



# Methods for protecting intellectual property of IP cores designers

Brice Colombier

## ► To cite this version:

Brice Colombier. Methods for protecting intellectual property of IP cores designers. Micro and nanotechnologies/Microelectronics. Université de Lyon, 2017. English. NNT : 2017LYSES038 . tel-02109304

**HAL Id: tel-02109304**

**<https://theses.hal.science/tel-02109304>**

Submitted on 24 Apr 2019

**HAL** is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.



N°d'ordre NNT : 2017LYSES038

**THESE de DOCTORAT DE L'UNIVERSITE DE LYON**  
opérée au sein de l'  
**Université Jean Monnet, Saint-Étienne**

**Ecole Doctorale 488**  
**Science Ingénierie Santé**

**Spécialité / discipline de doctorat :**  
Micro-électronique

Soutenue publiquement le 19/10/2017, par :  
**Brice Colombier**

---

**Méthodes pour la protection  
de la propriété intellectuelle des  
concepteurs de composants virtuels**

---

Devant le jury composé de :

Torres, Lionel	Professeur	LIRMM	Président
Güneysu, Tim	Professeur	Université de Brême	Rapporteur
Coussy, Philippe	Professeur	Lab-STICC	Rapporteur
Chotin-Avot, Roselyne	MCF	LIP6	Examinatrice
Le Gal, Bertrand	MCF	IMS	Examineur
Bossuet, Lilian	Professeur	LaHC	Directeur de thèse
Hély, David	MCF	LCIS	Co-encadrant de thèse
Fischer, Viktor	Professeur	LaHC	Invité



A thesis submitted for the degree of Doctor of Philosophy  
from University of Lyon



Doctoral School of Science, Engineering and Health no 488

**Topic : Microelectronics**

---

# Methods for protecting intellectual property of IP cores designers

---

**By : Brice COLOMBIER**

**Under the supervision of** Lilian BOSSUET, Maître de conférences, HDR,  
**and co-supervision of** David HÉLY, Maître de conférences.

<b>Thesis jury:</b>	Lilian BOSSUET	LaHC, France
	Roselyne CHOTIN-AVOT	LIP6, France
	Philippe COUSSY	Lab-STICC, France
	Viktor FISCHER	LaHC, France
	Tim GÜNEYSU	University of Bremen, Germany
	David HÉLY	LCIS, France
	Bertrand LE GAL	IMS, France
	Lionel TORRES	LIRMM, France

**Defense date :** October 19<sup>th</sup>, 2017





Cette thèse a été financée par  
la Région Auvergne-Rhône-Alpes





# Remerciements

*À ajouter dans la version finale du manuscrit*

# Table of contents

## Notations

<b>Introduction</b>	<b>1</b>
<b>1 Threats and protections for design data</b>	<b>13</b>
1.1 Parties involved in the design process and their roles . . . . .	15
1.2 Threats on design data . . . . .	18
1.3 Summary: association between parties and threats . . . . .	20
1.4 Threat models . . . . .	21
1.5 Design data protection methods . . . . .	24
1.6 Summary . . . . .	56
1.7 High-level requirements for a secure remote activation scheme . . . . .	57
1.8 SALWARE IP protection module . . . . .	57
<b>2 Combinational logic locking</b>	<b>59</b>
2.1 Definition . . . . .	61
2.2 Selection of the place of insertion . . . . .	66
2.3 Experimental results . . . . .	71
2.4 Discussion . . . . .	75
2.5 Conclusion . . . . .	81
<b>3 Centrality indicators for efficient and scalable combinational logic masking</b>	<b>83</b>
3.1 Definition . . . . .	85
3.2 A proposal for a masking efficiency evaluation metric . . . . .	85
3.3 Selection of the place of insertion . . . . .	87
3.4 Experimental results . . . . .	96
3.5 Possible improvements . . . . .	103
3.6 A priori evaluation of the masking potential . . . . .	105
3.7 Attacks aiming at recovering the activation word . . . . .	105
3.8 Conclusion . . . . .	106
<b>4 Key reconciliation protocols for error correction of silicon PUF responses</b>	<b>109</b>

4.1	Similarities between reconciliation in quantum key distribution and reliable shared key generation from a PUF response . . . . .	111
4.2	Error correction based on multiple parity checks and binary searches . . . . .	112
4.3	BINARY protocol . . . . .	113
4.4	CASCADE protocol . . . . .	116
4.5	Parameters of the CASCADE protocol . . . . .	117
4.6	Implementation . . . . .	122
4.7	Experimental results . . . . .	124
4.8	Security: attacks and countermeasures . . . . .	132
4.9	Discussion . . . . .	134
4.10	Conclusion . . . . .	135
<b>5</b>	<b>Complete hardware/software infrastructure IP for design protection</b>	<b>137</b>
5.1	Integration into EDA tools . . . . .	139
5.2	Hardware platform: HECTOR board . . . . .	146
5.3	Overall hardware implementation results . . . . .	147
5.4	Software interface . . . . .	149
5.5	Illustrative example . . . . .	151
5.6	Use case . . . . .	152
5.7	Conclusion . . . . .	153
	<b>Conclusion</b>	<b>155</b>
	<b>Publications and communications</b>	<b>163</b>
	<b>Bibliography</b>	<b>167</b>
	<b>Appendices</b>	<b>183</b>
	<b>Examples of graphs found in graph analysis for combinational logic locking</b>	<b>183</b>
	<b>List of Figures</b>	<b>187</b>
	<b>List of Tables</b>	<b>191</b>

# Sommaire

## Notations

<b>Introduction</b>	<b>7</b>
<b>1 Menaces sur les données de conception et méthodes de protection</b>	<b>13</b>
1.1 Acteurs impliqués dans le processus de conception et leur rôle . . . . .	15
1.2 Menaces sur les données de conception . . . . .	18
1.3 Résumé : liens entre acteurs et menaces . . . . .	20
1.4 Modèles de menace . . . . .	21
1.5 Méthodes de protection des données de conception . . . . .	24
1.6 Résumé . . . . .	56
1.7 Caractéristiques requises pour un système d’activation à distance sécurisé . .	57
1.8 Module de protection de la propriété intellectuelle SALWARE . . . . .	57
<b>2 Verrouillage de la logique combinatoire</b>	<b>59</b>
2.1 Définition . . . . .	61
2.2 Sélection du lieu d’insertion . . . . .	66
2.3 Résultats expérimentaux . . . . .	71
2.4 Discussion . . . . .	75
2.5 Conclusion . . . . .	81
<b>3 Indicateurs de centralité pour le masquage logique combinatoire efficace et adaptable</b>	<b>83</b>
3.1 Définition . . . . .	85
3.2 Proposition pour une métrique d’évaluation de l’efficacité du masquage . . .	85
3.3 Sélection du lieu d’insertion . . . . .	87
3.4 Résultats expérimentaux . . . . .	96
3.5 Améliorations possibles . . . . .	103
3.6 Évaluation a priori du potentiel de masquage . . . . .	105
3.7 Attaques visant à retrouver le mot d’activation . . . . .	105
3.8 Conclusion . . . . .	106

<b>4</b>	<b>Protocoles de réconciliation de clés pour la correction des erreurs dans les réponses des PUFs</b>	<b>109</b>
4.1	Similarités entre la réconciliation en distribution quantique de clés et la génération fiable de clé à partir d'une réponse de PUF . . . . .	111
4.2	Correction des erreurs basée sur des vérifications de parité et la recherche dichotomique . . . . .	112
4.3	Protocole BINARY . . . . .	113
4.4	Protocol CASCADE . . . . .	116
4.5	Paramètres du protocole CASCADE . . . . .	117
4.6	Implémentation . . . . .	122
4.7	Résultats expérimentaux . . . . .	124
4.8	Sécurité : attaques et contre-mesures . . . . .	132
4.9	Discussion . . . . .	134
4.10	Conclusion . . . . .	135
<b>5</b>	<b>Infrastructure matérielle/logicielle complète pour la protection des données de conception</b>	<b>137</b>
5.1	Intégration aux outils de conception électronique . . . . .	139
5.2	Plateforme matérielle : carte HECTOR . . . . .	146
5.3	Résultats d'implémentation matérielle globaux . . . . .	147
5.4	Interface logicielle . . . . .	149
5.5	Exemple illustratif . . . . .	151
5.6	Cas d'utilisation . . . . .	152
5.7	Conclusion . . . . .	153
	<b>Conclusion</b>	<b>159</b>
	<b>Publications et communications</b>	<b>163</b>
	<b>Bibliographie</b>	<b>166</b>
	<b>Annexes</b>	<b>183</b>





# Acronyms

**ASIC** application-specific integrated circuit.

**AW** activation word.

**BEOL** back end of line.

**CC0** combinational 0 controllability.

**CC1** combinational 1 controllability.

**CMOS** complementary metal-oxide-semiconductor.

**CO** combinational observability.

**CRP** challenge-response pair.

**DFF** D flip-flop.

**DIP** distinguishing input pattern.

**EDA** electronic design automation.

**EEPROM** electrically-erasable programmable read-only memory.

**FEOL** front end of line.

**FPGA** field-programmable gate array.

**FSM** finite-state machine.

**HDL** hardware description language.

**IP** intellectual property.

**LPN** Learning parities with noise.

**LUT** look-up table.

**NVM** non-volatile memory.

**OTP-NVM** one-time programmable non-volatile memory.

**PoS** Product-of-Sums.

**PUF** physical unclonable function.

**RO-PUF** ring oscillator PUF.

**ROM** read-only memory.

**SoC** System on Chip.

**SoP** Sum-of-Products.

**SRAM** static random access memory.

**TERO** transient effect ring oscillator.

**TERO-PUF** transient effect ring oscillator PUF.

# Notations

$[X]_K$  Ciphertext obtained after symmetric encryption of the plaintext  $X$  with the key  $K$ .

$[X]_K^{-1}$  Plaintext obtained after symmetric decryption of the ciphertext  $X$  with the key  $K$ .

$\text{HD}(A, B)$  Hamming distance between  $A$  and  $B$ .

$A[i]$   $i^{\text{th}}$  bit of vector  $A$ .



# Introduction

According to the World Semiconductor Trade Statistics (WSTS), the sales of the semiconductor market reached almost \$340 billion in 2016<sup>1</sup>. This ever-changing industry is characterised by a vigorous competitiveness, a steadily increasing complexity and a strong market pull. One of the main problems facing this industry today is the protection of design intellectual property rights. This is mainly due to the multiplicity of actors involved in the design, production and marketing of electronic products. In order to understand where the issue comes from, a historical and economical overview of the semiconductor industry is necessary.

## Historical and economical context

Moore's law, first published in 1965 [Moo65] and revised in 1975 [Moo75] states that the number of transistors that can be integrated on a unit area of integrated circuit doubles every two years. So far, even though a slight slowdown has been observed in recent years, the microelectronics industry followed this law. This is possible by making transistors smaller and smaller, 10nm being the technology node achieved in 2017<sup>2,3</sup>. Such a constant decrease is due to a strong market pull, which led customers to request more and more sophisticated, powerful and small devices.

A corollary of Moore's law is Rock's law, which states that the cost of a fabrication plant for integrated circuits doubles every four years. This emerges directly from the decreasing size of the transistors, making them harder and harder to manufacture. The cost of manufacturing plants now reaches tens of billion dollars<sup>4,5</sup>. With such considerable investments, control over

---

<sup>1</sup>Global Semiconductor Sales Reach \$339 Billion in 2016 [http://www.semiconductors.org/news/2017/02/02/global\\_sales\\_report\\_2015/global\\_semiconductor\\_sales\\_reach\\_339\\_billion\\_in\\_2016/](http://www.semiconductors.org/news/2017/02/02/global_sales_report_2015/global_semiconductor_sales_reach_339_billion_in_2016/)

<sup>2</sup>Samsung Starts Industry's First Mass Production of System-on-Chip with 10-Nanometer FinFET Technology. <http://news.samsung.com/global/samsung-starts-industrys-first-mass-production-of-system-on-chip-with-10-nanometer-finfet-technology>

<sup>3</sup>Intel Finds Moore's Law's Next Step at 10 Nanometers. <http://spectrum.ieee.org/semiconductors/devices/intel-finds-moores-laws-next-step-at-10-nanometers>

<sup>4</sup>China's Tsinghua Unigroup to build \$30 billion Nanjing chip plant. <http://www.reuters.com/article/us-tsinghua-plant-idUSKBN1532ED>

<sup>5</sup>Samsung Breaks Ground on \$14 Billion Fab. [http://www.eetimes.com/document.asp?doc\\_id=1326565](http://www.eetimes.com/document.asp?doc_id=1326565)

fabrication plants rose to a national priority in USA<sup>6,7</sup>, since most of the foundries are now located in Asia. Another consequence of this increasing up-front investment is the market domination of existing large corporations, where five of them (Intel, Samsung, Qualcomm, Broadcom, and SK Hynix) hold 41% of the marketshares in 2016<sup>8</sup>. The top two companies, Intel and Samsung, use the historical *Integrated Device Manufacturer* (IDM) model. A single company accomplishes the design, manufacturing and selling of the integrated circuit. However, the next two, Qualcomm and Broadcom, use the *fabless* model. As the name suggests, *fabless* companies do not own any fabrication facility. Instead, they rely on manufacturing plants own by third parties. Those companies, specialised in integrated circuits manufacturing, are called *foundries*. They are more and more important in the semiconductor industry, exceeding 50 billion dollars in sales in 2016, with an 11% increase compared to 2015<sup>9</sup>. Together, fabless designers and foundries form a new business model [Hod11], that appeared in the 1980s, when the process was split into two parts: design and manufacturing.

Semiconductors being a very competitive market, shorter and shorter time to market has been required. In conjugation with a strong market pull, the allotted time to design integrated circuits reduced significantly. In order to keep-up with this trend, integrated circuits designers massively switched to a design-and-reuse paradigm, also called core-based design [GZ97]. In this framework, a complex design is split into smaller functional blocks of manageable complexity. Thus two new types of companies appeared in the design process, dividing it further. Intellectual property (IP) providers design individual IP cores, implementing a precise function. For instance, one can find JPEG encoder or Ethernet controller IP cores. Those IP cores are typically purchased by system integrators, who integrate them into a single modular design. The different types of companies taking part in the design of an integrated circuit are shown in Figure 1. Of course, such a strict division does not perfectly match reality. For instance, a fabless designer might develop some IP cores in-house and purchase others from third party IP core providers.

The next section focuses on IP cores, detailing how they are distributed and the threats associated to this business model.

## IP cores distribution and business model

Following the global transition from an industrial economy to a knowledge economy [Dru69], the semi-conductor industry now relies heavily on the exchange and monetisation of intellectual

---

<sup>6</sup>Can the White House Make America's Chip Industry Great Again? <http://www.technologyreview.com/s/602768/can-the-white-house-make-americas-chip-industry-great-again/>

<sup>7</sup>Trump team backs call for crackdown on China over semiconductors. <http://www.ft.com/content/bca04dfe-de67-11e6-9d7c-be108f1c1dce>

<sup>8</sup>Five Suppliers Hold 41% of Global Semiconductor Marketshare in 2016. <http://www.icinsights.com/data/articles/documents/938.pdf>

<sup>9</sup>Pure-Play Foundry Market Surges 11% in 2016 to Reach \$50 Billion! <http://www.icinsights.com/data/articles/documents/945.pdf>

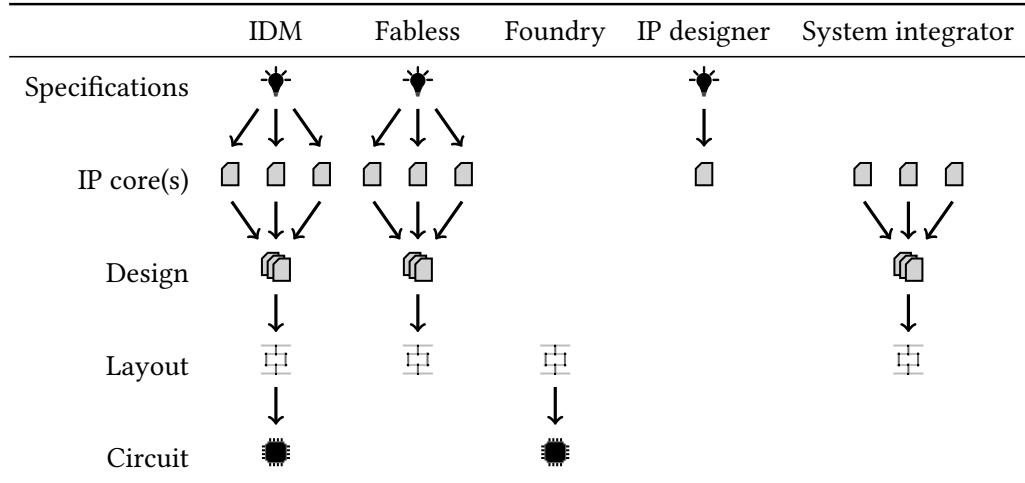


Figure 1 – Semiconductor companies and their respective positions in the integrated circuit design process.

property for the design of integrated circuits. Practically, IP cores are not provided alone but can come with application-specific integrated circuit (ASIC) synthesis scripts, field-programmable gate array (FPGA) place & route scripts, simulation scripts, testbenches, software models, test vectors, documentation, etc. Much like software companies, IP cores design companies now make the headlines for mergers and acquisitions worth millions of dollars. For instance, Intel acquired Altera and NXP was acquired by Qualcomm in the last two years. As stated in a recent research bulletin by IC Insights<sup>10</sup>, “*The dollar value of merger and acquisition agreements in 2015 and 2016 were both about eight times greater than the \$12.6 billion annual average of M&A announcements in the five previous years (2010-2014)*”.

Designers directly sell their IP cores to system integrators or rely on an intermediate IP broker. Those IP brokers, such as ChipEstimate<sup>11</sup>, Design-And-Reuse<sup>12</sup> or CAST<sup>13</sup>, maintain large catalogues of IP cores from multiple designers. System integrators then purchase IP cores from the brokers or from the designers directly. This is very similar to the way software products are sold.

However, even though the distributions of IP cores and pieces of software work in a similar way nowadays, their actual usage after distribution is entirely different. Indeed, proprietary pieces of software come with a license, either in the form of a key, a file or a server. Without them, the software cannot be executed. IP cores however, once they are sold by the designer, are much harder to keep control on. The main issue here is that once an IP core has been sold, the IP designer has no way of knowing how many times the IP core is actually instantiated.

This issue comes into great conflict with the knowledge economy principle stated above. Indeed, without knowing the number of IP core instances, IP designers must adopt a licensing

<sup>10</sup>2015-2016 deals dominate semiconductor M&A ranking. <http://www.icinsights.com/data/articles/documents/946.pdf>

<sup>11</sup><http://www.ipcatalog.com>

<sup>12</sup><http://www.design-reuse.com>

<sup>13</sup><http://www.cast-inc.com>



model with upfront payment. In this model, an IP designer demands a fixed amount of money from a system integrator before selling the IP core. Once it has been sold, the system integrator can instantiate the IP core as many times as needed. There are two issues with this business model. Firstly, it can inhibit small scale purchases for prototyping purposes or for small companies due to a too high initial investment. Secondly, it strongly limits the advantages brought by core-based design, which could benefit greatly from features typically found in software products like an evaluation period or a premium version of the IP core with enhanced performances.

Besides the limitations brought by upfront licensing, the main issue with the designer not knowing how many times the IP core is instantiated is that it can potentially be illegally copied. For example, a system integrator could sell a previously purchased IP core to business associates for a lower price, without the original designer knowing about it.

In order to exploit the full potential and benefits of knowledge economy and to prevent illegal copying, a designer must then be able to know how many times a particular IP core has been instantiated. Moreover, by allowing the designer to remotely activate an IP core, pay-per-use licensing would be possible. Finally, with remote activation comes pre-activation mode. If this mode is degraded, illegal copies can be effectively made useless until they are properly activated by the original IP designer. Obviously, such a remote activation scheme should also be secure, so that ill-intentioned users cannot circumvent it and use an illegal copy of the IP core. This is one of the objectives of the SALWARE project.

## **SALWARE project**

The SALWARE project is a 4-year project, supported by the French “*Agence Nationale de la Recherche*” and by the “*Fondation de Recherche pour l’Aéronautique et l’Espace*”. The title of the project is: “Salutary hardware design to fight against integrated circuit counterfeiting and theft”. The name of the project originates from the word *malware*, which stands for “malicious software”, and was turned into *salware* which stands for “salutary hardware”. The aim of the project is to design hardware components that provide intellectual property information and/or allow for remote activation of an integrated circuit or and intellectual property core after manufacturing. The hardware components designed in the framework of the SALWARE project aim at exhibiting the same features as a malware. Namely, they should be stealthy, or lightweight, in order to make the logic resources overhead as low as possible. This is a very strict requirement to make the proposed solution industrially usable. Moreover, they should be efficient at disturbing the operation of the circuit or the IP core, so that illegal copies are essentially useless. Finally, they should be sufficiently hard to remove or circumvent to deter malicious users.

Cédric Marchand, who was another PhD student working in the frame of the SALWARE project, defended his PhD in 2016 [Mar16]. His PhD thesis deals with watermarking, physical

unclonable function (PUF) design and lightweight cryptography implementation, which are essential components for salutory hardware, but more specifically targeted at preventing integrated circuits counterfeiting. This PhD thesis has complementary contributions, which are presented below, focusing more precisely on IP cores.

## Contributions

First of all, in order to ensure that a design data protection scheme is efficient, illegal copies must exhibit a very disturbed operation. The first option explored in this thesis to achieve this is to controllably force the outputs of a netlist to a fixed logic level. We call this logic locking. A very efficient algorithm to select the netlist nodes to modify based on the propagation of a controlling value in a graph is presented.

