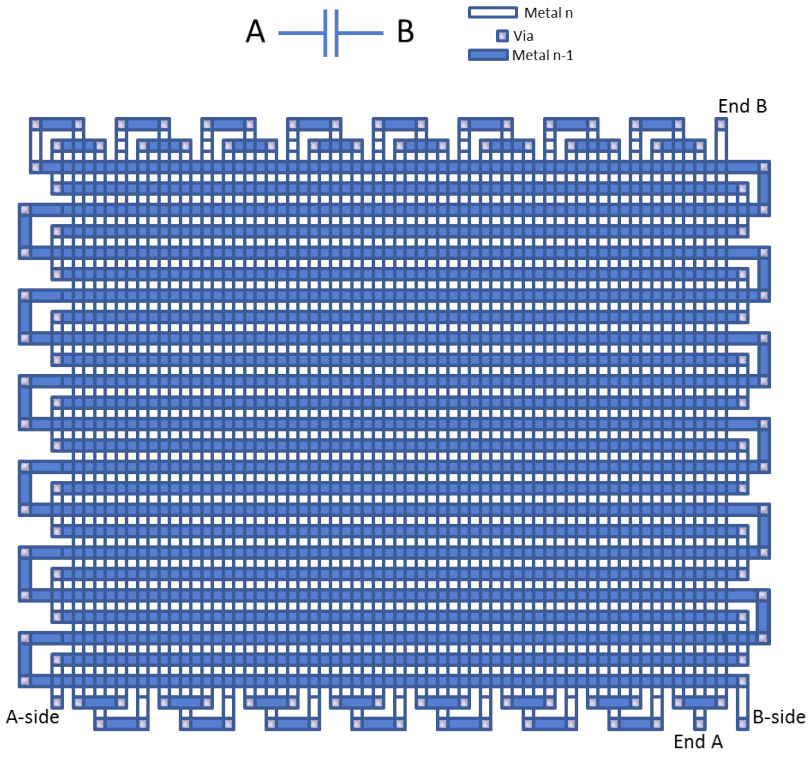


FIGURE 6.3 – Alternative two layer fringe cap with a single finger for each plate A and B

By placing an array of metal to metal fringe capacitors α (Figure 6.4) forming a shield directly above the circuits that one desires to secure, we protect them against invasive attacks.
The feedback capacitor β must not to be placed on the top layer of the metal stack to avoid any external modification as it is connected to the detector, this would easily allow an attacker access to the detector. So the feedback capacitor β is to be placed on a lower metal layer. This precaution also prevents an invasive attack by adding capacitance to the feedback capacitor thus reducing the gain of the circuit to a negligible level and allowing attacks through on the other input capacitors to the lower metal layers to go potentially undetected.

6.2.2 Evaluation circuit

The array of α capacitors would be connected to a switched-capacitor circuit (6.5), whose output is connected to a detector that measures a change in output voltage.

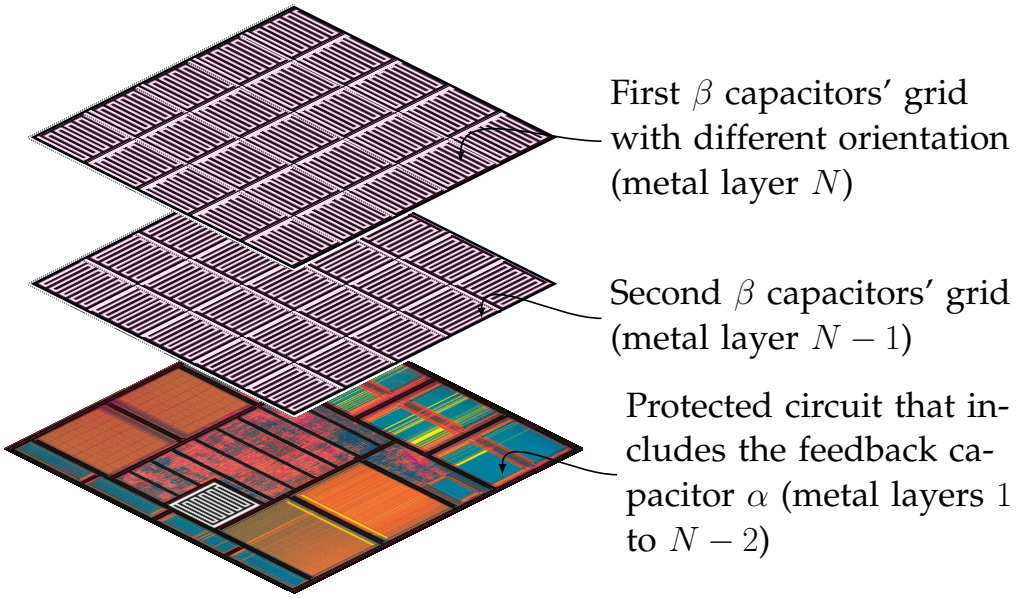


FIGURE 6.4 – Two layer metal to metal fringe capacitors array covering the protected circuit

Switched-capacitor amplifier circuit design techniques for detectors have been proposed in [81]. The output voltage will be affected by any change in the capacitor values at the input caused by an invasive attack on the IC, where the attack intends to eavesdrop with a probe through a hole in the capacitor metal shield to access signals on the metal tracks of the circuits below.

The switched-capacitor circuit we are using for capacitance measurement can have many multiplexed inputs so only single amplification circuit is needed to be used to protect a large secure area in the IC. Further embodiments based on multiple switched-capacitor amplifiers can be used, but would be wasteful of silicon area. The switched capacitor topology used can be singled ended or differential[2], both will give results which detect a change in the capacitor input values. The gain of the switched-capacitor amplifier is set by the ratio of the input capacitor and the feedback capacitor, we will adjust those parameters within the technology constraints to obtain a proper sensitivity for our purpose.

2. For the sake of conciseness we will only consider a single ended switched-capacitor amplifier embodiment in the rest of the paper.

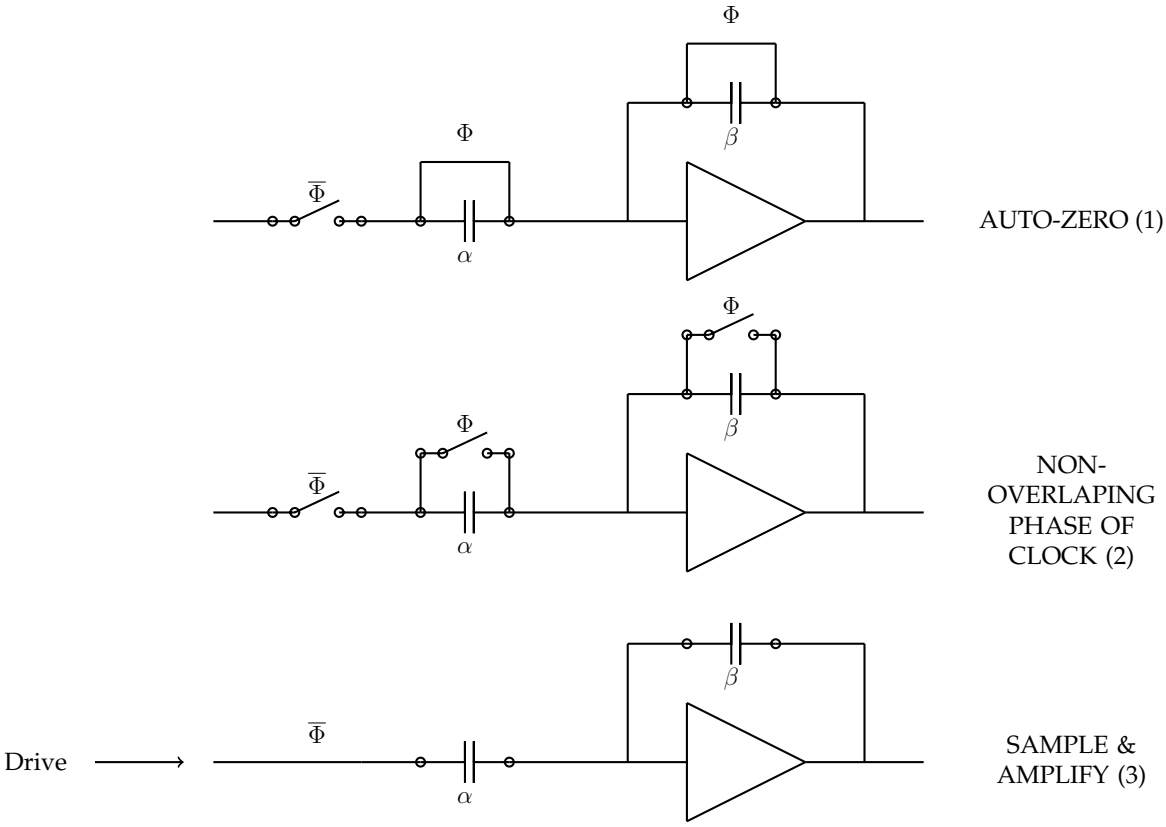


FIGURE 6.5 – Single ended switched-capacitor circuit (α being the measured capacitor and β the feedback capacitor)

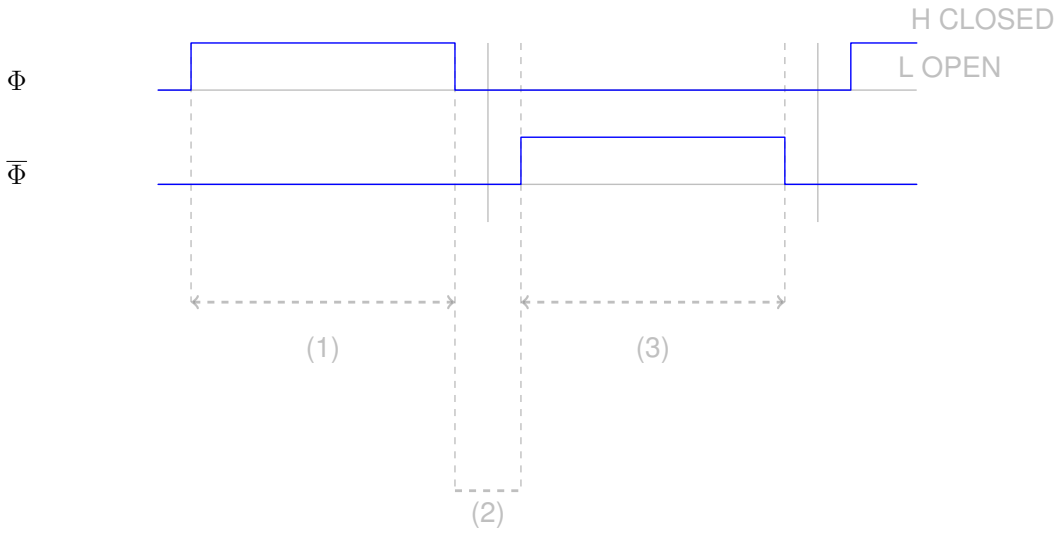


FIGURE 6.6 – Non-Overlapping Clocks

The switched-capacitor circuit is controlled by two non-overlapping clocks (Figure 6.6).

In the auto-zero phase (1) (Figure 6.5), the measured and feedback capacitors are discharged. In the sampling phase (3), we drive the input of the circuit and the detector samples the output and checks its value. If this voltage drop (Figure 6.9) exceeds a certain voltage range, the detectors trigger the safety mode. The preferred embodiment is a differential switched capacitor amplifier. As such an amplifier will amplify the difference between the two input capacitors, that are to be charged by the common reference level.

The gain ratio between the feedback capacitor β and the input capacitors α is desirable to be large, this will increase the sensitivity, as we do not care if the amplifier output hits the supply rails.

In order to select any capacitor for evaluation, the device incorporates a multiplexer to scan the plurality of input capacitors α_i together, so that they can be used with a single amplifier. Additional amplifiers could be used but it is more efficient to have a single multiplexer to select a plurality of switches and capacitors with a single amplifier feedback capacitor β.

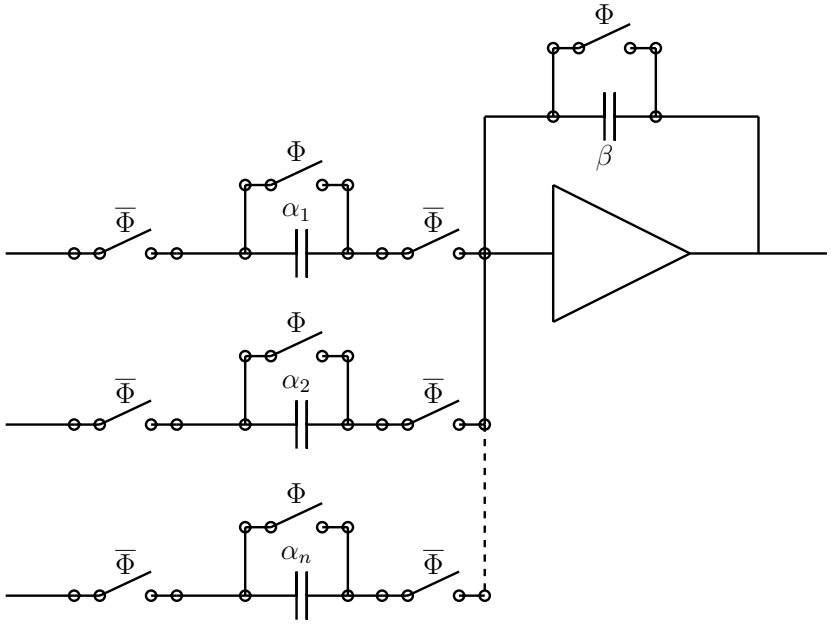


FIGURE 6.7 – Single ended switched-capacitor circuit measuring a network of n capacitors(α_i being the measured capacitors and β the feedback capacitor)

The security of the arrangement can be further increased by incorporating a sequencer to the multiplexer that controls the switches. This has the advantage that the multiplexed input capacitors are switched so that it is not the same pair of capacitors α_i being compared in respective measurement cycles. Thus, an attacker is not aware which capacitor is being compared at any given time. The sequencing of the multiplexer can also be changed or can be pseudo-random. This would further enhance the security against

a sophisticated adversary, who was prepared to invest significant time and resources in planning the attack. With the sequencer applied to the multiplexer the adversary would not know at what instant which pairs of capacitors in the array where being paired for comparison measurements.

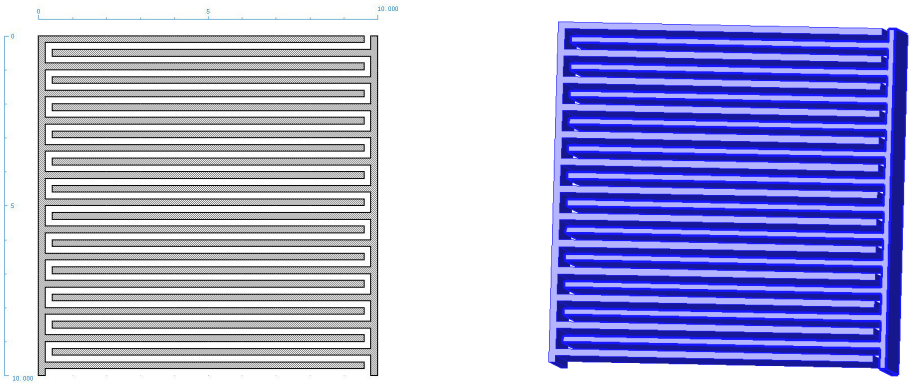
As shown in Figure 6.7, an additional switch is provided in series with the switched capacitor amplifier and the feedback capacitor β so that the feedback capacitor terminal is not directly available for access via a probe on the capacitor array α_i. It prevents an adversary from direct access to the feedback capacitor terminal.

6.3 Implementation

6.3.1 Sandwich Capacitor Design

The choice of the sandwich capacitor design has to deal with several constraint, first the sensitivity of the sensor, if the capacitor is too small, then parasitic capacitors with the circuit will not be negligible and modification in the capacitor value will be too small [3]. We also have manufacturing constraint for metal lines spacing. Each sandwich capacitor spreads on a square of 10μm $\times$ 10μm.

We evaluate the capacitor value through layout extraction on a 130nm technology, we get a value of $160fF$. To evaluate the sensitivity of this sandwich capacitor, we cut one finger of this capacitor and we got a capacitance value of $99{,}8fF$ (37,6% change).
If an attacker cut a piece of a fringe capacitor on the top layer to reach the transistor underneath, one piece of the capacitor will be floating and may even be charged if the attacker cut it while the IC is in function. We extracted the parasitic capacitance of such a floating capacitor to evaluate the impact on the sensitivity, but the phenomena revealed to be negligible.



(a) (b)

FIGURE 6.8 – 2D & 3D view of a single 160 fF top layer fringe cap (10μm$\times 10\mu$m)

3. Note that the parasitic capacitor between a sandwich capacitor and the protected circuit still have to be evaluated

6.3.2 Simulation Results

For simulation purposes we evaluate the fringe capacitor value from a layout extraction capacitor value on a 130nm for a full capacitor of 160fF and an attacked one of 99,8fF (as described in the previous section). We took a feedback capacitor of 80fF, giving a ratio of 2 for the output gain. We run the simulation over a single switched-capacitor amplifier, a differential would provide much more sensitivity to trim the sensors threshold to a lower level and determine the detector sensitivity. The transfer function gives us an output proportional to the ratio of the measured capacitor and the feedback capacitor so that we can adjust the second one to get the desired sensitivity ($V_{out} \propto \frac{C_\alpha}{C_\beta}$).

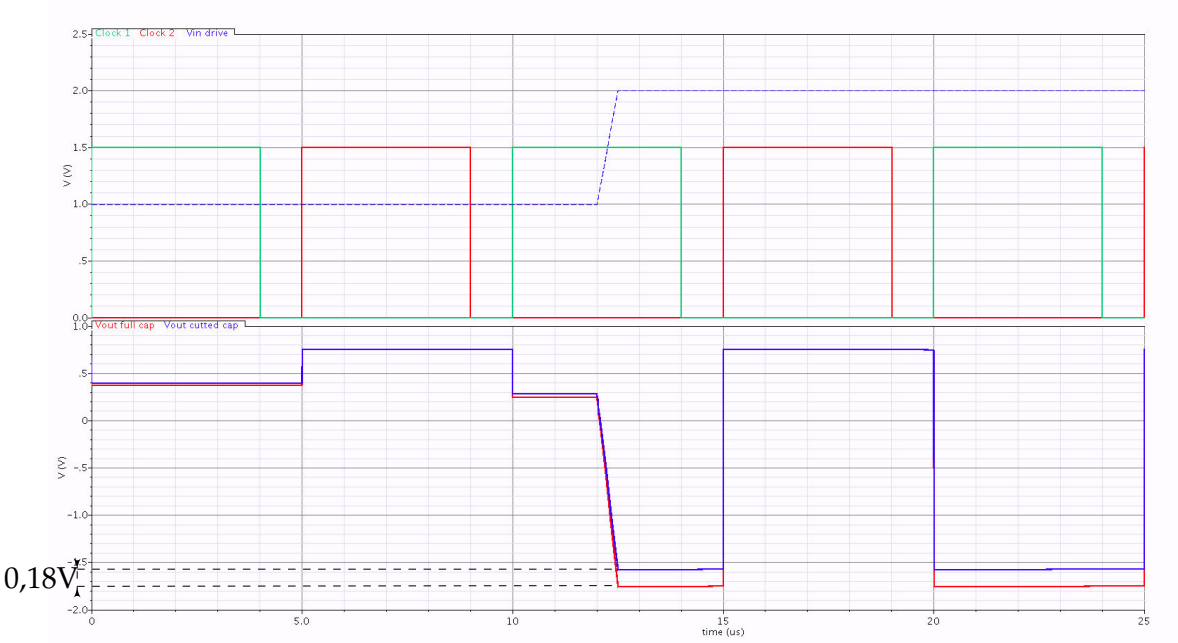


FIGURE 6.9 – Voltage values outputted by the switch-capacitor amplifier for full and cut fringe capacitor

We drive the input of the circuit with a 1 to 2V step voltage, changing the measured capacitance. We then evaluate the output and measure the voltage drop (We obtained a 0,18V drop, corresponding to a 10,25 % change in the output voltage value). Once the circuit is fully designed, we will have to take in consideration the capacitance added by the multiplexing wires.

6.3.3 Going to Silicon

A high scanning frequency is not required to measure the capacitance along the grid and we are using a unique switched-capacitor circuit, so the countermeasure's power consumption will be small regarding the overall chip consumption. The countermeasure will cost two or three masks depending on the embodiment of the sandwich capacitor chosen. We have to consider the capacitance value spread among different wafers to design the detector properly.

7 Random Shielding

7.1 Introduction

Cryptographic circuits must be protected against attacks that aim at extracting the information they conceal. Probing is a popular way to read or write data using a port (the probe tip) normally unavailable to the attacker. Thus, shielding protections were devised and implemented on top of most secure chips.

At the early ages of smart cards, their integrated nature had been already a good protection. It was believed that it is *a priori* harder for the average hackers to look into integrated circuits. Progressively, test tools, such as probing stations, normally used to debug live circuits, became increasingly available. Naturally, they turned to be relevant attack penetration tools. In the meantime, the circuits were using more and more advanced CMOS technologies, so that the feature size decreased and the number of interconnect layers increased. This deterred the probing, and has been indeed taken advantage of by designers to further obfuscate the circuit by doing a random placement and routing of sensitive gates. Thanks to recent technological advances, we witness since 2010 a revived interest in probing attacks. Especially, they have been aided by the increased popularity of the possibility to draw artificial pads that conduct directly into the inner parts of the circuit, thanks to a focused ion beam (FIB) tools. The shielding of circuits is thus still mandatory, and especially relevant nowadays.

This article is structured as follows. The motivation for an active shield is given in Sec. 7.2, along with an overview of known attacks. The specification of a feasible and secure shielding geometry is given in Sec. 7.3. A new solution, aimed at deterring trivial reconstruction of the shield, is explained in Sec. 7.4, and conclusions and perspectives are drawn in Sec. 7.5.

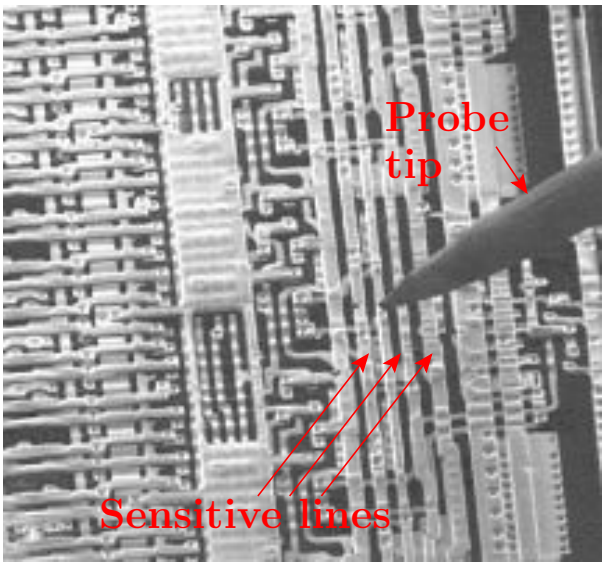


FIGURE 7.1 – Probing of a circuit thanks to prober tip, to read or force sensitive variables (courtesy of [42], Figure 4.1 of §4.2. at page 31).

7.2 Overview of Shielding

The goal of shielding is to prevent attacks that consist in

- either placing a probe on a resource (wire, gate, memory cell), for a subsequent probing (read and/or inject data into the device) during execution ;
- or modifying a chip (using a FIB), and then running it.

The first attack typically allows to spy data on a bus, or change access rights during memory writing, or alter opcodes read from program memory (Figure 7.1). The second attack can be used to unlock resources (Figure 7.2).

Conversely, shielding does not protect against reverse-engineering using methods such as layout recognition algorithms [98, 92]. While optical imaging allows to identify interconnect lines and gate functions, it usually fails to read non-volatile memory (NVM) contents (*e.g.* EEPROM cells). So, it is a safe practice to store keys in NVM memory.

Thus the general principle of shielding is to cover a sensitive area by metal lines, meant to detect intrusions. Secure logic is thus sandwiched between the shield (top) and the substrate (bottom) [1]. Backside attacks are more chancy, since the layout does not clearly stand out. Furthermore, some encapsulation techniques in the module do not allow

1. The optical PUFs [101] make use of the same strategy ; they consist of a transparent material containing randomly distributed scattering particles allowing to deviate the laser light. Nonetheless, this optical PUF technology requires light sources and detectors, and the depositions of specific materials. It is thus not compatible with mainstream CMOS processes.

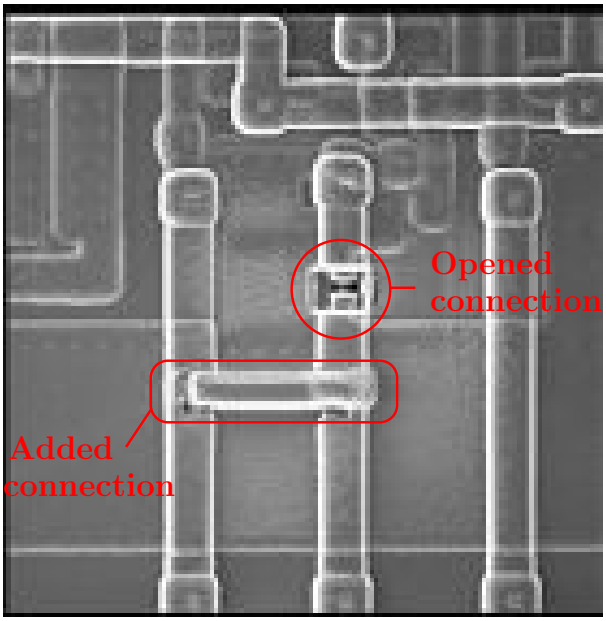


FIGURE 7.2 – Edition of a circuit thanks to a FIB, in a view to unlock the access to a memory (courtesy of [42], Figure 4.2 of §4.2. at page 31).

an accurate backside probing. Also, technologies such as SOI (Silicon on Insulator) are expected to forbid backside attacks. as well as other means of shielding, like 3D canaries [24]. The kind of protection we thus consider is sketched in Figure 7.3. The sensitive circuit is covered by metal line segments that guarantee the circuit is not functional if the shield is tampered with.

Passive shielding uses an analogue shield integrity measurement. For instance, the capacitive load of a line can serve as a signature. Passive shielding can however be defeated because it must tolerate some variations on the quantity being monitored. Thus, digital shielding (called active shielding) can be preferred. This consists in injecting random sequences of bits in a topmost metal circuit and checking that they arrive unaltered after their journey.

Nowadays threats are the attack of the active shield with the FIB. This protection can be defeated if the meaning of the lines is disclosed, since their geometry can be changed while keeping the functionality invariant (Figure 7.4). We refer to this kind of alteration by the term *shield rerouting attack*.

In practice, the identification of the identical lines is more complex than in the case of the single serpentine of Figure 7.4. But, it is often possible to recognize the equipotential lines with well structured shields (Figure 7.5).

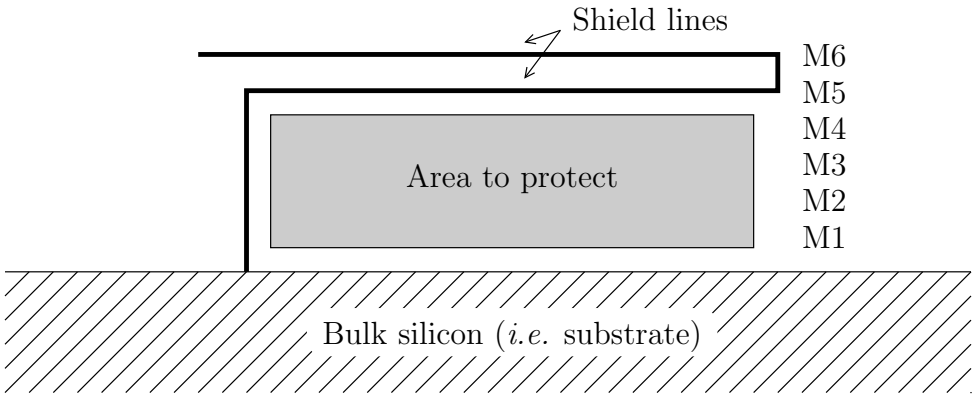


FIGURE 7.3 – General structure of a shield (sagittal view).

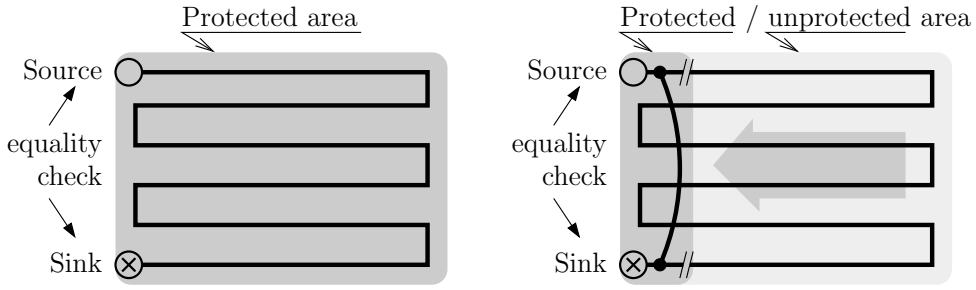


FIGURE 7.4 – Area protected by a snake active shield (*left*), and shrunk protected area (*right*) by shield extension reduction (with cuts // and connections • introduced by FIB), at constant functionality (view of the top of the shield).