The second option to provide a degraded mode of operation is to disturb the outputs of the netlist by controllably inverting specific internal nodes. We refer to this as logic masking. Specifically, a new method of selection of the nodes to invert based on centrality indicators from graph theory is shown. Compared to state-of-the-art selection heuristics, it scales better to large netlists and efficiently disturbs the circuit operation.

The third contribution of this thesis deals with unique identification of IP core instances using a PUF. PUFs are very interesting primitives since they allow to identify IP core instances by extracting device-specific manufacturing process variations, which are known to be random. However, those PUFs are subject to instability, and the extracted identifiers are not stable enough. To deal with this, we propose an innovative method based on the CASCADE key reconciliation protocol. Originally developed for quantum key exchange, we show that this interactive protocol can be successfully applied to error-correction of silicon PUF responses. Compared to existing error-correcting codes implementations, it is around an order of magnitude more lightweight in terms of required logic resources.

Finally, these contributions and those found in [Mar16] are bundled together in a complete design data protection module. We show that this module fulfils the requirements of a *salware* by being lightweight, secure and efficient at providing different degraded modes of operation for an IP core. Overall, this is an industrially viable solution for IP designers who wish to protect their design data from illegal copying at reduced cost.

## Outline

Chapter 1 presents the IP business model which is widely adopted by the semiconductor industry nowadays. We highlight the new threats on design data which emerge from this new business model and provide a detailed threat model. We then provide a state of the art of existing data protection methods for IP cores. We conclude this chapter by giving the requirements for the complete design data protection module and provide a high-level overview of the different

components required to fulfil these requirements. Chapter 2 describes the method based on graph-analysis for combinational logic locking of a netlist. Chapter 3 shows how centrality indicators from graph theory can be used to select the most suited nodes to modify by logic masking. Chapter 4 presents the similarities between two scenarios, quantum key exchange and error-correction for silicon PUF responses. In particular, we show how the CASCADE key-reconciliation protocol can be used to provide lightweight error correction of silicon PUF responses. Finally, Chapter 5 presents the complete hardware/software design data protection module, which is the objective of the SALWARE project.

# Introduction

D'après les Statistiques du Commerce Mondial des Semi-conducteurs, le marché des semi-conducteurs a atteint 340 milliards de dollars de vente en 2016<sup>14</sup>. Cette industrie en évolution constante est caractérisée par une compétitivité intense, une complexité en constante croissance et une forte demande du marché. L'un des principaux problèmes auquel cette industrie doit aujourd'hui faire face est la protection des droits de propriété intellectuelle sur les données de conception. Cela est dû majoritairement à la multiplicité des acteurs impliqués dans la conception, la production et la commercialisation de produits électroniques. Afin de comprendre d'où vient le problème, un aperçu de l'industrie des semi-conducteurs d'un point de vue historique et économique est nécessaire.

## Contexte historique et économique

La loi de Moore, publiée pour la première fois en 1965 [Moo65] puis révisée en 1975 [Moo75], dit que le nombre de transistors qui peuvent être intégrés sur une surface unitaire de circuit intégré double tous les deux ans. Jusqu'ici, même si un ralentissement certain a été observé récemment, l'industrie de la micro-électronique a suivi cette loi. Ceci est rendu possible en réduisant de plus en plus la taille des transistors, 10nm étant le nœud technologique atteint en 2017<sup>15,16</sup>. Cette diminution constante est due à une forte demande du marché, qui a amené les consommateurs à demander des équipements toujours plus sophistiqués, puissants et petits. Un corollaire de la loi de Moore est la loi de Rock, qui dit que le coût de fabrication d'une usine de fabrication de circuit intégrés double, lui, tous les quatre ans. Ceci est une conséquence directe de la diminution de la taille des transistors, ce qui les rend de plus en plus complexes à fabriquer. Le coût d'une usine de fabrication atteint aujourd'hui des dizaines

---

<sup>14</sup>Global Semiconductor Sales Reach \$339 Billion in 2016 [http://www.semiconductors.org/news/2017/02/02/global\\_sales\\_report\\_2015/global\\_semiconductor\\_sales\\_reach\\_339\\_billion\\_in\\_2016/](http://www.semiconductors.org/news/2017/02/02/global_sales_report_2015/global_semiconductor_sales_reach_339_billion_in_2016/)

<sup>15</sup>Samsung Starts Industry's First Mass Production of System-on-Chip with 10-Nanometer FinFET Technology. <http://news.samsung.com/global/samsung-starts-industrys-first-mass-production-of-system-on-chip-with-10-nanometer-finfet-technology>

<sup>16</sup>Intel Finds Moore's Law's Next Step at 10 Nanometers. <http://spectrum.ieee.org/semiconductors/devices/intel-finds-moores-laws-next-step-at-10-nanometers>

de milliards de dollars<sup>17,18</sup>. Avec des investissements aussi considérables, avoir la mainmise sur les usines de fabrication est devenu une priorité nationale aux États-Unis<sup>19,20</sup>, puisque la plupart d'entre elles sont maintenant situées en Asie. Une autre conséquence de l'augmentation de l'investissement initial requis est la domination du marché par les grandes entreprises existantes. Cinq d'entre elles (Intel, Samsung, Qualcomm, Broadcom et SK Hynix) possèdent ainsi 41% des parts de marché en 2016<sup>21</sup>. Les deux premières, Intel et Samsung, suivent le modèle historique du constructeur d'équipement intégré (*Integrated Device Manufacturer* ou IDM). Une seule entreprise assure la conception, la fabrication et la vente du circuit intégré. Néanmoins, les deux suivantes, Qualcomm et Broadcom, suivent le modèle *fabless*. Comme le nom l'indique, les entreprises *fabless* n'ont pas de moyens de fabrication. Elles s'appuient plutôt sur des entreprises tierces possédant des usines de fabrication. Ces entreprises, spécialisées dans la fabrication de circuits intégrés, sont appelées *fonderies*. Elles sont de plus en plus importantes dans l'industrie de la micro-électronique, dépassant les 50 milliards de dollars de vente en 2016, avec une augmentation de 11% par rapport à 2015<sup>22</sup>. Ensemble, les concepteurs *fabless* et les fonderies forment un nouveau modèle économique [Hod11], apparu dans les années 1980, où le processus global a été séparé en deux : conception et fabrication.

L'industrie des semi-conducteurs étant un marché très compétitif, les délais de commercialisation sont également réduits. Ce phénomène, associé à une forte demande des consommateurs, a réduit de manière significative le temps alloué à la conception des circuits électroniques. Afin de suivre cette tendance, les concepteurs ont rapidement adopté un modèle de conception modulaire [GZ97] basé sur la réutilisation de blocs existants (*design-and-reuse*). Dans ce cadre, un composant complexe est divisé en blocs fonctionnels de taille plus réduite et de complexité gérable. Ainsi, deux nouveaux types d'entreprises sont apparus dans le processus de conception, le divisant encore. Les *concepteurs de composants virtuels* conçoivent des modules implémentant une fonction spécifique. Par exemple, on peut trouver des composants virtuels de décodage JPEG ou de contrôle Ethernet. Ces composants virtuels sont typiquement achetés par des *intégrateurs système*, qui les associent dans un système modulaire. Les différents types d'entreprises prenant part à la conception d'un circuit intégré sont présentés dans la Figure 2. Évidemment une division stricte ne reflète pas parfaitement la réalité. Par exemple, un concepteurs *fabless* peut concevoir certains composants virtuels en interne et en obtenir d'autres de concepteurs tiers.

---

<sup>17</sup>China's Tsinghua Unigroup to build \$30 billion Nanjing chip plant. <http://www.reuters.com/article/us-tsinghua-plant-idUSKBN1532ED>

<sup>18</sup>Samsung Breaks Ground on \$14 Billion Fab. [http://www.eetimes.com/document.asp?doc\\_id=1326565](http://www.eetimes.com/document.asp?doc_id=1326565)

<sup>19</sup>Can the White House Make America's Chip Industry Great Again? <http://www.technologyreview.com/s/602768/can-the-white-house-make-americas-chip-industry-great-again/>

<sup>20</sup>Trump team backs call for crackdown on China over semiconductors. <http://www.ft.com/content/bca04dfe-de67-11e6-9d7c-be108f1c1dce>

<sup>21</sup>Five Suppliers Hold 41% of Global Semiconductor Marketshare in 2016. <http://www.icinsights.com/data/articles/documents/938.pdf>

<sup>22</sup>Pure-Play Foundry Market Surges 11% in 2016 to Reach \$50 Billion ! <http://www.icinsights.com/data/articles/documents/945.pdf>

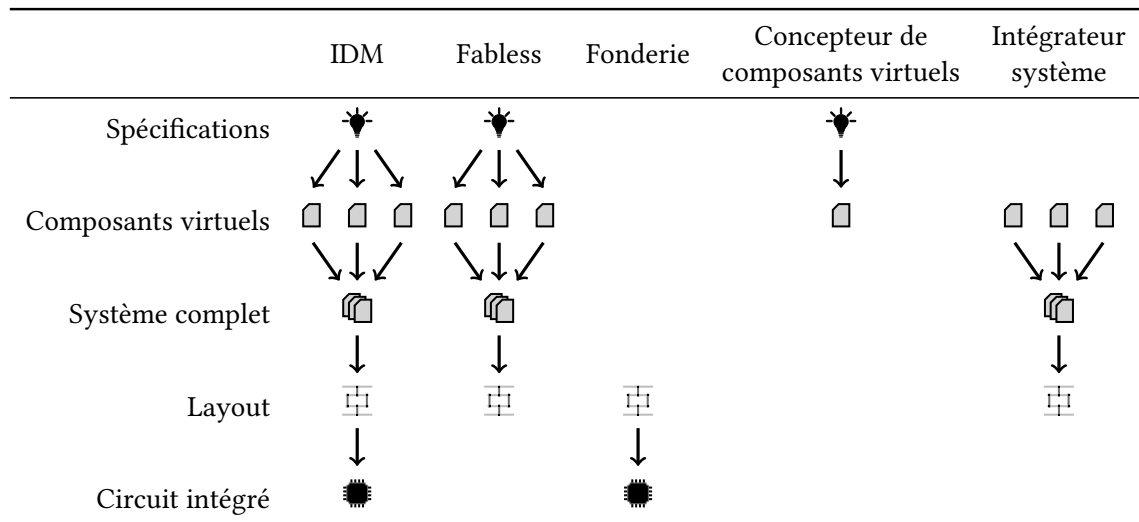


FIGURE 2 – Entreprises de la micro-électronique et leur position respective dans le processus de conception d'un circuit intégré.

La section suivante se concentre spécifiquement sur les composants virtuels, la manière dont ils sont distribués et les menaces associées à ce modèle économique.

## Distribution des composants virtuels et modèle économique

Suivant la transition globale d'une économie industrielle vers une économie de la connaissance [Dru69], l'industrie des semi-conducteurs s'appuie maintenant fortement sur l'échange et la monétisation de la propriété intellectuelle pour la conception des circuits intégrés. En pratique, les composants virtuels ne sont pas fournis seuls mais sont accompagnés de scripts de synthèse pour ASIC, de scripts de placement et routage pour FPGA, de scripts de simulation, de bancs de test, de modèles logiciels, de vecteurs de tests, de documentation, etc. Comme les sociétés de logiciels, les entreprises de conception de composants virtuels font maintenant la une avec des fusions et acquisitions atteignant des milliards de dollars. Par exemple, Intel a acquis Altera et NXP a été racheté par Qualcomm dans les deux dernières années. Comme mis en évidence dans un récent rapport par IC Insights<sup>23</sup>, le montant total des fusions et acquisitions en 2015 et 2016 était environ huit fois supérieur aux 12,6 millions de dollars qui constituaient la moyenne annuelle dans les cinq années précédentes (2010-2014).

Les concepteurs vendent leurs composants virtuels directement aux intégrateurs système ou s'appuient sur des grossistes intermédiaires. Ces derniers, tels que ChipEstimate<sup>24</sup>, Design-And-Reuse<sup>25</sup> ou CAST<sup>26</sup> maintiennent d'importants catalogues de composants virtuels de nombreux concepteurs. Les intégrateurs système acquièrent ensuite les composants virtuels

<sup>23</sup>2015-2016 deals dominate semiconductor M&A ranking. <http://www.icinsights.com/data/articles/documents/946.pdf>

<sup>24</sup><http://www.ipcatalog.com>

<sup>25</sup><http://www.design-reuse.com>

<sup>26</sup><http://www.cast-inc.com>

via ces grossistes ou directement auprès du concepteur. Ce mode de fonctionnement est très proche de la manière dont les logiciels sont vendus.

Néanmoins, même si les moyens de distribution des composants virtuels et des logiciels suivent un modèle similaire de nos jours, leur usage après distribution est entièrement différent. En effet, les logiciels propriétaires sont accompagnés d'une licence, sous forme de clé, de fichier ou de serveur. Sans ces derniers, le logiciel ne fonctionne pas. Les composants virtuels, cependant, une fois qu'ils sont vendus par le concepteur, sont beaucoup plus difficiles à contrôler. Le principal problème vient du fait qu'une fois qu'un composant virtuel a été vendu, le concepteur n'a aucun moyen de savoir combien de fois il sera instancié en pratique.

Ce problème s'oppose de manière directe au principe d'économie de la connaissance mentionné ci-dessus. En effet, puisqu'il ne connaît pas le nombre d'instances du composant virtuel, le concepteur doit se résoudre à adopter un modèle de licence à versement initial. Dans ce modèle, le concepteur demande un montant fixe à l'intégrateur système avant de lui fournir le composant virtuel. Une fois la transaction réalisée, l'intégrateur système peut instancier le composant virtuel autant de fois qu'il le souhaite. Deux problèmes apparaissent dans ce modèle économique. Premièrement, cela empêche les achats en quantité limitée qui peuvent être utiles pour le prototypage ou pour les petites entreprises du fait de l'investissement initial élevé. De plus, cela limite fortement les avantages apportés par la conception modulaire, qui pourrait bénéficier largement de possibilités typiquement présentes dans le domaine du logiciel telles qu'une période d'évaluation ou un version premium du composant virtuel avec des performances plus élevées.

Au delà des limitations induites par ce modèle de licence, le principal problème lorsque le concepteur ne sait pas combien de fois le composant virtuel a été instancié est qu'il peut être potentiellement copié de manière illégale. Par exemple, un intégrateur système pourrait vendre à des associés un composant virtuel qu'il a déjà acheté, à un prix réduit et sans que le concepteur original n'en ait connaissance.

Afin d'exploiter pleinement le potentiel et les avantages de l'économie de la connaissance et pour empêcher la copie illégale, un concepteur doit pouvoir savoir combien de fois un composant virtuel a été instancié. En outre, en permettant au concepteur d'activer à distance le composant virtuel, un modèle de licence basé sur l'usage serait possible. Enfin, permettre l'activation à distance implique la présence de modes de fonctionnement dégradés. Les copies illégales sont ainsi rendues inutilisables et donc inutiles jusqu'à ce qu'elle soient activées par le concepteur original. Évidemment, un tel système d'activation à distance doit également être sécurisé, de manière à ce que des utilisateurs mal intentionnés ne puissent pas le contourner et utiliser une copie illégale du composant virtuel. C'est l'un des objectifs du projet SALWARE.

## Le projet SALWARE

Le projet SALWARE est un projet de quatre ans, financé par l'Agence Nationale de la Recherche et la Fondation de Recherche pour l'Aéronautique et l'Espace. L'intitulé du projet est le suivant : "Conception de matériel salubre pour lutter contre la contrefaçon et le vol de circuits intégrés". Le nom du projet vient du mot *malware*, qui signifie logiciel malicieux, et qui a été changé en *salware* pour matériel salubre. L'objectif de ce projet est de concevoir des blocs matériels fournissant des informations de propriété intellectuelle et/ou permettant l'activation à distance d'un circuit intégré ou d'un composant virtuel après fabrication. Les blocs matériels conçus dans le cadre du projet SALWARE présentent les mêmes propriétés qu'un *malware*. Ainsi, ils doivent être discrets, ou légers, afin d'induire un coût supplémentaire en ressources logiques le plus faible possible. Ceci est une exigence très stricte pour rendre la solution proposée applicable dans un contexte industriel. En outre, ils doivent perturber efficacement le fonctionnement du circuit ou du composant virtuel, afin de rendre les copies illégales inutiles. Enfin, ils doivent être suffisamment difficiles à contourner ou à supprimer pour décourager les utilisateurs malveillants.

Cédric Marchand, qui était un autre doctorant travaillant dans le cadre du projet SALWARE, a soutenu sa thèse en 2016 [Mar16]. Sa thèse traitait du *watermarking*, de la conception de fonctions physiques non clonables (*Physical Unclonable Function* ou PUF) et de l'implémentation de cryptographie légère, qui sont des éléments essentiels pour du matériel salubre mais qui sont plus spécifiquement orientés vers la prévention de la contrefaçon de circuits intégrés. Cette thèse de doctorat a des contributions complémentaires, qui sont présentées ci-dessous, et se concentre plus spécifiquement sur les composants virtuels.

## Contributions

En premier lieu, afin de s'assurer qu'un système de protection des données de conception est efficace, les copies illégales doivent présenter un comportement très perturbé. La première option étudiée dans cette thèse pour permettre ceci est de forcer à une valeur logique fixe les sorties d'un composant virtuel, de manière contrôlée. Un algorithme très efficace permettant de sélectionner les nœuds de la netlist à modifier, basé sur la propagation d'une valeur de verrouillage dans un graphe, est présenté.

La seconde option pour fournir un mode de fonctionnement dégradé est de perturber les sorties du composant virtuel en inversant certains nœuds internes. Ceci est appelé masquage logique. Spécifiquement, une nouvelle méthode permettant de sélectionner les nœuds à inverser, basée sur les indicateurs de centralité en théorie des graphes, est proposée. Comparée aux heuristiques de sélection de l'état de l'art, elle s'étend plus efficacement à des composants virtuels de grande taille et altère le fonctionnement du circuit de manière efficace.

La troisième contribution de cette thèse traite de l'identification unique des instances



d'un composant virtuel en utilisant une PUF. Les PUFs sont des primitives très intéressantes puisqu'elles permettent d'identifier individuellement les instances en extrayant les variations apparaissant lors du processus de fabrication, qui sont spécifiques à chaque circuit produit et sont aléatoires. Toutefois, les PUFs présentent une certaine instabilité, et les identifiants extraits ne sont pas suffisamment stables. Pour résoudre ce problème, nous proposons une méthode innovante basée sur le protocole de réconciliation de clés CASCADE. Développé au départ pour l'échange quantique de clés, nous montrons que ce protocole interactif peut être utilisé de manière fructueuse pour la correction des erreurs présentes dans les réponses des PUFs. En comparaison des implémentations existantes de codes correcteurs d'erreurs, cette solution est une ordre de grandeur plus légère en terme de ressources logiques requises.

Enfin, ces contributions et celles de [Mar16] sont assemblées en un système complet de protection des données de conception. Nous montrons que ce système remplit les conditions pour être considéré comme un SALWARE, en étant léger, sûr et efficace pour proposer différents modes de fonctionnement dégradés pour un composant virtuel. Finalement, ceci constitue une solution industriellement viable pour les concepteurs de composants virtuels qui souhaitent protéger leurs données de conception de la copie illégale à moindre coût.

## Plan

Le chapitre 1 présente le modèle économique associé aux composants virtuels qui a été largement adopté par l'industrie des semi-conducteurs. Nous mettons en évidence les nouvelles menaces pour les données de conception qui émergent de ce nouveau modèle économique et proposons un modèle de menace détaillé. Nous présentons ensuite un état de l'art des méthodes de protection des données de conception pour les composants virtuels. Nous concluons ce chapitre en présentant les caractéristiques souhaitées pour un système de protection des données de conception et donnons un aperçu des différents éléments requis pour mettre en œuvre ces fonctions.

Le chapitre 2 décrit la méthode basée sur l'analyse de graphe pour le verrouillage combinatoire d'un composant virtuel. Le chapitre 3 montre de quelle manière les indicateurs de centralité de la théorie des graphes peuvent être utilisés pour sélectionner les nœuds les plus efficaces pour une modification par masquage logique. Le chapitre 4 présente les similarités entre deux scénarios, l'échange quantique de clés et la correction des erreurs dans les réponses des PUFs. En particulier, nous montrons que le protocole de réconciliation de clés CASCADE peut être utilisé pour fournir une solution légère de correction des erreurs pour les réponses des PUFs. Finalement, le chapitre 5 présente le système logiciel/matériel complet pour la protection des données de conception, ce qui constitue l'objectif du projet SALWARE.

# Chapter 1

## Threats and protections for design data

---

Following the fragmentation of the semiconductor design process mentioned before, multiple parties now participate and are involved at different stages. Such a multiplicity of actors comes with specific risks for design data. Intellectual property transfers between stakeholders, even though they are necessary to the new business model, are the cause of multiple threats. In order to further understand them, a review of the various parties and their individual role is needed.

Next, we give a description of the three main threats that can be identified against design data: overproducing, illegal copying and reverse-engineering. We then take the point of view of an IP core designer and identify which of the parties involved in the design process are likely to perform these illegal actions. This leads us to define two threats models, one shared between illegal copying and overproducing and a specific one for reverse-engineering. For each threat model, the attacker's and defender's objectives, capabilities or constraints are detailed.

We then give a state-of-the-art of existing methods that aim at the protection of design data. The methods are classified according to their efficiency at providing a complete protection against the aforementioned threats. This ranges from the simple identification of an IP core to thorough licensing schemes. We also present some solutions that are a combination of multiple design data protection methods.

This leads us to propose a set of requirements for a strong, lightweight and usable IP protection scheme. We then present how we propose to implement the features that fulfil these requirements in the SALWARE project.