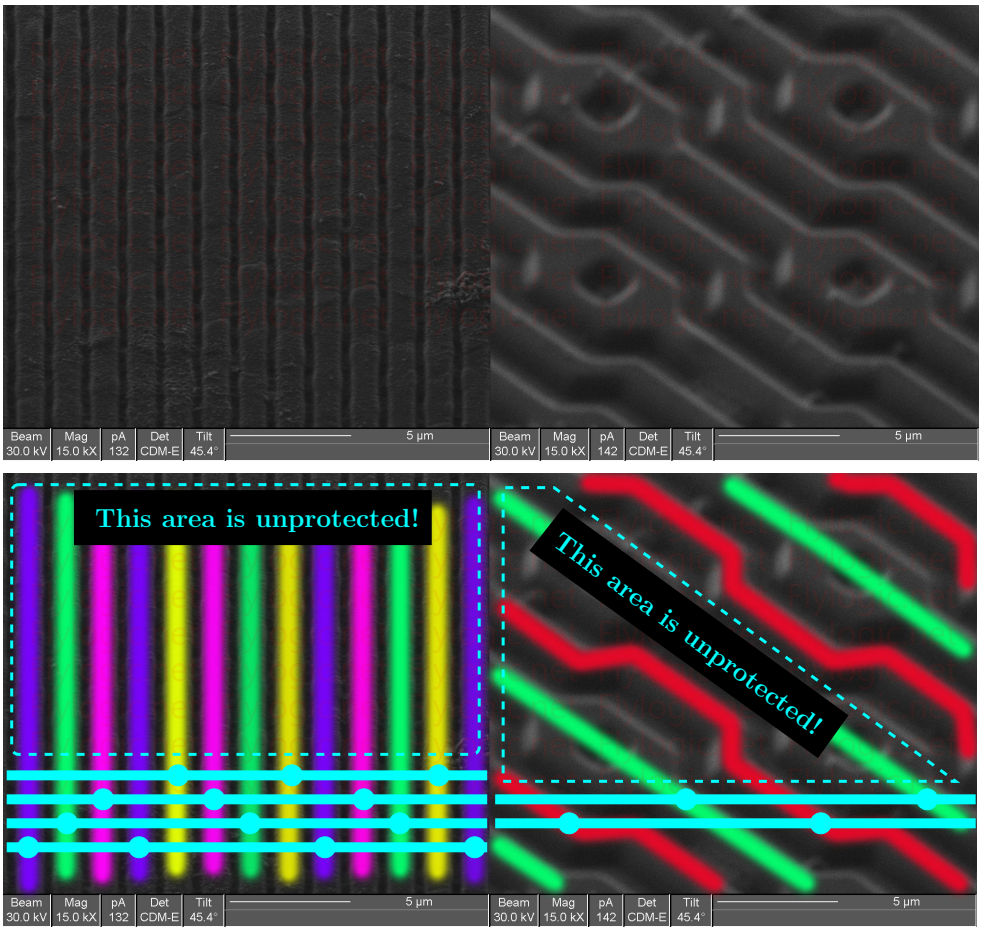


FIGURE 7.5 – Zoom at 15,000 magnification of shield structures by Infineon (*left*) and STMicroelectronics (*right*). On the bottom annotated picture, equipotential lines are underlined with the same color. [Source : [32]]. The *rerouting attack* principle is illustrated in cyan superimposed comments.

7.3 Requirements of Shielding

Little information is generally available publicly about shielding specifications. The main reason is that this topic is usually disregarded by the scientific community. As a matter of fact, researchers preferably look for other "secure by the design" protections, or consider that if an adversary has the power to probe lines, then most efforts to attempt to hamper him will only delay (but not prevent) a successful attack. The industry usually keeps the countermeasures secret, since, from a regulation point of view, it improves the attacks quotations (in terms of Common Criteria – CC [39]), and actually decreases the chance of a real-world attack once the product is on the field. Nonetheless, some companies patent shielding ideas, *e.g.* analog passive [61] or digital active [11] shield structures. Or we can also come across commercial brochures about active shielding [49], that explain some performance figures but not the protection rationale. Eventually, pirates sometimes disclose attacks, *e.g.* Tarnovsky (from FlyLogic [97]) on the Infineon SLE66 [32].

From this limited state-of-the-art, we nevertheless see that shielding is an industry requirement and a target of attacks. Here, we intend to explain the specification of shielding and provide a rigorous ground for design practices. The two main challenges when devising a shield are manufacturability and security. Cost and power consumption efficiencies are other constraints, but it appears in practice that shielding uses only scarce resources (compared to the principal factors for area and power, that are memory and their accesses, and the IO for power). We detail both aspects in the following subsections.

7.3.1 Manufacturability Requirements for the Shielding

The shield must comply with some design rules checks (DRCs), that are described in the following subsections. For the sake of illustration, we consider a 0.13 μm technology from STMicroelectronics, with 6 levels of metal. They are called M1 to M6, and can be connected respectively by the vias V12, V23, ..., V56. We recall that modern routing practices [104] consist in aligning the metal wires on a grid, that coincides with the possible positions for the vias. Thus the connectivity in the horizontal 2D plane is achieved by metal segments, whereas vertically, between Mi and Mj (where $j = i \pm 1$), it is achieved by a via Vij.

7.3.1.1 Metal extension beyond a via at end of lines

The metal lines cannot stop dead after a via : an extension is required by the design rules. Therefore, either one routing site is skipped after each via, or the width of the wire is increased to absorb the extension. These two strategies are illustrated in Figure 7.6. Here is a more accurate description of it :

(a) The extension after a via is a mandatory design rule, that makes up for possible masks misalignments during the fabrication process.
(b) The extension forbids to use all the possible slots available for vias.
(c) One solution, for instance used by automatic routers, is to forbid the use of the via next to the extension.
(d) Another solution is to make an abstraction of this extension by incorporating it into a new site placement grid (depicted in dotted fat gray lines in the figure).

7.3.1.2 Metal maximal parallel run length

Another design rule to consider is the maximum parallel run length. This rule aims at preventing two adjacent lines from being merged during fabrication. Thus the lines must be constrained to a given length. To respect this rule, some sites for M6 can be removed.

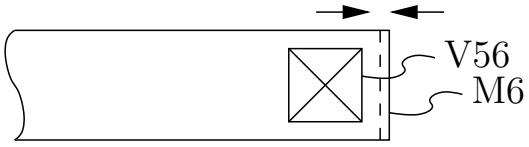
7.3.1.3 Density considerations

Lastly, the density rules must be verified. They state that the density must be neither too low nor too high, otherwise the chemical-mechanical planarization process will not manage to flatten the layer just deposited. However, by design, we are not concerned with this rule. Indeed, by using minimally (or near minimally) sized metal lines, we cannot reach the upper bound, due to the space between the wires. And as we wish to have most sites populated, we do not fall into the minimal density rule either.

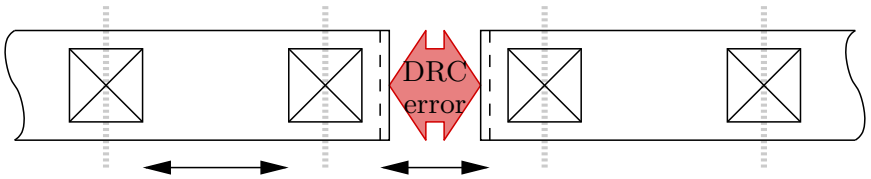
7.3.1.4 Antennae rules check

During fabrication, the wires collections (charged particles) from plasmas used in chemical vapor depositions fabrication stages. These charges can accumulate at the CMOS transistor gates and irreversibly damage them by perforating the oxide. Therefore, it is required that a maximal area of metal is exposed on transistor gates while not being connected to a drain. This situation is, by design, avoided, as the shield makes many zigzags (which coincides with the silicon founder recommended advice to reduce the

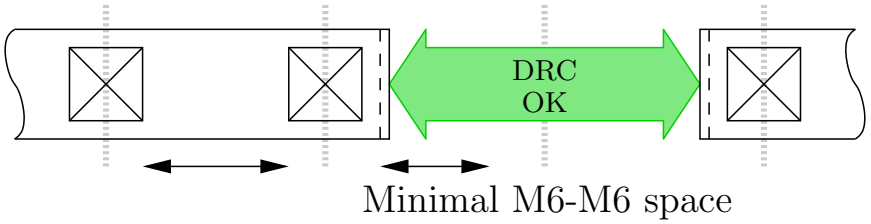
(a) **Mandatory extension after a via**



(b) **One via site is lost at every via**



(c) **Solution #1: skip a via**



(d) **Solution #2: fatten the wire and space the vias**

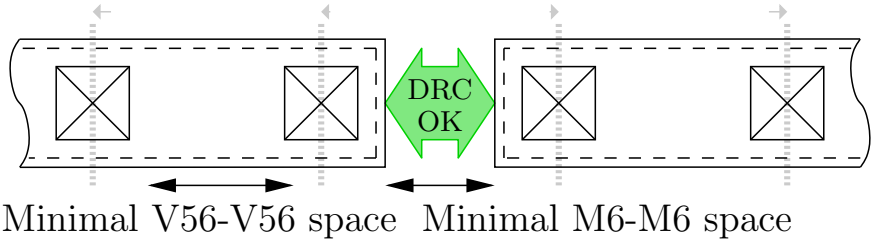


FIGURE 7.6 – Management of metal line extension beyond via end of line (extension).

antenna effect) between levels of metals, thus preventing large area wires to be connected directly to a few transistor gates.

7.3.2 Security Requirements for the Shielding

The shield feature size must be as small as possible, since FIBs have an edition capability of greater accuracy than the fabrication technology. Indeed, FIBs are intended to repair circuits, and must thus be able to alter them with a precision at least equal to that of the layout. This is normal because FIBs use a naively finer process : ions accelerated by FIBs have a smaller wavelength than the light used in lithographic fabrication processes. Thus, minimally-spaced M6 lines are employed for the chip protection.

Two other properties that an active shield must enjoy are

1. it must cover the circuit uniformly, and
2. it must resist against alteration. From Figure 7.5, it is clear that the greater the number the equipotentials, the more difficult "rerouting attacks" (that were sketched in Figure 7.4). Also, altering the geometry without modifying the connectivity will be all the harder as the shield seems more entropic.

7.4 Solution : Dense Random Spaghetti Active Shield

7.4.1 Rationale

The idea of dense random spaghetti active shield consists in

- defining a set of vertices and edges, that correspond to a set of M6 and M5 sites that are encapsulated in the area over the module(s) to protect,
- such that whatever subgraph (provided the smallest non-convex components are not singletons) is DRC compliant.

Then, for the compactness property of the shield (absence of holes), it must be ensured that all the vertices are visited once (and at most once, so as to avoid short circuits), hence a Hamiltonian path. The complete algorithm is detailed in Alg. 1.

Algorithm 1 Dense Random Spaghetti Routing.

1: **Input** N : number of different interleaved equipotentials.
2: **Step1** Build a graph whose vertices consist in free via slots and edges in the free routing slots.
3: **Step2** Label each edge by a random number.
4: **Step3** Solve the Traveling Salesman Problem (TSP) to get one Hamiltonian circuit.
5: **Step4** Cut the Hamiltonian circuit into N subpaths, and return those.
6: **Output** A random shield made up of N equipotentials.

7.4.2 Comments on the Approach

At step 1, the graph typically uses the available top-most layers, like M5 and M6. At step 3, The TSP can be run until an exact solution is found, or it can be used to yield only an approximation. Indeed, our requirement is to end up with a Hamiltonian circuit, but not necessarily optimal in terms of length. At the end of step 3, the shield consists in a single equipotential, that can thus easily be cut and shunted at will by an attacker. Thus the need for step 4, that consists in segmenting the found Hamiltonian circuit into many (N) subpaths, that are interleaved, because the graph is randomly annotated.

The number N of segments must be defined in accordance with the expected number of tries an attacker is willing to make in order to successfully edit the routing (of course without changing the connectivity !). Random shielding thus does not deter better than regular shielding (such as that displayed on the right side of Figure 7.5) against an attacker that knows its layout. But random shielding is a deterrent if its layout is unknown (for instance because its random routing makes it painful to unravel, as illustrated on Figure 7.7). Indeed, an attacker needs to recognize all the N equipotentials as a preliminary phase to start with a rerouting attack. Said differently, random shielding makes the identification phase hard. But once the shield topology is exposed, its exploitation is easy. It can be argued that this protection belongs to "security by obscurity" techniques. It is indeed ; but in the light of CC, it is valid : the identification phase is also quoted, and difficult identification improves the overall score of an evaluation. So, having admitted that the goal is merely to make the identification hard, we can assert that the spaghetti routing generated our Alg. 1 is efficient.

We focus in the sequel on special types of graphs, that we call cuboids. They consist in a juxtaposition of N_x, N_y, N_z cubes in the three dimensions. We consider the case where at least two dimensions exist (*i.e.* at least two sizes in N_x, N_y, N_z are greater or equal to two – otherwise no Hamiltonian circuit can exist because one vertex has degree one). For a cuboid to be Hamiltonian, $N_x \times N_y \times N_z$ must be even, that is to say that at least one amongst (N_x, N_y, N_z) must be even. Indeed, the value of $x + y + z$ and $x' + y' + z'$ of

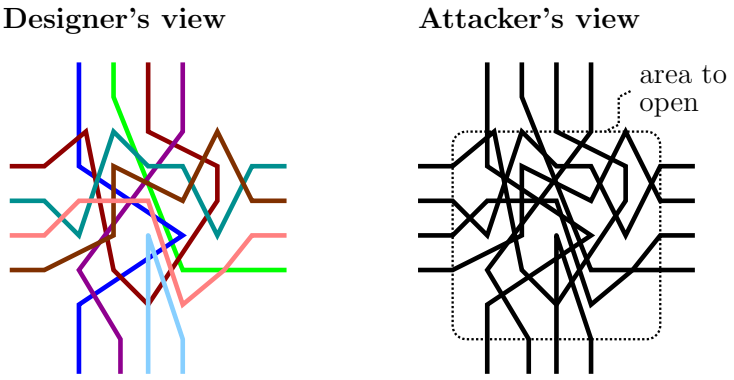


FIGURE 7.7 – The figure on the left illustrates the N segments making up an active random shield. The connectivity of these segments is unknown to the attacker ; the figure on the right shows the vision of an attacker who discovers the shield.

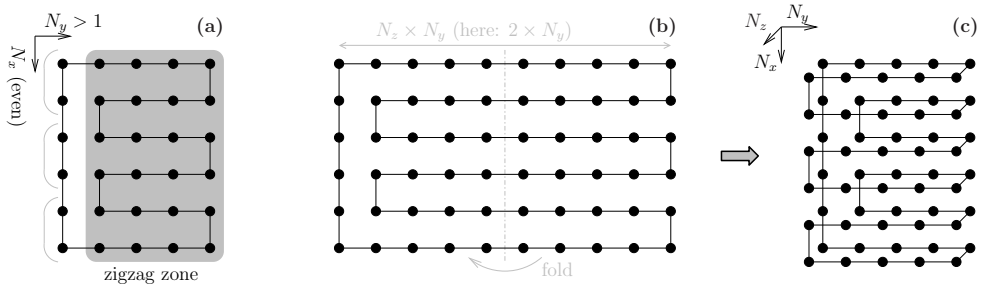


FIGURE 7.8 – Constructive Hamiltonian paths when N_x is even.

two adjacent vertices have different parity. Thus, any cycle must have a length of even parity ; such cycle cannot pass through all the vertices of the graph if the total number of vertices ($N_x \times N_y \times N_z$ in our case) is odd. The reciprocal assertion is also true : if $N_x \times N_y \times N_z$ is even, then the graph is Hamiltonian. Without loss of generality, we assume that N_x is even. Then Figure 7.8(a) shows one solution in the case of $N_z = 1$. When $N_z > 1$, a Hamiltonian circuit can be derived from a $N_x, N_z \times N_y, 1$ cycle built as previously (see Figure 7.8(b)), by folding the structure N_z times (see Figure 7.8(c)). Obviously, the algorithm 1 will find better randomized Hamiltonian circuits.

Lastly, we underline that in Alg. 1, it is important that a Hamiltonian path is found first and cut afterwards. Proceeding the other way around, no guarantee on the existence of solutions would exist. For instance, a subcycle (that passes only through some vertices) could be found ; but when removing it, the remaining graph might be separated in two unconnected parts, which is not wanted security-wise. Indeed, it creates isolated areas of the shield, while the goal is to try and spread the wires around the whole graph (not locally).

7.4.3 A Small Example

The result of an execution of Alg. 1 is shown in Figure 7.9. The plot (a) illustrates the non-planar graph. It is made up of a 4×6 sites area for the topmost metal (say M6), represented as squares, and the equivalent 4×6 sites area in M5, represented as dots. In this example, the vias are possible everywhere. To summarize, this graph has 48 vertices (24 drawn as squares and as many drawn as dots) and 100 edges. In plot (b), one Hamiltonian path, obtained by step 3 of Alg. 1, is drawn in purple. Eventually, the final results is shown in plot (c). The previous Hamiltonian path is cut in $N = 3$ paths (drawn in red, green and blue) at positions indicated by arrows (always in the lower level, *i.e.* M5). Here, it can be seen that the three equipotentials are well interleaved.

The generated layout is shown in Figure 7.10, where metal M6 is orange (resp. cyan), M5 yellow, and via V56 cyan (resp. gray) for CADENCE VIRTUOSOS (resp. GNU/ELECTRIC). These layouts are drawn in HCMOS9GP 130 nm technology (STMicroelectronics). Given the DRC rules to be obeyed (see Sec. 7.3.1), the shield drawing pitch is 97 nm, which is one order of magnitude smaller than the typical probe tip diameter ($\simeq 1$ μm being the minimum achievable).

This kind of spaghetti routing fosters long lines, and is thus very difficult to unravel for $10+$ different signals. Indeed, even if two equipotential wires are found, the overall cost to drill through a 10×10 pitch area requires nearly a complete shield reverse-engineering. Thus, we can assume this is not the primary attack path from a prospective attacker.

Incidentally, we notice that the shield lines can be heavily loaded. Indeed, at every via, that are massively instantiated (about for half of the connections in the shield), a resistance of about $1\ \Omega$ is added. However, the active shield does not require a speedy test. A functioning at 10 times the nominal frequency is still enough in most cases to ensure its usefulness. Additionally, as the capacitive load is high, this shield could also benefit from a passive integrity check. In this article, we leave this consideration as further research.

7.4.4 Performance on Larger-Scale Circuits

The high performance of the shield circuit generation is of utmost importance. Namely, step 3 of Alg. 1 is known to be NP-complete. Nonetheless, we actually do not need the shortest Hamiltonian path, but only one. In this case, some heuristic algorithms exist. For instance, the LKH software (based on [47]) allows to find a Hamiltonian path, but the graphs cannot be weighted. Said differently, it solves the TSP where edges are weighted only by 0 or 1. This method is quite fast, and allows to find Hamiltonian

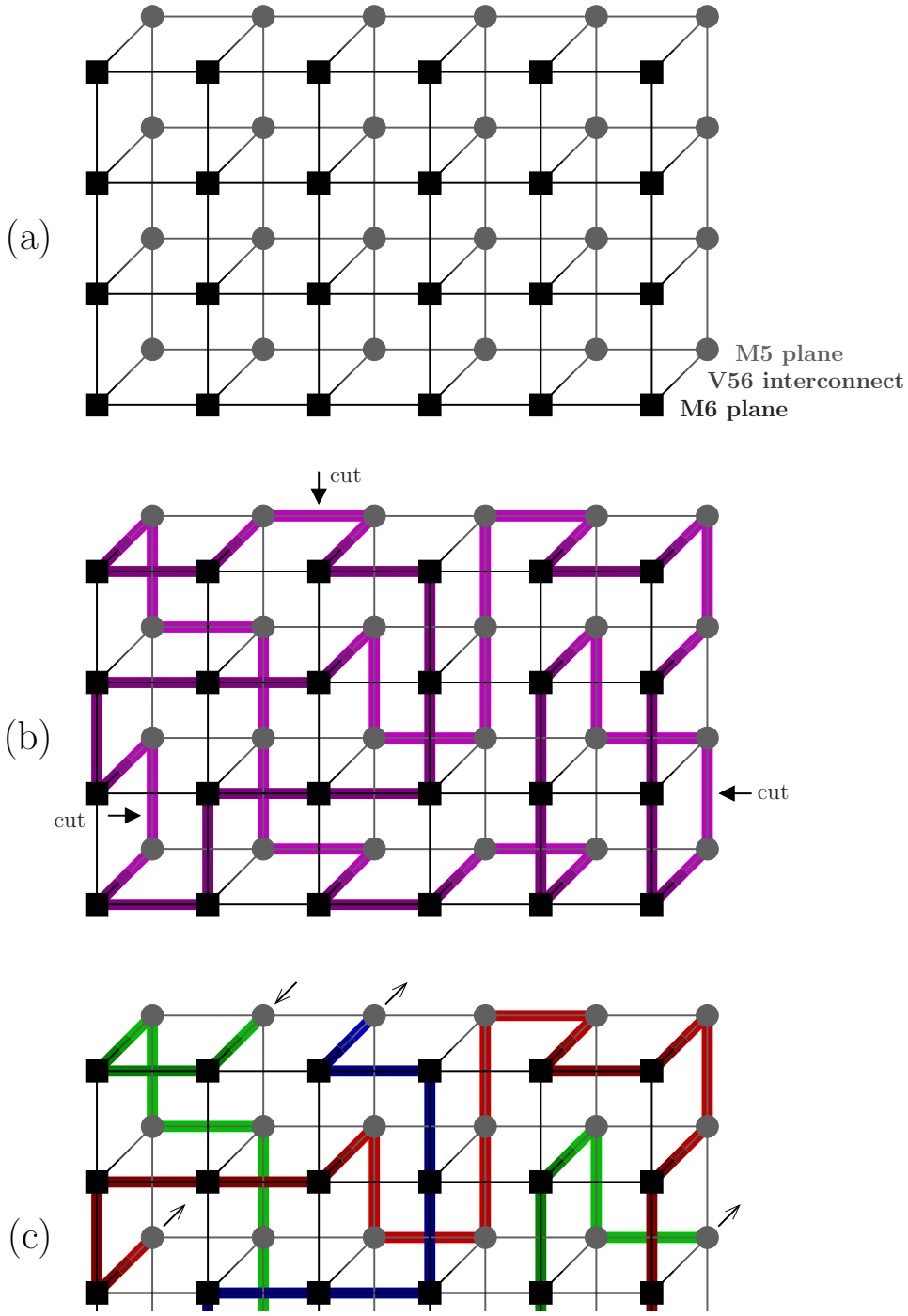


FIGURE 7.9 – Compact shielding, obtained by the execution of Alg. 1. In the final shield layout (c), the $N = 3$ segments are fed with unrelated random bit sequences.

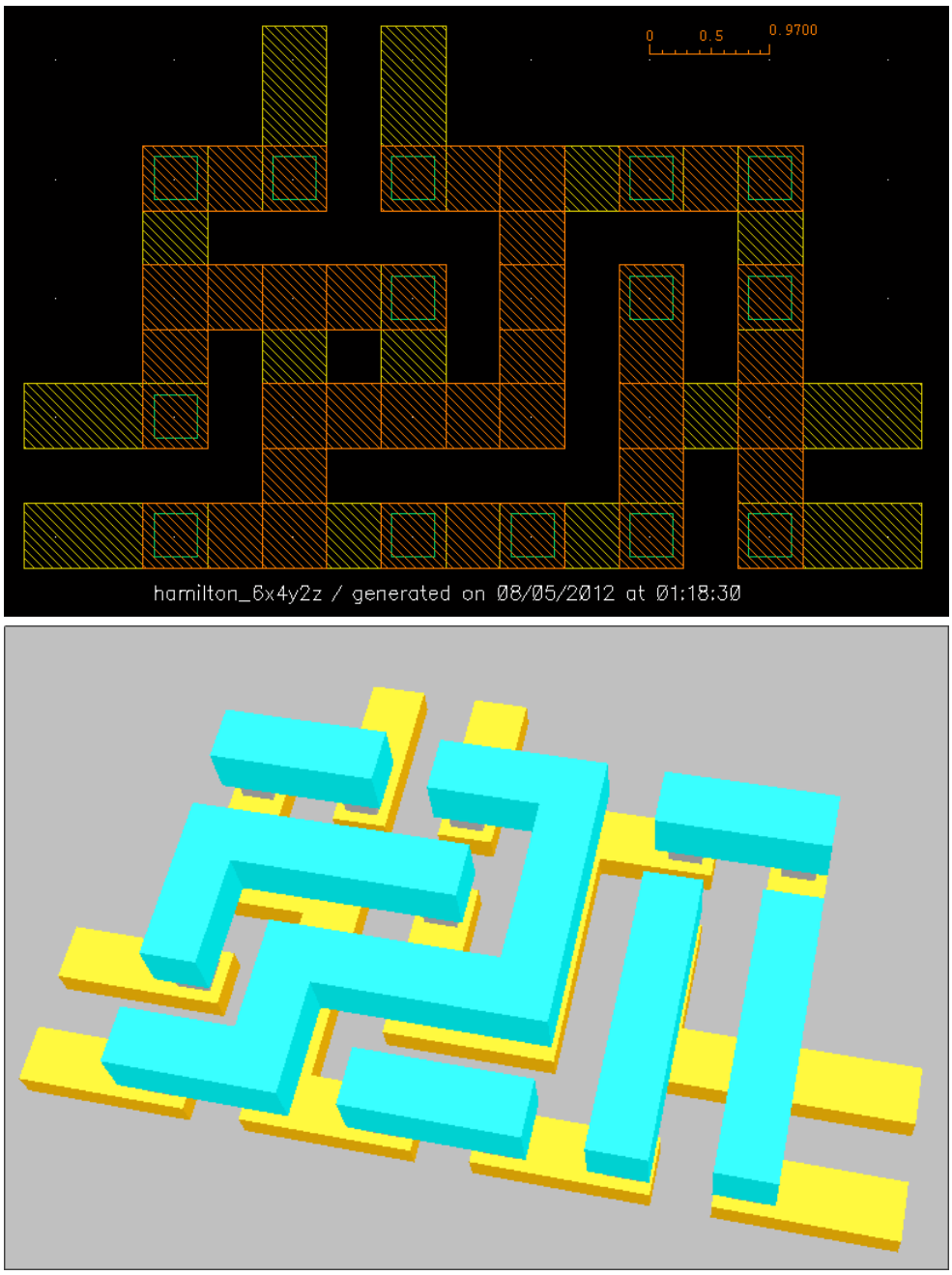


FIGURE 7.10 – Shield design under the CADENCE VIRTUOSO and GNU/ELECTRIC layout editors.

circuits for up to several thousand vertices. However, we observed that the result was not intricate enough.

Two key points for a scalable solution are thus to quantify the quality of a shield, and to find Hamiltonian circuits generation algorithms faster than those derived from the TSP.

7.4.4.1 Shield Quality

One feature to distinguish a random shield from a non-random one is the average isotropy of the routing. According to this criterion, the shield is all the better as the lines go almost equally in all directions. For this reason we propose the entropy of the directions as a metric. It is estimated as

$$H(C) = \sum_{d\in\{x,y,z\}} -P(d)\cdot\log_2 P(d)\,, \tag{7.1}$$

where $P(d)$ is the probability for the circuit to take this direction, evaluated as the ratio between the number of edges in that direction on the total number of edges (that is equal to the total number of vertices). When the shield is only 2D, $P(z) = 0$, thence we employ the limit $-P(z)\cdot\log_2 P(z) = \lim_{\epsilon\longrightarrow 0^+} -\epsilon\cdot\log_2\epsilon = 0$. As along one direction, there are as many edges going forward and backward, we intentionally neglect the notion of orientation for d. The optimal values are 1.000 bit for a 2D shield and approximately 1.585 bit for a 3D shield. To the authors' best knowledge, the question whether those bounds are tight for Hamiltonian circuits is open. We also underline that for a finite shield, the Eqn. (7.1) is approximate, since vertices on the borders cannot have edges in all directions. However, in practical cases (see Tab. 7.2), this metric is usable.

7.4.4.2 Genetic Algorithms

Genetic algorithms are random algorithms meant to approach an optimization problem by hybridizing solutions. In our case, they constitute a way to improve the Hamiltonian circuit's entropy. They are also interesting since we can start them from an existing Hamiltonian path (*e.g.* the one described in Figure 7.8), and not from a hard-to-generate one obtained by exact or approximate TSP. Our mutation consists in randomly finding two pairs of adjacent lines, and to switch the connections, as illustrated in Figure 7.11. This method requires a "bootstrap" Hamiltonian circuit (step 1). Then, it breaks the circuit into two circuits (step 2) twice. Eventually, the circuits are merged (step 3). The last step hopefully increases the Hamiltonian circuit's entropy. It is repeated until a sufficient entropy level is reached. For the sake of clarity, the same transformation is also shown in Figure 7.12. The Hamiltonian cycle is represented in an abstract way (there is no grid), and the pair of 4 neighbor sites is represented by squares and by diamonds :

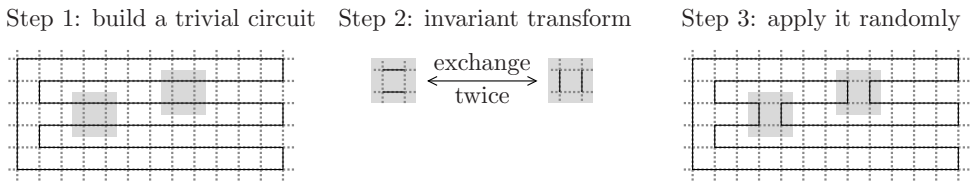


FIGURE 7.11 – Diversification of Hamiltonian circuits.