# Menaces sur les données de conception et méthodes de protection

---

Suite à la fragmentation du processus de conception de circuits intégrés mentionné précédemment, de nombreux acteurs sont aujourd'hui impliqués à différentes étapes. Cette multiplicité d'acteurs est accompagnée de risques spécifiques pour les données de conception. Les transferts de propriété intellectuelle entre les acteurs, quoique nécessaires au nouveau mode de fonctionnement de l'industrie, sont la cause de multiples menaces. Afin de mieux les comprendre, passer en revue les différents acteurs impliqués et leurs rôles est nécessaire.

Ensuite, nous décrivons les trois menaces principales qui ciblent les données de conception : surproduction, copie illégale et rétro-ingénierie. Nous nous plaçons ensuite du point de vue du concepteur de composants virtuels et identifions quels acteurs impliqués dans le processus de conception sont susceptibles de réaliser ces actions illégales. Cela nous conduit à définir deux modèles de menace, l'un commun à la copie illégale et la surproduction et l'autre spécifique à la rétro-ingénierie. Pour chaque modèle de menace, les objectifs, les possibilités et les contraintes des attaquants et des défenseurs sont détaillés.

Nous donnons ensuite un état de l'art des méthodes qui visent à protéger les données de conception. Ces méthodes sont classées d'après leur efficacité à fournir une protection complète contre les menaces mentionnées ci-dessus. Cela va de la simple identification d'un composant virtuel à des schémas de licence d'utilisation complets. Nous présentons également quelques solutions qui sont une combinaison de plusieurs méthodes de protection des données de conception.

Cela nous conduit finalement à proposer un ensemble de caractéristiques pour un module de protection des données de conception robuste, léger et utilisable. Nous présentons ensuite comment nous proposons d'implémenter les fonctionnalités satisfaisant à ces exigences dans le cadre du projet SALWARE.

## 1.1 Parties involved in the design process and their roles

Multiple parties are involved in the lifetime of an electronic device. We restrict ourselves to the ones present at design time. Therefore, we do not consider parties in charge of subsequent steps: manufacturing, testing, packaging, supplying, selling, recycling, etc.

### 1.1.1 IP Designer

The designer of the IP core is the first party to take part in the design process. From the specifications, which can be laid down by a customer, a standard or in-house, an IP core is designed. It consists in describing a hardware implementation of the specifications. Along with the implementation, the designer can supply test vectors, place and route scripts, testbenches, software models, documentation, etc. Together, these parts form the intellectual property material that is referred to as IP core. The actual implementation can come in three main forms:

**Software IP:** the IP core is provided in a hardware description language like VHDL, SystemVerilog or SystemC, before synthesis. This type of IP core offers the advantage of not being dependent on the final hardware target. These descriptions can be done at several levels of abstraction, with the constraint that they must be synthesisable. For example, a VHDL description can go down to the register transfer level, while the SystemC language allows one to do a high-level behavioural description. Software IP is described in a very high-level style, possibly using language features that are close to those of a programming language. However, a soft IP must be synthesisable, otherwise it is closer to a software model.

**Firmware IP:** a low level description after synthesis is given, in a netlist format such as EDIF<sup>1</sup>. It may be technology-dependent if the IP core instantiates vendor-specific primitives.

**Hardware IP:** this is the lowest level of abstraction to be found for an IP core. If the target hardware platform is ASIC, then a layout file in the GDS II, OASIS<sup>2</sup> or other format is provided. It directly represents the layout of the design as it will be used for the photo-lithography masks. Conversely, if the design is to be implemented on FPGA, a bitstream file is given. This bitstream file describes how the LUTs, switching matrices and RAM blocks inside FPGA must be configured to achieve the desired logic function.

These three types of IP cores represent different levels of abstraction. Examples of IP core designers are ARM<sup>3</sup>, Dolphin Integration<sup>4</sup>, Intrinsic ID<sup>5</sup> or Rambus<sup>6</sup>.

---

<sup>1</sup>Electronic Design Interchange Format

<sup>2</sup>Open Artwork System Interchange Standard

<sup>3</sup><http://arm.com>

<sup>4</sup><http://dolphin-integration.com>

<sup>5</sup><http://intrinsic-id.com>

<sup>6</sup><http://rambus.com>

### 1.1.2 Broker

A broker acts as a middleman between designers and system integrators. In order to provide more visibility to third party IP core designers, the broker maintains a catalogue of IP cores, which are classified by category according to the function they implement. For instance, a broker can offer several Ethernet controllers from various designers, reaching various performance targets. Some of these controllers can be low-power while others achieve very high throughput.

From a system integrator point of view, those online catalogues are very helpful. They allow to search and compare the IP cores from different vendors with criteria such as the performance/area/power consumption ratio, the technology node, the foundry, the hardware target, etc. Moreover, the organisations maintaining these catalogues can provide management software for those IP cores. For example, this type of software can manage an IP core repository and provide version control, so that the IP cores can be updated if revisions are done.

Examples of IP core brokers are AnySilicon<sup>7</sup>, ChipEstimate<sup>8</sup>, Design & Reuse<sup>9</sup> or OpenCores<sup>10</sup>, the latter being specialised in open-source IP cores. It is worth noting that FPGA manufacturers Xilinx<sup>11</sup> and Intel<sup>12</sup> also provide a catalogue of IP cores in their respective electronic design automation (EDA) tools Vivado and Quartus Prime.

### 1.1.3 System integrator

The system integrator purchases IP cores from a broker or directly from designers. These individual IP cores, which achieve a specific function, are then integrated together in a complex, modular system.

For a system integrator, previously mentioned IP core characteristics such as the process node or the foundry are crucial for flawless integration. If an IP core has already been manufactured and validated in silicon, it is said to be “silicon ready”. This information is provided by the broker or the designer and is of great help for the system integrator.

Nowadays, most of the designs integrate multiple IP cores. Therefore, the vast majority of electronics design companies are system integrators.

### 1.1.4 Trusted third party

In order to facilitate the interaction between the previously described parties, a trusted third party is sometimes involved. When two parties do not necessarily trust each other but still need to interact, they only need to trust this trusted third party.

---

<sup>7</sup><http://anysilicon.com>

<sup>8</sup><http://chipestimate.com>

<sup>9</sup><http://design-reuse.com>

<sup>10</sup><http://opencores.org>

<sup>11</sup><https://www.xilinx.com/>

<sup>12</sup><https://altera.com/>

Sometimes, in the case of IP cores that are meant to be integrated on FPGAs, the hardware manufacturer can act as a trusted third party. For instance, the hardware manufacturer can integrate secret keys given by the designer into the FPGA. These keys can then be used for IP licensing without the system integrator knowing them.

The role of the trusted third party is described in more details in section 1.5.5, dealing with IP licensing schemes.

### 1.1.5 Interaction between parties

Figure 1.1 shows how the previously described parties interact with one another in the typical semiconductor IP business. Specifically, it depicts how design data is transferred from one party to the other. However, there could be additional relations between these parties. For example, a system designer could request a specific IP to be designed by the IP core designer. A designer could also provide support to the system integrator to assist in the integration of the IP core. However, those relations do not deal with design data transfer.

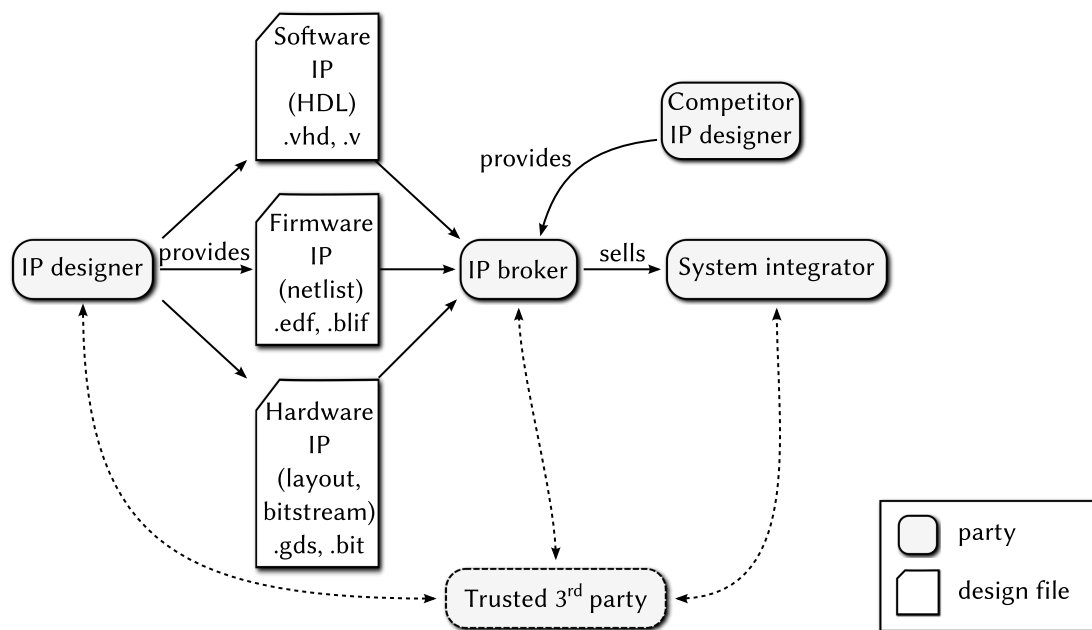


Figure 1.1 – Design data transfer in the semiconductor IP business.

### 1.1.6 Business models

Different types of business model can be found in the semiconductor IP market. They developed in the 1980s, when the semiconductor business started shifting to a knowledge economy. Previously, semiconductor devices were simply sold by manufacturers to system integrators who combined them on boards to design a final product. With the dematerialisation of IP cores, more sophisticated business models could emerge [Fab06]:

**Per-use:** In the per-use model, the IP core designer gives the system integrator the right to use the IP in a certain use scope. The scope must be defined very clearly and can consist in a specific project for example.

**Time-based:** The time-based model allows a system integrator to use an IP core as much as needed but only for a limited period of time. If needed, the contract can be later extended if it expired before the project is completed.

**Royalty-based:** In this model, the final price depends on the usage of the IP core. For example, this can be related to the number of manufactured integrated circuits. It can be very advantageous for both sides, since a system integrator can obtain an IP core for a low initial price but the original designer can also get paid significantly more if the final product is successful.

Even though they differ, all those business models have in common to require a transfer of design data. This comes with associated threats, detailed in the next section.

## 1.2 Threats on design data

With the semiconductor IP business model presented come specific threats on design data. This is first visible on Figure 1.1, where all the arrows representing design data transfer are one-way. This graphically conveys the idea that these design data transfers are asymmetrical. The IP designer provides the broker with an IP core, but in return the broker does not provide any intellectual property to the designer. Similarly, when a system integrator purchases an IP core, the intellectual property material is transferred from the IP designer to the system integrator in only one direction. This poses direct threats to design data since such asymmetric transfer gives rise to intellectual property infringements [SEM06; GDT14] which have severe economic and social impacts [Fro11]. These threats are described in the following subsections.

### 1.2.1 Overproducing

The first type of threat, emerging directly from the immaterial nature of IP cores is *overproducing*. It occurs when, in a per-use business model, the system integrator overrides the scope of use which was previously agreed upon. For example, an IP core which was used in a project is reused later in another design without mentioning it to the IP designer. In a time-based business model, this means that a system integrator keeps using an IP core even though the subscription period has elapsed. If royalties are owed by the system integrator to the IP core designer, the actual number of manufactured devices can be underreported to make the final cost lower. Consequently, for all these cases, the number of instances of the IP core reported to the designer does not match reality. This prevents proper billing and compensation.

### 1.2.2 Illegal copying

The next type of threat is *illegal copying*. This occurs when an IP broker or a system integrator copies an IP core in order to provide it or sell it to another party, unbeknown to the IP core designer. For example, it can be the case if an IP broker charges a system integrator for a certain number of instances of the IP core but in fact reports only half of these instances to the IP core designer. In this case, half of the instances are illegal copies of the IP core since the original designer is unaware of their existence. In large companies, different business units could also share IP cores between projects without reporting it. Finally, a system integrator who obtained an IP core from one designer could sell it to a competitor IP core designer. Similarly to overproducing, these case of illegal copying result in an actual number of IP core instances which is higher than the one reported to the IP core designer, preventing correct billing.

### 1.2.3 Reverse-engineering

The third threat against design data which can be identified is *reverse-engineering*. This is a direct threat to the intellectual property material itself, since it aims at recovering how a logic function is implemented. Therefore, reverse-engineering intends to find out the processes and techniques to go from the specifications to the implementation of the IP core.

Depending on the form in which the IP core is provided, reverse-engineering it can be more or less demanding. In the case of a soft IP described using a hardware description language, recovering the implementation is much easier than if only a layout is available. Similarly, a bitstream for an FPGA is usually hard to reverse-engineer completely [NR08; BSH12]. However, it is safe to assume that if a motivated attacker has sufficient resources and time, then reverse-engineering is always possible.

Reverse-engineering can also occur later, after the device has been manufactured. From high definition pictures of a delayered chip, automated image recognition software can recover the entire layout [MN08; TJ11; McL11]. More sophisticated imaging devices can be used such as microscopes that use scanning electron, scanning capacitance or X-rays technology [Qua+16]. Using X-rays for example allows an attacker to perform non-destructive reverse-engineering, since the chip is not damaged and can still operate after. On the other hand, a delayered chip is permanently damaged and cannot be used anymore.

In order to go further up in abstraction, recovering the netlist is necessary. This can be done from the bitstream [NR08; BSH12] or the layout. By observing the inputs and outputs of the device, the FSM can also be recovered [Bru+09; Smi+17].

Reverse-engineering can be done by the system integrator. This could help in future designs by not requiring the help of a contract IP core designer anymore. With the knowledge on how to implement a function, this can be done in-house.

A competitor IP core designer could also be interested in the internal architecture of an IP core and attempt to reverse-engineer it. This gives an advantage by reducing design time and



achieving equivalent performance if similar functions must be implemented in the future.

Reverse-engineering can be linked to illegal copying. Indeed, if the reverse-engineering step is successful, the attacker owns a version IP core without the original designer knowing it. The IP core can then be instantiated again, making it an illegal copy.

It is interesting to note that reverse-engineering can also have positive aspects [Qua+16]. It helps in failure analysis and detection. It can also be used to provide intellectual property information and prove that a particular IP core has been instantiated in a device [GDT14]. Moreover, reverse-engineering is often necessary to ensure that a design has not been infected by a hardware Trojan [Xia+16; BFS16]. Finally, this is also a great tool for educational purposes.

### 1.2.4 Limitations bypass

Since IP cores are increasingly following a software-like business model, another type of threat could emerge in the future. Just like pieces of software, IP cores could be distributed in evaluation mode, or offer a premium version with greater performances. So far, only software-assisted manipulations have been demonstrated. For instance, in 2015, a tool was able to disable the hardware locks of processing units of AMD Radeon GPUs<sup>13</sup>. This effectively allows to upgrade a graphics card.

We could not find purely hardware-based attacks, partly because multi-mode IP cores are rare. Therefore, this threat is not addressed in this thesis. However, in view of how well precedented these practises are for software, one can reasonably expect them to apply to IP cores too, once they reach such a level of refinement.

## 1.3 Summary: association between parties and threats

The semiconductor IP business presented in Figure 1.1 can now be extended by showing the different threats on it. Since we aim at providing IP core designers with means of protecting their intellectual property, we should now adopt their point of view when evaluating the trustworthiness of other parties. This is shown in Figure 1.2, in which untrusted parties *from the IP core designer's point of view* are highlighted in dark grey and threats are in red. We considered that the trusted third party, described previously, is indeed trusted. Thus it does not appear in Figure 1.2.

In an attempt to fight these threats, a precise threat model is required. This is presented in the following sections. For each of the previously described threats, a threat model is given, comprising an attacker and a defender model.

---

<sup>13</sup>New tool reawakens disabled hardware in high-end AMD Radeon graphics cards <http://www.pcworld.com/article/2960717/components-graphics/new-tool-reawakens-disabled-hardware-in-high-end-amd-radeon-graphics-cards.html>

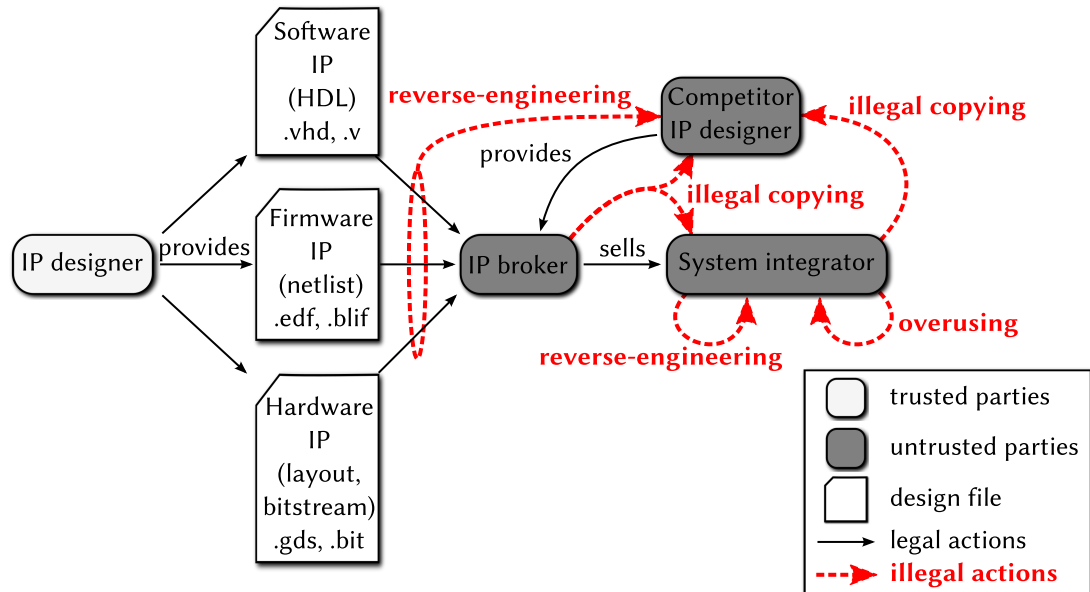


Figure 1.2 – Specific threats to design data in the semiconductor IP business. Trusted and untrusted parties are from the IP designer point of view.

## 1.4 Threat models

### 1.4.1 Threat model for design data exposed to illegal copying/overproducing

The threat model for illegal copying and overproducing is the same. Indeed, the final purpose of illegal copying, after illegal design transfer, is to instantiate the IP core without the designer knowing it. Thus it results in the same consequences as overproducing.

#### 1.4.1.1 Attacker model

**Attacker's objectives** When an attacker aims at carrying out illegal copying or overproducing of an IP core, its objective is to instantiate the IP core more times than agreed with the designer or the broker. From the attacker's point of view, a black box instantiation of a functional IP core is sufficient. Even though some technical characteristics of the core may be required, the knowledge of the internals is not needed to perform the attack.

**Attacker's capabilities** We assume that the attacker can obtain a copy of the IP core in a legal way. He also has the technical resources to instantiate it correctly. Namely, this means that he can obtain all the necessary technical information required such as the process node, the design rules, the foundry, etc.

#### 1.4.1.2 Defender model

**Defender’s objectives** The designer’s objective here is to prevent the attacker from proceeding to a black box instantiation of the IP core without reporting it to the designer. Practically speaking, the designer wants to know how many instances of the IP core exist. However, this does not prevent black box instantiation as described above. In addition, the defender must be able to control how many instances of the IP core actually operate. The fact to know how many instances of an IP core are operating is commonly referred to as *metering* [Kou11].

**Defender’s constraints** From the defender’s point of view, the main constraint to defend against illegal copying is the cost of the protection system. Indeed, adding extra components to the IP core in order to protect it increases the logic resources, the power consumption and possibly the latency of the core. This all comes at a cost, either because the IP core layout occupies a larger area and is more expensive to manufacture or because a higher power consumption or latency makes it less competitive. Therefore, the cost of the protection system must not exceed the financial losses caused by illegal copying or overproducing. However, the financial losses suffered by IP core designers can be hard to estimate.

### 1.4.2 Threat model for design data exposed to reverse-engineering

This threat model addresses reverse-engineering when committed with a malicious intent, in contrast with reverse-engineering aiming at educational purposes.

#### 1.4.2.1 Attacker model

**Attacker’s objectives** When an attacker attempts to reverse-engineer an IP core, the aim is to find out how specifications have been implemented in hardware. Namely, this includes revealing the types of logic gates used and the connections between them or the layout patterns on every layer. The objective is to gain knowledge of the practical implementation methods and techniques, in order to reduce time to market for a future in-house design while achieving similar performances to competitor devices.

**Attacker’s capabilities** The attacker can access both the digital and physical versions of the IP core. The digital version refers to the computer file which holds the design data. For example, this can be a VHDL, GDS II or bitstream file. The attacker can also have access to a physical implementation of the IP core in an integrated circuit. Depending on the financial support he gets, an attacker can use powerful techniques to recover design information [TJ11]. Some companies, such as Texplained<sup>14</sup> are specialised in providing this type of services.

---

<sup>14</sup><https://www.texplained.com/>

#### 1.4.2.2 Defender model

**Defender's objectives** From a defender's perspective, the objective is to conceal the architecture of the IP core. The ideal model for this is a black box, where only the inputs and outputs are visible. However, due to the way IP cores are distributed and supposed to be used, this objective is hard to fulfil.

A more relaxed version, which is at the same time more realistic given today's attackers capabilities, is to make the reverse-engineering as hard and time consuming as possible. Given that the parties who can perform reverse-engineering are the system integrator or a competitor IP designer, the objective of a designer is to make the reverse-engineering process more expensive than in-house development of the IP core.

**Defender's constraints** Similarly to the constraints detailed above for illegal copying and overproducing, the cost of the protection method against reverse-engineering must be lower than the potential financial losses caused by the intellectual property infringement.

As highlighted before, reverse-engineering can have salutary purposes like failure detection or tests. From a practical point of view, a protection against reverse-engineering can make such purposes harder to achieve.

## Conclusion on threats on design data

Due to the emergence of core-based design, overproducing/illegal copying and reverse-engineering have arisen or have been amplified. However, they have slightly different characteristics.

Reverse-engineering is quite a challenging task to perform, and will only become harder with the decreasing size of transistors and their increasing density. In addition, IP designers are more and more aware of this threat and have a large panel of possibilities to fight against it. Nevertheless, the development of automated reverse-engineering tools makes progress too. Therefore, the amount of time taken to reverse-engineer a design is only due to the manual intervention of people which is still required, since not everything can be automated. This still takes a good amount of time and skills. The required tools to physically de-package and process a circuit to reverse-engineer it are costly too. Overall, the potential financial losses for the IP core designer are high, but the increasingly fast time-to-market tends to reduce them if they are restricted to the intellectual property infringement.

Overproducing and illegal copying, on the other hand, do not require much time to be performed by an attacker. Indeed, after obtaining the design, copying it is trivial. However, obtaining it in the first place can cost some money. After the copy has been performed, overproducing a design requires no extra skills than the ones already present in most design houses. Thus the potential losses for the IP designer are much greater. Moreover, these potential losses can also originate from reverse-engineering being used to perform the illegal

copy, beyond the infringement of intellectual property mentioned above. Overall, these are a much more important threat for IP core designers than reverse-engineering. This is summarised in Table 1.1.

Threat	Requirements			Potential financial losses for the IP core designer
	Time	Money (equipment)	Skills	
Reverse-engineering	●●○	●●●	●●○	●●○
Overproducing/Illegal copying	●○○	●●○	●○○	●●●

Table 1.1 – Threats on design data.

In order to fight these threats, many design data protection schemes were developed. They consist in adding specific modules to a design or modifying it directly. These are developed in the following section.

## 1.5 Design data protection methods

Traditionally, design data protection methods are classified into passive and active methods. Passive protection means allow a designer to detect that an illegal action occurred. For example, by embedding an identifier inside an IP core, a designer who obtained a circuit can extract the identifier and prove that his IP core was instantiated. However, this does not prevent the illegal action to occur. Conversely, active protection means offer the designer a way to actively prevent the illegal action. For example, the circuit can exhibit an erratic behaviour until the correct activation word is fed to it.

Here, we chose to further refine this classification by sorting protection means according to the help they provide to the designer. Even though those helps are hard to classify strictly according to their efficiency, we broadly make an attempt to do it. The weakest methods allow to identify an IP core, but not individual instances. Identifying individual instances is necessary to count them and ensure precise metering. On the one hand, offering degraded modes of operation is a good way to prevent illegal copying and overproducing, since illegal copies are then essentially useless. On the other hand, concealing the internal architecture of the IP core can make reverse-engineering prohibitively expensive. Finally, the most efficient methods are referred to as licensing schemes. They are an attempt to transfer the licensing methods used for software to IP cores. This is shown in Figure 1.3.

As the pyramidal structure shown in Figure 1.3 suggests, the most efficient design data protection schemes are often built on top of weak ones, by combining them. For example, a good licensing scheme necessarily requires to identify individual instances of an IP core. Simple IP core identification is also useful in the first place to ensure that the IP core has been instantiated in a particular design. Therefore, we start with weak protections before gradually describing more and more efficient ones.

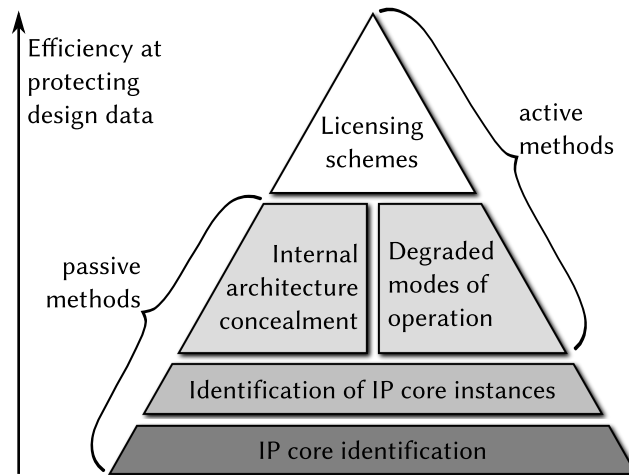


Figure 1.3 – Hierarchy of design data protection methods classified according to their efficiency at protecting design data.

### 1.5.1 Identification of an IP core

In order to detect that one IP core has been illegally copied, the original designer can embed an identifier inside it. Later on, when the designer suspects an IP core to be illegally integrated into a design, the identifier is retrieved to claim ownership. There are multiple ways to generate an identifier inside an IP core. The first one is to store it in a non-volatile memory (NVM) (see Figure 1.4a). The design can also be slightly modified in a way that is known only to the designer, so that this slight modification can later be detected. This is called watermarking and is shown in Figure 1.4b, where the watermark is retrieved via side-channel analysis. Those two techniques have the drawback to identify the IP core but not individual instances. This can be achieved by storing a unique identifier for each instance inside a one-time programmable non-volatile memory (OTP-NVM), as shown in Figure 1.4c. Finally, the physical characteristics of the silicon implementation can be exploited. This can be done by direct measurement, called fingerprinting (see Figure 1.4d) or by embedding a PUF. A PUF is structure that can be challenged, extracts the intrinsic entropy coming from manufacturing process variations and turns it into a binary identifier called a response. This is shown in Figure 1.4e.

With NVM and watermarking, the identifier is identical for all the instances of the IP core. These two methods are detailed below. Conversely, using an OTP-NVM, a PUF or performing fingerprinting allows one to identify individual instances. These methods are then studied in a specific section afterwards.

#### 1.5.1.1 Identifier stored in NVM

The first option to store a fixed identifier is to use an NVM, that set to a value at design time and is non-rewritable. Those memories are typically referred to as mask read-only memory (ROM). Complementary metal-oxide-semiconductor (CMOS) manufacturing process offers several technological possibilities to achieve physical hardwiring of an identifier.

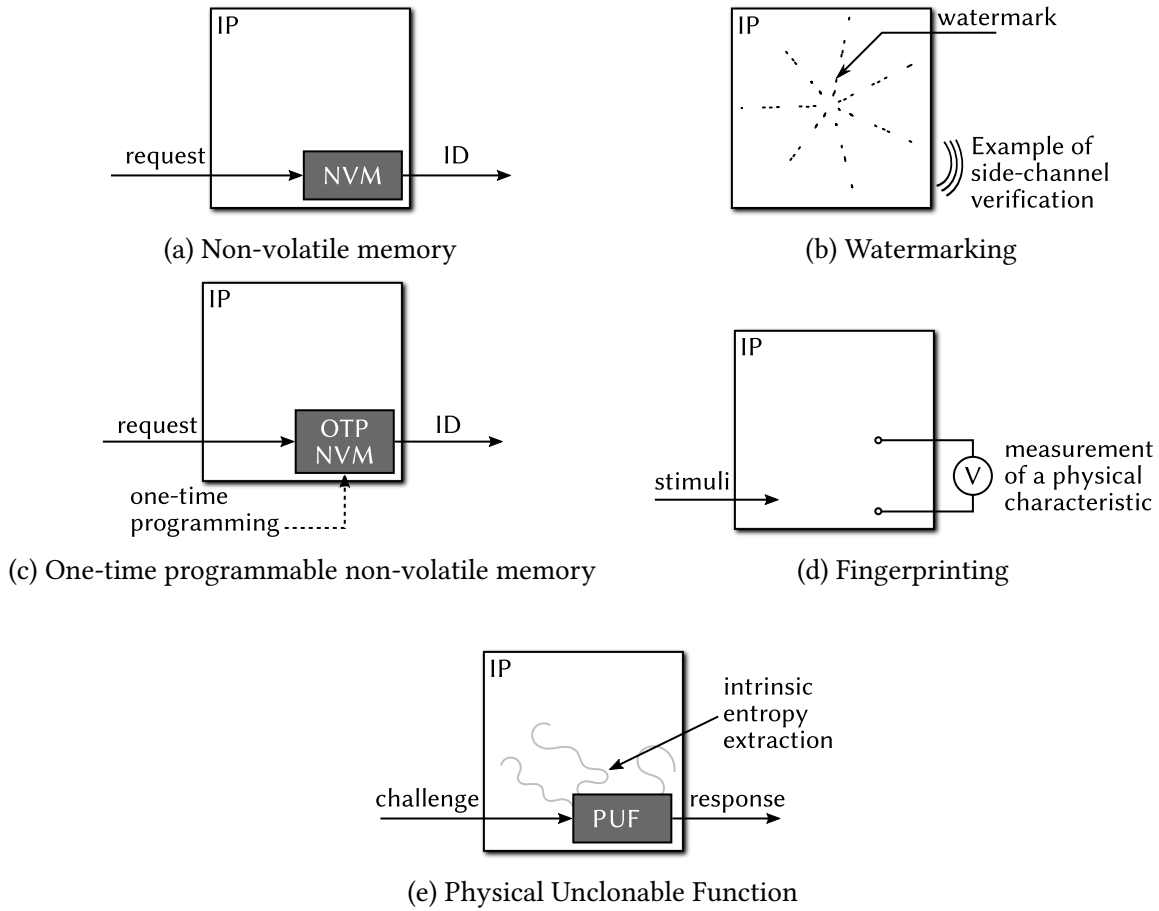


Figure 1.4 – Methods for identifying an IP core itself or the individual instances.

**Contact layer/via mask ROM** The first possibility to implement a mask ROM is to modify the vias, as shown in Figure 1.5a. This is done by removing the connection vias for certain transistors, leaving them unconnected.

**Active layer mask ROM** Mask ROM is implemented by not creating the channel for some transistors (see Figure 1.5b).

**Metal layer mask ROM** The first metallisation layer is used to create a short circuit between the source and drain contacts of the transistor (see Figure 1.5c).

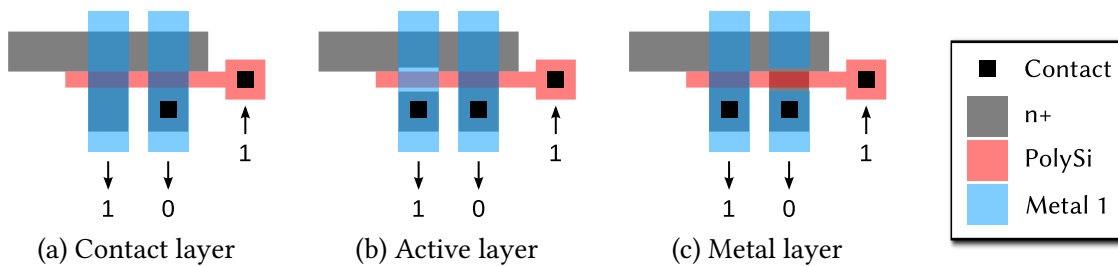


Figure 1.5 – Different types of mask ROM (adapted from [Yen14]).

Mask ROM has the advantage of not requiring any extra steps in the manufacturing process. Moreover, they have a very high density of one transistor per bit stored and require extensive physical processing to be reverse-engineered. Finally, they offer very good resilience to removal or perturbation attacks and can be easily read out.

However, they cannot be modified after the circuit has been manufactured. Reverse-engineering these ROMs is also feasible by de-packaging the circuit and delayering it [TJ11]. With sufficiently precise micro-photography, individual ROM bits are extracted by pattern recognition. Then, the common identifier is known and can be used to fake identity. Finally, designing mask ROM requires the IP core designer to carry out the design steps down to the layout level. This is not possible if the IP core is to be provided in a hardware description language (HDL) format for instance. Therefore, there is a requirement for means of embedding an identifier within an IP core at a higher level of abstraction. This can be done by watermarking techniques presented in the next section.

#### 1.5.1.2 Watermarking

Watermarking consists in modifying a design slightly in order to incorporate a mark into it. This watermark must fulfil eight properties [ATA04].

1. Its structure must be public,
2. It must exhibit a low false positive rate as well as being hard to forge,
3. It should not alter the functionality of the system,
4. It must be hard to modify,
5. It must contain enough data to claim ownership, typically enough bits,
6. It should not induce a too high implementation overhead,
7. It should be easy to detect and track,
8. It could be asymmetric, embedding both a public and a private part.

As said before, a watermark is embedded at a higher abstraction level than the transistor level. Such abstraction level ranges from the layout level up to the algorithmic level. All these are detailed below.

**Layout level watermarking** At the lowest level of abstraction, a watermark is embedded at the layout level. For example, [Kah+01] proposes to modify the placement and routing of an IP core and shows how these modifications can be easily integrated into the design flow of mainstream EDA tools. They demonstrate how configuration bits of unused output multiplexers, path timing constraints or column index may be modified to embed a watermark.



In [Jai+03], delay constraints are generated from the watermark and are embedded in the form of a fixed bit for the least significant bit of specific paths delay.

Those layout-level solutions allow a system integrator to verify that an IP core originates from the correct designer. However, once the IP core has been implemented and the circuit manufactured, those watermarks are hard to retrieve. This strongly limits the application of such techniques, and calls for more usable methods.

**Register transfer level watermarking** Alternatively, the watermarking scheme can be added at the register transfer level. Targeting FPGA designs, authors of [SZT08] proposed to store the watermark in unused LUT entries. This work highlights how a watermark may take advantage of existing unused resources. However, the same problem as for layout-level watermarking arises since a verifier needs access to the bitstream.

To increase verifiability, the test access ports were used. For instance, in [FT03], the watermark is generated along with output of the test circuitry and is verified at test time. The test infrastructure was also leveraged in [CQZ15] where the scan-chain is specifically modified. Depending on the watermark to insert, scan-chain D flip-flops (DFFs) are either connected together by their Q or Q' outputs. This modifies the output obtained from a given test input pattern, allowing one to verify that the watermark is indeed present. This approach has the advantage of incurring very low overhead. However, an access to the scan-chain is required to verify the watermark. This access could be exploited by an attacker to assist reverse-engineering.

To completely alleviate the need for a verification interface, side-channels are a powerful tool to verify a watermark. In 2008, Ziener et al. [ZT08] introduced a new watermarking technique which makes the watermark detectable in the power consumption traces. By driving a large shift register with a smaller one containing the watermark, characteristic power patterns are created. The electromagnetic channel is also suited to this purpose, as shown in [BBF15], where a tiny BFSK<sup>15</sup> transmitter is embedded inside a device to transmit information in a contactless manner. Thermal communication has also been considered by the company Algotronix [MKM08], but has a very low throughput.

**Finite state machine level** At the behavioural level, another good candidate to insert a watermark is the controller of a system, namely the finite-state machine (FSM). Indeed, as mentioned above, it usually has unused resources that can be exploited. For instance, if binary coding of the FSM states is used, then an FSM with  $m$  states requires  $\lceil \log_2(m) \rceil$  registers to store the current state. Therefore, there are  $2^{\lceil \log_2(m) \rceil} - m$  states which could potentially be encoded but are unused. Reading out the state register provides the watermark.

One more option is to add transitions to the state-transition graph that are passed through only after a certain sequence of inputs. The watermark is verified by observing the outputs

---

<sup>15</sup>Binary Frequency-Shift Keying

associated with the traversed states [Oli01; Cui+11].

A known graph can also be embedded into the state transition graph of the FSM [Lew+12]. Inserting the watermark boils down to a graph isomorphism problem or to finding the closest subgraph in order to modify it. Similarly, verifying the watermark requires to transition through the embedded graph states.

Finally, the states encoding itself may be modified to embed a watermark [ZC12]. The state encoding is then extracted by making the outputs dependent on it or by reading out the state register using a scan chain.

**Algorithmic level** Finally, at the highest level of abstraction, a watermark is embedded at the algorithmic level. Targeting digital signal processing applications, [CD00] proposed to modify the parameters of a finite impulse response filter according to the watermark to insert. The response of the filter is then slightly modified, allowing the watermark to be verified.

One more solution is to send out the watermark at the output when those are considered to be not valid [LB12].

#### 1.5.1.3 Conclusion on identification of an IP core

IP core identification allows a designer to embed an identifier into a design and claim ownership. They have the advantage to be deeply tied to the design and hard to remove. Nevertheless, their drawback is that they identify the IP core but not the actual instances themselves. Moreover, they can be hard to set up for software IP cores, for which the high level of abstraction does not allow for low level identification. This makes it impossible for a designer to identify and count instances individually. Therefore, metering, *i.e.* counting the number of operating instances of the IP core, is not possible with this approach.

In order to achieve metering, IP core instances must be identifiable individually. The methods that enable this are presented in the following section.

### 1.5.2 Identification of individual instances of an IP core

Distinguishing instances of an IP core is necessary to provide information feedback to the original designer. Without knowing how many times an IP core is used, the only licensing solution is a front-end payment. Therefore, a designer must be provided with ways to distinguish and count instances. The solutions proposed to this end are presented in the following sections.

#### 1.5.2.1 One-time programmable non-volatile memory

Instead of setting the content of the NVM at design time, a trusted third party or the designer himself writes it once the circuit has been manufactured. Yet this identifier must then be permanently stored inside the device to allow for a read-back later. A potential attacker should

also not be capable of rewriting an identifier of his choice. For this reason, so-called OTP-NVM must be used. There are three types of OTP-NVM available [Sko05]:

**Soft OTP-NVM** The memory is a standard electrically programmable ROM but without erasure interface. This way, the content can be written only once. A typical erasure interface is a quartz window above the die which allows ultraviolet light to erase the content of the memory (see Figure 1.6). By closing this window permanently, the memory cannot be erased anymore.



Figure 1.6 – Examples of integrated circuits embedding an electrically erasable ROM that can be erased by shining UV light through the quartz window<sup>16</sup>.

**Fuses** By default, the value stored in the cell is a logic 1. When setting a high voltage<sup>17</sup> across a conductor, it breaks and turns into an open circuit. Thus only logic 0s are programmed. Some technologies require a laser shot instead of a high voltage to blow the fuse. They have the disadvantage to be programmable only before die packaging, since the laser must be shot on the die directly.

**Anti-fuses** By default, the value stored in the cell is a logic 0. When setting a high voltage across an insulator, a conductive filament is created, turning the insulator into a conductor. Thus only logic 1s are programmed. Some non-volatile FPGAs make use of this technology for their configuration [Mic17a].

These methods, however, require physical access to the device in order to program the identifier into it. They also have a lower density than mask ROM since they require a write circuitry, which is used only once. Providing the high voltage necessary to program the OTP-NVM can lead to area overhead too.

Therefore, a new way to obtain instance-specific identifiers has emerged and is called fingerprinting.

<sup>16</sup>EPROMs 4M, 2M, 256k, 16kbit, by yellowcloud licensed under CC BY 2.0 <https://www.flickr.com/photos/yellowcloud/4525399624>

<sup>17</sup>Higher than normal operation, typically around a few volts.

### 1.5.2.2 Fingerprinting

Just like a human fingerprint is used to derive a unique identifier from random physical characteristics, fingerprinting aims at measuring the realisations of random variables that occurred in a circuit when it was manufactured. Such variations must be fixed so that the identifier is reliable enough. To this end, process variations inherent to CMOS manufacturing may be extracted. If the inter-device variation is sufficient, individual instances are reliably identified. Thus fingerprinting requires to characterise physical parameters of the device by a precise measurement of analog signals. Such parameters can be path delays or transistors threshold voltage for instance.

**Paths delay** The first solution to extract random process variations is to measure the delays of a chosen subset of the circuit paths. By applying the clock-sweeping technique [Tuz+12], individual path delays can be obtained. Gate level characterisation [WKP11] is another technique able to measure the delay at the gate level. It consists in measuring the delay of multiple paths containing a subset of  $n$  gates in order to build a system of  $n$  equations and solve it to recover the individual gates length.

**Transistors threshold voltage** The other parameter which is randomly influenced by process variations is the threshold voltage of transistors. Gate level characterisation is also useful here [WKP11]. By measuring the power consumption of small portions of a circuit involving a subset of  $n$  gates, a system of  $n$  equations can be built and solved to extract the threshold voltage  $V_{th}$  of individual gates.

**Conclusion on fingerprinting** Fingerprinting has the disadvantage of calling for measurement of analog signals to derive the device intrinsic parameters. These are highly dependent on the implementation and would be totally different from one technology node to another. Rapidly, structures have been proposed that can extract random physical process variations and provide a digital “digest”. Such a structure is called a PUF.

### 1.5.2.3 Silicon Physical Unclonable Functions

Formally, a PUF is a physical entity which produces a binary string as a *response* to a request called a *challenge*. Together, they form a challenge-response pair (CRP). As the term “physical” suggests, the information in the binary string depends on physical characteristics of the PUF.

Some PUFs accept a *challenge* before generating a response accordingly. The challenge is typically used to select which parts of the physical structure are operated to generate the response. Accordingly, CRPs  $(c_i, r_i)$  can be obtained. Depending on the number of challenges available, a number of CRPs can be recorded. This is done during the enrolment phase. Moreover, sending an identical challenge to multiple instances of an IP core results in different responses.

Thus those CRPs identify an IP core instance in a unique manner. Therefore, in order to authenticate an IP core, a server can send a challenge to it and wait for the associated response. If it matches the CRP stored in the database, the IP core is authenticated. A toy example of an authentication protocol is shown in Figure 1.7.

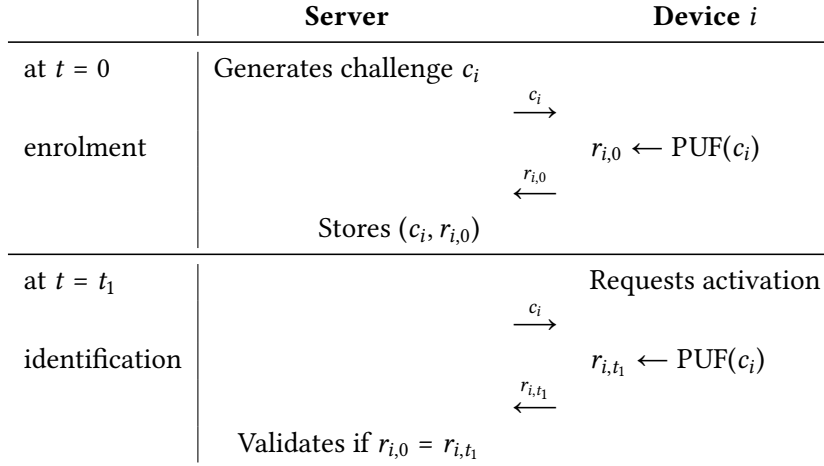


Figure 1.7 – Basic protocol for IP identification using a PUF.