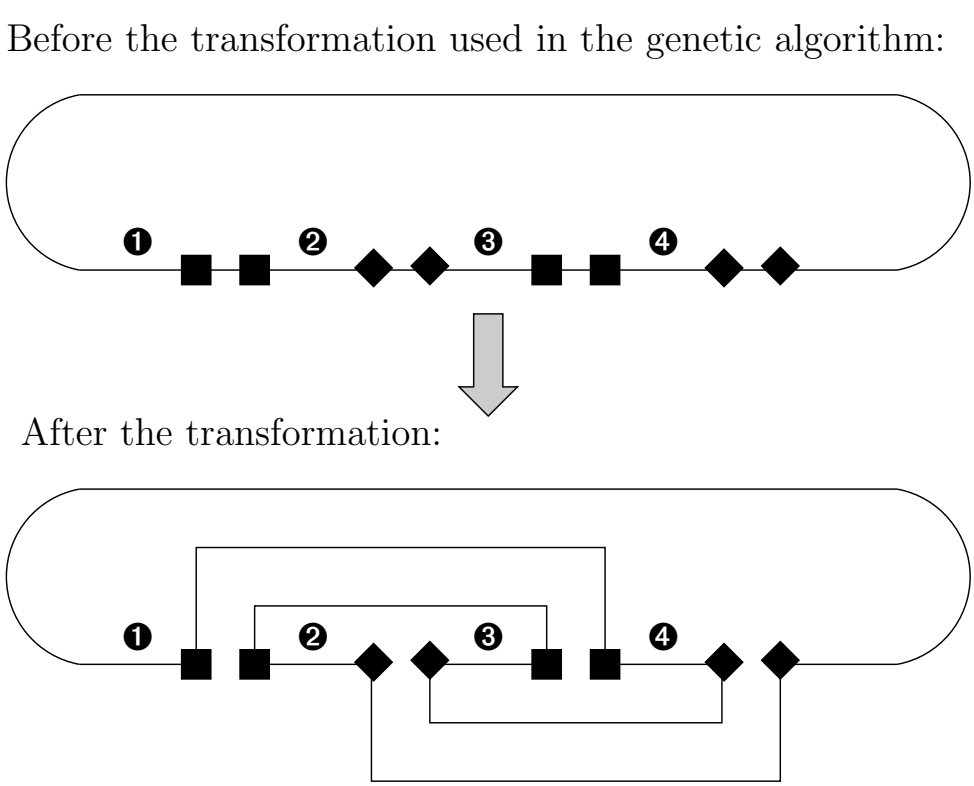


FIGURE 7.12 – Diversification process of Figure 7.11, seen topologically.

topologically speaking, it is possible to jump from one square (or diamond) to its neighbor with one hop. The original circuit uses the paths labeled ❶, ❷, ❸ & ❹ in this order (❶,❷,❸,❹). After transformation of the routing, the new circuit, (❶,❹,❸,❷), remains Hamiltonian. It requires that the squares and the diamonds be pairwise interleaved.

The table 7.1 shows that the genetic algorithm has indeed managed to increase the entropy. It is illustrated on 10 iterations (from left to right, and from top to bottom), where it is also apparent that the entropy (*c.f.* Eqn. (7.1)) appears to be a suitable metric to measure the entanglement of the N segments making up the active shield.

The performance of the shield generation has been prototyped on three examples of interest for the smart card industry

TABLE 7.1 – Evolution of a $x = 16$, $y = 16$, $z = 2$ shield with $N = 10$ segments (printed in different colors). Indicated are the entropy H and the time T for generation.

$H = 0.550$ bit, $T = 37$ ms.	$H = 0.673$ bit, $T = 129$ ms.
$H = 0.783$ bit, $T = 213$ ms.	$H = 0.903$ bit, $T = 316$ ms.
$H = 1.014$ bit, $T = 438$ ms.	$H = 1.126$ bit, $T = 599$ ms.
$H = 1.240$ bit, $T = 940$ ms.	$H = 1.349$ bit, $T = 1381$ ms.
$H = 1.454$ bit, $T = 2228$ ms.	$H = 1.556$ bit, $T = 4303$ ms.

1. a register containing a 128 bit key,
2. a 1 kB ROM, and
3. a DES cryptoprocessor.

In all cases, a $z = 2$ layer shield has been considered. The generation software is a prototype application, and thus written in a script language, without optimizations and all the "assertions checks" enabled. More precisely, it is a script PERL (version 5.10.1), executed on an Intel Xeon CPU cadenced at 2.13 GHz running GNU/Linux 2.6.32. The results are provided within Tab. 7.2. By rewriting the software with optimizations in mind, we could expect a two order of magnitude decrease in the execution time. The purpose of Tab. 7.2 is to show that with a limited memory footprint (less than 100 MB) and straightforward code, the active shield of large sizes can be generated by genetic algorithms. Eventually, Figure 7.13 illustrates the convergence speed to a solution of highest entropy (or its vicinity). The progression sometimes stalls, as for instance for the DES shield in the early minutes. The reason is that when choosing a first transformation site (step 2 of Figure 7.11), this selection can leave few choices for the second transformation site, hence numerous trials-and-errors. Also, it can be seen that the curves stop when a maximal value (for our version of a genetic algorithm) is reached and no other positive mutation can be found. It is remarkable to notice that the final value of the entropy is very close to the maximal theoretic one, which proves the mutation method presented in Figure 7.11 is relevant. We also notice that genetic algorithms allow for a trade-off *security* versus *computation time*. A less entropic shield can be found faster than an optimal (or near optimal) one, by stopping the genetic algorithm if the entropy is considered high enough (for instance, a threshold of 1.550 bit can be set).

Even larger circuits can be protected at a lower computational cost by abutting several instances of the smaller active shield problem. Although less elegant, this solution still is provided with an increased security level since the instance is *a priori* one from each

TABLE 7.2 – Computation time to generate a Hamiltonian circuit that can serve as shield for several sensitive modules of a smartcard.

Circuit	Area	Number of vertices	Time for the generation	Entropy — Eqn. (7.1)
128-bit register file	$10,000\ \mu m^2$	17,200	1 h 45 min	1.574 bit
1 kB ROM	$15,000\ \mu m^2$	25,760	2 h 43 min	1.564 bit
DES crypto-accelerator	$21,000\ \mu m^2$	33,792	3 h 54 min	1.554 bit

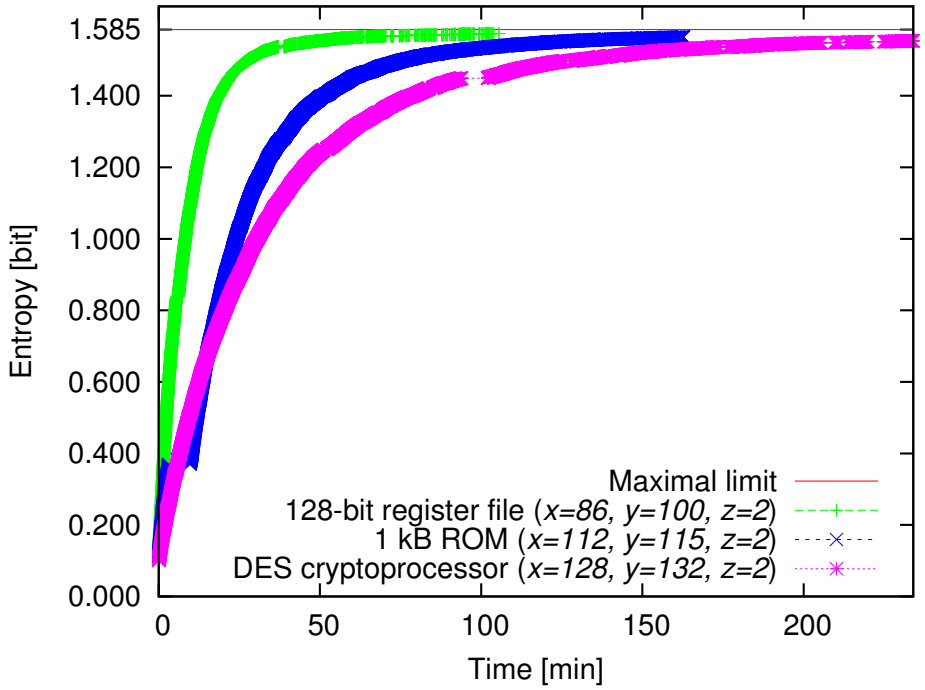


FIGURE 7.13 – Convergence rate of three real-world random active shields.

other, making artificial FIB rerouting connections chancy.

7.5 Conclusions and Perspectives

The adequate shielding of secure integrated circuits is of great importance, in regard with current exploits, for instance from FlyLogic. Active shielding is a known technique, that consists in injecting random data through top-metal wires and checking that they arrive uncorrupted. However, most publicly disclosed active shields are structured, hence their topology can be inferred by attentive attackers. In this article, we propose a method to achieve intricate spaghetti routing of a dense mesh of wires. The density is actually optimal in the sense that no routing site is left empty. This is achieved by computing a Hamiltonian circuit. The entanglement comes from randomized constraints given in the Hamiltonian circuit generation algorithm.

It is worthwhile to notice that we have employed graph-agnostic algorithms (only the genetic one requires an initial Hamiltonian circuit). However, most of the time, the graphs are highly structured, for instance, they are often "lattices", *i.e.* a (finite) repetition of a fixed pattern. Certainly the algorithms can be guided to converge faster if they are aware of the graph's topology. But given that general (graph-agnostic) algorithms finish rapidly [2], this refinement is left as a perspective for future incremental improvement on top of Alg. 1.

Eventually, we notice that the active shield presented in this paper is statically random. An improvement could consist in changing the division of the Hamiltonian circuit into several N segments, depending on a "shield configuration" random variable. The feasibility of this dynamically random routing is of interest for the future generations of active shields.

2. As compared with other CAD tools used in ASIC design.

8 Reconfigurable Digital Shielding

3D integration is a promising advanced manufacturing process offering a variety of new hardware security protection opportunities. This chapter presents a way of securing 3D ICs using Hamiltonian circuits as hardware integrity verification sensors. As 3D integration consists in the stacking of many metal layers, one can consider surrounding a security-sensitive circuit part by a wire cage.

After exploring and comparing different cage construction strategies (and reporting preliminary implementation results on silicon), we introduce a "hardware canary". The canary is a spatially distributed chain of functions F_i positioned at the vertices of a 3D cage surrounding a protected circuit. A correct answer $(F_n \circ \ldots \circ F_1)(m)$ to a challenge m attests the canary's integrity.

8.1 Introduction

3D integration is a promising advanced manufacturing process offering a variety of new hardware security protection opportunities. This paper presents a way of securing 3D ICs using Hamiltonian circuits [1] as integrity verification sensors. 3D integration consists in the stacking of many metal layers. Hence, one can consider surrounding a security-sensitive circuit part by a wire cage, for instance a Hamiltonian circuit connecting the vertices of a cube (Fig. 8.1). In this paper, different algorithms to construct cubical Hamiltonian structures are studied ; those ideas can be extended to other forms of sufficiently dense lattices.
Since 3D integration is based on the vertical stacking of different dies, a Hamiltonian cage can surround the whole target and protect its content from physical attacks. 3D ICs are relatively hard to probe due to the tight bonding between layers [102]. Moreover, the 3D circuit can even penetrate the protected circuit and connect points in space between the protected circuit's transistors.

1. A Hamiltonian circuit (hereafter "cage" or simply "path" for the sake of conciseness) is an undirected path passing once through all the vertices of a graph.

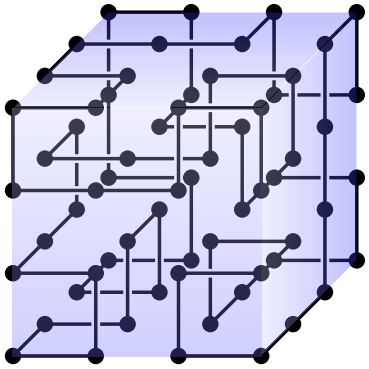


FIGURE 8.1 – Hamiltonian cycle passing through the vertices of a $4 \times 4 \times 4$ cube

A circuit running through different metal layers and different dies can thus serve as a *digital* integrity verification sensor allowing the sending and the collecting of signals. In addition, the wire can be used to fill gaps in empty circuit parts to increase design compactness and make reverse-engineering harder.

Such a protection proves challenging in terms of design as it requires devising new manufacturing and synthesis tools to fit the technology used [1, 4]. However the resulting structures prove very helpful in protecting against active probing (*cf.* Section 1.2.7).

Throughout this paper n will represent the number of points forming the edge of a cubical Hamiltonian structure. We will focus our study on cubical structures, but the algorithms and concepts that are presented hereafter can in principle be extended to many types of sufficiently dense lattices of points.

8.2 Generating Random 3D Hamiltonian Circuits

8.2.1 General Considerations

The problem of finding a Hamiltonian circuit in arbitrary graphs (HAMPATH) is NP-complete. Membership in NP is easy to see (given a candidate solution, the solution's correctness can be verified in quasi-linear time). We refer the reader to [18] for more information on HAMPATH.

If a graph contains a Hamiltonian circuit, then it contains a Hamiltonian path, obtained for instance by removing one vertex. In the applications we consider, we are looking for Hamiltonian paths whose ends are close one from each other. Thus, we focus on Hamiltonian paths resulting from the opening of Hamiltonian circuits. A quick glance

reveals that a cube's n^3 vertices, potentially connectible by a mesh of $3n^2(n-1)$ edges, break-down into four categories, illustrated in Figure 8.2 [2] :

- $(n-2)^3$ vertices corresponding to the cube's innermost edges (*i.e.* not facing the outside) can be potentially connected in any of the possible 3D directions (right, left, up, down, front, rear).
- $6(n-2)^2$ vertices, facing the cube's outside in exactly one direction, can be potentially connected in five possible directions.
- $12(n-2)$ vertices, facing the cube's outside in exactly two directions, can be potentially connected in four possible directions.
- 8 extreme corner vertices can be connected in only three possible manners.

Indeed : $(n-2)^3 + 6(n-2)^2 + 12(n-2) + 8 = ((n-2)+2)^3 = n^3$

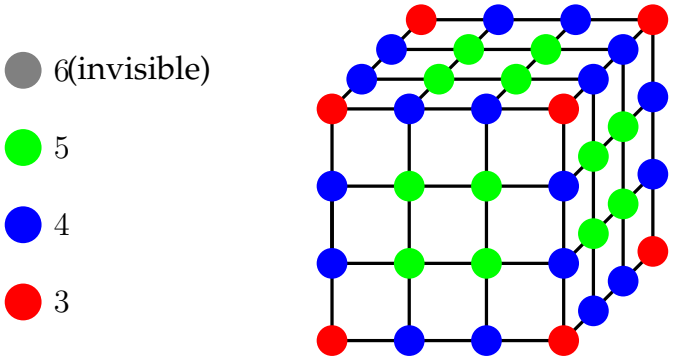


FIGURE 8.2 – Potential edge connectivity

We observe that for HAMPATH to be solvable in a cube, n must be even. If we depart from point the $(0,0,0)$ and reach a point of coordinates (x,y,z) after visiting i vertices, then $x+y+z$ and i have the same parity. Given that the path must collect all the n^3 cube's vertices, the parity of 0 (circuit start) and of n^3 (index of the circuit after traveling back to the origin) must be the same. Thus n must be even.

8.2.2 Odd Size Cubes

The above observation excludes the existence of odd-size cubes unless one skips in such cubes an edge (x,y,z) such that $x+y+z \equiv 1 \bmod 2$. To extend the construction to odd $n = 2k+1$ while preserving symmetry, we *arbitrarily* decide to exclude the central vertex (*i.e.* at coordinate (k,k,k)) when n is odd.

2. The depicted cube is shown as a solid opaque object for the sake of clarity.

Assume that we color vertices in black and white alternatingly (the cube's 8 extreme vertices being black) with black corresponding to even-parity $x+y+z$ and white corresponding to odd parity $x+y+z$. Here $0 \leq x, y, z \leq 2k$. In other words, a $(2k+1)$-cube has $4k^3+6k^2+3k$ white vertices and $4k^3+6k^2+3k+1$ black vertices.

The coordinate of the cube's central vertex is (k,k,k) which parity is identical to the parity of k. When k is even, vertex (k,k,k) is black and when k is odd vertex (k,k,k) is white. If we remove vertex (k,k,k) it appears that :

- When k is even, (*i.e.* $n = 2k+1 = 4\ell+1$) we have as many black and white vertices (namely $4k^3+6k^2+3k$).
- When k is odd, we have $4k^3+6k^2+3k+1$ black vertices and $4k^3+6k^2+3k-1$ white vertices.

Noting that each edge causes a color switch, we see that Hamiltonian circuits in cubes of size $4\ell+3$ cannot exist. Note that if one extra black vertex is removed [3] then (the now asymmetric) construction becomes possible for all k.

It remains to prove that cubes of size $n = 4\ell+1$ exist for all $\ell \neq 0$. This is seen to be true given the extensible structure shown in Figure 8.3. If ℓ is increased, the structure can be re-scaled by enlarging each floor by four units and piling up four additional floors (two at the top and two at the bottom).

As a purely theoretical side-note, although we have not fully analyzed the constructibility problem in higher dimensions, it seems that 4D cubes of all sizes are "constructible". A hypercube of dimension d has n^d vertices with a central vertex at coordinate $(dk, \ldots, dk)$. Hence when d is even the parity issue *seems* to vanish.

3. *e.g.* one of the cube's extreme edges which is necessarily black.

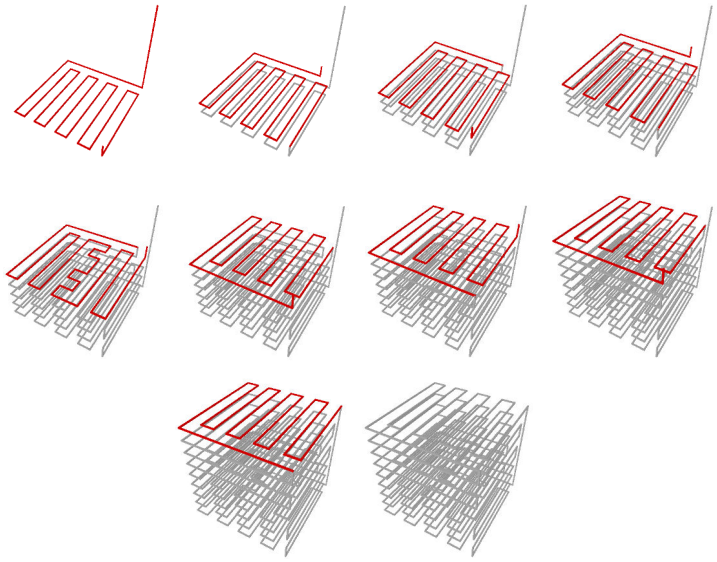


FIGURE 8.3 – Constructive proof that cubes of size $n = 4\ell + 1$ exist for all $\ell \neq 0$

8.3 A Toolbox for Generating 3D Hamiltonian Cycles

8.3.1 From Two to Three Dimensions

We start by presenting a first algorithm for constructing random Hamiltonian cycles in graphs having a minimum degree equal to at least half the number of their vertices. The entropy of a random Hamiltonian circuit generator $\mathcal{G}(n)$ for cubes of size n is difficult to estimate, it is given by :

$$H(\mathcal{G}(n)) = -\sum_{i=1}^{u_n} p_i \log_2(p_i)$$

Where u_n denotes the number of distinct circuits constructible within a cube of size n and p_i is the probability that, when queried, $\mathcal{G}(n)$ will output circuit number i. This definition is however of little use given that we know of no estimates of u_n in the literature (let alone estimates the p_i's for the algorithms proposed in this paper).

Our application requires an efficient algorithm that outputs cycles passing through a very large number of vertices. The first algorithm reduces the problem's complexity by using smaller cycles that we will progressively merge to form the final bigger cycle. Consider the elementary Hamiltonian cycle forming a simple 2×2 square. To combine two such squares all we need are two parallel edges. Merging (denoted by the operator

$\leftrightsquigarrow$) can be done in two ways as shown Figure 8.4. Note that this association not only preserves Hamiltonicity but also extends it.

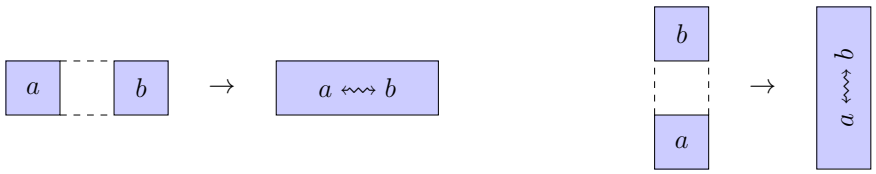


FIGURE 8.4 – Association of squares along the x axis (leftmost figure), or the y axis (rightmost figure)

In other words, at each step two different Hamiltonian cycles in adjacent graphs are merged, and a new Hamiltonian cycle is created. The process is repeated until only one Hamiltonian cycle remains. We implemented this process in C. As explained previously, our program cannot find Hamiltonian cycles for odd cardinality values simply because such cycles do not exist (see Algorithm 2). The code starts by filling the lattice with 2×2 squares, and then associates them randomly. The program ends when only one cycle is left (Figure 8.5).

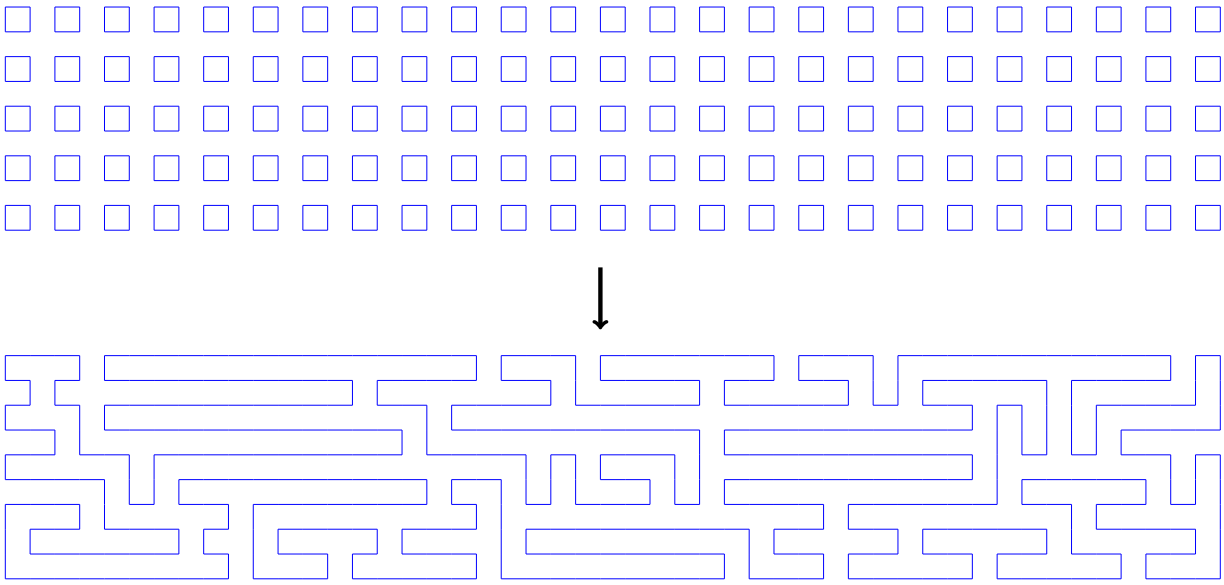


FIGURE 8.5 – Rewriting 125 squares filling a 50×10 lattice as a Hamiltonian cycle using Algorithm 1

Algorithm 2 Cycle Merging

1: **Input** $p, q \in 2\mathbb{N}$.
2: **let** $Q = Q_1, ..., Q_v$ be the $v = \frac{pq}{4}$ squares of size 2 filling the lattice of $p \times q$ points.
3: **while** Card$(Q) \neq 1$ **do**
4: choose randomly $\{a, b\} \in Q^2$ with $a \neq b$.
5: **if** a and b have at least one couple of neighbouring parallel edges **then**
6: Break a randomly chosen couple of parallel neighbouring edges, verify that they form a single Hamiltonian circuit and merge $c = a \rightsquigarrow b$.
7: **let** $Q = Q \cup \{c\} - \{a, b\}$
8: **else**
9: **goto** line 4
10: **end if**
11: **end while**

The algorithm is pretty fast, and we were able to build Hamiltonian cycles of 10^5 points using a laptop [4] within few seconds. For some p and q values, we observed some runtime spikes in single measurements due to convergence issues. Figure 8.6 shows the average runtime over 100 measurements as well as the standard deviation at each point in red.

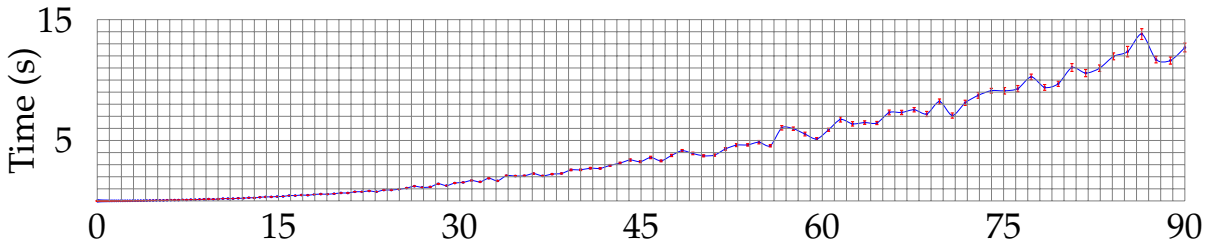


FIGURE 8.6 – Cycle Merging runtime as a function of the number of points $\times 10^3$ (average over 100 measurements)

To transform a rectangular 2D Hamiltonian cycle into a 3D one, we run Algorithm 1 for $\{p, q\} = \{p, p^2\}$ to get a $p \times p^2$ rectangle $\mathcal{L}$ similar in nature to the one shown in Figure 8.5.

Then, letting (x_i, y_i) denote the Cartesian coordinates of points in $\mathcal{L}$, with the first point being $(0, 0)$, we fold $\mathcal{L}$ into a 3D structure of coordinates (x'_i, y'_i, z'_i) using the following transform where $j = \lfloor \frac{x_i}{p} \rfloor$ and $\ell \equiv j \bmod 2$:

4. MacBook Air 1.8 GHz Intel Core i7.

$$\varphi = \left\{ \begin{array}{rcl} x'_i & = & (-1)^\ell (x_i - jp) + \ell(p-1) \\ y'_i & = & y_i \\ z'_i & = & j \end{array} \right.$$

The result is shown in Figure 8.8.

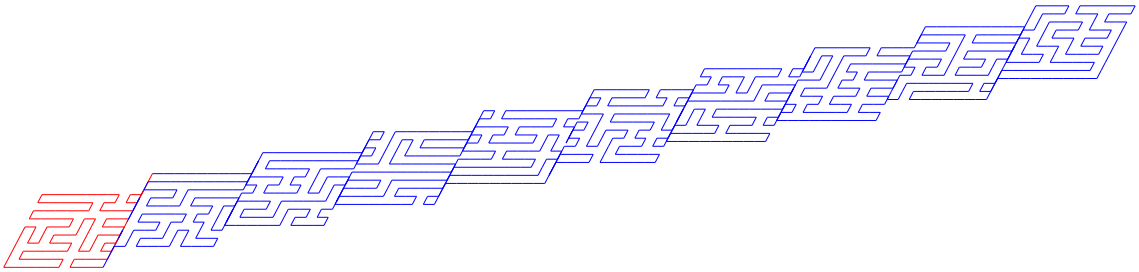


FIGURE 8.7 – 10×100 Hamiltonian rectangle $\mathcal{L}$ prepared to be folded

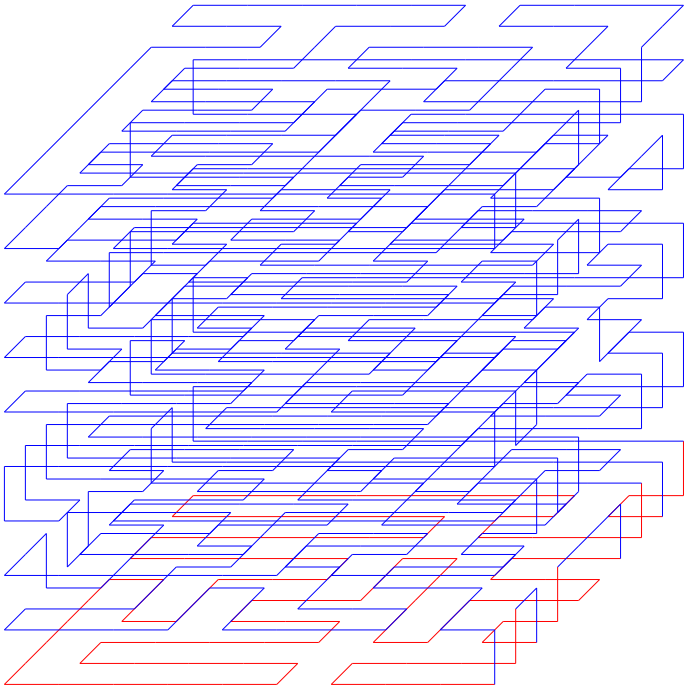


FIGURE 8.8 – $10 \times 10 \times 10$ Hamiltonian cube $\varphi(\mathcal{L})$ obtained by folding Fig. 8.7

It remains to destroy the folded nature of the construction while preserving Hamiltonicity. This is done as follows : identify anywhere in the generated structure the red pattern shown at the leftmost part of Figure 8.9 where at positions a, b, c, d edges take any of the blue positions, iteratively apply this rewriting rule *along any desired axis*

until the resulting structure gets "mixed enough" to the designer's taste. Evidently, this is only one possible rewriting rule amongst several.

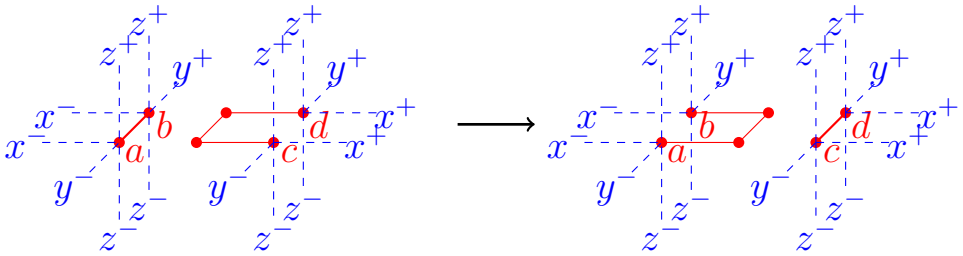


FIGURE 8.9 – Rewriting rule

Note that the zig-zag folding φ is only one among many possible folding options as φ may be replaced by any 2D (preferably random) plane-filling curve of size $p \times p$ (*e.g.* a Peano curve [71]).

8.3.2 Random Cube Association

Another approach consists in generalizing Algorithm 1 to the associating of elementary 3D cubes. As shown in Figure 8.11, one can fill the target lattice by a random sampling of six elementary Hamiltonian cubes (Figure 8.10), associate them randomly and further randomize the resulting structure by rewriting.