The internal structure of a PUF, since it directly relies on random manufacturing process variations, is supposedly “unclonable”. However, modelling attacks have been mounted [Rüh+10], highlighting the gap between theoretical and practical security for PUFs [Bec15].

In order to evaluate a PUF, two metrics are commonly used [MGS13]. The first one, *steadiness*, characterises the stability of the PUF response over time by giving the average ratio of unreliable response bits. It is given by Equation (1.1), where  $\bar{r}_i$  is a reference response of device  $i$  obtained by averaging the  $m$  samples  $R_{i,t}$ . The difference between a response and the average is given by the Hamming distance HD.

$$\text{steadiness} = \frac{1}{m} \sum_{t=1}^m \frac{\text{HD}(R_{i,t}, \bar{r}_i)}{n} \quad (1.1)$$

The target value for steadiness is 0, which corresponds to a PUF that generates identical responses to the same challenge over time.

Besides stability of responses over time, another criterion which is used to evaluate a PUF is *uniqueness*. It indicates how different the responses obtained from two PUFs implemented on separate devices are. Given  $n$  devices, pairwise comparison of responses obtained from devices  $i$  and  $j$  leads to a definition for uniqueness given in Equation (1.2)

$$\text{uniqueness} = \frac{1}{n(n-1)m} \sum_{i=1}^n \sum_{\substack{j=1 \\ j \neq i}}^n \sum_{t=1}^m \frac{\text{HD}(R_{i,t}, \bar{r}_j)}{n} \quad (1.2)$$

Those two criteria are the most commonly accepted. Some other works proposed to test randomness but the small amount of data which can be gathered leads to a lack of statistical

significance. In order to implement silicon PUFs, several architectures have been proposed. They are presented in the following sections.

**Arbiter PUF** Arbiter PUFs compare the delay of two manufactured paths which were designed to be of identical length [Gas+02]. Due to manufacturing process variations, two paths of the exact same length at design time have a slightly different one after manufacturing. Therefore, by comparing the time of arrival of a signal after it propagated through those two paths, one bit of information can be extracted, depending on which path is the shortest.

In order to obtain the bit of information, an arbiter is used. It is a two-input one-output component which output is 0 if its A input is asserted first or 1 if its B input is asserted first. This is an ideal component.

This can be implemented using a DFF that samples the signal from one path while using the signal from the other path as a clock. The first path is then connected to the D input of the DFF while the second path is connected to the CLK input. If the signal going through the first path arrives first, then the rising edge on the signal of the second path will sample a logic 1. The extracted bit will then be a 1. Conversely, if the rising edge on the clock input occurs while the signal on the first path did not arrive at the DFF yet, a logic 0 is sampled. The extracted bit will then be a 0. Compared to the ideal arbiter component, a DFF can behave erratically if the two signals arrive very close from one another. This could violate the setup and hold times of the DFF, leading to metastability.

Propagation paths can be shared with switch boxes. A switch box has three inputs, two for data,  $i_0$  and  $i_1$ , one for selection,  $sel$ , and two outputs  $o_0$  and  $o_1$ . The output values depend on the  $sel$  input:

$$o_0 = \begin{cases} i_0, & \text{if } sel = 0 \\ i_1, & \text{if } sel = 1 \end{cases} \quad o_1 = \begin{cases} i_1, & \text{if } sel = 0 \\ i_0, & \text{if } sel = 1 \end{cases} \quad (1.3)$$

The path from input  $i_0$  to output  $o_0$  has an identical length to the path from input  $i_1$  to output  $o_1$ . Similarly, the path from input  $i_0$  to output  $o_1$  has an identical length to the path from input  $i_1$  to output  $o_0$ . The associated delays should be equal too. Let us denote the delay from input  $a$  to output  $b$  as  $t_{a,b}$  then we have for the switch box:

$$\begin{aligned} t_{i_0,o_0} &= t_{i_1,o_1} \\ t_{i_0,o_1} &= t_{i_1,o_0} \end{aligned} \quad (1.4)$$

Thus the selection input of the switch box works as a challenge input, allowing one to select either one or the other pair of internal paths.

In an arbiter PUF, multiple switch boxes are chained. The set of selection inputs of all the switch boxes can be seen as an  $n$ -wide challenge input when  $n$  switch boxes are used. One final arbiter is used to sample the signals as described previously. Figure 1.8 shows a schematic of chained switch boxes along with an arbiter, establishing an arbiter PUF.

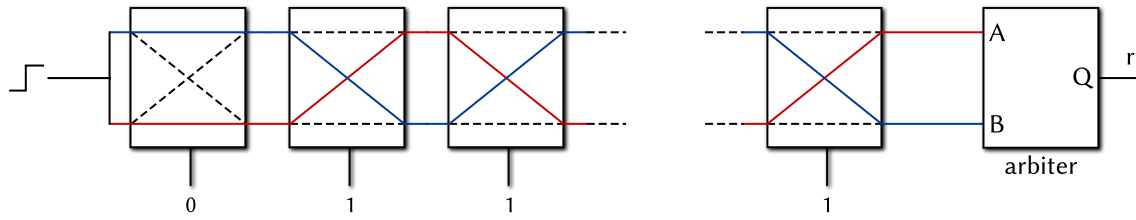


Figure 1.8 – Arbiter PUF with challenge “011 . . . 1” applied, comparing the blue and red path.

On the one hand, arbiter PUFs are easy to implement on ASIC where the path length is geometrically measurable. Two theoretically identical paths can then be easily constructed, as well as a balanced switch box for which Equation (1.4) is verified. On the other hand, the intrinsically constrained routing found in FPGAs prevents such a structure to be implemented on this type of hardware platform [Che+13]. Indeed, two routing paths cannot be made of perfectly equal length on an FPGA, leading to a bias toward 0 or 1 at the PUF output.

Arbiter PUFs have the advantage to incur low area overhead due to the density brought by switch boxes. They provide an exponential number of challenges with respect to the number of switch boxes used, although the responses obtained from these challenges are correlated. Moreover, they extract process variations efficiently and lead to high uniqueness. They also exhibit low steadiness.

**Ring oscillator PUF** Some PUF structures are much more suited for implementation on FPGA targets. Among them, the ring oscillator PUF (RO-PUF) [SD07] structure is easy to implement. It generates a response bit by comparing the frequency of two ring oscillators selected from a pool of theoretically identical ones. A ring oscillator is a chain of an odd number of inverters. In order to make it controllably activable, an AND gate is usually inserted in the chain, with one of its inputs connected to a control signal. This allows to stop the oscillations in the ring oscillator when it is not used, which is useful to limit power consumption and interference between ring oscillators. The output of the ring oscillator cell is tapped from the output of one of the inverters of the chain. A ring oscillator cell is shown in Figure 1.9a.

In order to compare the frequency of two ring oscillator cells, their respective outputs are sent to two counters of the same size. The first counter to overflow shows which ring oscillator has the highest frequency. The result of this comparison is one bit of the PUF response. In a ring oscillator PUF, ring oscillator are then compared pairwise. A multiplexer is used to select which ring oscillators are compared and activate them. The general architecture of a ring oscillator PUF is shown in Figure 1.9b.

Ring oscillator PUFs have the advantage to be easy to design, both on ASIC and FPGA. They exhibit low steadiness and high uniqueness [Mai+10]. Moreover, further refinement in the architecture gives the possibility to extract more than one response bit per comparison. Indeed, instead of simply comparing the frequencies, the counter values can be subtracted [KL16]. Some



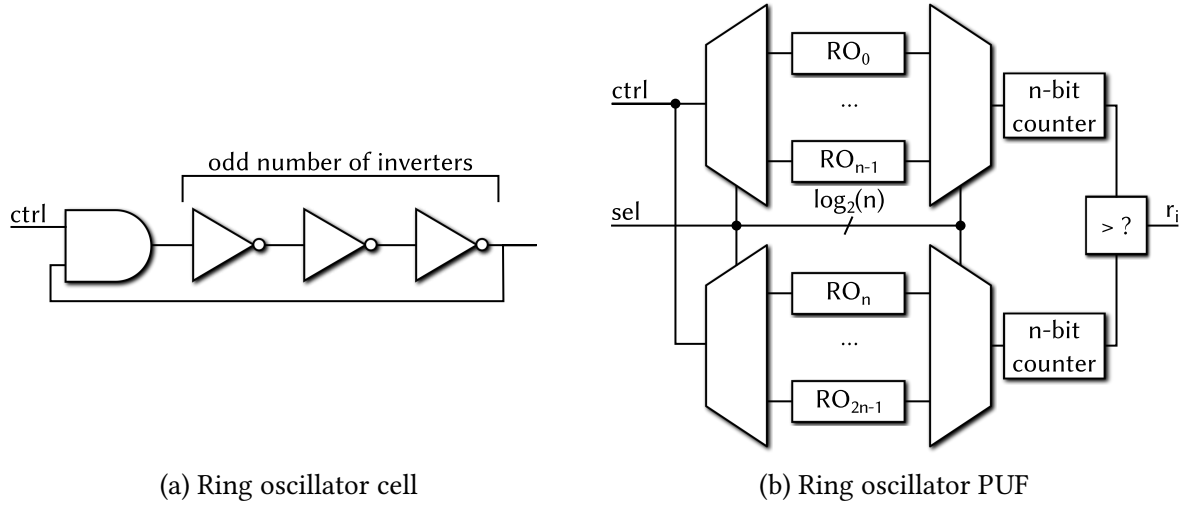


Figure 1.9 – Ring oscillator cell and PUF.

bits of the difference can be exploited as response bits. Previously, with simple comparison, only the sign bit was extracted. However, precise characterisation can determine which other bits are worth using. Indeed, the least significant bits of the difference are greatly affected by noise and can not be used reliably. Yet the most significant bits are not useful either because the counter might never reach sufficiently high values if the counter is over-sized. Characterisation helps to determine the optimal counter size and exploit as many bits as possible.

Nevertheless, ring oscillator PUFs also have drawbacks. They come with high overhead, since the number of possible independent CRPs grows only linearly with the number of ring oscillator cells instantiated. Ring oscillators also have a strong electromagnetic emanation, and are sensitive to electromagnetic attacks in return [Bay+12]. Their frequency can then be modified by electromagnetic injection. Moreover, when multiple ring oscillators are implemented close to one another, they tend to synchronise their frequencies [Boc+10], just like two mechanical pendulums do when they are attached on the same wall. This phenomenon is referred to as “locking”. If two ring oscillators are oscillating at the same frequency, comparing their frequencies obviously makes no sense.

In order to avoid the locking phenomenon, ring oscillators that oscillate only temporarily have been proposed. They are presented in the following section.

**Transient effect ring oscillator PUF** The transient effect ring oscillator (TERO) cell [VDF13] is a controlled ring oscillator but with the control input fed at two stages of the chain (see Figure 1.10). Both top and bottom branches of the TERO cell must have the same propagation delay. When the control signal is asserted, two events propagate in the loop. After some time, one of the events catches up with the other, stopping the oscillations. The number of oscillations is stable enough to be exploited by a PUF.

The transient effect ring oscillator PUF (TERO-PUF) architecture presented in [Bos+14] is similar to the one shown in Figure 1.9b, but a subtractor is used instead of a comparator.



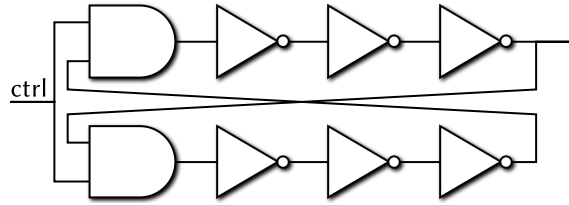


Figure 1.10 – Transient effect ring oscillator cell.

Similarly, multiple response bits can be extracted for each subtraction performed.

TERO-PUFs have very good uniqueness and steadiness characteristics (see [CBM16] for ASIC and [MBC16] for FPGA implementations). However, they are hard to implement on FPGAs, where balancing the two branches of the TERO cell is challenging. Similarly to RO-PUFs, the number of challenges grows linearly with respect to the number of TERO cells.

All the PUFs presented so far require an additional structure to be added to the circuit. Reusing an existing structure could reduce the area overhead. This is what the static random access memory (SRAM) PUF attempts to.

**SRAM PUF** Due to the mismatch between the two inverters of an SRAM cell, when first powered, a logic 0 or 1 is stored. An SRAM PUF exploits this random start-up state of an SRAM array as a response. Obtaining the PUF response then consists only in reading the uninitialised value found at a specific address. The address at which the value is read is the challenge. Therefore, the number of challenges available only grows linearly with the number of SRAM cells.

An initialisation pattern which could be observed in an SRAM array is shown in Figure 1.11. Black cells store a logic 0, white cells store a logic 1. There are some cells, however, which start-up state is not stable. Those grey cells store either a logic 0 or a logic 1. Grey cells are unstable bits of the PUF response while black and white ones are stable.

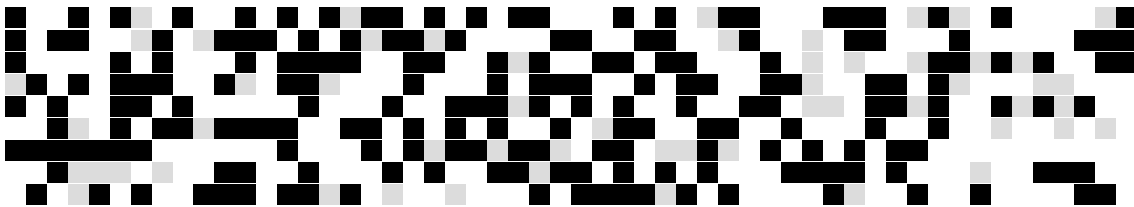


Figure 1.11 – Typical initialisation pattern observed in an SRAM array.

SRAM PUFs exhibit quite high steadiness in general, with a typical error-rate that can reach 10% [CLB11]. Moreover, the argument that it uses existing resources is tenuous since an SRAM array used for a PUF should be reserved for this usage only. Indeed, using it for common temporary data storage can lead to uneven stress of the SRAM cells. This increases both the bias for some bits of the PUF response and the PUF error-rate, since the behaviour of some cells can vary over time.

Nevertheless, they are very easy to implement since no specific tuning step is required. They can be implemented on any electronic system where a memory is present, be it an ASIC, and FPGA or even a micro-controller. Among PUFs, only SRAM PUFs found their way to industrial products, offered by companies such as Intrinsic ID<sup>18</sup>.

#### 1.5.2.4 Conclusion on PUFs

PUFs have been extensively studied in the last twenty years and have proved to be an efficient way to extract an instance-specific identifier for IP cores. The advantages and drawbacks of the considered PUF architectures are summarised in Table 1.2.

Architecture	Advantages	Drawbacks	Industrial adoption
Arbiter PUF	<ul style="list-style-type: none"> <li>• easy to implement on ASIC</li> <li>• low area</li> <li>• high uniqueness and low steadiness</li> </ul>	<ul style="list-style-type: none"> <li>• hard to implement on FPGA</li> <li>• correlation between responses</li> </ul>	×
RO-PUF	<ul style="list-style-type: none"> <li>• easy to implement on ASIC</li> <li>• easy to implement on FPGA</li> <li>• multiple response bits per challenge</li> <li>• high uniqueness and low steadiness</li> </ul>	<ul style="list-style-type: none"> <li>• large area</li> <li>• strong EM interaction</li> <li>• frequency locking [Boc+10]</li> </ul>	×
TERO-PUF	<ul style="list-style-type: none"> <li>• easy to implement on ASIC</li> <li>• multiple response bits per challenge</li> <li>• high uniqueness and low steadiness</li> </ul>	<ul style="list-style-type: none"> <li>• hard to implement on FPGA</li> <li>• large area</li> </ul>	×
SRAM-PUF	<ul style="list-style-type: none"> <li>• easy to implement on ASIC</li> <li>• easy to implement on FPGA</li> <li>• use existing resources</li> <li>• high uniqueness</li> </ul>	<ul style="list-style-type: none"> <li>• high steadiness</li> </ul>	✓

Table 1.2 – Advantages and drawbacks of the considered PUF architectures.

Most of the time, the uniqueness observed is satisfactory and allows to uniquely identify the instances. The problem lies in the steadiness. Indeed, a PUF with a perfectly stable response to the same challenge over time does not exist. As a consequence, some sort of error-correction mechanism must be integrated as well. Classical error-correction codes can be used to this end and are presented in the following section.

#### 1.5.2.5 Error-correction codes for PUFs

The advantage of storing an identifier in an OTP-NVM is that it can be reliably retrieved on demand. In the case of a PUF, however, some of the response bits are not perfectly stable and vary over time with an identical challenge. This change can be caused by power supply voltage variations or environmental electromagnetic noise. Nevertheless, when one needs to authenticate a circuit, the identifier must be reliable.

<sup>18</sup><http://www.intrinsic-id.com>

Traditionally, the way to tackle this issue is to generate helper data from the PUF response obtained at the enrolment phase. Later on, when the PUF is queried again with an identical challenge, this helper data is used to get the error-prone response to match with the response stored on the server.

A very good and thorough overview of helper data algorithms usage with PUFs is given in [Del+15]. These helper data are generated by secure sketches that employ the code-offset or the syndrome construction [Dod+08]. A secure sketch is a primitive which includes two procedures: sketch and recover. The sketching procedure outputs a string  $s$  from an input  $w$ :  $SS(w) = s$ . Later on,  $s$  is used in the recovery procedure to correct the errors in a noisy version  $\tilde{w}$  of the input:  $Rec(\tilde{w}, s) = w$ . Table 1.3 gives details about those two procedures for the code-offset and syndrome constructions.

	Sketch $SS(w)$	Recover $Rec(w', s)$
Code-offset	Select random codeword $c$ (or encode random word) $SS(w) = w \oplus c = s$	$c' = w' \oplus s$ Correct $c'$ to $c$ $w = c \oplus s$
Syndrome	$SS(w) = syn(w) = w.H^T = s$	Find $e$ such that $syn(e) = syn(w') \oplus s$ $w = w' \oplus e$

Table 1.3 – Sketch (SS) and recover (Rec) procedures for code-offset and syndrome constructions of secure sketches.

Proposed schemes found in literature employ either the syndrome [SD07; Her+12; MHV12; Hil+15] or the code-offset [Bös+08; MTV09a; MTV09b; LPS12] construction. The underlying error-correcting codes employed can be a BCH<sup>19</sup> [SD07; Her+12], Reed-Muller [MTV09b; LPS12] or convolutional code [HYS16] for example. In order to increase error tolerance, concatenated codes were used in other works. Typically, a repetition code is concatenated with a BCH [MHV12] or a Reed-Muller code [Bös+08]. In 2015, Hiller et al. [Hil+15] used generalised concatenated Reed-Muller and repetition codes.

Specifically when applied to PUFs, several suited encoding methods have been proposed. Index-based syndrome (IBS) coding [YD10] incorporates bit-specific confidence information and picks the most reliable bit among  $q$ . Complementary index-based syndrome coding [Hil+12] improves on it by repeatedly applying IBS coding to blocks of PUF bits and picking the most and less reliable bit alternatively. Systematic low leakage coding [HYP15] hides the data bits of a codeword by XORing them with other remaining PUF bits. In 2016, another technique called differential sequence coding [HYS16] stores the distance between stable PUF bits and the exclusive-or of the PUF bit and a known codeword bit. Although these solutions can reduce the error-rate, an additional error-correcting code is always required to reach acceptably low failure rate values.

<sup>19</sup>Bose, Chaudhuri, Hocquenghem

All these methods, however, have the drawback to occupy a significant amount of resources on the device side. Moreover, they often need a great amount of PUF bits in order to obtain sufficient final entropy to generate a 128-bit key. Table 1.4 shows implementation results of the presented schemes on FPGA when given in the original articles. The implementations that achieve the best performance for the considered criteria are in bold. These schemes can accommodate quite high error-rates, around 15% on average. With constant improvements coming for PUF implementations, such high error rates are less likely. Typically, RO-PUF [Mai+10] and TERO-PUF [MBC16] implementations have an average error-rate below 5%. Reducing the acceptable error-rate leads to less complex codes and more efficient hardware implementations in terms of occupied logic resources.

Article	Logic resources (Slices)		Block RAM Bits	Failure rate	Acceptable error-rate	PUF bits required for 128-bit entropy
	Spartan 3	Spartan 6				
[Bös+08]	168		0	$1.49 \times 10^{-6}$	15%	4640
[MTV09b]	164		192	$10^{-6}$	15%	1536 (12×128)
[MHV12]		221	0	$10^{-9}$	13%	2226
[Hil+12]	250		0	$10^{-6}$	15%	>1536 (12×128)
[Her+12]		<b>&gt;59</b>	0	$10^{-6.97}$	<b>21.6%</b>	1785
[Hil+15]		179	0	$1.48 \times 10^{-9}$	14%	<b>&gt;130</b>
[HYS16]	<b>75</b>	<b>27</b>	10752	$10^{-6}$	15%	974

Table 1.4 – Logic resources required by the presented error-correction schemes on FPGA.

## Conclusion on identification of IP core instances

In order to uniquely identify IP core instances, taking advantage of random manufacturing process variations is definitely a good solution. To this end, PUFs are very good candidates. Most of them exhibit good uniqueness which means that the probability that two instances share an identical identifier is negligible. Therefore, individual IP core instances can be identified, which is the basic requirement to achieve metering.