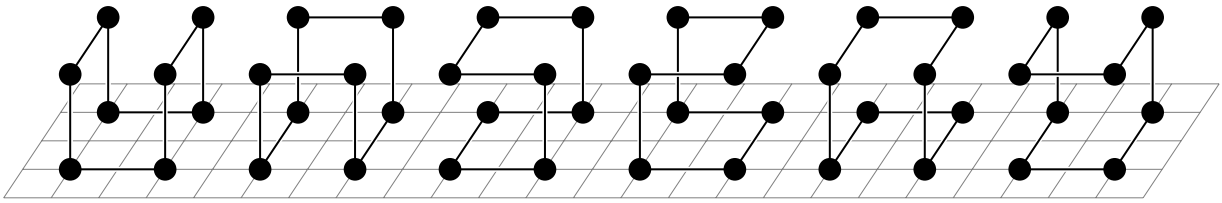


FIGURE 8.10 – The six elementary Hamiltonian cubes of size 2

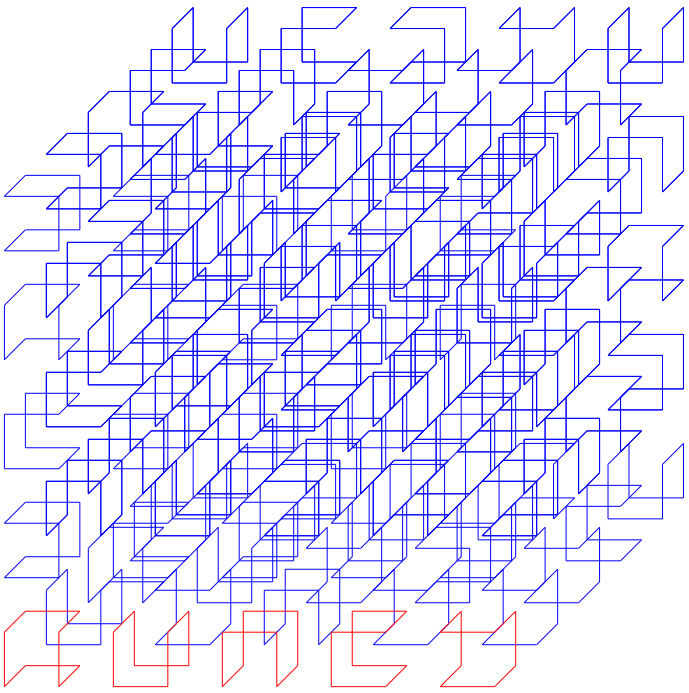


FIGURE 8.11 – Elementary 2×2 cubes filling the lattice of points forming a cube of size $n = 10$

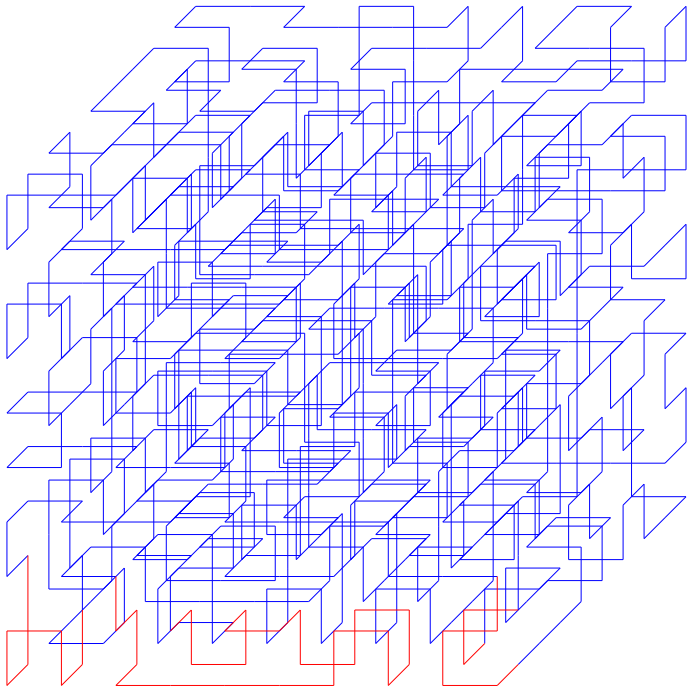


FIGURE 8.12 – An $n = 10$ Hamiltonian circuit obtained by randomly associating Fig. 8.11

The algorithm proves very efficient (Figure 8.13) and takes a few seconds [5] to compute a random Hamiltonian cube of size 50 (125 000 points).

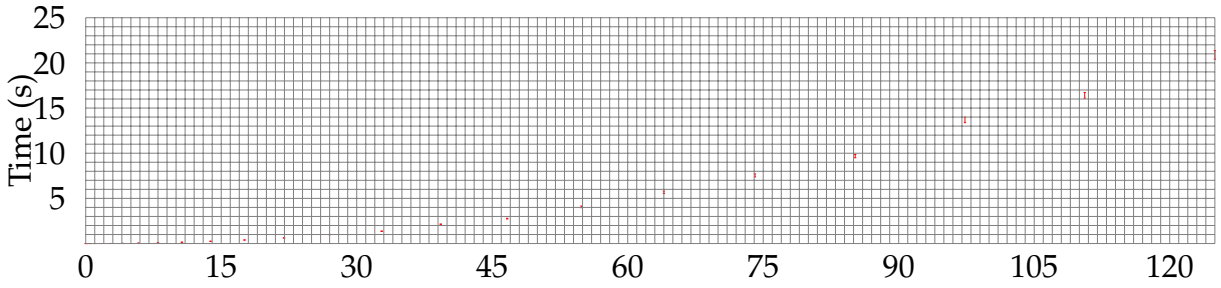


FIGURE 8.13 – Random Cube Association runtime as a function of the number of points $\times 10^3$ (average over 100 measurements)

The algorithm picks random parallel edges from different Hamiltonian cycles and attempts to associate them in one new structure. By opposition of the 2D case, the 3D case presents a new difficulty which is that in some cases associable parallel edges suddenly cease to exist. To force termination we abort and restart from scratch if the number of iterations executed without finding a new association exceeds the upper bound p^3. To

5. MacBook Air 1.8 GHz Intel Core i7.

compute structures over huge lattices (*e.g.* $n = 100$), one might need to introduce additional association rules (*e.g.* the rule shown in Figure 8.14) to avoid such deadlocks.

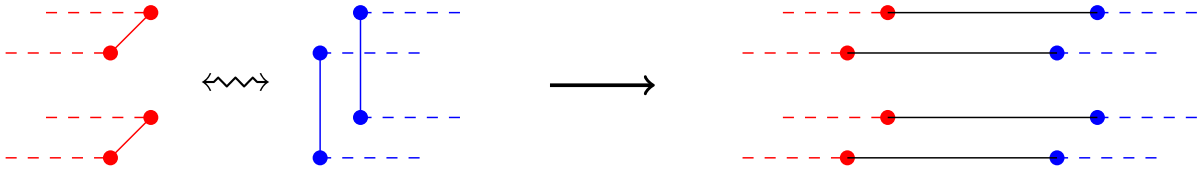


FIGURE 8.14 – An additional association rule (example)

8.3.3 Cycle Stretching

Our third algorithm maintains and extends a set of edges E initialized with the four edges defined by the square of vertices $(0,0,0)$, $(0,1,0)$, $(1,1,0)$ and $(1,0,0)$. At each iteration, the algorithm selects a random edge $e \in E$ and one of the four extension directions shown in Figure 8.15. If such an extension is possible (in other words, by doing so we do not bump into an edge already in E) then E is extended by replacing e by three new edges (one parallel to e and two orthogonal to e in the chosen extension direction). If e cannot be replaced, *i.e.* none of the four extensions is possible, we pick a new $e' \in E$ and try again.

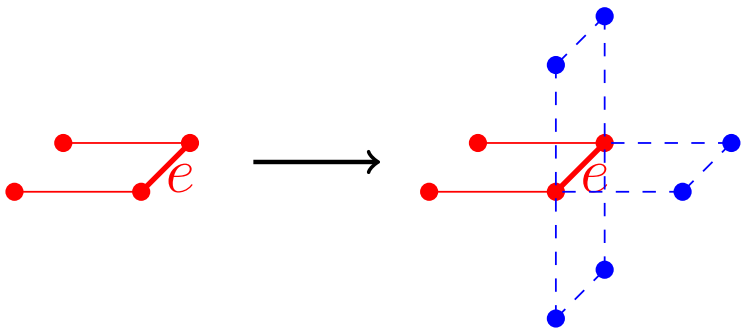


FIGURE 8.15 – Extension options

The algorithm keeps track of a subset of E, denoted B, interpreted as the set of potentially stretchable edges of E. B avoids trying to stretch the same e over and over again.

At each stretching attempt the algorithm picks a random $e \in B$. As the algorithm tries to stretch e, e is removed from B (no matter if the stretching attempt is successful or not). If stretching succeeded, e is also removed from E and three new edges replacing e are added to B and E.

The algorithm halts when $B = \varnothing$. If upon halting $|E| = n^3 - (n \bmod 2)$ then the algorithm succeeds, otherwise the algorithm fails and has to be re-launched. Since at most $3n^2(n-1)$ vertices can be added to B, the algorithm will eventually halt.

A non-optimized implementation running on a typical PC found a solution for $n = 6$ in about a minute and a solution for $n = 8$ in 30 hours. The same code was unable to find a solution for $n = 10$ in three weeks. An empirical human inspection of the obtained cubes shows that the resulting structures seem very irregular. Hence, an interesting strategy consists in generating a core cube of size $n = 8$ by cycle stretching, surrounding it by elementary size 2 cubes and proceeding by random cube association and rewriting.

Algorithm 3 Edge Stretching

let $E =$ the four vertices defined by the square $(0,0,0)$, $(0,1,0)$, $(1,1,0)$, $(1,0,0)$.
let $B = E$.
while $B \neq \varnothing$ **do**
  let $e \in_R B$, we denote the vertices of e by $e = [e_1, e_2]$.
  let $B = B - \{e\}$
  let dir $= \{\leftarrow, \rightarrow, \uparrow, \downarrow, \nearrow, \swarrow\}$
  while dir $\neq \varnothing$ **do**
    let $d \in_R$ dir
    let dir $=$ dir $- \{d\}$
    if d and e are not aligned and stretching is possible **then**
      $E = E - \{e\}$.
      $E = E \cup \{[e_1, v_1], [v_1, v_2], [v_2, e_2]\}$.
      break
    end if
  end while
end while

In the above algorithm the sentence "*stretching is possible*" is formally defined as the fact that no edges in E pass through the two vertices v_1,v_2 such that the segment $[v_1, v_2]$ is parallel to e in direction d. Arrows represent right, left, up, down, front and backwards directions, *i.e.* $\leftarrow$ $\uparrow$ $\nearrow$ $\rightarrow$ $\swarrow$ $\downarrow$

8.3.4 Constraining Existing Hamiltonian Circuitfinding Algorithms

A fourth experimented approach consisted in adapting existing HAMPATH solving strategies. (Dharwadker) [36] presents a polynomial time algorithm for finding Hamiltonian circuits in certain classes of graphs. Assuming that the graphs that we are interested in are in such a class, we tweaked [36]'s C++ code to find Hamiltonian cycles in

cubes. The resulting code succeeded in finding solutions, but these had a too regular appearance and had to be post-processed by re-writing.

We hence constrained the algorithm to work in a randomly chosen subgraph E of the full n^3 cube. We define a *density factor* $\gamma \leq 1$ allowing to control the number of edges in E to which we apply [36]. The ratio of edges in E and n^3 is expected to be approximately γ. Note that because of the heuristic corrective step (9), meant to reduce the odds that certain points remain unreachable, E's density is expected to be slightly higher than γ. The corresponding algorithm is the following :

Algorithm 4 Edges Selection Routine

1: $E = \varnothing$
2: **for** each vertex $v = (x, y, z)$ of the full cube **do**
3: **for** each move $\mathrm{d}v = (\mathrm{d}x, \mathrm{d}y, \mathrm{d}z)$ in $\{(1,0,0), (0,1,0), (0,0,1)\}$ **do**
4: **generate** a random $r \in [0, 1]$
5: **if** $r < \gamma$ and $(0,0,0) \leq v + \mathrm{d}v \leq (n-1, n-1, n-1)$ **then**
6: **add** edge $[v, v + \mathrm{d}v]$ to E
7: **end if**
8: **end for**
9: **if** loop 3 didn't add to E any edge having v as en extremity **then**
10: **goto** line 3
11: **end if**
12: **end for**

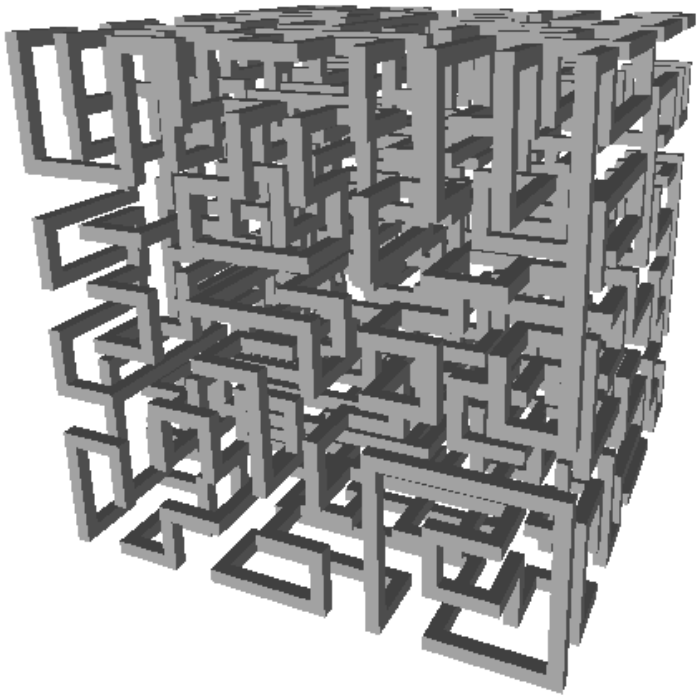


FIGURE 8.16 – A $n = 10$ Hamiltonian cycle obtained by a modified version of Dharwadker's algorithm [36]

Practical experiments show indeed that as γ diminishes, the generated Hamiltonian cycles seem increasingly irregular (for high (*i.e.* $\simeq 1$) γ values the algorithm fills the cube by successive "slices"). Finding solutions becomes computationally harder as γ diminishes, but using a standard PC, it takes about a second to generate an instance for $\{\gamma = 0.8, n = 6\}$ and an hour to generate a $\{\gamma = 0.86, n = 10\}$ one. Figure 8.17 is presenting several experimental results.

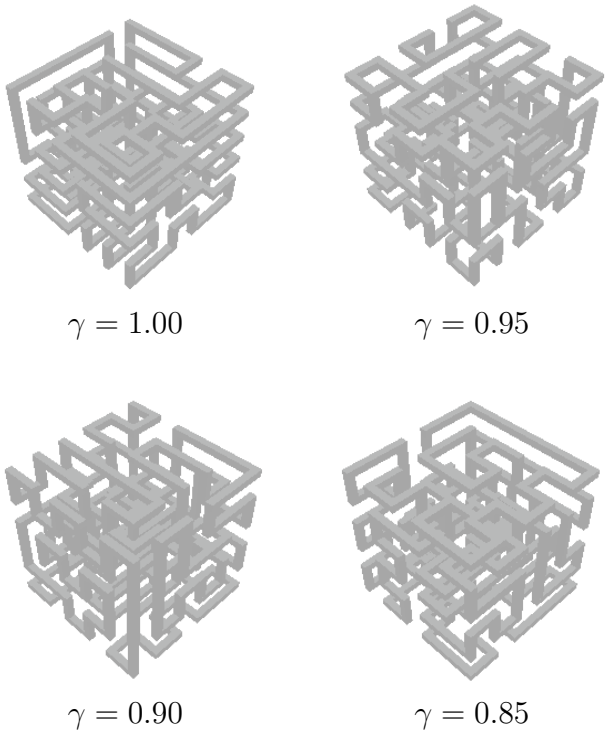


FIGURE 8.17 – Structures obtained for several γ values.

8.3.5 Branch-and-Bound

Another experimented approach was the use of branch-and-bound : Using a recursive function, we can try all different cycles. Given a connected portion of a potential Hamiltonian circuit, this function tries to add all the possible new edges and calls itself recursively. If the function is called with a complete circuit, the job is done.

We added several heuristic improvements to this method :

1. If the set of vertices unlinked by the current circuit is disconnected, it is clear that we won't be able to find any Hamiltonian circuit, and thus we can stop searching.
2. If this set is not connected to the extremities of the current circuit, we can also halt.
3. The existence of an Hamiltonian circuit containing a given sub-circuit only depends on the extremities and on the set of vertices in the circuit. We can hence use a dynamic programming approach to avoid redundant computations.
4. We tried multiple heuristics to chose the order of recursive calls.

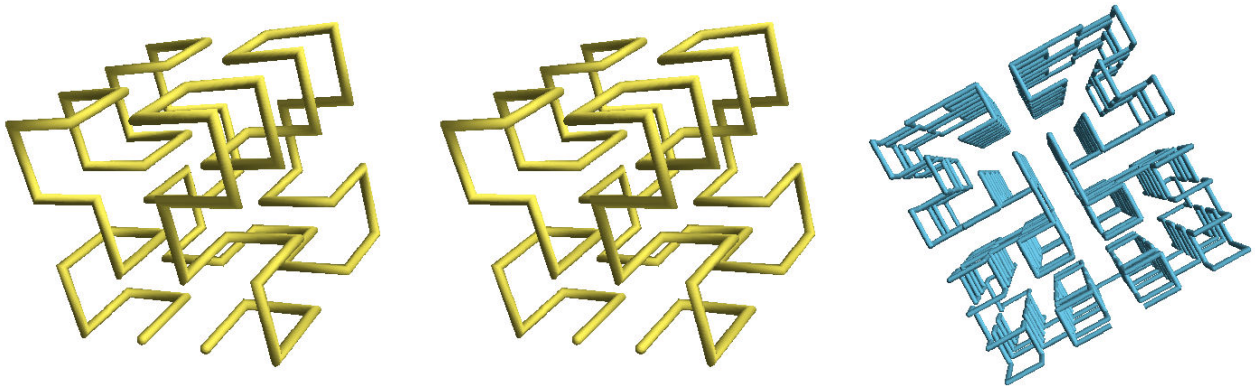
However, those approaches proved much slower than cycle stretching : it appears that

the branch-and-bound algorithm makes decisive choices at the beginning of the circuit without being able to re-consider them quickly. We tried to count all the Hamiltonian cycles when $n = 4$ using this algorithm, but the code proved too slow to complete this task in a reasonable time.

Those results suggest a meta-heuristic approach that would be intermediate between branch-and-bound and stretching : we can make a cycle evolve using meta-heuristics until we obtain an Hamiltonian cycle. Using this method (that we did not implement) we should be able to re-consider any previous choice without restarting the search process.

8.3.6 Rewriting 3D Moore Curves

Finally, one can depart from a know regular 3D cycle (*e.g.* a 3D Moore curve as shown in Figure8.18) and rewrite it. Moore curves are particularly adapted to such a strategy given that the maze entrance and exit are two adjacent edges. However, as shown in Figure8.18c (a top-down view of Figure8.18b), Moore curves are inherently regular and must be re-rewritten to gain randomness.



(a) (b) (c)

FIGURE 8.18 – Example of Moore Curves [37]

8.4 Silicon Experiments

8.4.1 Experimental Pre-Silicon Models

Having obtained several construction plans, we decided to try and construct concrete examples using copper supplies before migrating to silicon. We used an industrial robot to cut 12mm$\varnothing$ copper segments of various sizes. A measurement of the dimensions of off-the-shelf right angle connectors (Figure 8.19) revealed that if a 1-unit segment is h millimeters long, then an i-unit segment has to measure $(h + 16) \times i - 16$ millimeters.

FIGURE 8.19 – Angle connector

Layering and visualizing the prototypes (and chip metal layers) was done using an *ad-hoc* software suite written in C and in Processing [6]. The software allows decomposing a 3D structure into layers and rotating it for inspection.

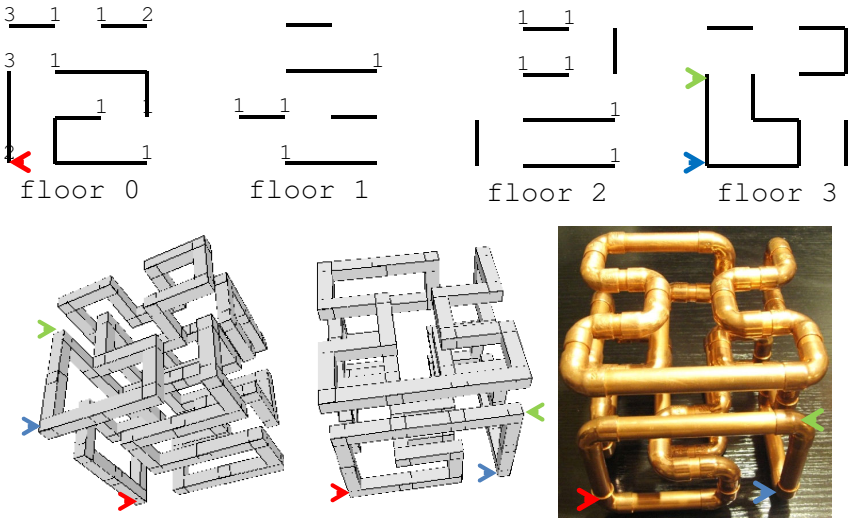


FIGURE 8.20 – Layering, visualizing and constructing the prototypes.

Segments were assembled using several techniques ranging from soldering to super-glue. The disadvantage of welding was the risk of unsoldering an angle connector while

6. http://processing.org/

soldering the nearby one (and this indeed happened at times). Super-glue happened to be less risky but called for dexterity as the glue would harden in a couple of seconds and thereby make any further correction impossible. All in all super-glue was preferred and allowed the generation of a variety of experimental pre-silicon cubes shown in Figure 8.21. 3D printing using stereolitography or thermoplastic extrusion (fused deposition modeling) were considered as well.

FIGURE 8.21 – Experimental pre-silicon cubes

When assembling a 3D cage with glue (or soldering) it is very easy to make mistakes that add-up. A small angular deviation in the assembly of an angle along the x axis will mix with a small angular deviation along the y axis and quickly result in a distorted cage. To avoid this, we assembled structures using a process that consists in slicing the generated structure along the three axes and identifying the longest planar (2D) parts in the target construction. Each planar part is laid on a table and is hence glued according to two axes only (*i.e.* with a lesser degree of freedom). This makes 2D angular errors avoidable (in theory) or at least much smaller (in practice). As the 2D parts are dry and ready, they are glued to each other to form the final cage. As it turns out, this indeed yields much straighter constructions.

8.4.2 Going To Silicon

To test manufacturability in silicon we created a first passive cage meant to protect an 8-bit register. We notice that the compactness of the cage provides a very good reverse-engineering protection.

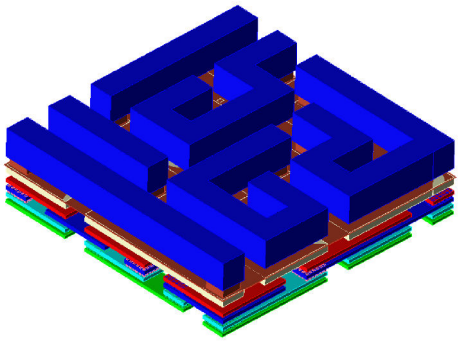
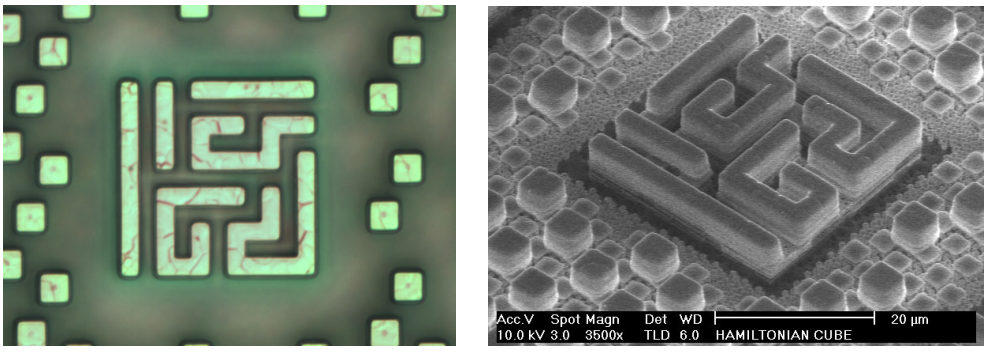


FIGURE 8.22 – 3D layout of a cage of size 6 (130nm, 6 Metal Layers Technology)

The implemented structure (Figure 8.22) is a $6 \times 6 \times 6$ Hamiltonian cube stretching over six metal layers, the first four metal layers are copper ones, and the last two metal layers are thicker and made of aluminum (130nm RF technology, Figure 8.23). The cube is 26μm wide and covers an 8 bit register.

As will be explained in the next Section, this first prototype is not dynamic, the Hamiltonian circuit is not connected to transistors. The implementation of a simplified dynamic structure as described in section 8.5 is underway and does not seem to pose insurmountable technological challenges. Moreover, all layers of the prototype are processed in one side of the silicon, so this implementation does not prevent backside attack. Backside metal deposit and back to back wafer stacking must thus be investigated to thwart backside attacks as well.



(a) (b)

FIGURE 8.23 – Top layer view (a) and tilted SEM view (b) of a 26μm wide $6 \times 6 \times 6$ cage implemented in a 130nm technology ($\times$2500) [7]

This preliminary structure was handmade and the covered area is very samll. As the area to protect on a full cryptoprocessor will be very large, creating a DRC-compliant

7. The structure implemented in silicon is surrounded by fill shapes used as a gaps filler, due to manufacturing constraints (polishing).

shield layout by hand over several metal layers is unpractical. In order to automate this shield generation and integrate it in the design flow, we integrate our algorithm in the analog CAD tools. We've been using Cadence Virtuoso[25] scritpting language SKILL to generate our Hamiltonian structure on several metal layers. Using skill commands we can select the metal layer, define the path of the wire, define the type ofcontacts between layers and set the width of the wires. This is gonna be extremely usefull to custom a shield to an IC need. This is also allowing us to generate the mesh between spaces of the other layers, thus using existing layers for shielding and saving mask cost.

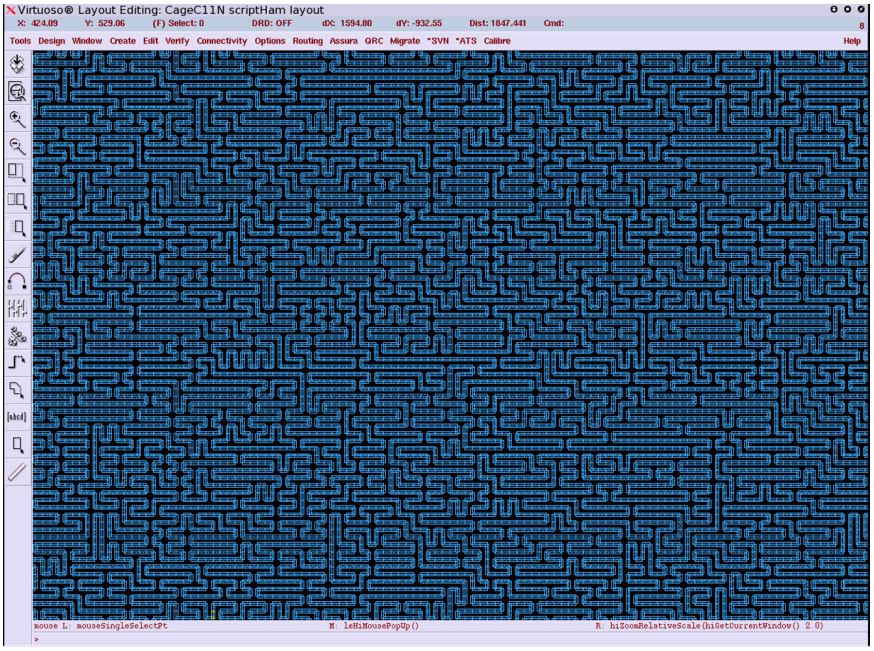


FIGURE 8.24 – Simple Hamiltonian Path Spreading Over Metal 6 (130nm, 6 Metal Layers Technology)

Figure 8.24 is represents a first simple to generate a unique wire over the top metal layers. Instead of building a Hamiltonian cycle, we build an Hamiltonian path to be able to send a signal in the wire, probably it will simply be carrying VDD at first. We then increase the complexity by generating this wire on a much larger surface, and this time spreading over two metal layers (Figure 8.25 and 8.26).

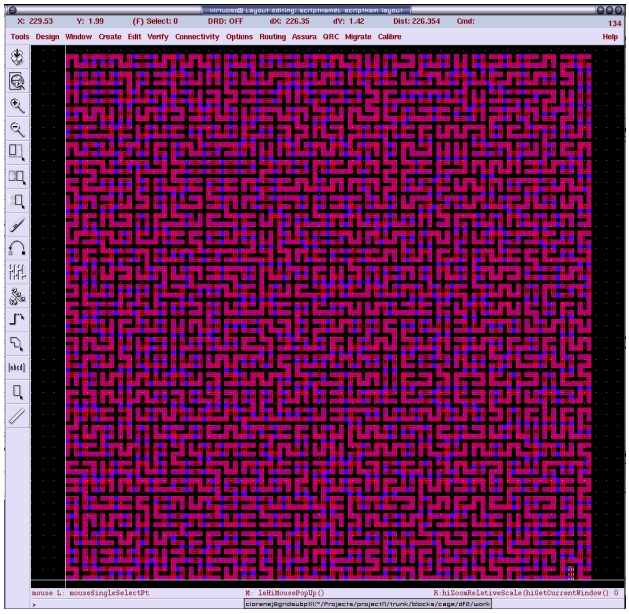


FIGURE 8.25 – Large Hamiltonan Path Spreading Over Metal 5 and Metal 6 (130nm, 6 Metal Layers Technology)

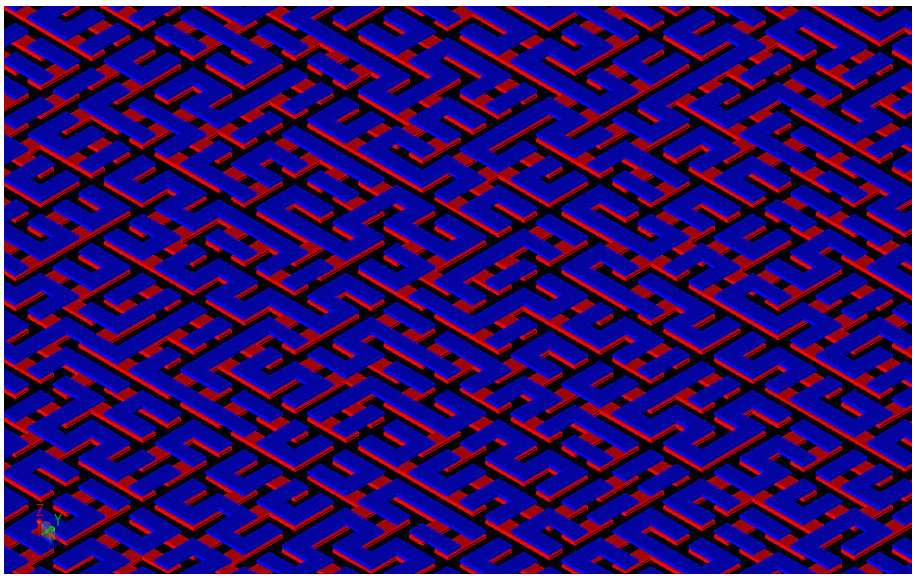


FIGURE 8.26 – Close-up view of The Hamiltonian Path Running Over Metal 5 And Metal 6 (contacts are not displayed for better clarity)

8.5 Dynamically Reconfigurable 3D Hamiltonian Circuits

A canary is a binary constant placed between a buffer and stack data to detect buffer overflows. Upon buffer overflow, the canary gets corrupted and an overflow exception is thrown. The term "canary" is inherited from the historic practice of using canaries in coal mines as toxic gas biological alarms. The dynamic structures presented in this section are hardware equivalents of biologic canaries : our "hardware canary" is formed of a spatially distributed chain of functions F_i positioned at the vertices of a 3D cage surrounding a protected circuit. In essence, a correct answer $(F_n \circ \ldots \circ F_1)(m)$ to a challenge m will attest the canary's integrity. The device described in this section relies on a library of circuits precomputed using the toolbox of algorithms described in the previous section.