The errors observed in PUF responses, however, are an issue. Indeed, by requiring the instantiation of an error-correction core, the logic resources occupied by the PUF grow dramatically. Thus low overhead error-correction solutions are developed and progressing [Her+12; Hil+15]. They are required to improve the stability of PUF responses and make PUFs a usable hardware root of trust. Nevertheless, identifying instances of an IP core is not enough to prevent illegal copying or reverse-engineering. To this end, modifying the design itself is necessary. The aim of these modifications is to prevent the illegal action from happening, making it prohibitively hard to carry out. Indeed, it is important to note that the goal here is not absolute security. Instead, making illegal actions sufficiently costly is considered sufficient.

### 1.5.3 Internal architecture concealment

Protecting the intellectual property against reverse-engineering can be done at different levels. The aim is to prevent an attacker from recovering the internal architecture of a design.

### 1.5.3.1 Split manufacturing

In order to hide the architecture of a design, the first method is to perform split manufacturing. Manufacturing a chip comes in two parts, the front end of line (FEOL) and the back end of line (BEOL), as shown in Figure 1.12.

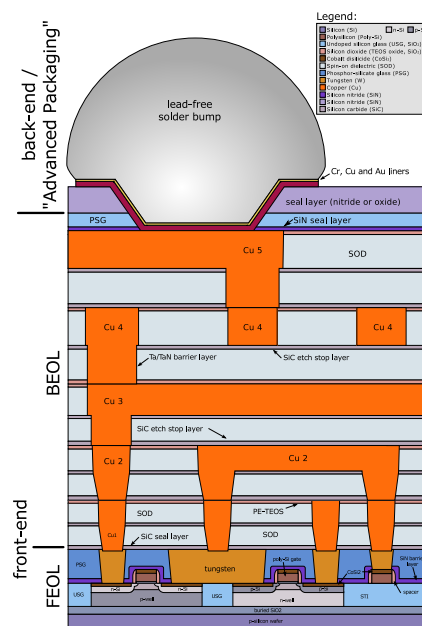


Figure 1.12 – Front end of line and back end of line layers in the CMOS manufacturing process<sup>20</sup>.

The FEOL is the set of layers that incorporate the smallest features like transistors, capacitors and resistors, without interconnect. The BEOL includes all interconnects, which are larger.

Split manufacturing consists then in having the untrusted foundry to manufacture only the FEOL part. Thus the finest process node available can be used to implement the individual transistors. Afterwards, the devices are shipped to a trusted foundry, which performs the remaining manufacturing steps of the BEOL [Ime+13], where the features do not need to be so small. 3D integration allows for a good assembly of the parts that were manufactured in different foundries [Huf+08]. An attacker who has access only to the FEOL design would have to reconstruct the whole interconnect network.

However, the security of split-manufacturing is questioned. Indeed, FEOL features that are connected are usually not far from one another, leading to the possibility of mounting a so-called proximity attack [RSK13]. Therefore, a way to modify the design as a whole is required. This is the aim of logic obfuscation.

<sup>20</sup>Cmos-chip structure in 2000s, by Cepheiden licensed under CC BY-SA 3.0 [https://commons.wikimedia.org/wiki/File:Cmos-chip\\_structure\\_in\\_2000s\\_\(en\).svg](https://commons.wikimedia.org/wiki/File:Cmos-chip_structure_in_2000s_(en).svg)

### 1.5.3.2 Logic obfuscation

The second way to hide the internal architecture of an IP core is to use logic obfuscation. In a software context, a definition of obfuscation is proposed by [Hac03]:

*Transform a program  $P$  into another program  $P'$  harder to reverse engineer with the same observable behaviour.*

We can apply this definition to our use case simply by replacing the program by the IP core. The observable behaviour are the outputs of the core. Making the design harder to reverse-engineer can be done at several level of abstraction, from the gate-level to the source code. Optimally, this should only allow for a black box usage of the IP core.

**Obfuscation of the hardware implementation** At the lowest level of abstraction, the logic function of individual logic gates can be obfuscated. For example, the company Syphermedia [Coc+14] offers logic gates that look the same even though they achieve a different logic function. By modifying the gate topology, as shown in Figure 1.13, the NAND (Figure 1.13a) and the NOR (Figure 1.13b) gates look the same.

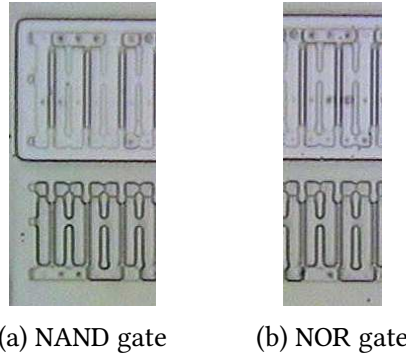


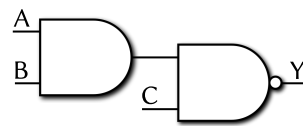
Figure 1.13 – Active layer of Syphermedia gates [Coc+14].

Exploring this idea further, a standard structure can be made programmable to turn it into any logic gate. This reconfigurable element can simply be a  $k$ -input look-up table (LUT), as presented in [BTZ10]. In [Raj+13], a structure which can act as an XOR, NAND or NOR gate is described. It contains 19 contacts that change the functionality of the gate depending on which of them are real or dummy. The number of achievable logic functions was extended in [McD+16] by implementing a so-called *polygate*. The polygate is described as a  $\{0, 1\}^2 \times \{0, 1\}^3 \rightarrow \{0, 1\}$  function. It implements any of the standard 2-input logic gates with 3 configuration bits. Another idea, developed in [SHF14], consists in changing the dopants polarity to configure a diffusion programmable ROM cell, to either 0 or 1. This allows to design cells that act as an inverter or a buffer [Mal+15]. All the layout layers are the same except the dopant layer, making reverse-engineering from a layered circuit very difficult. These cells are aggregated around a 4-input NAND gate, one on each input and one at the output [Mal+15]. Depending on the dopants, up to 162 different logic functions can be implemented.

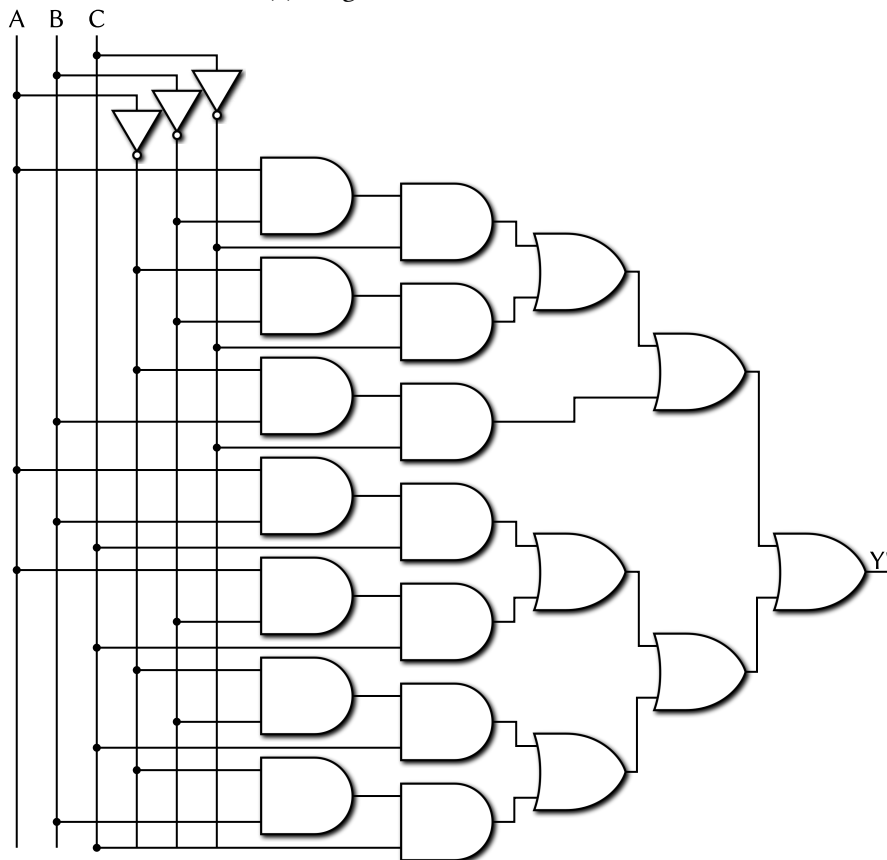
The more complex the solutions the larger the induced area overhead. In [McD+16], the area overhead ranges from 200 to 1800%. In [Mal+15], it goes from 311 to 770%. To maintain a reasonable overhead, only a few strategic gates of the circuit can be modified. For example, in [Mal+15], only the S-boxes of the PRESENT cipher are obfuscated.

Another solution to make reverse-engineering harder is to exploit the laws of Boolean algebra “backward” [CBH16a]. For example, the implementation can follow the disjunctive normal form or the conjunctive normal form strictly, using only AND and OR gates and inverters. The function  $Y = \overline{A} \cdot \overline{B} \cdot \overline{C}$ , whose schematic is shown in Figure 1.14a, can be rewritten in canonical disjunctive normal form (see Equation (1.5)), using AND, OR and NOT gates. The associated schematic is shown in Figure 1.14b.

$$Y' = A \cdot \overline{B} \cdot \overline{C} + \overline{A} \cdot \overline{B} \cdot \overline{C} + \overline{A} \cdot B \cdot \overline{C} + A \cdot B \cdot \overline{C} + A \cdot \overline{B} \cdot C + \overline{A} \cdot \overline{B} \cdot C + \overline{A} \cdot B \cdot C \quad (1.5)$$



(a) Original boolean function

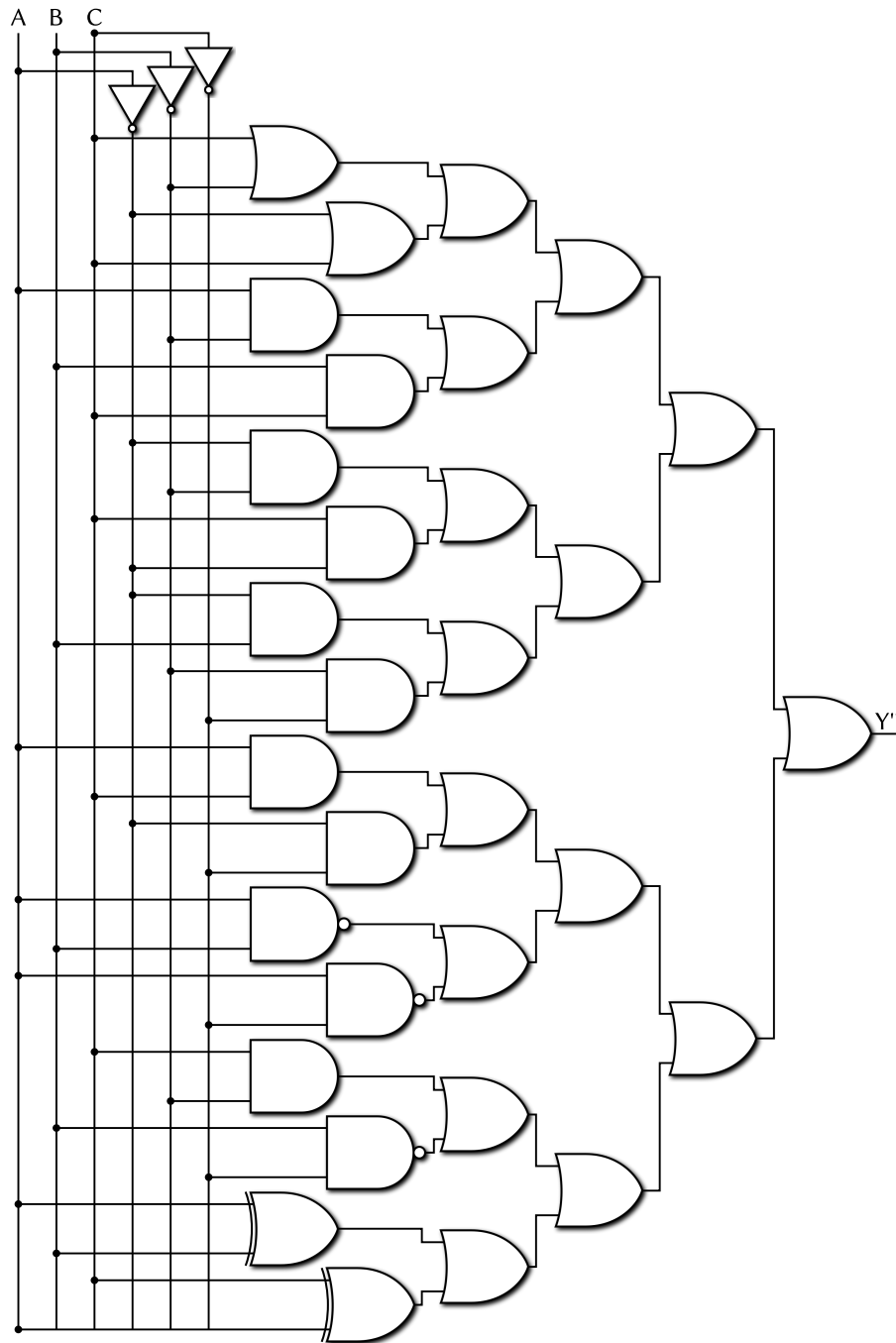


(b) First step of obfuscation

Figure 1.14 – Logic obfuscation of a boolean function

A second step of logic obfuscation can further obfuscate the logic function. The aim is to increase the number of gates used for the implementation. An example of backward usage of boolean laws is given in Equation (1.6), with the corresponding schematic shown in Figure 1.14c.

$$Y'' = A \cdot \bar{B} + \bar{A} \cdot \bar{B} + \bar{A} \cdot B + \bar{B} \cdot \bar{C} + \bar{A} \cdot \bar{C} + A \cdot C + \bar{B} \cdot C + \bar{A} \cdot C + B \cdot C + \bar{A} + \bar{B} + C + \bar{A} \cdot \bar{B} + A \oplus C + A \oplus B + \bar{A} \cdot \bar{C} + \bar{B} \cdot \bar{C} \quad (1.6)$$



(c) Second step of obfuscation

Figure 1.14 – Logic obfuscation of a boolean function.



Following this concept a step further, only universal logic gates, NAND or NOR, can be allowed for implementation [PVK16]. Obviously, the area overhead remains very high in all these cases.

Dummy logic cells can also be inserted into the layout [Coc+14; PVK16]. By making the layout very dense, those additional gates are hard to distinguish from the original ones.

Structural information may be obfuscated too. In [PVK16], they propose to make the routing look “generic” by placing the logic gates on a grid. It makes routing less identifiable by reverse-engineering tools. When an IP core is implemented, the boundaries of individual sub-components is usually visible. A boundary-blurring technique is presented in [Par+10] that makes sub-components overlap.

All those layout-level techniques can be efficient but they all require significant area overhead. Therefore, they cannot be applied to a whole design but must be focused on strategic locations instead.

**Design files obfuscation** When an IP core is not provided as a layout, it is usually in the form of a file written in an HDL. To obfuscate these files, several techniques exist, mostly inspired by those already used in software engineering. Those modifications [OM95; BY07; Mey+11] include replacing locally static expressions by their values, adding dummy structural layers, adding dummy variables, renaming variables<sup>21</sup>, loop unrolling, etc. An example of VHDL obfuscation is given in Figure 1.15, where the variables name have been changed and the indentation has not been followed.

```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;

ENTITY full_adder IS
    PORT (
        a      : IN  STD_LOGIC;
        b      : IN  STD_LOGIC;
        c_in   : IN  STD_LOGIC;
        q      : OUT STD_LOGIC;
        c_out  : OUT STD_LOGIC);
END ENTITY full_adder;

ARCHITECTURE rtl OF full_adder IS
BEGIN -- ARCHITECTURE rtl

    q <= a XOR b XOR c_in;
    c_out <= (a AND b) OR (c_in AND (a XOR b));
END ARCHITECTURE rtl;
```

(a) VHDL description of a full-adder

```
library ieee;use ieee.std_logic_1164.all;entity
II1I100O0O is port(iOOO0101oioi,OOO000iIIiOoO1
,I101OO1OOI:in std_logic;II1I1000OO,I11II0000O:
out std_logic);end entity II1I100O0O;
architecture OOOOO of II1I100O0O IS begin
II1I1000OO<=iOOO0101oioi xor OOO000iIIiOoO1
xor I101OO1OOI;I11II0000O<=(iOOO0101oioi and
OOO000iIIiOoO1)or(I101OO1OOI and(iOOO0101oioi
xor OOO000iIIiOoO1));end architecture OOOOO;
```

(b) Obfuscated VHDL description of a full-adder

Figure 1.15 – An example of VHDL design files obfuscation.

<sup>21</sup>A classic example consists in replacing variable names by a sequence of ones, zeroes, and the letters 'I' and 'O'.

Although these modifications are very easy to achieve, they are essentially useless after the design is synthesised. This is indeed one role of the synthesiser to get rid of all the dummy elements that were added. Moreover, the software engineering ecosystem is full of tools that can automate de-obfuscation. Specifically for FPGAs, the bitstream used to program the target can be compressed [VMV13]. Even though a bitstream can look undecipherable at first sight, it turns out to be quite structured and easy to remap to a netlist [NR08; BSH12].

To reach a higher level of concealment, encryption must be used instead of simple obfuscation. This is detailed in the next section.

### 1.5.3.3 Design files encryption

In order to conceal the architecture of a design, encryption is a useful tool. It goes further than obfuscation by preventing black-box instantiation of a design without a valid decryption key.

Most of the EDA tools integrate encryption and decryption capabilities for design files. For instance, Cadence offers ncprotect while Mentor Calibre can also encrypt and decrypt design files. These tools make use of the principles of public-key cryptography so that designers can distribute their design files securely.

For FPGAs, bitstream encryption is a very common feature nowadays. Both Intel [Alt09] and Xilinx [Wil15] EDA tools allow a designer to encrypt a bitstream. Since FPGAs are more and more complex, they now integrate a symmetric cryptographic core which is in charge of decrypting the bitstream when the FPGA is configured.

The adoption of bitstream encryption for most of the products by FPGA vendors is quite recent. Previously, solutions originating from academia have also been proposed [Gas+12; MSV12; BCM16]. They all exploit partial reconfiguration features to allow for secure configuration.

The wide adoption of bitstream encryption by EDA tools vendors shows that this IP protection scheme is effective. With the cost per transistor constantly decreasing, implementing a symmetric cipher in an FPGA is now easily feasible. However, Moradi *et al.* [Mor+11; Mor+13; MS16] showed that those implementations are vulnerable to side-channel attacks.

### 1.5.3.4 Conclusion on internal architecture concealment

Hiding the internals of a design can prove very efficient at deterring attackers. The previously described methods are well implemented and handled at different stages of the design process. For example for FPGAs, the bitstream encryption is done by the EDA tool while the decryption is done at runtime by the hardware. The impact on the standard design flow is then limited. Measuring the efficiency of protection consists in estimating the amount of time required by an attacker with a specific amount of funds available to reverse-engineer the design. The adoption of these also depends on the impact they have on the performance of the IP core. Although bitstream encryption for instance does not alter the performances, split manufacturing can induce additional delay in the interconnections [Hil+13].

### 1.5.4 Degraded modes of operation

The other solution for modifying a design to prevent illegal actions is to incorporate a degraded mode of operation into it. By default, the design operates in degraded mode. For normal usage, it can then reach the correct mode of operation but only on certain input conditions. These specific input conditions should be sufficiently hard to achieve from an attacker point of view but easy to provide for the original designer. This effectively makes the design *activable*. Most of the time, the activation is done by setting a specific value on a dedicated activation input.

There are two possibilities to make a design unusable. The first one is to alter the outputs in a seemingly random way, so that the correlation between the normal and altered outputs is as low as possible. We refer to this as *logic masking*. In this case, the outputs are altered as much as possible and the alteration depends on the value fed to the activation input of the design. For all input combinations but the correct one, the outputs of the netlist are altered. The second solution is to force the outputs of the design to a fixed value. We refer to this as *logic locking*. As opposed to masking, for locking, no matter what the value that is fed to the activation input is. For all input combinations but the correct one, the outputs of the IP core remain the same.

#### 1.5.4.1 Logic masking

Logic masking was first proposed in 2008 [RKM08a; RKM10]. Several terms are found in literature for this method. Originally coined as “logic locking”, even though no actual *locking* is performed, it has been successively called “logic obfuscation” [LT15], or “logic encryption” [Raj+15], although this cannot be related at all to obfuscation as defined in [Hac03] or to encryption in a cryptographic context. A formal definition of these terms is proposed in [CBH16a]. Logic masking consists in inserting linear (XOR, XNOR) gates at specific locations in the netlist, controlled by an activation input on which an activation word (AW)<sup>22</sup> must be fed. This makes it possible to controllably invert some nodes, altering the internal state of the netlist. The inserted gates have one of their inputs connected to the node, while the other acts as a activation input. The choice between inserting an XOR or an XNOR gate is dependent on the associated activation bit. If the activation bit is a 0, then an XOR gate is inserted. The node is then inverted if the wrong activation bit, a logic 1, is fed. Similarly, if the activation bit is a 1, an XNOR gate is inserted. This is summarised in Figure 1.16.

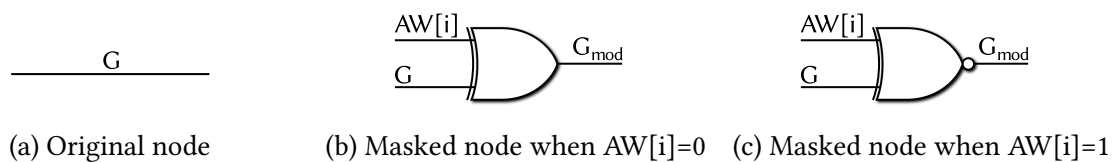


Figure 1.16 – Original and masked nodes depending on the associated activation bit.