8.5.1 Reconfigurable 3D Mazes

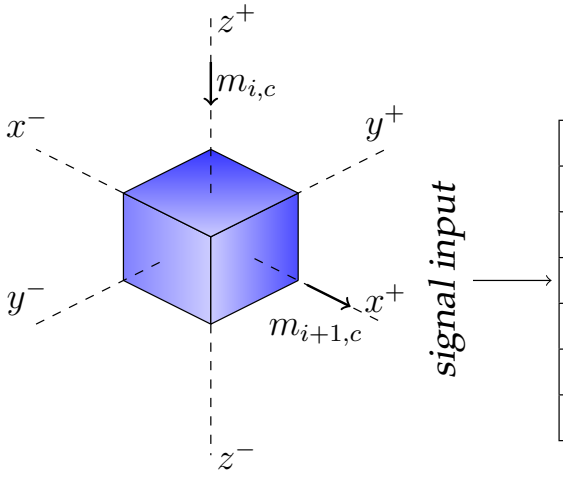
The construction of a 3D dynamic grid begins with the description of a Network On Silicon (NOS) with speed, power and cost constraints [50, 103]. As described in [55, 82], metal wires are shared, or made programmable, by introducing *switch-boxes*, that serve as the skeleton of the dynamic Hamiltonian circuit. Each switch-box is an independent cryptographic cell that corresponds to a vertex of the graph. The switch-boxes are reconfigurable and receive reconfiguration information as messages flow through the Hamiltonian circuit during each session c. All boxes are clocked [8], and able to perform basic cryptographic operations. Six cell-level parameters are used to define each switch-box :

- A *coordinate identifier* i is a positive integer representing the ordinal number of the box's Cartesians coordinates : *i.e.* $i = x + ny + n^2z$.
- A *session identifier* c is an integer representing the box's configuration : this value is incremented at each new reconfiguration session.
- A *key* k_i shared with the protected processor located inside the cage.
- A *routing configuration* $w_{i,c}$ chosen between the thirty possible routing positions of a 3D bi-directional switch (Figure 8.27) [9].
- A *state variable* $s_{i,c}$ computed at each clock cycle from the incoming data $m_{i,c}$ (see hereafter) and the preceding state, $s_{i,c-1}$. The state $s_{i,c}$ is stored in the switch-box's internal memory [10].

8. We denote by t the clock counter.
9. For switch-boxes depicted in red, blue and green (Figure 8.2) the number of possible configurations drops to (respectively) 6, 12 and 20.
10. Upon reset $s_{i,0} = 0$ for all i.

$$\begin{cases} m_{i+1,c} &= F(m_{i,c}, k_i, w_{i,c}, s_{i,c}) \\ s_{i,c+1} &= G(m_{i,c}, k_i, w_{i,c}, s_{i,c}) \end{cases} \tag{8.1}$$

The output data $m_{i+1,c}$ is computed within box i using the input data $m_{i,c}$ and an integrated cryptographic function F, serving as a lightweight MAC. The final output $m_{n^3,c}$ attests the cage's integrity during session c.



signal input →

signal output ↑

w_i ↘	x^+	x^-	y^+	y^-	z^+	z^-
x^+	-	00	01	02	03	04
x^-	10	-	05	06	07	08
y^+	11	15	-	09	0A	0B
y^-	12	16	19	-	0C	0D
z^+	13	17	1A	1C	-	0E
z^-	14	18	1B	1D	1E	-

FIGURE 8.27 – Example of a 3D switch-box programmed with a routing configuration w_i = 0x13

Each switch-box comprises five logic parts (Figure 8.28) that serve to route the integrity attestation signal through the box's six IOs and successively MAC the input values $m_{i,c}$:

- Two multiplexers routing IOs, with three state output buffers to avoid short-circuits during re-configuration.
- A controller commanding the two multiplexers' configuration.
- A MAC cell for processing data and a register for storing results.
- A register storing the state variable $s_{i,c}$, the key k_i, the present configuration $w_{i,c}$, the next box configuration $w_{i,c+1}$ and the clock counter t.

The input message $m_{0,c}$, sent through the Hamiltonian circuit, is composed of two parts serving different goals (Figure 8.29) :

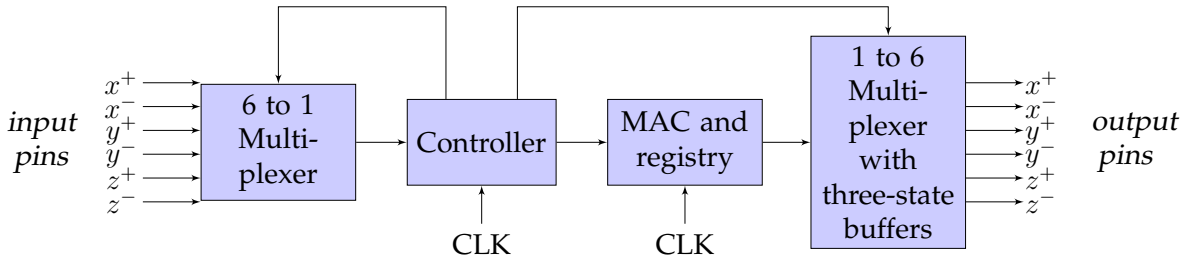


FIGURE 8.28 – Logic diagram of a 3D switch-box

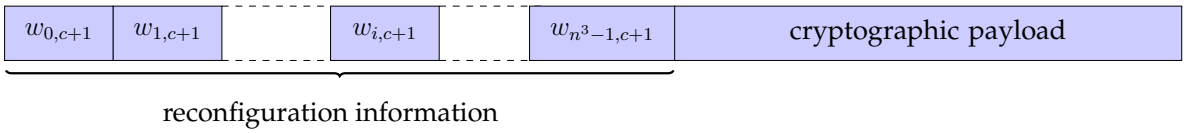


FIGURE 8.29 – Structure of message $m_{0,c}$

- The first message part is dedicated to reconfiguring the grid. For a cube of size n, the reconfiguration information has n^3 parts, each containing the next routing configuration $w_{i,c+1}$ of switch-box i. As the routing information of each switch-box can be coded on 5 bits, the reconfiguration information is initially $5n^3$ bits long [11]. Basically, this message part carries the position of all switches for the next Hamiltonian circuit of session $c + 1$.

- The second message part (cryptographic payload) is used to attest the circuit's integrity, the 64-bits payload will be successively MACed by all switch-boxes and eventually compared to a digest computed by the protected circuit. If possible, one should select a function F that simplifies after being composed with itself to reduce the protected circuit's computational burden.

8.5.2 Description of the Dynamic Grid and the Integrity Verification Scheme

Upon reset, each switch-box is in a default configuration $w_{i,0}$ corresponding to an initial predefined hardwired Hamiltonian circuit for session $c = 0$. The input and the output boxes (S_0 and S_{n^3-1}) are only partially reconfigurable ; namely, the routing of S_0's input and the routing of S_{n^3-1}'s output cannot be changed. To clarify the reconfiguration dynamics, we denote by t the number of clock ticks elapsed since system reset assuming a

11. Note that the reconfiguration information part of the $m_{i,c}$'s gets shorter and shorter as i increases, *i.e.* as the message approaches the last switch-box.

one bit per clock tick throughput ; given that 5 bits are dropped at each "station", a full reconfiguration route (session) claims

$$5\sum_{j=0}^{n^3-1}(n^3-j)=\frac{5}{2}n^3(n^3+1)$$

clock ticks, which is the time needed for the reconfiguration information to flow through all n^3 switch-boxes *i.e.* the number of clock ticks elapsed between the entry of the first bit of $m_{0,c}$ into S_0 and the exit of the last bit of $m_{n^3,c}$ from S_{n^3-1}. Note that this figure does not account for the time necessary for payload transit [12].

At $t=0$ **:** A new session c starts and the first bit of $m_{0,c}$ is received by S_0 form the protected processor.

For $5\sum_{j=0}^{i-1}(n^3-j)=\frac{5}{2}i(2n^3+1-i)\leq t\leq 5\sum_{j=0}^{i}(n^3-j)-1=\frac{5}{2}(i+1)(2n^3-i)-1$ **:** All switch-boxes except S_{i-1} and S_i are inactive (dormant). S_{i-1} sends the message $m_{i-1,c}$ to S_i which performs the following operations :

- Store the reconfiguration information $w_{i,c+1}$, for the next Hamiltonian route of session $c+1$.
- Compute $m_{i+1,c}$ and update $s_{i,c+1}$ as defined in formula (8.1).

At $t=5\sum_{j=0}^{n^3-1}(n^3-j)=\frac{5}{2}n^3(n^3+1)$ **:** The first bit of message $m_{n^3,c}$ emerges from the grid (from S_{n^3-1}) and all switch-boxes re-configure themselves to the new Hamiltonian circuit $c+1$. $m_{n^3,c}$ is received by the protected processor who compares it to a value computed by its own means. At the next clock tick a new message $m_{0,c+1}$ is sent in, and the process starts all over again for a new route representing session number $c+1$.

12. $p(n^3+1)$ where p is the payload size in bits.

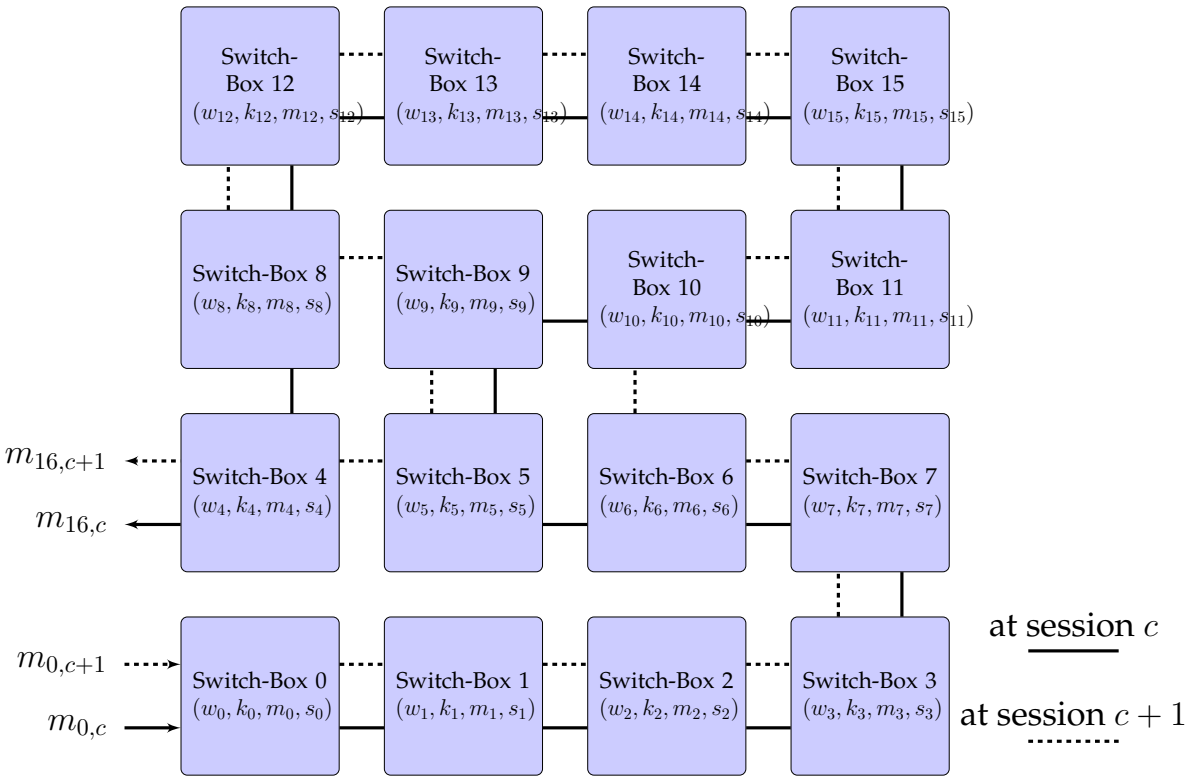


FIGURE 8.30 – 4×4 dynamic switch-box grid routed at c and $c + 1$ (illustration)

If one of the switch-boxes is compromised then the digest output by the circuit will be altered with high probability and the fault will be detected by the mirror verification routine implemented in the protected processor (Figure 8.31). The device could then revert to a safe mode, and sanitize sensitive data.

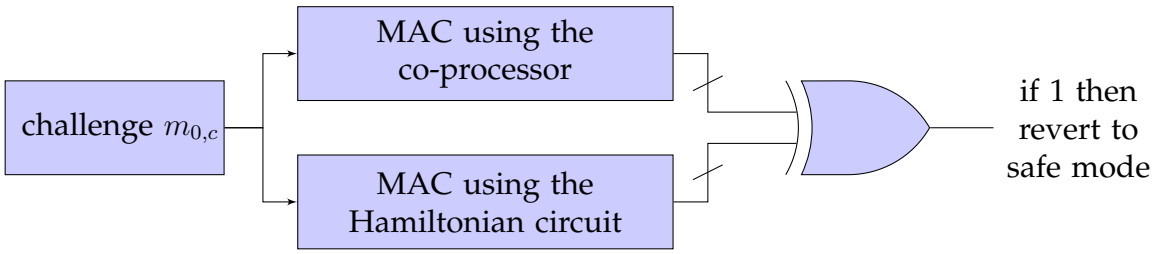


FIGURE 8.31 – Device integrity verification scheme

The verification circuit's size essentially depends on the MAC's size and complexity. Note that the XOR gate is a weak point : if it is bypassed the entire canary becomes pointless. Luckily, the XOR is spatially protected by the Hamiltonian circuit that surrounds it.

8.5.3 Vulnerability to Focused Ion Beam (FIB) Attacks

The proposed dynamic structure complies with the Read-Proof Hardware requirements described in [100] : the structure is easy to evaluate, relatively cheap (in some case no additional masks would be required) and can't be easily removed without damaging the chip.

Even though an attacker might modify some switch-box interconnections using FIB equipment, one cannot bypass a switch-box without modifying the digest computation logic and thus triggering the canary. In theory, an attacker may microprobe the input of the first switch-box to get the reconfiguration circuit, feed it into an FPGA simulating the grid and re-feed the MAC into the target, thus bypassing the canary. The state function s_i implemented in each switch-box should prevent such attacks by keeping state information. Moreover, switch-boxes are defined at transistor level (first metal level) : to microprobe each cell the attacker has to bypass many interconnections, making such an attack very complex. Figure 8.32 describes schematically the dynamic grid concept.

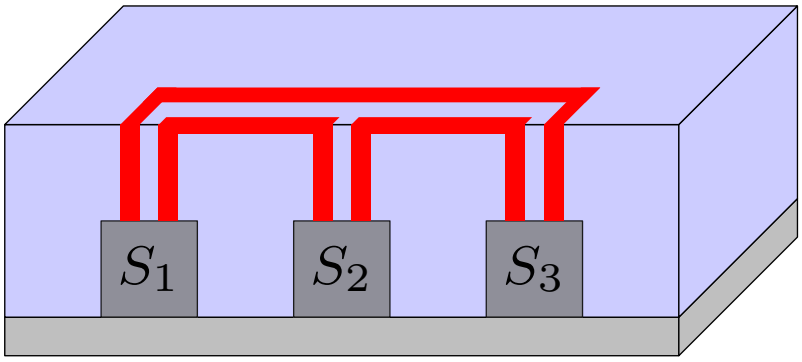


FIGURE 8.32 – Three switch-boxes embedded at substrate level with interconnections over the top layers

The successive grid configurations are precomputed by an external Hamiltonian circuit generator using the strategies described in Section 8.3. This configuration data should be stored in a non-volatile memory located under the cage.

8.5.4 Vulnerability to backside FIB attacks

Despite reported hacks [95], conventional digital shielding provides a good protection against probing and FIB attacks, but is actually inefficient regarding backside attacks. Backside attacks have been known for a long time, they consist in performing optical analysis, probing and injecting faults the back of the circuit (*i.e.* from the silicon substrate end) without altering its function.

Backside access requires a special chip preparation. There are several techniques [33] that can be used to perform controlled backside thinning of silicon circuit without damaging the chip ; it can be thinned up to 10 μm depending on the nature of the FIB that will be used for the edit.

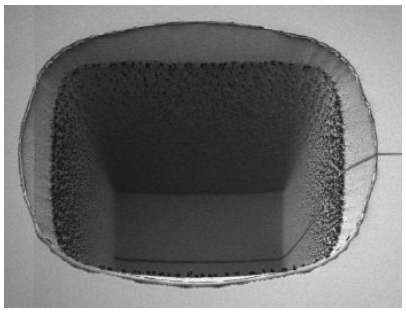


FIGURE 8.33 – A backside trench performed on an thick die[83]

Once the circuit is thinned, the attacker can observe backside photons emission [86], he can perform localised fault injection, eavesdrop signal and even modify the circuitry using FIB. Some recent works have shown that circuit edit is even possible on thick silicon devices [83], thus avoiding the advanced thinning process normally needed in backside attacks (Figure 8.33).

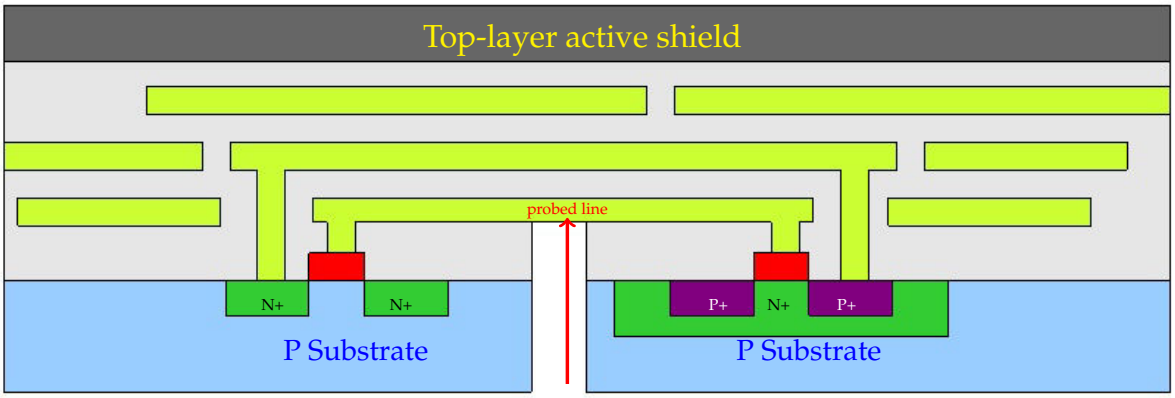


FIGURE 8.34 – Side view of a thinned substrat probed on backside

Figure 8.34 is presenting a piece of circuitry where an attacker has gained physical access to metal lines using backside FIB attack, after thinning the substrate and identifiyng the targeted line, the attacker digs a funnel using FIB to probe the inverter input. The attacker has succesfully bypassed the top layer active shield and has gained physical access to metal lines, where he can read data and inject faults in a fully working circuit.

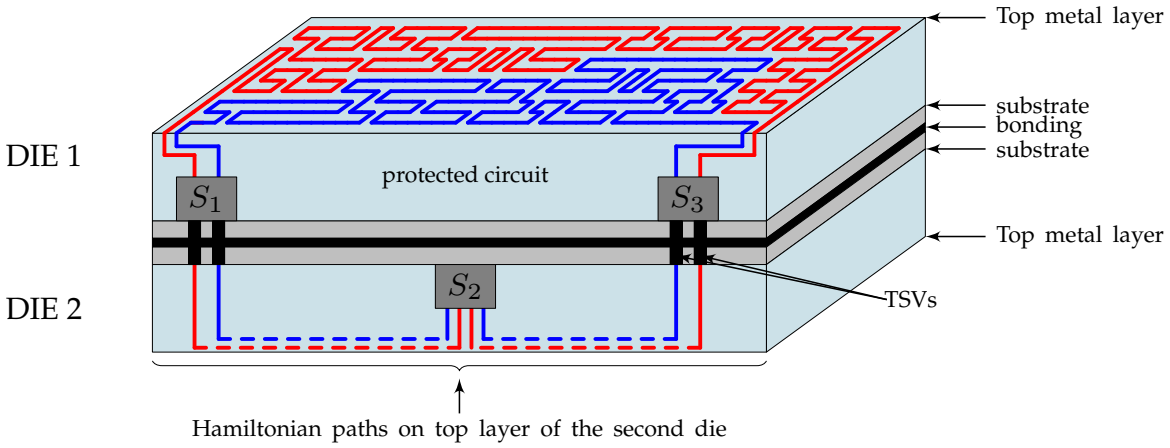


FIGURE 8.35 – 3D active shield spreading over several dies

8.5.5 Improvement thanks to SIP technology

In order to improve active shield's protection, we can extend our dynamic active shield design towards 3D integration. Conventional active shields are inefficient regarding backside attacks, but if our serpentines are spreading over several dies then we can protect a whole chip stack against all kinds of probing attacks.

3d integration technology also offers a very good protection against micro probing ; it is not possible to tear down a chip stack and maintain the overall chip functionning, the glue used to bond dies together and the vias will be irreversibly damaged. As FIB attacks are still possible, we need to consider protecting the chip furthermore by spreading meshes over several layers.

There are several types of 3d technologies and depending on the one we are using, we can create different ways of shielding the chip stack. Figure 8.35 presents a back to back embodiement. Each die's top-layer is used to create a cage around the circuit. We can also think of an embodiment using face-to-back wafer stacking.

8.5.6 Manufacturing constraint

In addition to usual DRC rules, we have to pay a particular attention to the so-called antenna effect [13] as our hamiltonian wires' surfaces are very large. To avoid damage of transistors connected to Hamiltonian pathes during manufacturing, we need to comply with the design rule check known as antenna rule. We will make sure that any CMOS gate is connected to a diffusion before Metal 1 is processed.

13. plasma induced gate oxyde damage

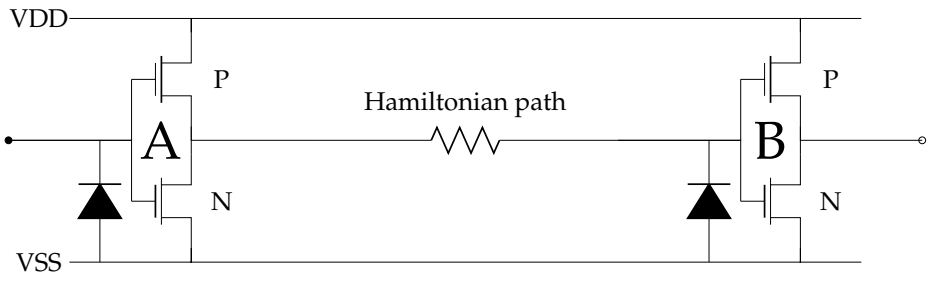


FIGURE 8.36 – Representation of the gates connected to our serpentine

Figure 8.36 represents the gates surrounding our serpentine, we need to protect those gates by connecting the floating grids of inverter A and B before Metal 1 is processed. For that purpose we use NAC diodes(I.E. Net Area Check diodes). NAC diodes are usually picked as small as possible to minimize leakage but depending on the technology and the floating wires surfaces, we may have to increase the diode's value.

It is important to mention that those antenna diodes are to the grid's well. Depending on the 3D technology being used, the exposition of TSVs on back of the wafer during process may cause some antenna effect as well [57].

8.6 Perspectives and Open Problems

Hardware canaries present an advantage with respect to analog integrity protection such as PUFs and sensors : being purely digital, hardware canaries can be integrated at the HDL-level design phase be portable across technologies. The proposed solution would, indeed, increase manufacturing and testing complexity but, being purely digital, would also increase reliability in unstable physical conditions, a common problem encountered when implementing analog sensors and PUFs.

The previous sections raise several sophistication ideas. For instance, instead of having the processor simply pick a reconfiguration route in a pre-stored table, the processor may *also re-write* the chosen route before configuring the canary with it. Devising more rewriting rules and developing lightweight heuristics to efficiently identify where to apply such rules is an interesting research direction.

Another interesting generalization is the interleaving in space of several disjoint Hamiltonian circuits. Interleaved canaries will force the attacker to overcome several spatial barriers. It is always possible to interleave a cube of size $n-1$ in a cube of size n without having the two cubes intersect each other [14] as illustrated in Figure 8.37.

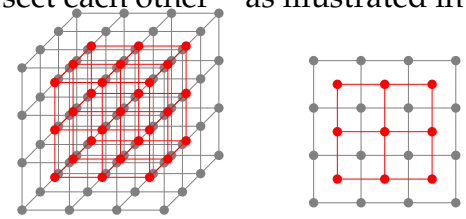


FIGURE 8.37 – A size 4 cube interleaved with a size 3 cube (3D and front view)

Figure 8.38 shows the result of such a (laborious !) physical interleaving for a cube of size 4 and a cube of size 5. Note that interleaving remains compatible with a dynamic evolution of *both cubes* as canaries do not touch each other nor share any hardware (edges or vertices).

FIGURE 8.38 – Interleaving a Hamiltonian cube of size 4 and a Hamiltonian cube of size 5

Finally, functions F for which the evaluation of $F(x) = (F_{n^3-1} \circ \ldots \circ F_0)(x)$ is faster than n^3 individual evaluations of F_i are desirable for efficiency reasons. XOR, bit permutation, addition, multiplication and exponentiation (*e.g.* modulo 251) all fall into this

14. Remove the (k, k, k) point from the center of the odd cube as explained before.

category [15]. Note that $F_i(x) = k_i \times x^{k'_i} \bmod p$ works as well.
In the first dynamic prototype the F_i's will be formed of XORs and bit permutations. Devising computational shortcuts taking into account an evolving internal state $s_{i,c}$ are also desirable.

15. Evidently, input should be nonzero for multiplication, nonzero and $\neq 1$ for exponentiation etc.

9 Cryptographic Shielding

9.1 Introduction

Using the probing attack, the attacker can come to read the data of the circuit, extract the cryptographic key hence breaking the IC security (Figure 9.1). It is because of this problem, that shields were created. It is formed by a mesh of metal lines on the top most metal layer of IC, which can prevent the reading and writing of attacker via probing attack. But with the innovation of attack techniques, the shield can be still attacked. Recently with a technology called Focused Ion Beam (FIB), attackers can draw artificial pads that conduct directly into the inner parts of the circuit and hence spy data of ICs. So we should provide a countermeasure for this kind of attack.

Actually, we can classify the shields into two categories : either passive or active. Passive shielding uses an analogue shield integrity and then use capacitor load as the signature of the shield. In [62], P. Laackmann and H. Taddiken present an analog passive shield. This method is based on an analog transmitter, an analog receiver, a drive and evaluation device. Then it uses a capacitive measurement method to evaluate the shield. But this shield covers partially IC so it is possible for an attacker for attack. Another problem with passive shield is : some variations can be monitored on passive shield. To mitigate this problem, the digital (active) shielding was created. It consists in injecting random sequences of bits in a topmost metal circuit and checking that they arrive unaltered after their journey. For active shields, we can find some propositions in [11] and [49]. But these shields are menaced using FIB technique [32]. Recently, a new active shield structure (called random active shield) has been proposed [23, 30, 24]. This method achieves intricate spaghetti routing of a dense mesh of wires hence making the geometry of the shield difficult to recognize. But it requires two top most metal layers for creation of mesh wires. In a compact IC, it could give a big problem for IC routing. Another problem of this method is that the nature of random numbers is not detailed and it is predictable if these numbers are proceeded by an LFSR. In this research, we present another active shield structure that requires only one metal layer for mesh wires and random numbers is generated by a cipher block. Moreover, we show a me-

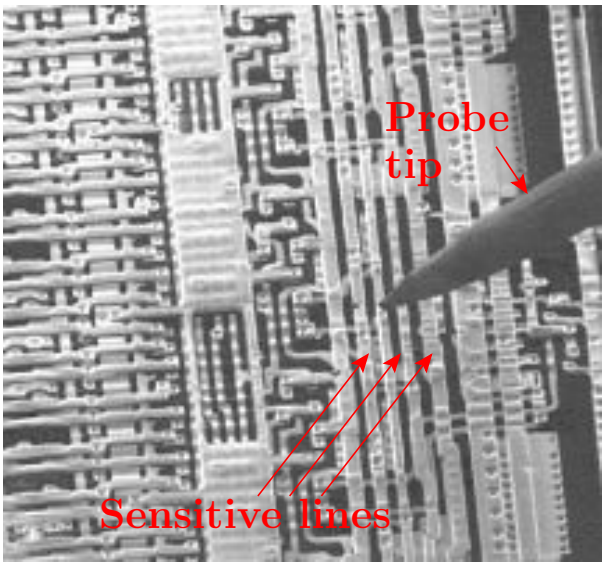


FIGURE 9.1 – Probing attack ([42], Fig. 4.1 of §4.2. at page 31).

thodology for preventing backside attacks with 3D hardware canaries.

9.2 Cryptographically Secure Shield

9.2.1 Rationale

As observed above, the current active shields, presented in Section 9.1, can not assure the IC's security. The attackers can attack these shield thanks to FIB technique. The principle of this attack is to short-circuit the equipotential lines of the shields thus creating unprotected areas on the circuit [23].

To mitigate this problem, our active shield will use on a SIMON block in Cipher Block Chaining (CBC) mode to generate random numbers and independent shield mesh lines. The Figure 9.2 shows our shield's structure (Note that both ALICE and BOB parts are located behind the shield mesh). It is composed of 3 parts :

- ALICE (Transmitter) which has a SIMON block to generate n random bits and is located behind the shield mesh.
- BOB (Receiver) which has a SIMON block and a n bits comparator and is located behind the shield mesh.
- Shield mesh which is composed of n lines on the last metal layer, used as a communication channel between ALICE and BOB, protect the integrated circuit inside.

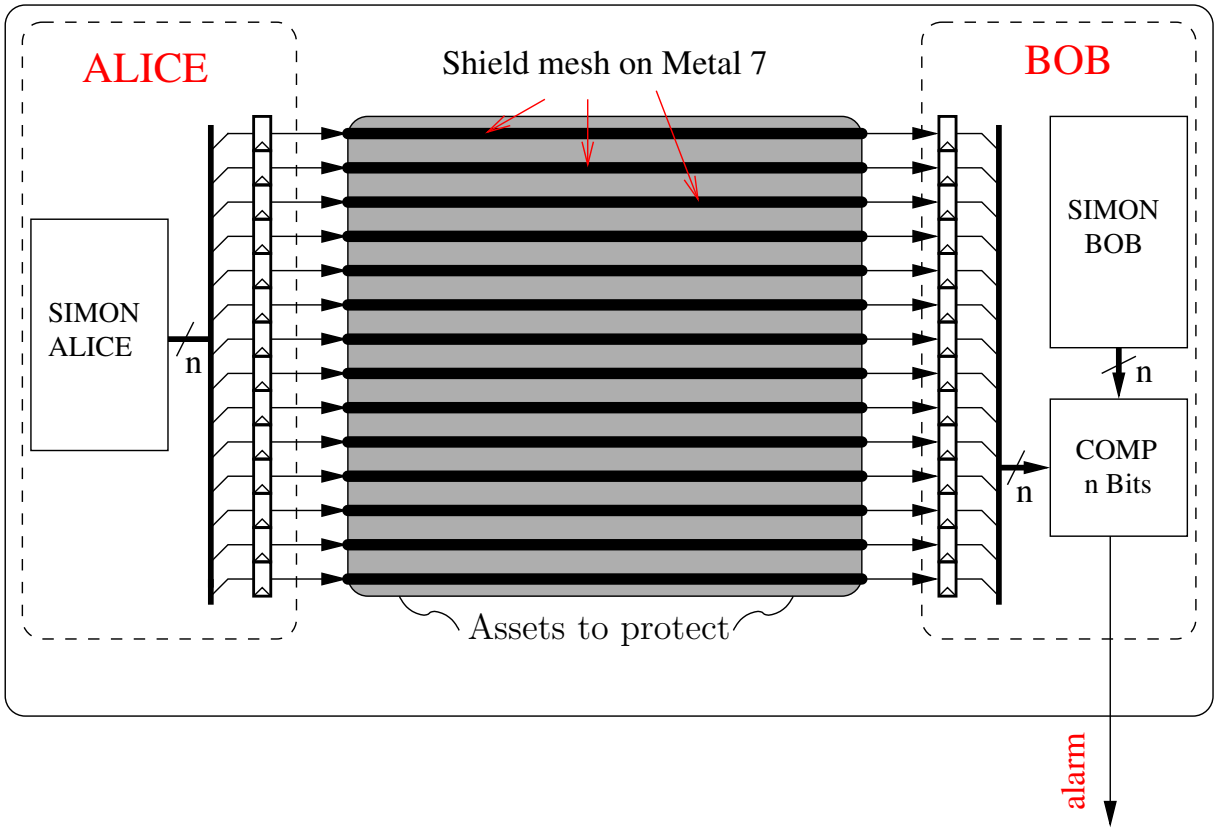


FIGURE 9.2 – Cryptographically secure shield

The idea of our shield is simple. We use 2 SIMON block on CBC mode (one of ALICE and other of BOB). These SIMONs realize the same computation (same key and plaintext on the beginning) to generate n random bits. These random bits are the results of some successive ciphertexts. After, n random bits of ALICE will be sent to BOB via n shield lines. BOB, in its turn, receives these n bits and compare them with n random bits generate by its own SIMON block. If these 2 n random bits are the same so the alarm will be '0'. If these 2 n random bits are not the same so the alarm will be '1' which indicate that there is a problem with the shield lines.

In the Figure 9.2, we use 2 SIMON blocks to well separate ALICE (on the left of the circuit) and BOB (on the right of the circuit). This separation is done to minimize the routing of the active shield on the circuit. It is routed using only the last metal layer (M7 on our case) thus there is no lines that pass through the circuit in the lower metal. It will facilitate the placement and routing of the final circuit because we must place and route all others IPs before insert the shield.

The advantages of this shield are the following :

- The bit sequences exchanged in the shield's lines are the cipher results of SIMON so there is no reuse of the random bits because cipher text change after each encryption computation. If an attacker wants to know these bits, he must attack SIMON to extract the cipher key.
- Each line of the shield presents one bit of cipher message. Theoretically, there are no equipotential lines on shield mesh. So it is very difficult for an attacker to attack the circuit with FIB. We could say that IC's security will be improved with this shield.
- This method is very low cost in entropy. The SIMON block is a lightweight cipher block regarding AES block. With the CBC mode, we need chose only the key and we start with a plain text '0'.

9.2.2 Structure

9.2.2.1 Logic Level

This section describes the logic path of our cryptographic shield. By optimizing, we found another structure of the shield which present the same function of the one on Figure 9.2. Its function is presented in Fig. 9.3. It is composed of three paths :

- Shield interface : composed of a SIMON 128 bits block, a Finite-State Machine (FSM) and one 128 bits comparator.
- ALICE block : composed of buffers.
- BOB block : composed of buffers and one $n \times 128$ bits multiplexer.

Figure 9.3 represents each component in a separated area, whereas in the actual layout, all the logic (including that driving the shield) is located *behind* the shield mesh.

The operating of our shield is the following : At the beginning, we send the a cryptographic key of 128 bits to the shield via an Application Programming Interface (API). When the FSM receives this key. It will send it to the SIMON block and start the cipher computation with the plaintext '0'. When SIMON finishes his computation, its 128 bits cipher text will be sent to BOB block. BOB, in his turn, will connect these 128 bits to the n packet of 128 lines of the shield (for example in our case, the last metal layer of the circuit is filled with 5 packets of 128 lines). These n packets of 128 lines will arrive to ALICE block. ALICE, with a multiplexer n x 128 bits, will choose one packet between n packets (thanks to index sent from FSM block) and send it to SHIELD INTERFACE. SHIELD INTERFACE will compare the packet come from ALICE and the packet sent to BOB via a comparator 128 bits. If theses packets are the same so the result of the comparator is '0' (alarm = '0'). If theses packets are not the same so the result of the comparator will be '1' which indicate that there is a problem with the shield. At the

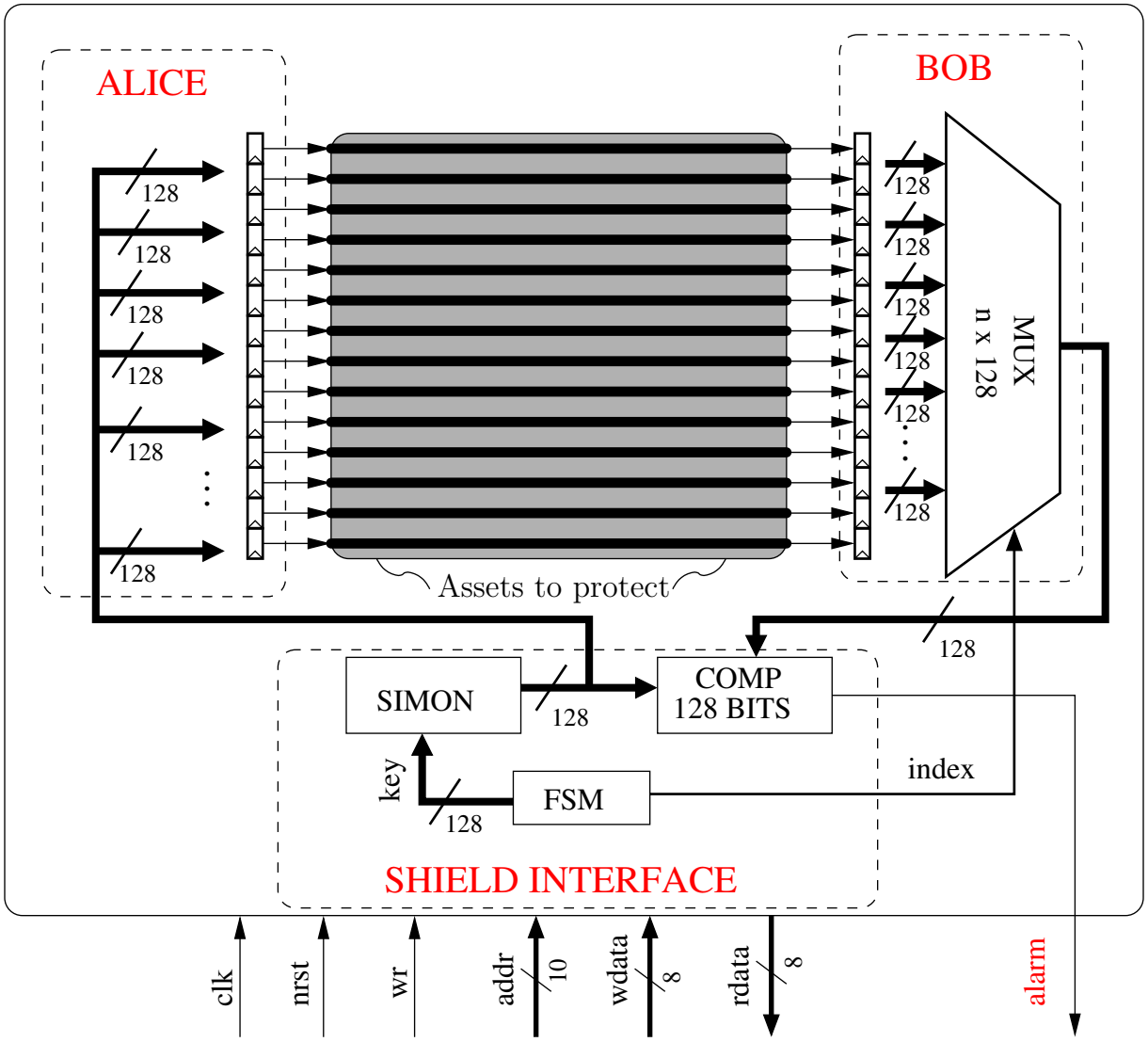


FIGURE 9.3 – Optimized cryptographically secure shield

TABLE 9.1 – API of cryptographically active shield

Instructions	Function description
Set_key	write cryptographic key to SIMON block of the shield
Enable/Disable	enable or disable shield block
Get_status	Read alarm register of the shield
Set_clock	Configure clock frequency of the shield

same time, SIMON continues to calculate using the cipher text as plaint ext. After each computation, we repeat the loop between SHIELD INTERFACE, BOB and ALICE. For each time, "index" will be changed to assure that all lines packets will be sampled.

The active shield, in the Section 9.2.1, has 3 separated parts to minimize its routing on lower metal layers to avoid the problem with the routing of final circuit. But this new shield structure can be routed automatically without problem by using Encounter Cadence scripts. Moreover, this one has 2 advantages over the other :s

- It use only one SIMON block.
- The new one update alarm after each cipher computation while other update alarm after $n/128$ cipher computation.

9.2.3 Connexion to the System

We communicate and control our shield via an Application Programming Interface (API). The table 9.1 presents all abilities of the API for our shield. It allows us to write (or to change) the cipher key of SIMON. We can enable or disable our shield. We can read our alarm register which contains the comparison values of each packet of 128 bits and can help us to locate the modifications on the shield (if it exist). For example, the shield mesh of our circuit has 640 lines which are divided in 5 packets of 128 lines so our alarm register has 5 bits. Reading this register, we can know which packet of 128 lines has a problem (which alarm register bit is "1"). The alarm register verification can be done by hardware or software via API. We can also configure the shield clock to a desired value via this API to reduce its power consumption. This topic will be explained in Section 9.3.3.

9.3 Test chip and Performances

9.3.1 Layout Level

To evaluate our shield, we created an ASIC which is composed of 8 IPs using the technology 65 nm. Circuit size is (1000 μm $\times$ 1000 μm). For the chosen technology, there is 7 metal layers available. The 7th metal layer is used for shield mesh. We use Encounter Cadence for the placement and routing of our ASIC circuit. The placement and routing of the shield is done by scripts.

Ont the layout level, there are 640 topmost metal layers parallel, of minimal width and with minimal spacing, as allowed by the design rules. The Figure 9.4 shows the logic part of our circuit. The image (a) of Figure 9.5 presents the circuit layout of the first cryptographically secure shield structure presented in 9.2. We can find that ALICE part (in yellow on the left) and BOB part (in yellow on the right) are well separated from this structure.

The image (b) of Figure 9.4 shows the circuit layout with the optimized shield structure Section 9.3. ALICE part is on the left. BOB part and Shield interface part are on the right of the circuit.

The logic part of the shield is presented in yellow. We can notice that the area occupied by the optimized shield smaller than the other shield structure. We notice also that shield logic area takes a small area in the circuit for both two layouts. The shield parallel lines are driven by the output of one SIMON [10] operating in CBC mode.

The Figure 9.5 shows the size and the location of each shield part. Image (c) on the Figure 9.5 presents the shield interface block. It is composed as follows :

- SIMON block with the key and message of 128 bits, 32 computation rounds
- Comparator 128 bits.
- wrapper to communicate with UART.

ALICE (the emitter) is made up of

- 128 registers with enable ;
- 5×128 buffers `HS65_LS_BFX18` to drive the lines.

The image (a) on the Figure 9.5 shows the location of ALICE block. We can notice that a vertical line (on the left side of the circuit), is 640 buffers `HS65_LS_BFX18` which are placed manually via scripts.

BOB, image (b) on the Figure 9.5,(the receiver) is made up of :

- 5×128 buffers `HS65_LS_BFX2` ;

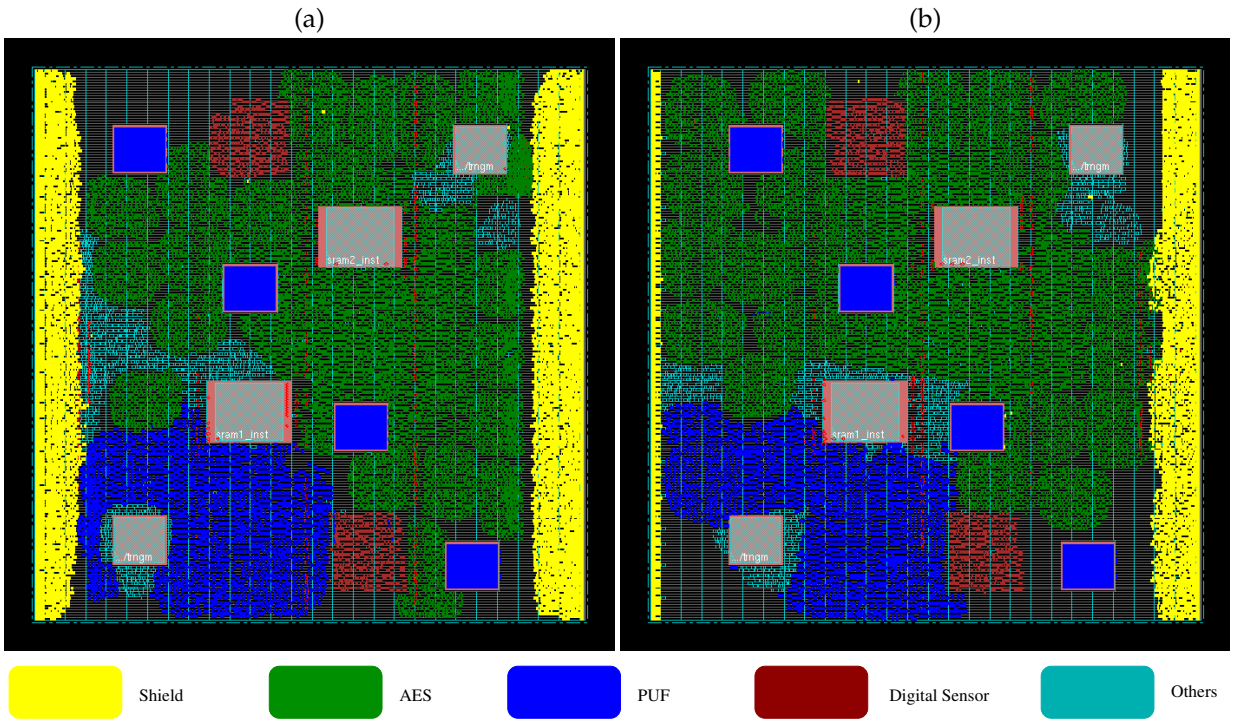


FIGURE 9.4 – Logic part of final circuit (a) and Shield mesh on 7th metal layer (b) (1000 μm $\times$ 1000 μm) with 30% core utilization rate

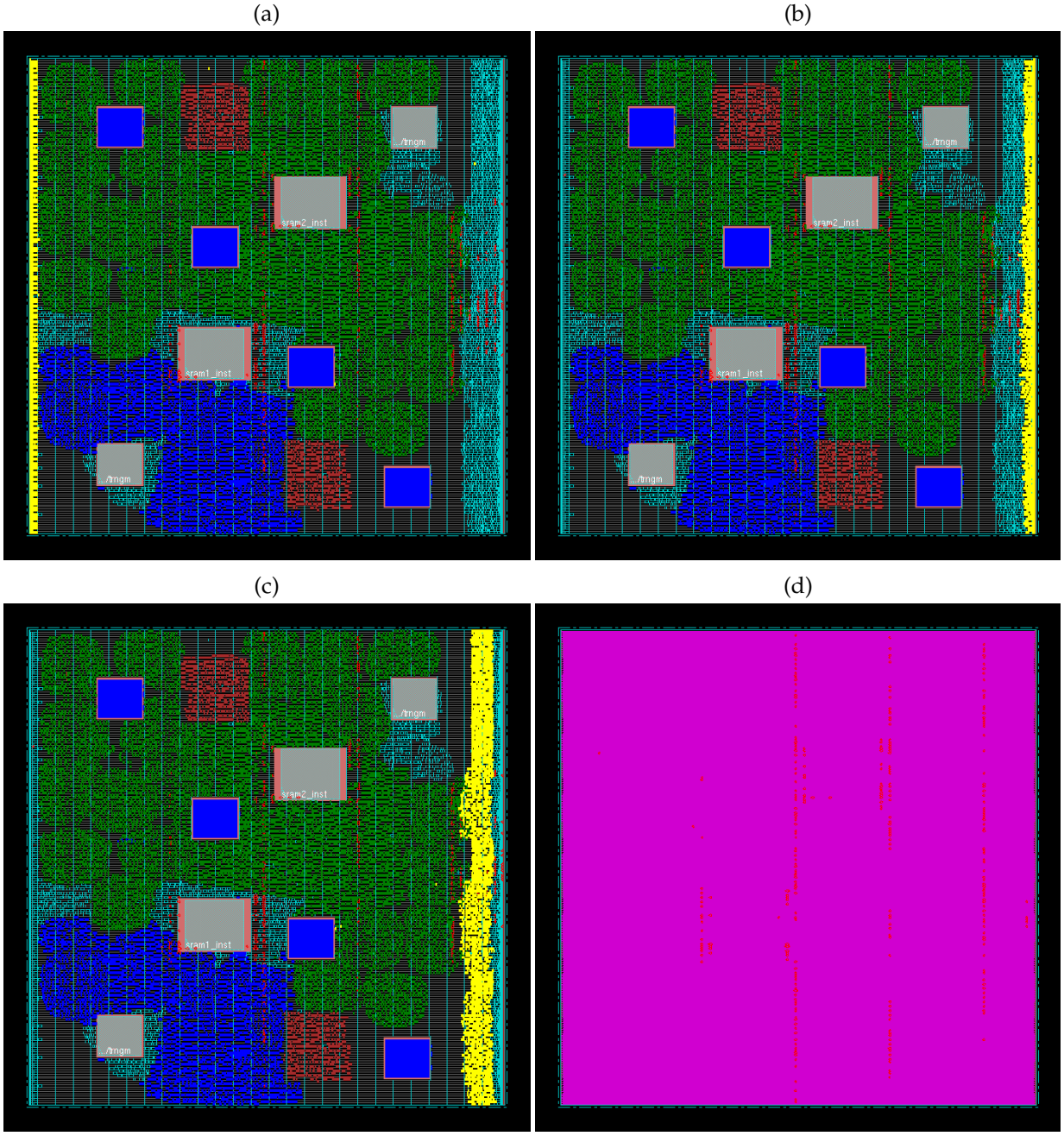


FIGURE 9.5 – Shield logic part in yellow (1000 μm $\times$ 1000 μm) (a) ALICE block, (b) BOB block, (c) Shield Interface block, (d) Shield mesh lines on Top most Metal layer (M7)

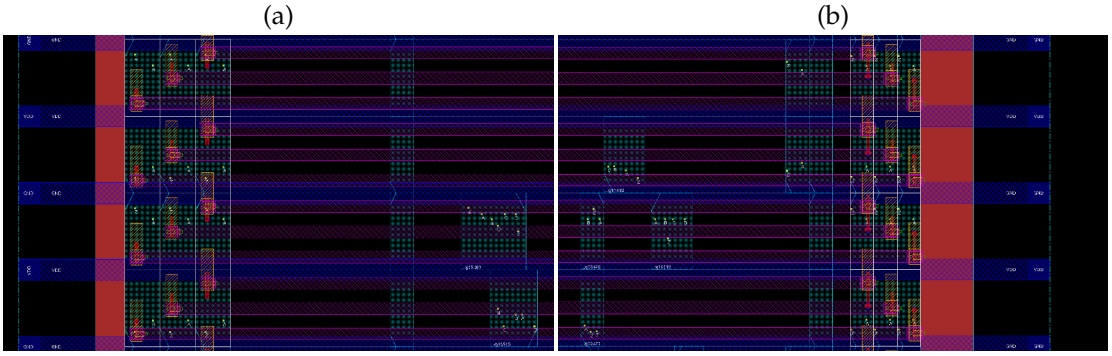


FIGURE 9.6 – ALICE & BOB buffers placement and routing with 30% core utilization rate (a) ALICE Buffers (b) BOB Buffers

- 128 registers with enable ;
- Multiplexers to choose 128 bits amongst the 640 ($5 \rightarrow 1$).

In the image (b) on the Figure 9.5, we observe the same vertical line of 640 buffers. This buffers placement will facilitate the routing of shield mesh. Others logics parts are placed automatically with Encounter.

The image (d) on the Figure 9.5 shows the shield mesh lines (in violet) of our shield. We notice that it fills the circuit area on the top most metal layer (7th layer) and then protect all IP inside the circuit (Shield logic part including)

The Figure 9.6 shows the placement and routing of ALICE buffers (on the left), BOB buffers (on the right), and shield mesh creation. Theses buffers are placed vertically by packet of 3. Then parallel lines (shield mesh) on the topmost layer (M7 in violet color) are drawn from ALICE buffers to BOB buffers with a minimal width (0.4 μm) and with minimal spacing (0.4 μm). All placements and routing are done by scripts. We notice that ALICE buffers are bigger than BOB buffers because we use `HS65_LS_BFX18` for ALICE and `HS65_LS_BFX2` for BOB.

9.3.2 Area

To created the shield, we must sacrifice the topmost metal layer (Metal 7 in our study case). That is the cost for all shield structure. The table 9.2 gives an overview of the size and surface area occupied by some IPs of our ASIC circuit. For shield, ALICE block needs 768 cells, BOB needs 1032 cells and shield interface need 1858 cells. In total, we need 3658 cells to create our cryptographically secure shield. It is still small comparing with other IPs in our circuit (example AES needs 10203 cells). The shield takes less than 10 % of the final circuit (Figure 9.5). Note that the size of our circuit is 1000 μm $\times$ 1000 μm so it is a small circuit. The average density of the circuit is around 30 % With

TABLE 9.2 – Size and area occupied by IPs

Instance	Cells	Cell Area	Net Area	Total Area
shield_inst	3658	18357	14525	32882
shield_interface_inst	1858	10463	7270	17733
simon_inst	1083	5351	3761	9113
dp_i	1051	5177	3469	8645
zgen	21	97	62	159
ctrl_i	32	175	123	297
BOB	1032	4300	3249	7549
ALICE	768	3594	1120	4714
aes_inst	10203	39591	50685	90276
secure_clock_inst	104	356	399	755

TABLE 9.3 – Power consumption of IPs

Instance	Internal Power (mW)	Switching Power (mW)	Leakage Power (mW)	Total Power (mW)	Percentage (%)
shield	0.521	0.07643	0.008701	0.6061	16
shield interface	0.3476	0.05406	0.003984	0.4057	10.71
BOB	0.07959	0.01035	0.001839	0.09177	2.422
ALICE	0.09332	0.009351	0.002863	0.1055	2.785
aes	1.039	0.7332	0.0151	1.787	47.16
puf	0.3368	0.08735	0.005215	0.4293	11.33
others	0.4532	0.50945	0.010984	0.973634	25.51

a bigger circuit and a bigger density, the percentage will be decreased because shield interface is the same for every circuit and only ALICE and BOB blocks will increase (the number of buffers-used for creating shield mesh-and multiplexer).

9.3.3 Power

The Table 9.3 gives an overview of the circuit power consumption using Encounter Cadence. We notice that, for a frequency of 100 MHz, the shield consumption is 16 % of the total consumption.

$P_{\text{total}} = P_{\text{dynamic}} + P_{\text{static}}$

$P_{\text{dynamic}} = p_t \cdot (C_{out} \cdot VDD^2 \cdot F_{clk} + t_{sc} \cdot I_{sc} \cdot VDD)$

$P_{\text{static}} = I_{DDQ} \cdot VDD$

The formulas above show the power computations of ICs. We notice that the clock frequency has an important role in dynamic power consumption of a circuit. For our shield, it does not need to turn at 100 MHz. It must only generate the random bits and realize the check for a convenient time interval. So we can reduce considerably the shield frequency (a few hundred kHz) via our API presented in Section 9.2.3 to reduce its power consumption.

As it could be operated at a very low frequency, there is no critical part in the shield. So we can replace all shield standard cells by a low- power standard cells to reduce the power consumption.

Another possibility is that we can disable the shield all the time and activate it only when making critical computations (for example cryptographic computations) via API Sec. 9.2.3.

For this first shield version, we did not make specific efforts to optimize the shield structure and synthesis. The shield clock tree was made with the same constraints of the protected circuit (100 MHz). With another relaxed constraints, we can reduce the numbers of buffers used for this clock tree. For instance, the comparator on BOB's side is either on 128 or on 5×128 bits, but it could clearly be made smaller by trading silicon area for more time. So with these approaches and optimizations, we can reduce considerably the shield area and consumption.

Conclusion And Perspectives

In this thesis we investigated different classes of countermeasures and protections related to side-channels and invasive attacks, our aim being to propose some new ones.

In the first chapter, we had an overlook on the state of the art of hardware attacks and the countermeasures against them. We saw in particular, that other solutions than the traditional boolean masking exist for protecting integrated circuits. as balancing circuit logics and also by adding random noises. We also presented some researches concerning the protection again invasive attacks.

Thanks to latest fab techniques, these attacks become more and more powerful, accordingly the chip industry needs for continuously improving solutions to protect secure circuits. If conventional shields worked for years, nowadays, in the face of recent sophisticated attacks, they revealed to be insufficiently efficient, thus the need for countermeasures became an imperative requirement. Digital shielding seems to offer an attractive solution as it combines two worlds. One one hand, that of analog anti-tampering robust solutions which can only be overcome through a great amount of efforts and machinery. On the other hand, the digital signatures that prove the integrity of the different nodes of the shield. We introduced a cryptographically active shield architecture. This shield, based on a SIMON cipher lightweight block, insures the hardware security against probing and FIB attacks.

We also demonstrated the implementation of this shield on a ASIC circuit 1000 μm $\times$ 1000 μm. The extra cost in terms of surface is less than 10 % and the extra cost in terms of power consumption could be smaller at a low frequency. Therefore, the implementation cost is acceptable. With a bigger circuit, these costs should be smaller.

In addition to cryptographically securing of the shield, various types of detectors can be integrated in such a network as for example, light detectors, FIB detectors, temperature sensors, a.s.o. Of course, a part of a circuit can be tampered by using FIB without detection, but if the shield is used as a a PUF with a high sensitivity to any modification, then the offered protection is really high. Most of the work done in this work is still open to research since there is not a perfect (ideal) solution for protection of an IC, mainly because any protection depends on the attack models taken into consideration.

Some of the ideas we introduced in this thesis, have not been implemented in hardware. Actually, It would be interesting for applications to implement them and offer afterwards to the best IC hackers out there. The cost of such a solution is rather high the mask cost is not negligible in smaller technology nodes. Moreover, the area and the power used for the shield canary logic and sensors will obviously, impact the overall chip specifications. Nevertheless, if a chip needs security certifications, such protections are required to pass evaluations.

Table of Figures

Appendix

A Using Hamiltonian Totems as Passwords

Physical authentication brings extra security to software authentication by adding real-world input to conventional authentication protocols. Existing solutions such as textual and graphical passwords are subject to brute force and shoulder surfing attacks, while users are reluctant to use biometrics for identification, due to its intrusiveness. This paper uses Hamiltonian tokens as authentication means. The proposed token structure offers many possible configurations (*i.e.*, passwords) and is small enough to be carried on a physical keychain. After presenting our general idea, we describe an efficient algorithm to produce these tokens. Our procedure was validated by running a recognition campaign on a wide batch of synthetic samples, and experimented on prototypes manufactured using a commercial 3D-printer.