<sup>22</sup>We deliberately use “activation word” instead of “key” to not imply any cryptographic property.

Alternatively, instead of modifying simple nodes, inverters can be replaced. In this case, the corresponding activation bit is inverted. If an XOR gate is inserted to replace an inverter, the associated activation bit is 1. Similarly, if an XNOR gate is inserted to replace an inverter, the associated activation bit is 0.

**Masking efficiency evaluation metrics** By modifying the internal state of the IP core, the outputs are modified too. The point is to disrupt them greatly so that they differ as much as possible from the non-masked outputs. Originally, in [RKM08a; RKM10], this was dealt with by ensuring that only one AW is valid, *i.e* able to make the IP core operate normally. Let  $C(x)$  be an  $l$ -input combinational netlist and  $C(x, k)$  be a masked version of it. Then ensuring that only one AW is valid can be expressed by Equation (1.7).

$$\exists! AW_{\text{valid}} \mid \forall x \in \{0, 1\}^l, C(x, AW_{\text{valid}}) = C(x) \quad (1.7)$$

However, this requirement is not restrictive enough to ensure strong logic masking. Indeed, while it imposes a condition on the valid AW, it does not deal with invalid ones. Namely, there is no requirement on the degree of disturbance observed at the outputs when the wrong AW is fed to the IP core.

Later on, [Raj+12a; Raj+13] a criterion on the Hamming distance was proposed. On average, when a wrong AW applied, the Hamming distance between the normal and masked outputs should be as close as possible to 50%. Exhaustive search over the input patterns, both activation inputs and primary inputs, is not feasible, so simulation must be carried out with  $m$  random input patterns. In [Raj+12a; Raj+13] for example, 1000 input patterns were simulated.

When simulating, a subset in primary input patterns is chosen. Let us denote such set as  $I_{\text{inputs}}$ . This set is a subset of  $\{0, 1\}^l$  and has a cardinality of  $r$ . A subset of activation input patterns is chosen. Let us denote such set as  $I_{\text{AWs}}$ . This set is a subset of  $\{0, 1\}^n$  and has a cardinality of  $s$ . Then the requirement on the Hamming distance (HD) between normal and masked outputs is expressed in Equation (1.8).

$$\forall I_{\text{inputs}} \subset \{0, 1\}^l, \forall I_{\text{AWs}} \subset \{0, 1\}^n \setminus \{AW_{\text{valid}}\} \mid \#I_{\text{inputs}} = r, \#I_{\text{AWs}} = s, \quad (1.8)$$

$$\lim_{(r,s) \rightarrow (2^l, 2^n)} \frac{1}{r \cdot s} \sum_{\substack{x \in I_{\text{inputs}} \\ AW \in I_{\text{AWs}}}} \text{HD}(C(x, AW), C(x)) = 0.5$$

In order to fulfil these requirements, the locations of the inserted masking gates matters a lot. Several heuristics have been proposed over the years and are presented in the following section.

As a side note, logic masking can also be considered to be a form of internal architecture concealment. Indeed, the functionality of the inserted gates being unknown, it makes reverse-engineering the netlist harder.

**Nodes selection heuristics** In the original article by Roy, Koushanfar and Markov [RKM08a], the netlist nodes to mask were selected at random. However, as pointed out in [Raj+12a], this method is not very efficient at altering the outputs and the Hamming distance between normal and masked outputs remains low. Rapidly, new heuristics were proposed to select more suitable nodes. In 2009, in the HARPOON design methodology [CB09], the fan-in and fan-out cones of nodes were exploited. A so-called *suitability* metric is computed, shown in Equation (1.9), where FI and FO are the fan-in and fan-out values for the considered node,  $FI_{\max}$  and  $FO_{\max}$  are the maximum fan-in and fan-out values found in the netlist and  $w_1$  and  $w_2$  are normalisation weights which are best set to 0.5. Intuitively, this metric is maximised for nodes that have either a large fan-in or fan-out, or both.

$$M_{\text{node}} = \left( \frac{w_1 \cdot FO}{FO_{\max}} + \frac{w_2 \cdot FI}{FI_{\max}} \right) \times \frac{FO \cdot FI}{FI_{\max} \cdot FO_{\max}} \quad (1.9)$$

Later on, [Raj+12b] improved on the random selection heuristic. They identify several cases in which the masking gates are not inserted optimally, allowing an attacker to propagate the activation bit at one of the primary outputs. They define the notion of *interference graph* to represent the interaction between masking gates. Ideally, this graph should be complete<sup>23</sup>, indicating that the masking gates have maximum interaction with one another. This was refined in [Raj+13] with a *corruptibility* metric, ensuring that the outputs are corrupted when the wrong AW is fed to the design. All these approaches have the advantage that their associated metric is easy to compute. Thanks to this, large netlist can be handled and masked. The masking efficiency, however, is quite low for these methods, and the correlation between normal and masked outputs remains high.

A different approach was adopted in [Raj+12a; Raj+15] and is based on fault-analysis. This time, the metric computed for every node of the netlist is called the *fault impact*, detailed in Equation (1.10). The number of patterns that detect a stuck-at-0 fault at the output of the gate is called  $NoP_0$ , while the total number of output bits affected by this fault is called  $NoO_0$ .  $NoP_1$  and  $NoO_1$  are defined in a similar way for stuck-at-1 faults.

$$\text{fault impact} = NoP_0 \cdot NoO_0 + NoP_1 \cdot NoO_1 \quad (1.10)$$

Since it exploits fault analysis, this method requires a dedicated fault simulator to compute the values of  $NoP_0$ ,  $NoO_0$ ,  $NoP_1$  and  $NoO_1$ . The tasks performed by such software are usually computationally demanding. Moreover, authors of [Raj+12a; Raj+15] propose to insert the masking gates iteratively. After inserting a masking gate on the node that maximises the fault impact, the fault impact values are recomputed for all the nodes in the netlist. Therefore, the nodes selection heuristic is at the same time computationally expensive and intrinsically sequential. In [Raj+15], it is reported that it takes two hours to analyse and mask a netlist of

<sup>23</sup>A graph is *complete* if every pair of vertices is connected by a unique edge.

3,500 nodes. Thus even though this method achieves efficient masking, integrating it in EDA tools is unrealistic. A possible speed-up is presented in [GGY15] but requires to implement a masking gate on every node of the netlist before programming it and performing an emulation on FPGA. For very large netlists, this is clearly impractical. A summary of the strict separation between masking efficiency and computational simplicity for existing nodes selection heuristics is shown in Table 1.5.

Heuristic	Masking efficiency	Computational simplicity
Random [RKM08a]	×	✓
Fan-in/out [CB09]	×	✓
Interference graph [Raj+12b]	×	✓
Corruptibility [Raj+13]	×	✓
Fault analysis [Raj+15]	✓	×

Table 1.5 – Masking efficiency opposed to computational complexity for existing nodes selection heuristics. The symbol × means that the property is not fulfilled, the symbol ✓ means that the property is fulfilled.

Those solutions aim at being integrated into EDA tools. This way, designers could add masking gates to their design on the fly. The computational complexity of the selection heuristic is then a strong requirement. Obviously, the masking efficiency should be optimised as well. From what can be observed in Table 1.5, there is room for selection heuristics that offer a trade-off between masking efficiency and computational complexity.

The other solution to make an IP core unusable is to force the outputs to a fixed logic value until the valid unlocking word is fed. This is referred to as logic locking and presented below.

#### 1.5.4.2 Logic locking

Logic locking, just like logic masking, aims at making an IP core unusable until the valid AW is fed to it. However, instead of disrupting the outputs as much as possible, those are simply forced to a fixed logic value. This is expressed in Equation (1.11) where  $y_{locked}$  is the value at which the outputs are forced when the IP core is locked. Equation (1.7) relative to the uniqueness of  $AW_{valid}$  still holds for logic locking.

$$\exists! y_{locked} \mid \forall AW \in \{0, 1\}^n \setminus \{AW_{valid}\}, \forall x \in \{0, 1\}^l, C(x, AW) = y_{locked} \quad (1.11)$$

The works presented in Chapter 2 of this thesis are the first to deal with logic locking at the combinational level. Previous work focus on higher levels of abstraction and are presented in the following sections.

**Locking FSM** The first proposition is named *boosted* FSM [AK07]. It consists in adding states before the start-up state of an FSM. This is pictured in Figure 1.17, in which the original state machine is in light grey while the added states are in black.



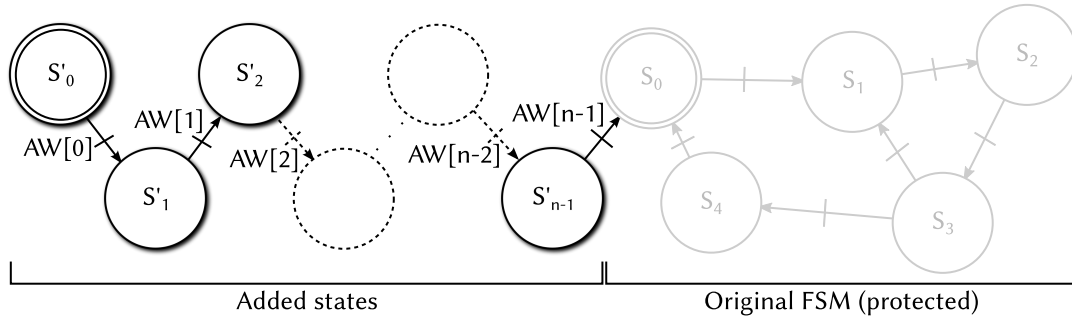


Figure 1.17 – Boosted FSM with added states and transitions in black and original states and transitions in light grey.

In the original article [AK07], the start-up state is determined by setting the state register with the output of a so-called random unique block, which is in fact a PUF. If the number of added states is large compared to the number of original states, then the probability to start in the added states is great. When a system integrator must activate the IP core, the state register value is sent to the designer, who then sends back the sequence of activation bits that lead to the original start-up state. In order to maximise the number of traversed extra states, the FSM can be set to the added initial state  $S'_0$  when reset.

While the system is in the added states, it does not operate. The outputs can be locked while the system is in these states, achieving logic locking. These states can also be used to apply logic masking on certain nodes [CB09]. When the original FSM is reached, the system operates normally.

The boosted FSM can be extended with so-called *black-hole* states [Kou12]. Once the system reached one of these states, it cannot come back to the original FSM anymore. It prevents brute-forcing of the sequence of activation bits. However, an attacker can then reset the system and start again.

Implementing locking at the FSM level has the advantage to be able to exploit unused states which can be encoded in the state register. However, all the extra transitions to add between these states still need combinational logic, leading to quite high overhead [Kou12].

**Input/output locking** The inputs and outputs of the circuit can integrate anti-fuses to achieve locking. As shown in [BZB14], an anti-fuse can be easily integrated in a general purpose input-output pin of a circuit. When the correct key is fed to the device, the correct anti-fuses are blown and the associated ports are unlocked. Otherwise, if programmed with the wrong key, the port is unusable.

Adding anti-fuses to a circuit requires specific write circuitry with a higher voltage than the device core. Thus placing the fuses at the input-output ports is a good option since higher voltages can be found there. This solution has also the advantage to be able to detect recycled circuits. Indeed, if some fuses of the circuit are already blown when a customer receives it, then it is clear that the circuit has already been used before.

**Communication bus locking** In complex IP cores, a communication bus is usually used to interconnect the modules efficiently. For instance, the AMBA architecture is from ARM [ARM17], Intel has a bus system called Avalon [Int17] and even the Opencore open-source repository proposes the Wishbone bus [Ope10]. By controllably scrambling the bus, the information transiting in it can be corrupted and made unusable [RKM08b]. This is achieved using a Beneš network, which is a grid of switch boxes as used by an arbiter PUF, described in Equation (1.3). This solution has the drawback to insert extra components on the paths where information transits on a chip. This necessarily induces delay, which is often critical for interconnection buses.

#### 1.5.4.3 Conclusion on degraded modes of operation

Offering degraded modes of operation for an IP core is a way to implement an activation scheme. Before activation, the system does not operate correctly. Once the correct activation word is fed to it, it reaches normal operation. This is the first step toward a licensing scheme.

Some more advanced degraded modes of operation are also possible. For example, following the model proposed for pieces of software, a demonstration mode with limited functionality or performance can be available [Par+09]. Another possibility is to offer the demonstration mode for a limited period of time [CK06]. These possibilities would pave the way for more fine-grained licensing models, but are still not implemented. More limited licensing schemes were developed though, but mostly focus on the security. They are presented below in the following section.

### 1.5.5 Licensing schemes

All the previously described methods deal with a specific aspect of intellectual property protection. However, some more holistic works proposed complete licensing schemes. Depending on how they make sure that the overall process is secure, they can be divided in two categories. Some of them require a trusted third party, while others make use of public key cryptography.

When classifying the methods as either using a trusted third party or public key cryptography, this is with respect to how the IP is protected. When a trusted third party is present, it usually manages keys which are used to encrypt the design and decrypt it on board. Conversely, public key cryptography is mostly used to send activation keys to the implemented design directly. Details are given in the following sections.

#### 1.5.5.1 Public key cryptography

The first option is to make use of public key cryptography. Most of the times, it is used to encrypt an activation word. For example, in [RKM08a], a unique pair of public and private keys is generated by a TRNG embedded in the device. The activation word is encrypted by the



IP core designer using the device public key and his private key. It is then decrypted inside the device using the IP core designer public key and the private key of the device.

In [HL08], targeting ASICs, the IP core designer embeds his public key inside the design and distributes it. Later on, when a system integrator wants to activate the IP core, he enters his private key, which is concatenated with a PUF response and hashed to generate an activation word. This activation word is encrypted by the designer's public key and sent back to the designer. The designer can then authenticate the system integrator with other techniques, decrypt the activation word and send it back to the designer. Since the activation word is device-specific, it is of no use for overproducing the IP core.

Instead of being integrated in the IP core itself, public key cryptography can be leveraged by the EDA tool. This is done in [Gua+09], where the existing Synplicity Open IP protocol is improved and another IP sharing protocol is presented. This protocol is detailed in Figure 1.18. In this case, the IP core encryption and decryption is handled inside the EDA tool.

Using public key cryptography offers strong security guarantees, but is very heavy to implement on-chip [HL08; RKM08a]. Typically, an RSA or elliptic-curve core is implemented and occupies a lot of logic resources. On the other hand, integrating these capabilities into the EDA tool can enforce the use of a specific piece software.

#### 1.5.5.2 Trusted third party

In some protocols, a trusted third party is required. A trusted third party participates in the protocol and behaves fairly. It serves as an intermediate and is supposed to be trusted by all parties, without these parties trusting each other. The existing protocols usually deal with IP cores provided for FPGAs.

In [Kea02], the trusted third party is the FPGA vendor, which is responsible for assigning a unique key to each FPGA and maintaining a database of keys. FPGA bitstreams are then encrypted with these keys, allowing them to be decrypted on only one hardware target that owns the key.

In [SS06], the trusted third party has multiple roles. It handles the hardware enrolment, which consists in obtaining a list of CRPs from the PUF. It then uses one response to encrypt authentication data and sends another response to the IP provider to encrypt the IP core. Since the responses are only accessible from the hardware, only the FPGA can decrypt these messages.

A specific metering architecture is presented in [MSV12]. In this case, the trusted third party enrolls both the hardware and the IP cores. A device-specific metering bitstream is then generated and handles the secure configuration of different IP cores on the same device. This is summarised in Figure 1.19, in which the trusted third party is called “metering authority”, or MA.

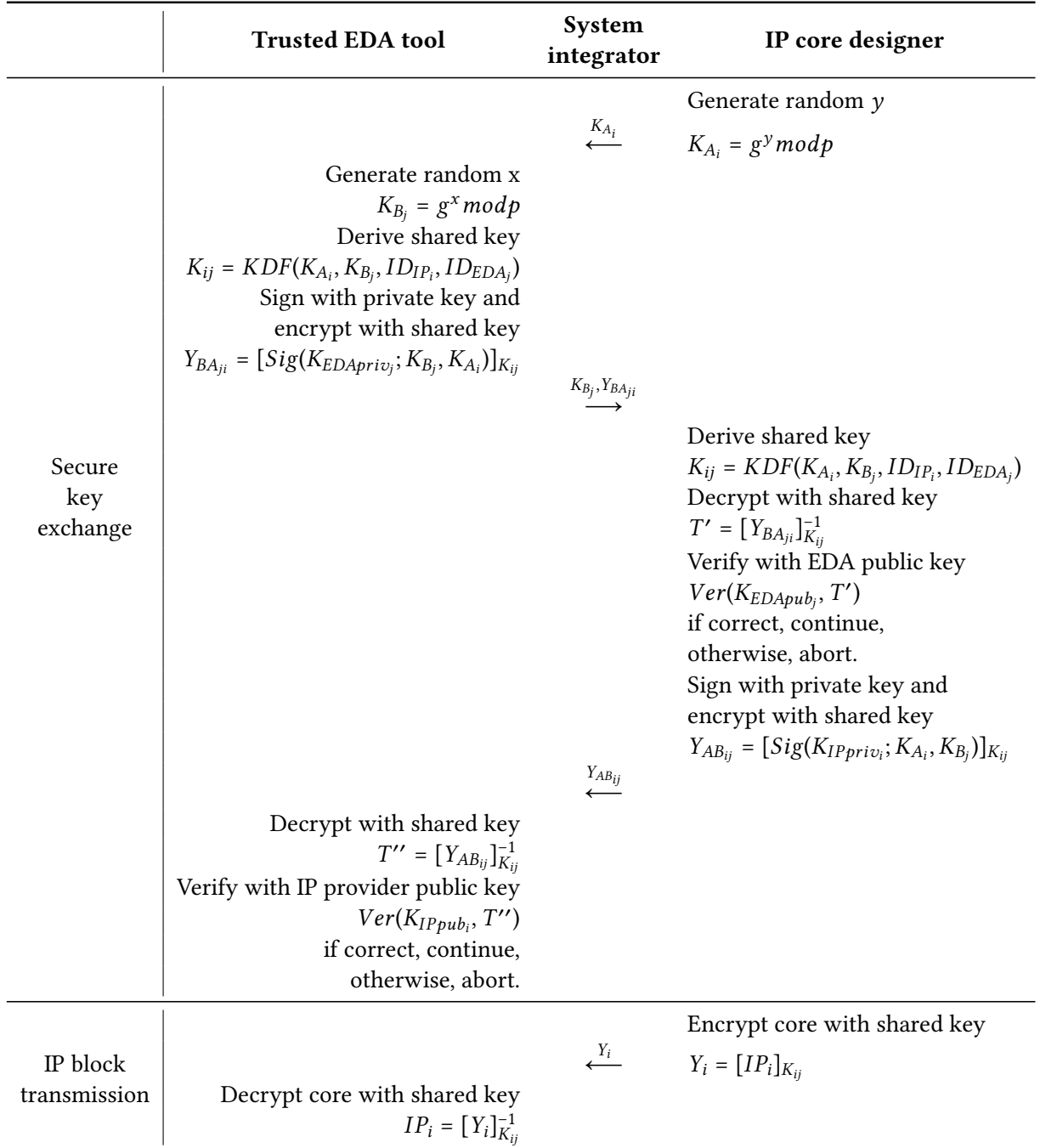


Figure 1.18 – Example of public-key cryptography usage in the EDA tool for a secure key exchange and IP block transmission (adapted from [Gua+09]).

The metering authority has multiple roles:

- Embedding a device-specific key into every device,
- Program every device with an encrypted metering bitstream,
- Enrol and register IP core along with their specific IP key,
- Provide system integrators with the encrypted IP-specific key  $[K_{IP}]_{K_M}$

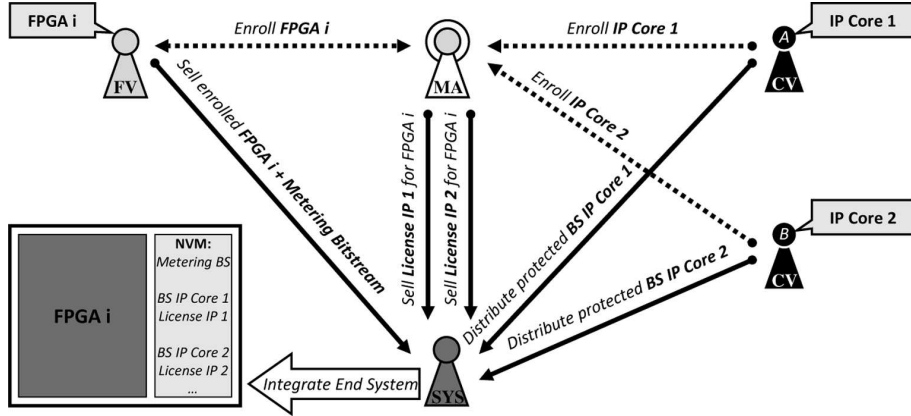


Figure 1.19 – Example of the implication of a trusted third party (MA) in the transactions between an FPGA vendor (FV), a system integrator (SYS) and two IP core designers (CV) (from [MSV12]).

Table 1.6 summarises which keys are known to which parties and integrated into which devices. This clearly highlights that the only party that owns all the keys is the metering authority. All other parties rely on it for trusted communication.

Key	Party				Devices	
	FPGA vendor	IP core designer	System integrator	Metering authority	Empty FPGA	FPGA + metering bitstream
Device-specific $K_{FPGA}$	×	×	×	✓	✓	✓
Metering key $K_M$	×	×	×	✓	×	✓
IP-specific key $K_{IP}$	×	✓	×	✓	×	✓
Encrypted metering key $[K_{IP}]_{K_M}$	×	×	✓	✓	×	✓
IP core $B$	×	✓	×	✓	×	✓
Encrypted IP core $[B]_{K_{IP}}$	×	✓	✓	✓	×	✓

Table 1.6 – Knowledge of the keys and encrypted data among parties (✓: known, ×: unknown).