A.1 Visual passwords

This paper introduces a new user authentication method well-suited for mobile devices such as smartphones. In some sense, the concept described here can been seen as the illegitimate child between biometric recognition and passwords. From the first, it borrows the pattern matching algorithms that handle data, and from the second, their secrecy. The concept, called *visual passwords*, is also less privacy intrusive than biometrics while keeping most of their characteristics. Biometric systems rely on physical characteristics of an individual to identify him amongst a large population [52]. As physical characteristics are generally considered public and impossible to renew, this can raise some privacy issues as the link between the individual and the application using this biometric must be protected.

Our visual password authentication proposal relies on freely chosen objets. The underlying principle of visual passwords is quite simple :

- At registration, the user chooses *something* as password and takes a photography

of it. In reference to the movie Inception, we call this image a *totem*. The user freely choose something that he has under his hand. Obviously, the choice of the totem has to remain secret. Once chosen, the totem is sent to the authentication service,
- when the user wants to authenticate, he takes another photography of his totem for a comparison image vs image with the reference.

To increase the totem's entropy, totems can be chosen among a great variety of objects [65] or be objects that lend themselves to a sufficient diversity. We decided to take the second option for two reasons : it enables us to develop specialized recognition algorithms for the kind of totems we chose and is that we do not want to get back the passwords drawback with hard to remember totems.

There are other attemps related to our visual passwords scheme. For instance, with graphical passwords [94], the user is asked to select a certain number of images from a set of random pictures. He then must select them among some decoy pictures (see [38] for a detailed analysis of graphical passwords on mobile devices). In [53], the user creates a password by clicking on several arbitrary pixels of an image where some tolerance is accepted to correct his inaccuracy in the selection of pixels. The main difference with our visual passwords scheme is that we rely on an effective image processing algorithm for the verification step, i.e. we want to determine the degree of similarity between the image stocked during the enrollment step and the fresh one taken for the authentication purpose. This enables us to more diversity in the choice of our visual passwords.

The paper is organized as follows : Section 2 describes the Hamiltonian totems which are the main contribution of this article. Section 3 details our recognition algorithms.

A.2 Hamiltonian Totems

A Hamiltonian circuit is a circuit running through all the vertices of a graph. The high entropy provided by Hamiltonian graphs makes them a very suitable totem candidates.

The problem of finding a Hamiltonian circuit in arbitrary graphs (HAMPATH) is known to be NP-complete. Membership in NP is easy to verify (given a candidate solution, the correctness of a solution can be verified in linear time). We refer the reader to [56] for more information on HAMPATH.

Solving the HAMPATH problem is a special case of the famous traveling salesman problem. However, generating Hamiltonian paths for specific structures, e.g. cubes, can bedone efficiently [24, 36].

At the beginning we have thought of creating a Hamiltonian cube but the recognition algorithm did not detect inner layers when plunging inside the cube. We hence decided to limit the Hamiltonian circuit to the cube's surface, namely to its four vertical faces.

Dirac's theorem on Hamiltonian cycles states that an n-vertex graph in which each vertex has degree at least $n/2$, must have a Hamiltonian cycle. In our particular case, the only constraint to obtain a solution to HAMPATH is to choose an even sized cube.

Algorithm 5 Random Hamiltonian Circuit Generator

1: **Input** n size of the cube
2: **Output** Hamiltonian Totem
3: **Let** $Q = Q_1, ..., Q_v$ be the $v = n(n-1)$ squares of size 2 filling the lattice of $4n(n-1)$ points.
4: **while** Card$(Q) \neq 1$ **do**
5: Choose randomly $\{a, b\} \in Q^2$ with $a \neq b$.
6: **if** a and b have at least one couple of neighboring parallel edges **then**
7: Break a randomly chosen couple of parallel neighboring edges, verify that they form a single Hamiltonian circuit and merge $c = a \leftrightsquigarrow b$.
8: **Let** $Q = Q \cup \{c\} - \{a, b\}$
9: **else**
10: **goto** line 4
11: **end if**
12: **end while**

The Algorithm 5 solves HAMPATH for our structure in a very short time by associating elementary Hamiltonian squares mapped on the four vertical faces of the cube. At each step two different Hamiltonian cycles in adjacent graphs and a new Hamiltonian cycle are created. The process is repeated until only one Hamiltonian cycle remains. We implemented this process in C. The code starts by filling the lattice with 2×2 squares, and then associates them randomly. The program ends when only one cycle is left.
The Hamiltonian cycle spreads on the cube's four vertical faces and we place two plates on the upper and lower faces to increase the structure's rigidity. This causes a loss in entropy for recognition but in return, we succeed to create a solid totem structure.
The entropy of a random Hamiltonian circuit generator $\mathcal{G}(n)$ for cubes of surface of size n is difficult to estimate, and is given by the following formula :

$$H(\mathcal{G}(n)) = -\sum_{i=1}^{u_n} p_i \log_2(p_i),$$

where u_n denotes the number of distinct circuits constructible within a cube of size n and p_i is the probability that, when queried, $\mathcal{G}(n)$ will output the circuit number i. However, this definition is of little use since to the best of our knowledge, there are no estimates of u_n in the literature. An efficient security analysis of our solution requires an estimation of the key space, *i.e.*, of the number of possible Hamiltonian circuit configurations.

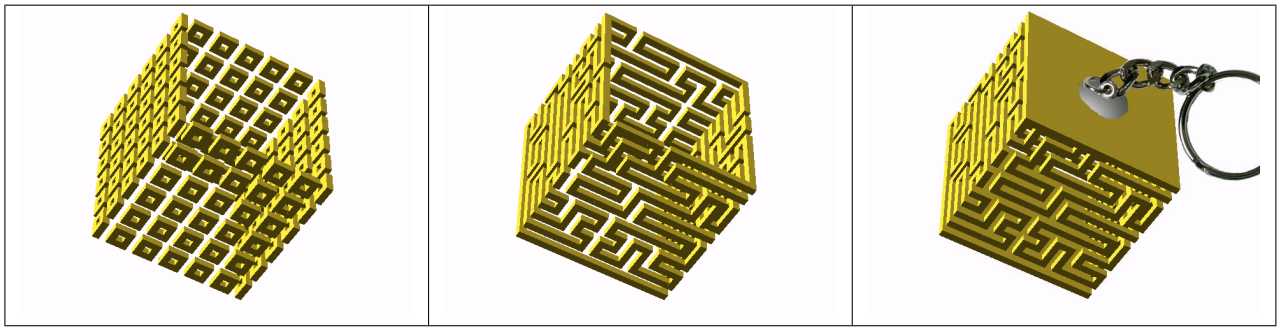


FIGURE A.1 – The three steps of the token generation

Figure A.1 shows the three steps of the token generation, we used OPENSCAD [75] for generating 3D-printable files of our totems.
We first filled the four vertical faces with elementary squares, then we associated them randomly using the square association algorithm [24]. Finally, we added the upper and lower faces of the cube to improve totem rigidity.

A.3 Recognition Algorithm

A generic feature extraction (FE) algorithm could be implemented to encode Hamiltonian cubes. A generic FE algorithm classically relies on extracting characteristics points. These points are usually located on high gradient areas. The nature of the descriptors extracted at these points could vary but they must usually have the following properties : relative invariance to scale, rotation, translation and illumination. The literature is extremely rich on this topic, see for instance, [64, 70, 99]. Points must be discriminant and their relative positions must be compared using classical algorithms that compute the deformation between clouds of points.

Nevertheless, better performances should be obtained by resorting to FE algorithms specific to the problem to be solved. Hamiltonian cubes are objects with multiple constraints that could be leveraged to improve robustness to the extraction process.

The FE problem on Hamiltonian cubes in a video can be decomposed as follow :

- Select of the best representative for each side of the cube and geometry correction.
- Binarize each and every side of the cube
- Error correction considering constraints on the Hamiltonian path

As we are working with a synthetic data (the background is stable, the movement of the cube is regular and controlled), the first FE step is relatively simple. A combination of Hough transform on the gradient map computed by using Sobel filter and a simple

frame binarization using Otsu's algorithm [77] combined with a morphological operation (closing filter to remove noise), is used to determine on which frame each face of the cube should be extracted. Similar tricks such as those used for QR codes (patterns easily detected, specific colours used for the corners and the connection on the edges of the cube) [96] could be used to ease the rectification process in real conditions.

As soon as the face's corner is accurately detected, a homograhy is calculated to rectify the face to its frontal view by using DLT algorithm [45].

A grid is set on the cube to try and determine if each grid element of the grid is filled or not. The result of Otsu's algorithm rectified with the homography is used to vote for each element.

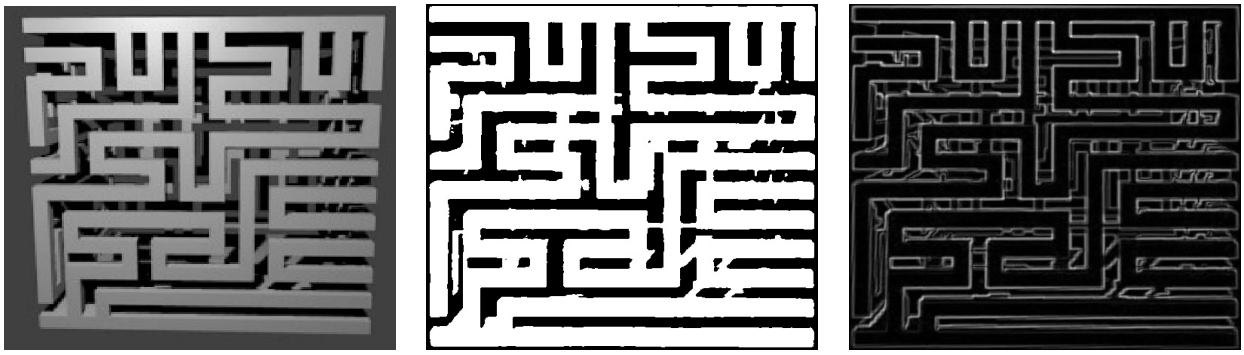


FIGURE A.2 – Selected image. Binarized image after rectification. Gradient image after rectification

The Hamiltonian path that we want to extract is not random. This means that not every configuration is possible. Masks of possible configurations are used to correct wrongly detected elements. In particular, every vertex should be connected to exactly two neighbors. A greedy algorithm is used to vote for edges which are unambiguous and determine which remaining edges are impossible and which ones are still possible. Ambiguous edges are determined at the end of the process when only a few possibilities remain.

More sophisticated algorithms could employed and the choice could be made to balance between object entropy and robustness of the feature extraction process.

As soon as the Hamiltonian paths are extracted on each face of the cube, we have extracted 4 binary vectors that can be compared using Hamming distance. In real conditions, to tolerate occlusion, the chosen comparison function could be the one used classically to compare iris codes [35], taking only into account only good quality areas where the cube was robustly extracted.

A.4 Tests on Synthetic Data

We run an acquisition campaign on 100 samples (Figure A.3), to avoid printing each token, we generated videos of the rotating samples. An algorithm has been developed using the opencv function to evaluate the approach's validity. On synthetic data, the algorithm developed in the previous section leads to perfect extraction performances. No error on the Hamiltonian path extracted was observed on a synthetic database of 100 different samples. This mean that in this specific campaign, a threshold of 0 on the hamming distance could be used to separate perfectly genuine and impostor tests and lead to a False Reject Rate of 0% compared to a False Accept Rate of 0%. However, the difficulty of the extraction process should be evaluated in real conditions to determine

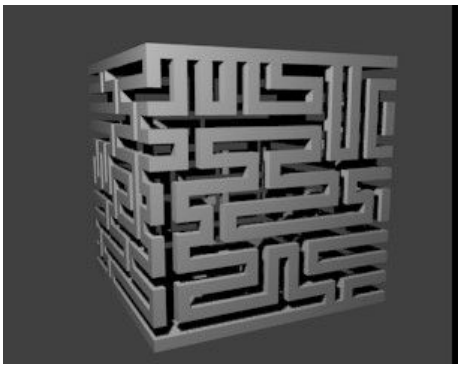


FIGURE A.3 – Snapshot from one the acquisition campaign's video

the real performances and find out how to customize the tokens to improve robustness in difficult real-life acquisitions conditions (*e.g.* uncontrolled lights, non-uniform background, uncontrolled acquistion scenarios).

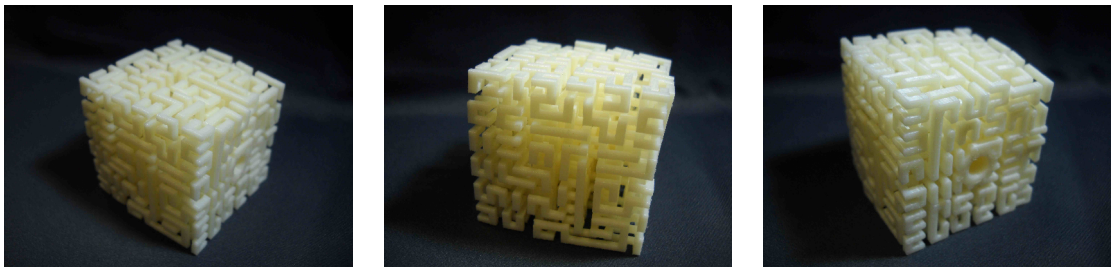


FIGURE A.4 – Some examples of totem prototypes generated by our 3D-printer

A.5 Prototyping

A.6 Further Research

The experiments done so far merely use the cube as a collection of four independent QR-codes. This does not exploit all the potential complexity of 3D structures. The authors are currently working on how to design hamiltonian cubes to be able to extract more information than the surface. This could be done by playing on the transparency, the thickness and the color of the edges as a function of the depth.

Further ideas could be the generation of opaque keys containing a complex 3D form that could be detected using x-rays or other 3D scanning techniques. We can also think to extend our work from Hamiltonian totems to artifical fingerprints [27, 28] (not necesseraly with a realistic fingerprint texture, *i.e.* which corresponds to a real human fingerprint) to exploit the fingerprint sensor technology which comes with new smartphones. Whenever the texture artificially generated (ridges frequency, precense of bifurcations and endings) is sufficiently similar to real fingerprints, then the underlying feature extractor and comparison algorithm will enable us to replace fingerprint authentication by authentication with these new totems. We here assume that we can find a suitable material [9] in order to counter the integrated liveness detection technology to have an image of the totem inside the smartphone of sufficient quality.

B Communicating Covertly through CPU Monitoring

As the number of applications running simultaneously on an operating system is increasing, more and more interest is given to the central processing unit (CPU) monitoring. Since modern computers architectures are build around multi-cores processors, softwares are running on one or several cores depending on their implementation, giving unbalanced work-loads among the CPU cores.
In this section we will be considering CPU work load monitoring as a potential covert channel on multi-core architecture, and we will study how malicious softwares may leak information through CPU usage, and could be use for steganography.

B.1 History

Covert channels, introduced by Lampson [63] are communication channels not intended for information transfer. [63] was the first to point out that varying the input/output computing ratio of a CPU could allow covert information exchange.
Recently Okamura and Oyama presented a CPU load covert channel between Xen virtual machines [74] where clients are connecting to different virtual machines running on the same physical CPU-core. This covert channel exploits the fact client Alice is paused when client Bob is scheduled, allowing Bob to know if Alice is computing something.
In this column we investigate CPU load covert channels between clients running on a *multi-core* machine. We will show that how covert channels using CPU load are also possible between clients connected to a multi-core remote server.

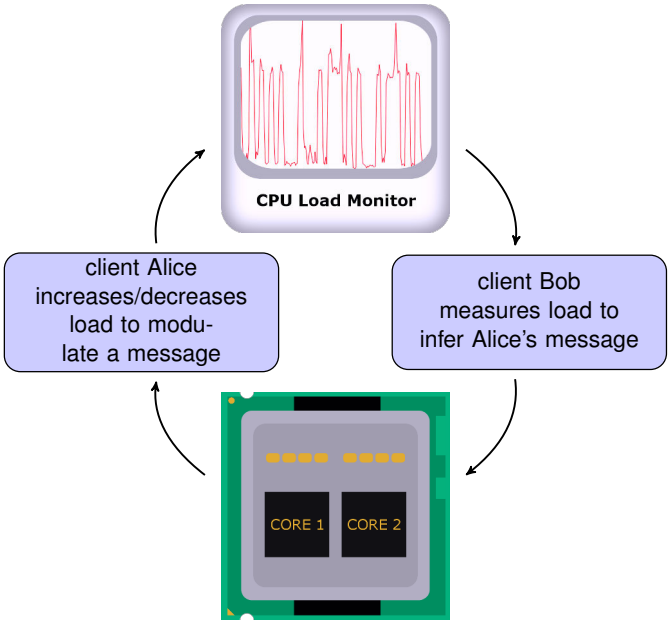


FIGURE B.1 – CPU cores communicating through work load monitoring

B.2 Description

Due to an increasing demand in computing ressources and necessities of managing costs, companies turned to a business model where they centralize or outsource their computing ressources. In the case of a client connected through a virtual machine, the host is transparent. All users are partitionned in their own environment and shouldn't be aware of other clients performing tasks on the same computation grid.
But this principle is questioned in the case of remote connection to a server, where different cores of a same computation grid are shared by different remote users. Users can indeed communicate through CPU usage monitoring. Figure B.1 represents how two CPU-cores can communicate : the first core is running Alice's malicious software that modulates a message through CPU usage, while Bob retreives information by monitoring Alice core's load by running an application on the second core.

B.3 Implementation

To validate this idea we need to implement it on a remote server. To monitor CPU performance in a Linux system we used SYSSTAT packages's SAR shell command [85], it allows any remote user to measure the CPU workload core by core in real time without

administrator privilege. It allows an update rate of one second for CPU usage monitoring. Figure B.2 presents the covert communication channel's general idea, Alice and Bob are connecting to the remote server as clients.

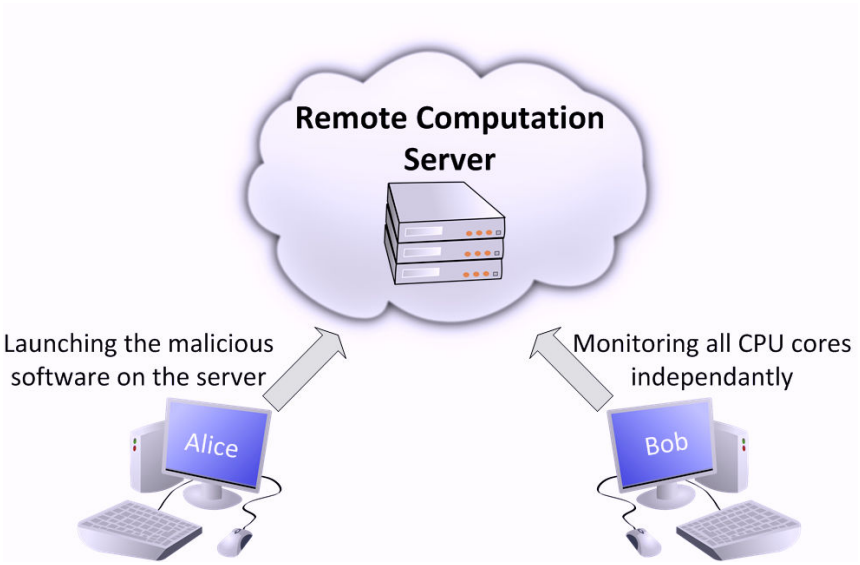


FIGURE B.2 – Client Alice and client Bob covertly communicating via CPU load modulation

Alice runs the malicious software to send a message and Bob monitors CPU load on all cores separately to receive a potential message. We looked over different ways of coding each character ; the conventional way is to code each character on a byte using the ASCII encoding, but at first we chose to use Morse code for illustration purposes, we will the use ASCII encoding for the software implementation.

A	B	C	D	E	F	G	H	I	J	
• —	— • • •	— • — •	— • •	•	• • — •	— — •	• • • •	• •	• — — —	-
N	O	P	Q	R	S	T	U	V	W	
— •	— — —	• — — •	— — • —	• — •	• • •	—	• • —	• • • —	• — —	—

TABLE B.1 – Morse code

Morse code converts characters into as a series of short and long pulses, in our case the CPU usage over a certain period of time. It is based on a succession of shorter and longer events corresponding to specific letters as presented in Table B.1. The malicious program will control the CPU-load to send a message.

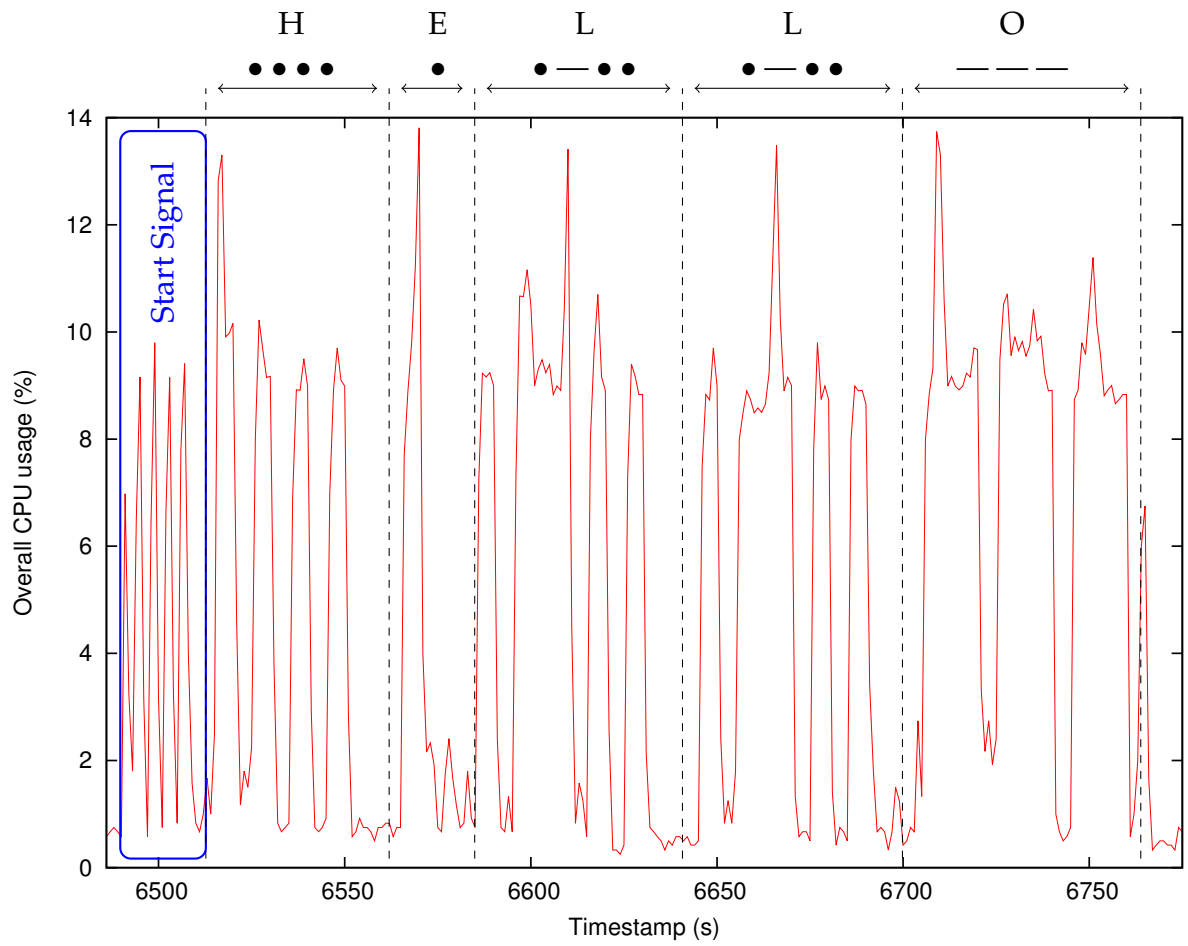


FIGURE B.3 – Overall CPU load monitoring of a 12-core CPU where one core is running a malicious software

To perform the dots and bars of the Morse code, we simply need to run shorter and longer computation tasks. Let p be the duration of a dot in the morse code and $b = 3p$ be the bar's division. The spacing between two elements in the same character is p, between two letter in the same word $3p$ and between two words $7p$.

Figure B.3 shows an implementation of the idea where $p = 4$ seconds. Alice is launching processes that use a single CPU-core to create load patterns. In this case overall CPU load amplitude is low compared to Alice's load, making message transmission possible.
Monitoring the overall CPU consumption has the drawback that the message may not be readable. To counter this problem we can monitor solely the core that runs the malicious software, thus we get a clear signature of the message independently of the other tasks that are taking place in the CPU (Figure B.4).

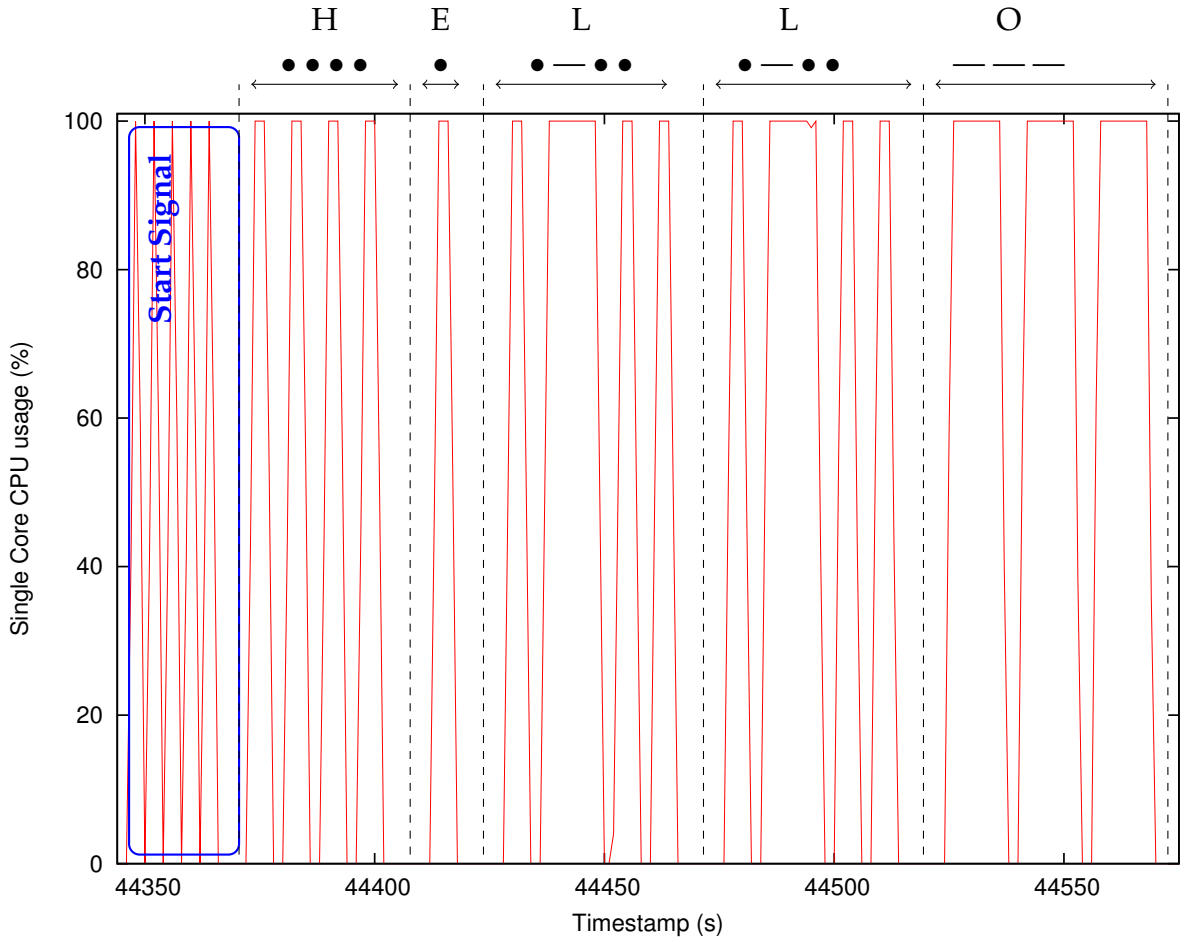


FIGURE B.4 – CPU load monitoring of a single core running a malicious software

The use of ASCII encoding is of course more interesting as it allows to use way more characters than Morse.
The malicious key-logger software is trying to send the user password's "BOB" through the CPU work load in ASCII format. The sequence is initiated by five one-second full CPU load followed by two seconds of pause and terminated by a five one-second full CPU load preceded by two second pause. The bit sequence is composed of zeros corresponding to a two second pause, and ones corresponding to a two seconds of full load, spaced by two seconds silence. The message "BOB" encoded in ASCII gives "01000010 01001111 01000010", it has a length of 24 bits so it will take about a minute to transfer this message (Figure B.5).

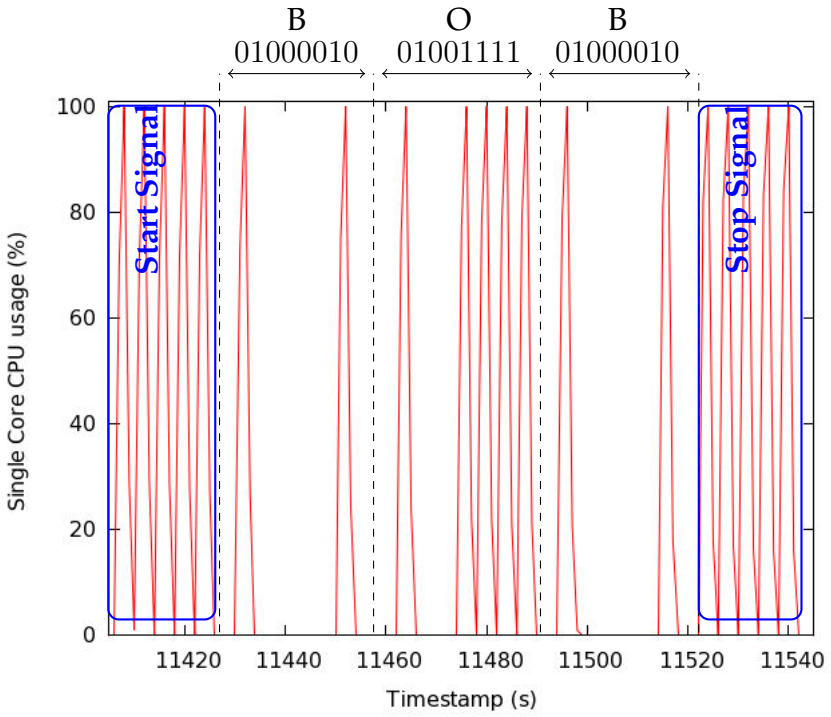


FIGURE B.5 – CPU load monitoring of a single CPU-core running a malicious software

B.4 CPU-load-messenger

The CPU-load-messenger is a Linux software that allows to send and receive a message through CPU load covert-channel (the source code can be found here [69]). It allows a user to send an ASCII message on a multi-core system, the user is able to select the core on which he'll run the covert channel and the system is returning him the time necessary to transmit his message.
The sender and receiver need to agree on the application or the core that is going to carry the message. The communication protocol is composed of three phase.

- Initializing sequence composed of five $2p$ seconds full CPU load spaced by $2p$ seconds load
- Transmission of the message
  "1" bits are transmitted using a p seconds load followed by a p seconds pause
  "0" bits are transmitted using a $2p$ seconds pause
- Terminating sequence composed of a $6p$ seconds load, $2p$ seconds load and $6p$ seconds load spaced by $2p$ seconds load

For this implementation we took $p = 2$ seconds but this parameter is adjustable, Figure B.6 shows the message "`Hello World!`" being transmitted, the overall transmission takes 280 seconds, including start and stop message sequence. An other user is running the CPU-load-messenger but in receiver mode. The only input he has to give in is for how long is willing to monitor the system for a message. The software will monitor all cores of the system and will detect if one is carrying a message.

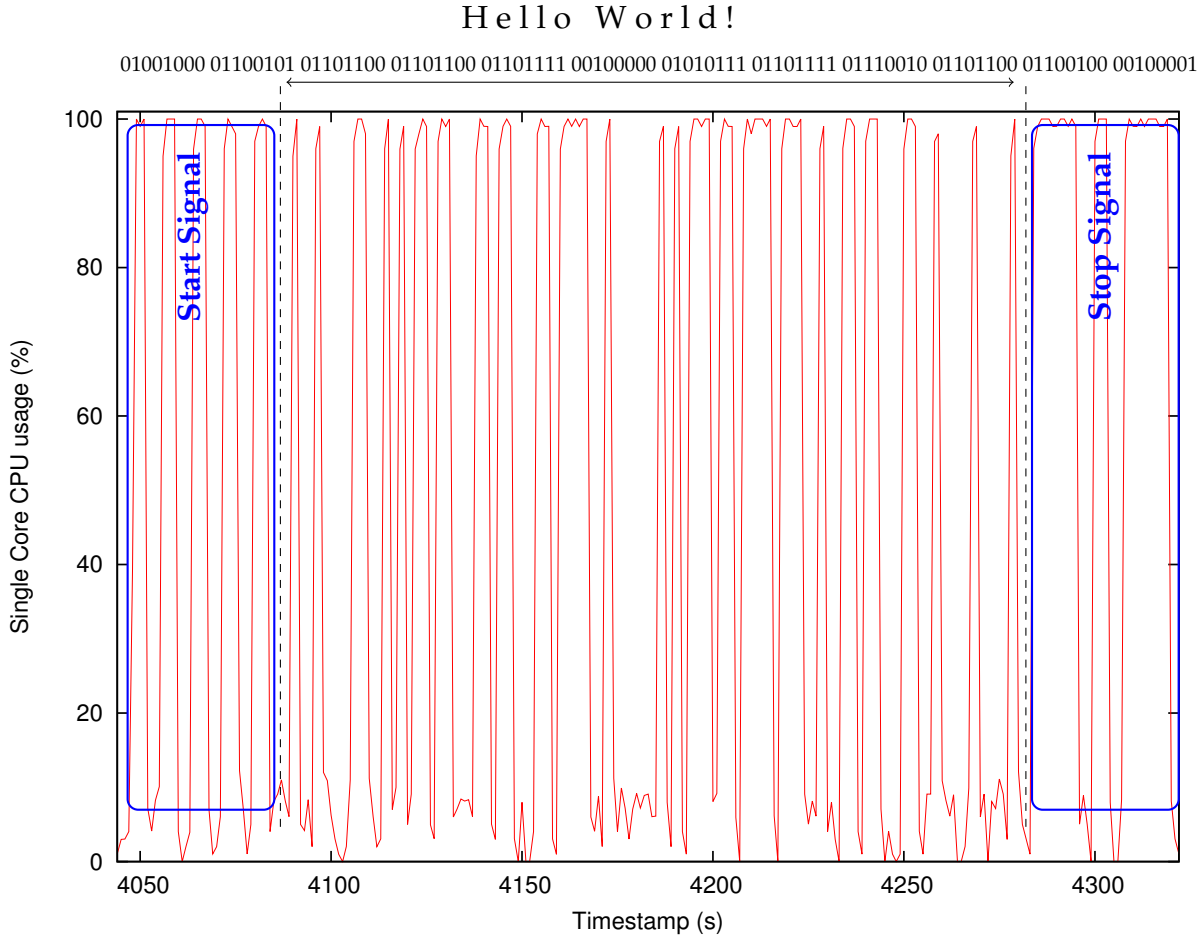


FIGURE B.6 – CPU load monitoring of a single core running a malicious software

B.5 Improvements

The covert channel we discussed in this column is presenting two serious backdraws : Firstly signal to ratio loss to kept under a certain level for the covert channel to be possible. The malicious software needs to have at least 51% of the targeted core available

to send a clear signal. Figure B.7 is representing this limit, the malicious software has over 51% of the CPU-core computing capability to send a message, the message is still extractable from the other task running. During a transmissionany CPU load under 51% is considered as a “0” and over 51% is a “1”.

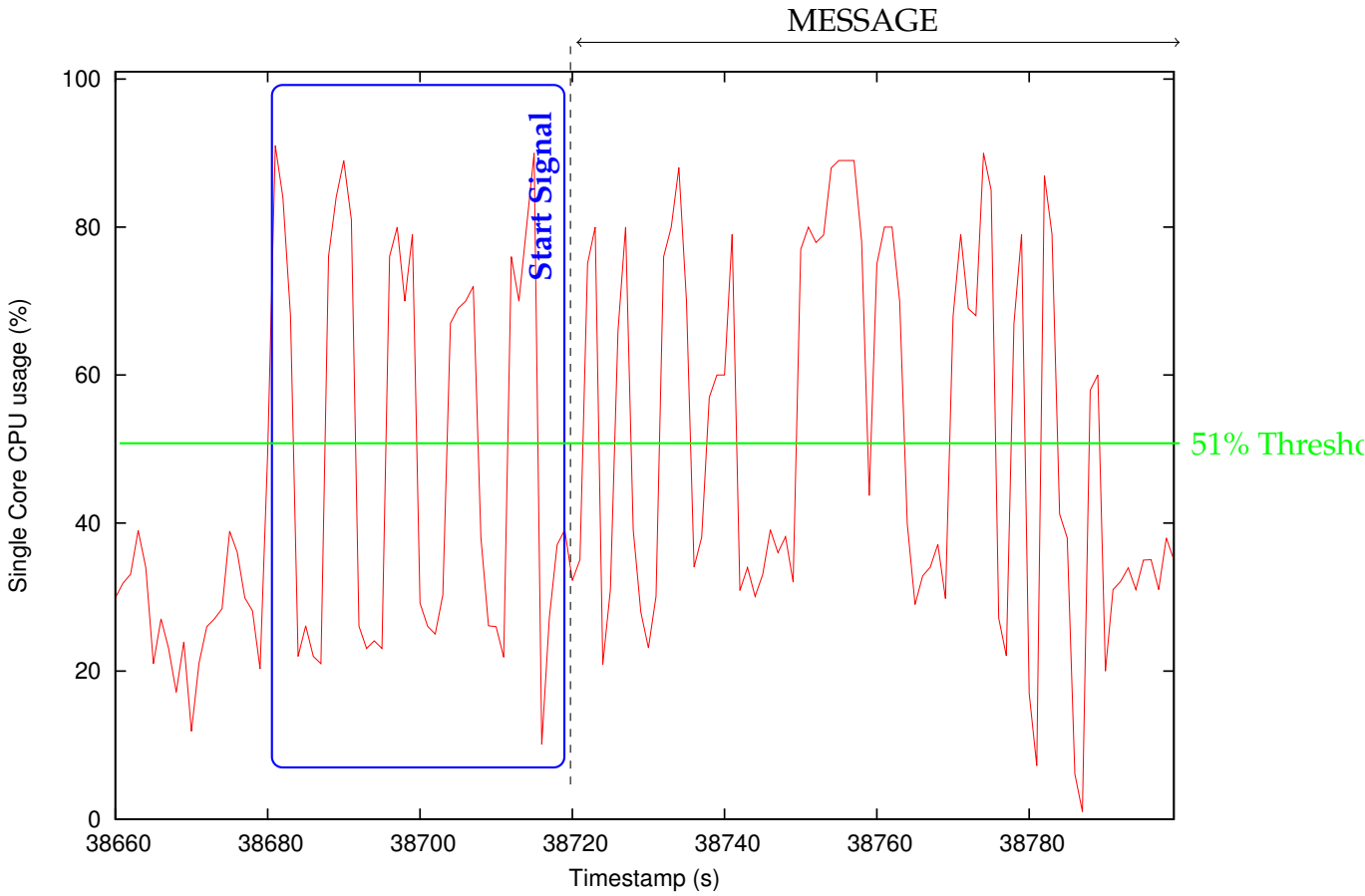


FIGURE B.7 – Malicious software having more than 51% of the overall load

Secondly the throughpu

B.6 Conclusions

In this column we discussed the idea of using CPU load monitoring as a covert channel. In this first approach, we set on purpose a malicious software that leaks information spied on the system. But as hardware evolves, CPUs’ monitors update rates and precision increase, getting closer from a real-time CPU monitoring making this covert

channel more feasible. In the future CPU, usage may become an important source of leakage, giving information of the actual data being computed.

Bibliography

[1] Cristinel Ababei, Yan Feng, Brent Goplen, Hushrav Mogal, Tianpei Zhang, Kia Bazargan, and Sachin S. Sapatnekar. Placement and routing in 3d integrated circuits. *IEEE Design and Test*, 22, 2005.

[2] Michel Agoyan, Jean-Max Dutertre, David Naccache, Bruno Robisson, and Assia Tria. When Clocks Fail : On Critical Paths and Clock Faults. *Lecture notes in computer science*, Volume 6035/2010 :182–193, April 2010.

[3] M. Akkar and C. Giraud. *An Implementation of DES and AES, Secure Against Some Attacks*. In *Lecture Notes in Computer Science*, pages 309–318. Cryptographic Hardware and Embedded Systems (CHES), Springer, 2001.

[4] Michael J. Alexander, James P. Cohoon, Jared L. Colflesh, John Karro, Edward L. Peters, and Gabriel Robins. Placement and Routing for Three-Dimensional FPGAs, 1996.

[5] Altera. Cyclone ll fpga starter development kit.

[6] Ross Anderson and Markus Kuhn. Low cost attacks on tamper resistant devices. In *Security Protocols*, pages 125–136. Springer, 1998.

[7] Christian Aumüller, Peter Bier, Wieland Fischer, Peter Hofreiter, and J-P Seifert. Fault attacks on rsa with crt : Concrete results and practical countermeasures. In *Cryptographic Hardware and Embedded Systems-CHES 2002*, pages 260–275. Springer, 2003.

[8] Hagai Bar-El, Hamid Choukri, David Naccache, Michael Tunstall, and Claire Whelan. The sorcerer's apprentice guide to fault attacks. *Proceedings of the IEEE*, 94(2) :370–382, 2006.

[9] Claude Barral and Assia Tria. Fake fingers in fingerprint recognition : Glycerin supersedes gelatin. In *Formal to Practical Security*, volume 5458 of *Lecture Notes in Computer Science*, pages 57–69. Springer Berlin Heidelberg, 2009.

[10] Ray Beaulieu, Douglas Shors, Jason Smith, Stefan Treatman-Clark, Bryan Weeks,

and Louis Wingers. The simon and speck families of lightweight block ciphers. Cryptology ePrint Archive, Report 2013/404, 2013. http://eprint.iacr.org/.

[11] Andrea Beit-Grogger and Josef Riegebauer. Integrated circuit having an active shield. In *United States Patent number 6,962,294A*, 2005.

[12] 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. In *IEEE Transactions on Computers*, volume 52, April 2003.

[13] G. Bertoni, L. Breveglieri, I. Koren, and V. Piuri. Fault detection in the advanced encryption standard. In *Massively Parallel Computing Systems*, pages pp. 92–97, 2002.

[14] Guido Bertoni, Luca Breveglieri, Israel Koren, Paolo Maistri, and Vincenzo Piuri. On the propagation of faults and their detection in a hardware implementation of the advanced encryption standard. In *ASAP*, pages 303–, 2002.

[15] Ingrid Biehl, Bernd Meyer, and Volker Müller. Differential fault attacks on elliptic curve cryptosystems. In *Advances in CryptologyCRYPTO 2000*, pages 131–146. Springer, 2000.

[16] Eli Biham and Adi Shamir. Differential fault analysis of secret key cryptosystems. In *Advances in CryptologyCRYPTO'97*, pages 513–525. Springer, 1997.

[17] Johannes Blömer and Jean-Pierre Seifert. Fault based cryptanalysis of the advanced encryption standard (aes). In *Financial Cryptography*, pages 162–181. Springer, 2003.

[18] B. Bollobás. *Graph Theory, An Introductory course*. Springer-Verlag, New York, Heidelberg, Berlin, 1 edition, 1979.

[19] D. Boneh, R. DeMillo, and R. Lipton. *On the Importance of Checking Cryptographic Protocols for Faults*. In Springer-Verlag, editor, *Advances in Cryptology : Proceedings of EUROCRYPT'97*, pages 37–51, May 1997.

[20] Dan Boneh, Richard A DeMillo, and Richard J Lipton. On the importance of checking cryptographic protocols for faults. In *Advances in CryptologyEUROCRYPT97*, pages 37–51. Springer, 1997.

[21] 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.

[22] Dan Boneh, RichardA. DeMillo, and RichardJ. Lipton. On the importance of checking cryptographic protocols for faults. In Walter Fumy, editor, *Advances in Cryp-*

tology EUROCRYPT 97, volume 1233 of *Lecture Notes in Computer Science*, pages 37–51. Springer Berlin Heidelberg, 1997.

[23] Sebastien Briais, Jean-Michel Cioranesco, Jean-Luc Danger, Sylvain Guilley, David Naccache, and Thibault Porteboeuf. Random active shield. *2013 Workshop on Fault Diagnosis and Tolerance in Cryptography*, 0 :103–113, 2012.

[24] Sébastien Briais, Stéphane Caron, Jean-Michel Cioranesco, Jean-Luc Danger, Sylvain Guilley, Jacques-Henri Jourdan, Arthur Milchior, David Naccache, and Thibault Porteboeuf. 3d hardware canaries. In *In Cryptographic Hardware and Embedded Systems CHES*, pages 41–57. LNCS Volume 7428, 2012.

[25] Cadence. Virtuoso analog design environment.

[26] G. Canivet. *Analyse des effets d'attaques par fautes et conception sécurisée sur plateforme reconfigurable*. 2009.

[27] Raffaele Cappelli, A Erol, D Maio, and D Maltoni. Synthetic fingerprint-image generation. In *Pattern Recognition, 2000. Proceedings. 15th International Conference on*, volume 3, pages 471–474. IEEE, 2000.

[28] Raffaele Cappelli, Dario Maio, and Davide Maltoni. Synthetic fingerprint-database generation. In *Pattern Recognition, 2002. Proceedings. 16th International Conference on*, volume 3, pages 744–747. IEEE, 2002.

[29] Hamid Choukri and Michael Tunstall. Round reduction using faults. *FDTC*, 5 :13–24, 2005.

[30] Jean-Michel Cioranesco and David Naccache. Protection of an integrated circuit against invasive attacks. In *Patent EP 2624296 A1*, August 7 2013.

[31] Jim Colvin. Functional failure analysis by induced stimulus. In *INTERNATIONAL SYMPOSIUM FOR TESTING AND FAILURE ANALYSIS*, pages 623–630. ASM International ; 1998, 2002.

[32] Christopher Tarnovsky Infineon / ST Mesh Comparison.

[33] E. I. Jr. Cole D. L. Barton and K. Bernhard-Höfer. Flip-chip and backside sample preparation techniques. In *Microelectronics Failure Analysis, Desk Reference, 5th Ed.*, pages 42–48, 2004.

[34] F. Darracq, T. Beauchene, V. Pouget, H. Lapuyade, D. Lewis, P. Fouillat, and A. Touboul. Single-event sensitivity of a single sram cell. *Nuclear Science, IEEE Transactions on*, 49(3) :1486–1490, Jun 2002.

[35] John Daugman. How iris recognition works. *IEEE Transactions on Circuits and Systems for Video Technology*, 14 :21–30, 2002.

[36] A. Dharwadker. *The Hamiltonian Circuit Algorithm*. In *Proceedings of the Institute of Mathematics*, page 32. Springer, 2011.

[37] R. Dickau. *Hilbert and Moore 3D Fractal Curves*. http://demonstrations.wolfram.com/HilbertAndMoore3DFractalCurves.

[38] Paul Dunphy, Andreas P. Heiner, and N. Asokan. A closer look at recognition-based graphical passwords on mobile devices. In *Proceedings of the Sixth Symposium on Usable Privacy and Security*, SOUPS '10, pages 3 :1–3 :12, New York, NY, USA, 2010. ACM.

[39] Common Criteria for Information Technology Security Evaluation (ISO/IEC 15408).

[40] P. Fouillat. *Contribution à l'étude de l'interaction entre un faisceau laser et un milieu semiconducteur : applications à l'étude du latchup et à l'analyse d'états logiques dans les circuits intégrés en technologie CMOS*. 1990.

[41] Benedikt Gierlichs, Lejla Batina, Pim Tuyls, and Bart Preneel. Mutual information analysis. In *Cryptographic Hardware and Embedded Systems–CHES 2008*, pages 426–442. Springer, 2008.

[42] Christophe Giraud. Attaques de cryptosystèmes embarqués et contre-mesures associées, 2007.

[43] Sylvain Guilley, Laurent Sauvage, J-L Danger, Nidhal Selmane, and Renaud Pacalet. Silicon-level solutions to counteract passive and active attacks. In *Fault Diagnosis and Tolerance in Cryptography, 2008. FDTC'08. 5th Workshop on*, pages 3–17. IEEE, 2008.

[44] Donald H Habing. The use of lasers to simulate radiation-induced transients in semiconductor devices and circuits. *Nuclear Science, IEEE Transactions on*, 12(5) :91–100, 1965.

[45] R. I. Hartley and A. Zisserman. *Multiple View Geometry in Computer Vision*. Cambridge University Press, ISBN : 0521540518, second edition, 2004.

[46] Clemens Helfmeier, Dmitry Nedospasov, Christopher Tarnovsky, Jan Starbug Krissler, Christian Boit, and Jean-Pierre Seifert. Breaking and entering through the silicon. In *Proceedings of the 2013 ACM SIGSAC Conference on Computer &#38 ; Communications Security*, CCS '13, pages 733–744, New York, NY, USA, 2013. ACM.

[47] Keld Helsgaun. An effective implementation of the lin-kernighan traveling salesman heuristic. In *European Journal of Operational Research*, pages 106–130, 200.

[48] Texas Instrument. Launchpad msp430 development board.

[49] digital IP INVIA Active Shield IP and analog IP that detects invasive attacks.

[50] I. Verbauwhede J. Kim and M.-C. F. Chang. *Design of an Interconnect Architecture and Signaling Technology for Parallelism in Communication*. IEEE Trans. VLSI Syst. 15(8), 1 edition, 2007.

[51] M Smola J. Otterstedt, M Richter and M. Eisele. *Protection circuit for an integrated citcuit*. Patent No. US 6,496,119 B1, December 2002.

[52] A.K. Jain, P. Flynn, and A.A. Ross. *Handbook of Biometrics*. Springer, 2007.

[53] Nebojsa Jojic, Darko Kirovski, and Paul Roberts. Click passwords, July 10 2007. US Patent 7,243,239.

[54] Marc Joye and Michael Tunstall, editors. *Fault Analysis in Cryptography*. Information Security and Cryptography. Springer, 2012.

[55] A. Peeters K. Goossens, J. van Meerbergen and P. Wielage. *Networks on Silicon : Combinig Best-Effort and Guaranteed Services*. In *Proceedings of Design Automation and Test Conference (DATE)*, pages 423–425. Springer, 2002.

[56] RichardM. Karp. Reducibility among combinatorial problems. In RaymondE. Miller, JamesW. Thatcher, and JeanD. Bohlinger, editors, *Complexity of Computer Computations*, The IBM Research Symposia Series, pages 85–103. Springer US, 1972.

[57] G. Katti, A. Mercha, J. Van Olmen, C. Huyghebaert, A. Jourdain, M. Stucchi, M. Rakowski, I. Debusschere, P. Soussan, W. Dehaene, K. De Meyer, Y. Travaly, E. Beyne, S. Biesemans, and B. Swinnen. 3d stacked ics using cu tsvs and die to wafer hybrid collective bonding. In *Electron Devices Meeting (IEDM), 2009 IEEE International*, pages 1–4, 2009.

[58] P. Kocher. *Timing Attacks on Implementations of Diffie-Hellman, RSA, DSS, and Other Systems*. In *Advances in Cryptology - CRYPTO'96*, volume 1109 of *LNCS*, pages 104–113, 1996.

[59] P. Kocher, J. Jaffe, and B. Jun. *Differential Power Analysis*. In *Advances in Cryptology - CRYPTO'99*, volume 1666 of *LNCS*, pages 388–397, 1999.

[60] Oliver Kömmerling and Markus G. Kuhn. Design principles for tamper-resistant smartcard processors. In *Proceedings of the USENIX Workshop on Smartcard Technology on USENIX Workshop on Smartcard Technology*, WOST'99, pages 2–2, Berkeley, CA, USA, 1999. USENIX Association.

[61] P. Laackmann and H. Taddiken. *Apparatus for protecting an integrated circuit formed in a substrate and method for protecting the circuit against reverse engineering*. Patent No. US 6,798,234 B2, September 2004.

[62] Peter Laackmann and Hans Taddiken. Apparatus for protecting an integrated

circuit formed in a substrate and method for protecting the circuit against reverse engineering. In *United States Patent number 6,798,234*, February 19 2003.

[63] Butler W. Lampson. A note on the confinement problem. *Commun. ACM*, 16(10) :613–615, October 1973.

[64] David G. Lowe. Distinctive image features from scale-invariant keypoints. *Int. J. Comput. Vision*, 60(2) :91–110, November 2004.

[65] madmoizelle.

[66] S. Mangard, E. Oswald, and T. Popp. *Power Analysis Attacks - Revealing the Secrets of Smart Cards*. Springer, 2007.

[67] Timothy C May and Murray H Woods. A new physical mechanism for soft errors in dynamic memories. In *Reliability Physics Symposium, 1978. 16th Annual*, pages 33–40. IEEE, 1978.

[68] A. Menezes, P. van Oorschot, and S. Vanstone. *Handbook of Applied Cryptography*. CRC Press, 1996.

[69] CPU Load Messenger.

[70] Krystian Mikolajczyk and Cordelia Schmid. Scale and affine invariant interest point detectors. *International Journal of Computer Vision*, 60(1) :63–86, 2004.

[71] E. H. Moore. *On Certain Crinkly Curves*. Trans. Amer. Math Soc., 1 edition, 1900.

[72] National Bureau of Standards. *Data Encryption Standard*, January 1977.

[73] National Institute of Standards and Technology (NIST). Announcing the Advanced Encryption Standard (AES), November 2001.

[74] Keisuke Okamura and Yoshihiro Oyama. Load-based covert channels between xen virtual machines. In *Proceedings of the 2010 ACM Symposium on Applied Computing*, SAC '10, pages 173–180, New York, NY, USA, 2010. ACM.

[75] OPENSCAD.

[76] Elisabeth Oswald, Stefan Mangard, Christoph Herbst, and Stefan Tillich. Practical second-order dpa attacks for masked smart card implementations of block ciphers. In David Pointcheval, editor, *Topics in Cryptology CT-RSA 2006*, volume 3860 of *Lecture Notes in Computer Science*, pages 192–207. Springer Berlin Heidelberg, 2006.

[77] Nobuyuki Otsu. *IEEE Transactions on Systems, Man and Cybernetics*, (1) :62–66.

[78] Dan Page and Frederik Vercauteren. A fault attack on pairing-based cryptography. *Computers, IEEE Transactions on*, 55(9) :1075–1080, 2006.

[79] Gilles Piret and Jean-Jacques Quisquater. A differential fault attack technique against spn structures, with application to the aes and khazad. In *Cryptographic Hardware and Embedded Systems-CHES 2003*, pages 77–88. Springer, 2003.

[80] V. Pouget and Université de Bordeaux I. *Simulation expérimentale par impulsions laser ultra-courtes des effets des radiations ionisantes sur les circuits intégrés*. 2000.

[81] K. W. Martin R. Gregorian and G. C. Temes. *Switched-capacitor circuit design*. In *Proceedings of the IEEE*, pages 941–966. Springer, 1983.

[82] E. Rijpkema, K. G. W. Goossens, A. Rdulescu, J. Dielissen, J. van Meerbergen, P. Wielage, and E. Waterlander. Trade offs in the design of a router with both guaranteed and best-effort services for networks on chips. In *Proceedings of Design, Automation and Test Conference in Europe - DATE*, pages 350–355, 2003.

[83] Chad Rue, Steven Herschbein, and Carmelo Scrudato. Backside circuit edit on full-thickness silicon devices. In *Proc. 34th Int. Symp. Test and Failure Analysis - ISTFA*, page 141, 2008.

[84] D. Samyde, S. Skorobogatov, R. Anderson, and J.-J. Quisquater. On a new way to read data from memory. In *Security in Storage Workshop, 2002. Proceedings. First International IEEE*, pages 65–69, Dec 2002.

[85] Linux User Command Manual sar(1).

[86] Alexander Schlösser, Dmitry Nedospasov, Juliane Krämer, Susanna Orlic, and Jean-Pierre Seifert. Simple photonic emission analysis of aes. In *In Cryptographic Hardware and Embedded Systems CHES*, pages 41–57. LNCS Volume 7428, 2012.

[87] B. Schneier. *Applied Cryptography : Protocols, Algorithms, and Source Code in C*. John Wiley & Sons, Inc., 1996.

[88] Nidhal Selmane, Sylvain Guilley, and J-L Danger. Practical setup time violation attacks on aes. In *Dependable Computing Conference, 2008. EDCC 2008. Seventh European*, pages 91–96. IEEE, 2008.

[89] Sergei P Skorobogatov. Semi-invasive attacks-a new approach to hardware security analysis. *Technical report, University of Cambridge, Computer Laboratory*, 2005.

[90] Sergei P Skorobogatov and Ross J Anderson. Optical fault induction attacks. In *Cryptographic Hardware and Embedded Systems-CHES 2002*, pages 2–12. Springer, 2003.

[91] Sergei P Skorobogatov and Ross J Anderson. Optical fault induction attacks. In *Cryptographic Hardware and Embedded Systems-CHES 2002*, pages 2–12. Springer, 2003.

[92] Martin Schobert GNU software DEGATE.

[93] W. Stallings. *Cryptography and Network Security Principles and Practices*. Prentice Hall, 2005.

[94] Xiaoyuan Suo, Ying Zhu, and G. Scott. Owen. Graphical passwords : A survey. *Computer Security Applications Conference, Annual*, 0 :463–472, 2005.

[95] C. Tarnovsky. *Hacking the smartcard chip*. https://www.blackhat.com/html/bh-dc-10/bh-dc-10-archives.html.

[96] ISO/IEC 18004 :2006 Information technology. Qr code 2005 bar code symbology specification.

[97] Christopher Tarnovsky How to Reverse-Engineer a Satellite TV Smart Card.

[98] Randy Torrance and Dick James. The state-of-the-art in ic reverse engineering. In *In Cryptographic Hardware and Embedded Systems CHES*, pages 363–381. LNCS Volume 5747, 2009.

[99] Zhuowen Tu and AlanL. Yuille. Shape matching and recognition using generative models and informative features. In Tomá Pajdla and Jií Matas, editors, *Computer Vision - ECCV 2004*, volume 3023 of *Lecture Notes in Computer Science*, pages 195–209. Springer Berlin Heidelberg, 2004.

[100] Pim Tuyls, Geert jan Schrijen, Boris Skori, Jan Van Geloven, Nynke Verhaegh, and Rob Wolters. Read-proof hardware from protective coatings. In *In Cryptographic Hardware and Embedded Systems CHES*, pages 369–383. Springer, 2006.

[101] Pim Tuyls, Boris Skoric, and Tom Kevenaar. Security with noisy data : Private biometrics, secure key storage and anti-counterfeiting. In *1st Edition, ISBN 978-1-84628-983-5*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2007.

[102] Jonathan Valamehr, Ted Huffmire, Cynthia Irvine, Ryan Kastner, Çetin Kaya Koç, Timothy Levin, and Timothy Sherwood. A qualitative security analysis of a new class of 3-d integrated crypto co-processors. In *Festschrift Jean-Jacques Quisquater, LNCS vol. 6805*, pages 364–382. Springer, 2011.

[103] I. Verbauwhede and M.-C. F. Chang. Reconfigurable interconnect for next generation systems. In *The Fourth IEEE/ACM International Workshop on System-Level Interconnect Prediction - SLIP*, pages 71–74. Proceedings, 2002.

[104] Neil H.E. Weste and David Harris. Cmos vlsi design : A circuits and systems perspective. In *3rd edition*. Addison Wesley, 2004.

[105] Sung-Ming Yen, Sangjae Moon, and Jae-Cheol Ha. Hardware fault attack on rsa with crt revisited. In *Information Security and CryptologyICISC 2002*, pages 374–388. Springer, 2003.

[106] J. F. Ziegler and W. A. Lanford. Effect of cosmic rays on computer memories. *Science*, 206(4420) :776–788, November 1979.