In [GMP07], a key establishment scheme derives the FPGA-specific key from the secret key of the hardware manufacturer, the secret key of the IP core designer and the device ID, as shown in Equation (1.12) from the hardware manufacturer point of view or in Equation (1.13) from the point of view of the IP core designer. The FPGA then decrypts the bitstream internally.

$$K_{FPGA} = \text{key}(PK_{IPO}, SK_{HM}, ID) \quad (1.12)$$

$$= \text{key}(PK_{HM}, SK_{IPO}, ID) \quad (1.13)$$

All these solutions show that it is not impractical to implicate a trusted third party in the design process. Moreover, an existing party like the hardware manufacturer can play this role, making the implementation and adoption easier. Alternatively, trusted third parties could be implemented just like certificate authorities are for software.

### 1.5.5.3 IEEE 1735

It is worth pointing out that the IEEE<sup>24</sup> released a standard for “Recommended Practice for Encryption and Management of Electronic Design Intellectual Property” in 2015 [Soc14]. This standard specifies some capabilities that could be added to EDA tools or to HDLs to enforce IP protection. If adopted, this would allow EDA tools to conform to a common set of IP protection techniques.

This document is divided into several chapters that deal with different aspects. Chapter 5 defines a set of pragmas added to the HDL code to specify interoperability parameters. Chapter 6 defines how keys are managed between parties, while chapter 7 defines how rights are handled and granted to parties. In chapter 8, a license system is described that implements rights management. Chapter 9 defines how the visibility of the IP core components is managed, in particular the characteristics of a model that can replace the actual IP core for simulation purposes. Finally, chapter 10 defines common rights that all tools of the design flow should be able to handle. Some companies have implemented this standard into their EDA tools, like Xilinx in Vivado [Xil13] or Microsemi in Libero SoC [Mic17b].

### 1.5.5.4 Association of solutions

Finally, the last option to ensure a form of IP protection is to combine previously described solutions. We give some examples found in the literature, showing how the combination is actually implemented. For instance in [HL08], the activation inputs of a logic masking module are controlled by the response obtained from a PUF. This response is compared to a value stored in memory, fed by the system integrator. Following the principles of public key cryptography, the system integrator obtained the PUF response from the designer after it has been encrypted on chip by the designer’s public key and decrypted by the designer with his private key. In [CB09], a locking FSM is used to control the activation inputs of a logic masking module. The transitions between the extra states of the boosted FSM depend on a PUF response. In [Kou12], a boosted FSM is integrated with a PUF, so that the start-up state depends on the PUF response. This makes the set transitions to the original start-up state dependent on the PUF response. Therefore, the set of transitions is device-specific and is a condition to unlock the IP core. Finally, in [BZB14], a key is common to all instances of the IP core. This key is compared to one stored in an NVM and the result of this comparison triggers the blowing of specific anti-fuses located in the input-output blocks. These associations of previously described solutions are summarised in Table 1.7.

---

<sup>24</sup>Institute of Electrical and Electronics Engineers

	[HL08]	[CB09]	[Kou12]	[BZB14]
Identification of an IP core				key
Identification of IP core instances	PUF	PUF	PUF	
Degraded mode of operation	logic masking	FSM locking + logic masking	FSM locking	Anti-fuse locking
Public key cryptography	Elliptic curve			

Table 1.7 – Association of solutions to achieve complete IP protection.

## 1.6 Summary

After presenting which solutions exist in literature to provide IP protection, we can relate them to the two previously described threats: illegal copying and reverse-engineering. This is shown in Table 1.8, where the number of black dots refers to the efficiency of the solution at fighting the associated threat. For example, identifying an IP core is not very efficient at preventing illegal copying since obtaining the key for one IP core unlocks them all. On the other hand, offering degraded modes of operation is a very efficient solution to deter potential adversaries. Identifying individual instances of an IP core is a must for design data protection. It is the basis of metering. Hiding the internals of a design can prevent reverse-engineering but a sufficiently motivated and funded adversary will always manage to extract information anyway. Finally, licensing schemes are efficient but usually require a lot of logic resources on the device.

Threats	Solutions				
	IP core identification	IP core instances identification	Internal architecture concealment	Degraded modes of operation	Licensing schemes
Illegal copying	●○○	●●○		●●●	●●●
Reverse engineering			●●○	●○○	

Table 1.8 – Suitability of IP protection solutions at addressing different threats.

The most efficient combination seems to be the one integrating a unique identifier for every IP core instance along with a controllable degraded mode of operation. By adding a symmetric cipher on top of this, security can be guaranteed. Essentially, a secure remote activation scheme must be built.

## 1.7 High-level requirements for a secure remote activation scheme

In the framework of the SALWARE project<sup>25</sup>, the main objective is the industrial feasibility of the proposed solutions. From the existing state-of-the-art, we can derive the following high-level requirements for the secure remote activation scheme.

First of all, it must be easy to operate the activation scheme in a normal way. Namely, a legitimate system integrator should be able to activate an IP core easily if it has been obtained under the standard procedure. Contrarily, from an attacker point of view, the protection scheme should be sufficiently hard to circumvent, that is to say obtaining a functional copy of the IP core. This is closely related to the security level reached by the cryptographic primitives implemented in the system. Instead of aiming at long-term security, a moderate security level should be the target here. Typically, symmetric ciphers would employ 80-bit keys.

In addition to those two basic requirements which form the basis of the IP protection scheme, we can add some additional specifications. When an attacker obtained an IP core in an illegal way, the IP core must operate in a very disturbed manner, as far as possible from its original behaviour. All the modes of operation should be affected. However, when the IP core has been unlocked, the protection scheme should have no impact on the performances.

Another characteristic of the protection scheme that is determinant to foster its adoption by industrial partners is the amount of hardware resources it occupies. Clearly, we aim at making the whole module as lightweight as possible, so that it does not incur too high additional costs for the IP core to be protected. Similarly, ease of integration into standard design flow is also necessary. In particular, the protection scheme should be as universal as possible and be able to deal with all sorts of IP cores.

## 1.8 SALWARE IP protection module

An overview of the IP protection module proposed in the framework of the SALWARE project is shown in Figure 1.20. On the right-hand side, an integrated circuit that integrates three IP cores is shown. One of them is protected by the module detailed on the left-hand side, which communicates with a remote server shown at the bottom. This module comprises the following components:

**Lightweight block cipher** It decrypts the encrypted activation word sent by the remote server. The encryption key is the PUF response.

---

<sup>25</sup> <http://www.univ-st-etienne.fr/salware/>

**Logic locking/masking module** It locks or masks the protected IP core and makes it unusable when not activated yet.

**PUF** It generates a unique identifier for the IP core instance.

**Interactive error correction** It makes the device-side and server-side responses ( $r$  and  $r_0$ ) match by carrying out a key reconciliation protocol.

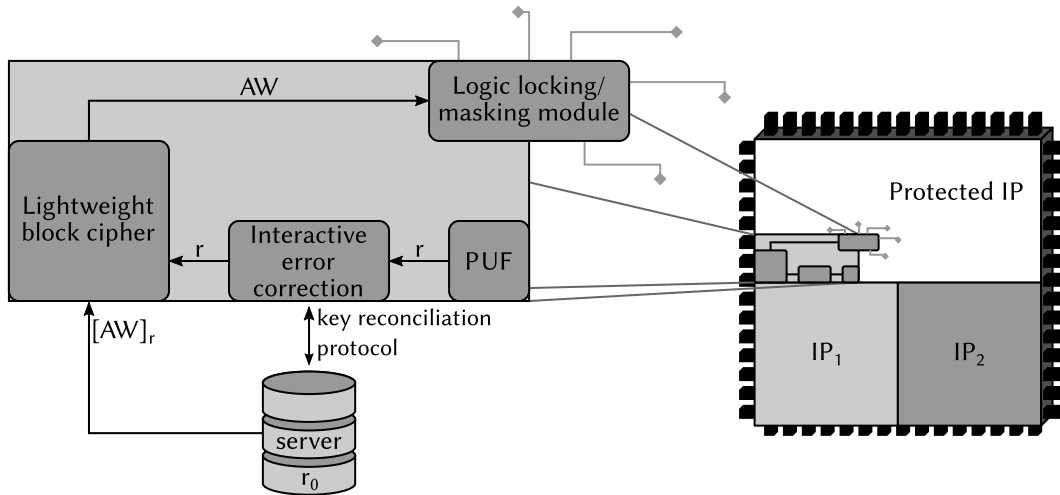


Figure 1.20 – Overview of the IP protection module designed in the framework of the SALWARE project.

# Chapter 2

## Combinational logic locking

---

Among the degraded modes of operations presented in the previous chapter, logic locking consists in setting the outputs of a design to a fixed logic level unless the correct activation word is fed. So far, high level features, such as the FSM, the input/outputs or the communication bus, were targeted. This comes with a lack of generality, since most of the techniques are dependent on the architecture or the features of the design to protect.

To overcome this limitation, directly acting at a lower level, on the combinational logic, is a solution. The method presented in this chapter leverages the representation of a netlist as a directed acyclic graph. By inserting so-called “locking gates”, the outputs of the netlist can be forced to a fixed value. The contribution of this chapter is an algorithm that selects which nodes must be modified based on the propagation of a locking value through a sequence of nodes. The nodes selection and insertion process proves to be very computationally efficient, allowing to process large combinational netlists of up to 200 000 nodes. At the same time, the logic resources overhead induced by the extra logic gates is 3% on average.



# Verrouillage combinatoire de la logique

---

Parmi les modes de fonctionnement dégradés présentés dans le chapitre précédent, le verrouillage consiste à forcer les sorties d'un composant virtuel à un niveau logique fixe tant que le mot d'activation correct n'a pas été fourni. Jusqu'à présent, des caractéristiques de haut niveau, telles que la machine à états finie, les entrées/sorties ou le bus de communication, étaient ciblées. Ces techniques sont difficiles à généraliser, car la plupart dépendent de l'architecture ou des caractéristiques du composant virtuel à protéger.

Pour dépasser cette limitation, agir directement à un niveau plus bas, celui de la logique combinatoire, est une solution. La méthode présentée dans ce chapitre s'appuie sur la représentation d'une netlist comme un graphe orienté acyclique. En insérant des "portes de verrouillage", les sorties du composant virtuel peuvent être forcées à une valeur logique fixe. La contribution de ce chapitre est un algorithme qui sélectionne les nœuds à modifier en se basant sur la propagation d'une valeur de verrouillage à travers une suite de nœuds. Le processus de sélection et d'insertion est très efficace et permet de traiter des composants virtuels combinatoires contenant jusqu'à 200 000 nœuds. Dans le même temps, le surcoût en ressources logiques induit par les portes logiques supplémentaires est de 3% en moyenne.

## 2.1 Definition

Logic locking is defined as the fact to controllably force the outputs of a design to a fixed logic value unless the correct AW is fed to the dedicated inputs. There can be two definitions of logic locking, depending on the actual number of outputs that are locked.

Let  $y$  be the output of the netlist and  $AW_{\text{valid}}$  the correct activation word, then *total* logic locking is defined in Equation (2.1). When total logic locking is applied to IP core, *all* the outputs are forced to a fixed logic level unless the correct activation word is fed. The output value is then  $y_{\text{fixed}}$ .

$$\exists! y_{\text{fixed}} \in \{0, 1\}^m \mid \forall AW \neq AW_{\text{valid}} : y = y_{\text{fixed}} \quad (2.1)$$

A weaker definition of logic locking can be derived in the case where some outputs are not affected when a specific AW is provided. This is the more general case.  $AW_{\text{valid}}$  unlocks all the outputs (see Equation (2.2)), its complement  $\overline{AW_{\text{valid}}}$  locks all the outputs (see Equation (2.3)) and all the other possible AWs lock only a fraction of the outputs (see Equation (2.4)).

$$\exists! AW_{\text{valid}} : I_y = I_{y_{\text{unlocked}}} \quad (2.2)$$

$$\exists! \overline{AW_{\text{valid}}} : I_y = I_{y_{\text{locked}}} \quad (2.3)$$

$$\forall AW \notin \{AW_{\text{valid}}, \overline{AW_{\text{valid}}}\} : I_y = I_{y_{\text{locked}}} \cup I_{y_{\text{unlocked}}} \quad (2.4)$$

In this general case (Equation (2.4)), the set of output bits,  $I_y$ , can be seen as the union of two subsets. The set  $I_{y_{\text{locked}}}$  corresponds to the set of outputs which are forced to a fixed logic value by a *specific* AW. The set  $I_{y_{\text{unlocked}}}$  corresponds to the set of outputs which are not forced to a fixed logic value by this specific AW. The cardinality of the sets  $I_{y_{\text{locked}}}$  and  $I_{y_{\text{unlocked}}}$  depends on the AW fed. Some AWs will lock more outputs than others.

In order to implement combinational logic locking, so-called *locking gates* are inserted inside the original netlist. We first describe a naive implementation of the weak definition of logic locking, before formalising and giving details about a more efficient method based on graph analysis. We then provide means of achieving the definition of logic locking shown in Equation (2.1).

### 2.1.1 Naive description

To force an output of a design to a fixed logic value, one of the inputs of the final logic gate must be set to its corresponding controlling value. For example, setting a logic 0 to one of the inputs of a NAND gate forces its output to 1. Table 2.1 gives the controlling value for the usual non-linear logic gates. Indeed, only linear logic gates have a controlling value. The output of linear logic gates like XOR or XNOR cannot be set to a fixed logic value by controlling only one of the inputs.

Logic gate	Controlling value	Forced output value
AND	0	0
OR	1	1
NAND	0	1
NOR	1	0

Table 2.1 – Controlling value of non-linear logic gates and the associated forced output value

In order to force the controlling value, locking gates are inserted. If the controlling value is 0, an AND gate is inserted to controllably force it to 0. If the controlling value is 1, an OR gate is inserted to controllably force it to 1. An example of how an output is modified is given in Figure 2.1. In Figure 2.1a, the final gate before the output is a 2-input NAND. The controlling value of an NAND gate being 0, an AND gate is added for logic locking to be able to controllably force this input,  $X_0$  here, to 0. The lockable output is shown in Figure 2.1b.

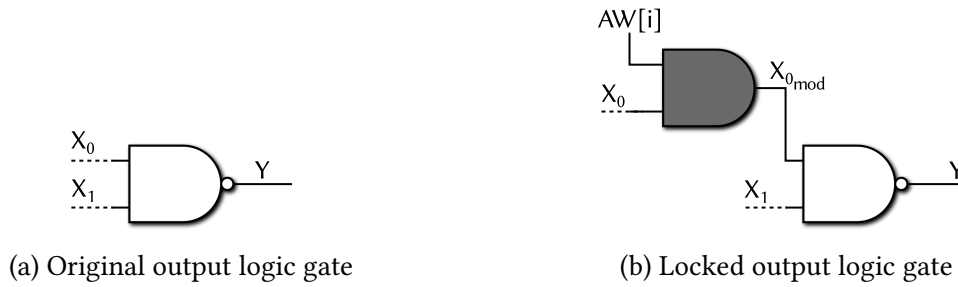


Figure 2.1 – Modification of an output logic gate

When the locking input  $AW_i$  of the locking gate (in dark grey) is set to 0, which is the locking value, the wire that propagates the controlling value,  $X_{0\_mod}$  is forced to 0. Since 0 is the controlling value of the original output gate (in white), the original output  $Y$  is forced to 0. Conversely, when the locking input of the locking gate is set to 1 which is the unlocking value, the wire that propagates the controlling value has the same logic value as the other input  $X_0$ . In this case, the overall NAND logic function is preserved. By repeating this process for all the output gates of a design, all the outputs can be controllably forced to a fixed logic value.

### 2.1.2 Logic function analysis using Boole's expansion theorem

Boole's expansion theorem [Boo54] states that an  $n$ -input boolean function can be split into two parts containing two cofactors, later called Shannon cofactors. This is shown in Equations (2.5) and (2.6), where  $F$  is the boolean function, and  $F_x$  and  $F'_x$  are the two cofactors. The positive cofactor  $F_x$  is equal to  $F$  with the variable  $x$  set to 1. The negative cofactor  $F'_x$  is equal to  $F$  with the variable  $x$  set to 0. Equation (2.5) shows the Sum-of-Products (SoP) form, while Equation (2.6) shows the Product-of-Sums (PoS) form.

$$F = x \cdot F_x + \bar{x} \cdot F'_x \quad (2.5)$$

$$= (x + F'_x) \cdot (\bar{x} + F_x) \quad (2.6)$$

It is possible to highlight logic locking in both these decomposition.

**Lemma 1** *A boolean function  $F$  is locked to the value  $y_{locked}$  by the variable  $x$  when  $x = 0$  if  $F$  can be written as:*

$$F = x \cdot F_x + \bar{x} \cdot y_{locked} \quad (2.7)$$

*in SoP form, or as:*

$$F = (x + y_{locked}) \cdot (\bar{x} + F_x) \quad (2.8)$$

*in PoS form.*

**Lemma 2** *A boolean function  $F$  is locked to the value  $y_{locked}$  by the variable  $x$  when  $x = 1$  if  $F$  can be written as:*

$$F = x \cdot y_{locked} + \bar{x} \cdot F'_x \quad (2.9)$$

*in SoP form, or as:*

$$F = (x + F'_x) \cdot (\bar{x} + y_{locked}) \quad (2.10)$$

*in PoS form.*

Any boolean function that can be identified with Equation (2.7), (2.9), (2.8), (2.10), where  $y_{locked}$  is a *constant*, can be locked by the  $x$  variable.

For a 2-input AND gate, we can write  $Y = X_0 \cdot X_1 + \bar{X}_0 \cdot 0$ . This highlights, according to Equation (2.7), that the output of an AND gate can be locked to 0 by setting its input  $X_0$  to 0.

Similarly, for a tree of seven 2-input OR gates we can write the following equality:

$$\begin{aligned} Y &= X_0 + X_1 + X_2 + X_3 + X_4 + X_5 + X_6 + X_7 \\ &= (X_0 + X_1 + X_2 + X_3 + X_4 + X_5 + X_6 + X_7) \cdot (\bar{X}_0 + 1) \end{aligned}$$

This shows that such a structure can be locked by setting its input  $X_0$  to 1, according to Equation (2.10). All the other inputs could be used as well.

Figure 2.2 shows two slightly different 4-input logic functions.

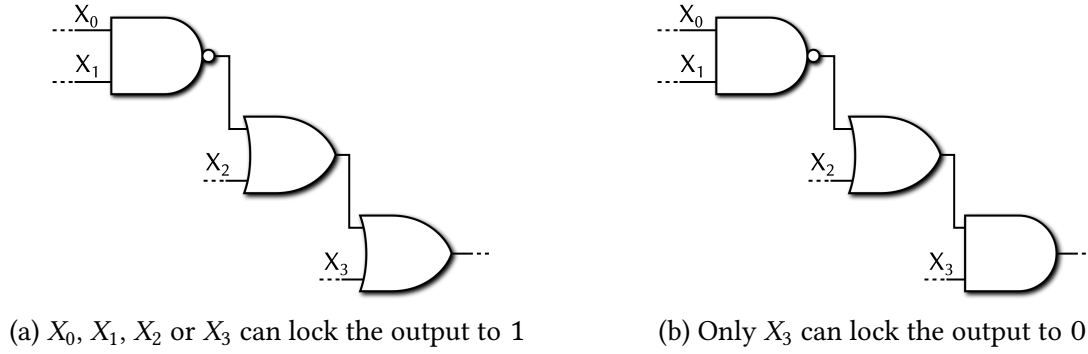


Figure 2.2 – Two examples of logic functions and the inputs that can lock their output

On the left-hand side, Figure 2.2a, the logic function is:

$$\begin{aligned}
 F &= \overline{(X_0 \cdot X_1)} + X_2 + X_3 \\
 &= \overline{X_0} + \overline{X_1} + X_2 + X_3 \\
 &= (X_0 + 1) \cdot (\overline{X_0} + \overline{X_1} + X_2 + X_3) \\
 &= (X_1 + 1) \cdot (\overline{X_1} + \overline{X_0} + X_2 + X_3) \\
 &= (\overline{X_2} + 1) \cdot (X_2 + \overline{X_0} + \overline{X_1} + X_3) \\
 &= (\overline{X_3} + 1) \cdot (X_3 + \overline{X_0} + \overline{X_1} + X_2)
 \end{aligned}$$

These PoS forms can be identified with Equation (2.8) and (2.10), showing that the output of this function can be locked to 1 by forcing  $X_0$  or  $X_1$  to 0 or  $X_2$  or  $X_3$  to 1.

Conversely, on the right-hand side, Figure 2.2b, the logic function is:

$$\begin{aligned}
 F &= ((\overline{X_0 \cdot X_1}) + X_2) \cdot X_3 \\
 &= X_3 \cdot (\overline{X_0} + \overline{X_1} + X_2) + \overline{X_3} \cdot 0
 \end{aligned}$$

These SoP form can be identified with Equation (2.7), showing that the output of this function can be locked to 0 by forcing  $X_3$  to 0.

Finding such identities in the logic equation of the outputs of a circuit is tedious, since this requires the manipulation of complex equations. Moreover, most of the boolean functions cannot be locked. Finally, this does not favour the nodes that are far from the outputs. This is an issue since the locking gates could be very easily identified in the netlist if they are very close to the outputs. For instance, for the function shown in Figure 2.2a,  $X_0$  or  $X_1$  are better suited than  $X_3$  for combinational logic locking. In order to overcome this, another point of view can be taken. By considering the schematic of the boolean function implementation, we can highlight interesting sequences of nodes in the netlist that are capable of propagating a locking value.

### 2.1.3 Schematic view: propagation of a controlling value

Inserting the locking gates deeper in the netlist, as far as possible from the outputs, requires to identify sequences of nodes that can propagate a locking value. An example depicting how a sequence of nodes can propagate a locking value is shown in Figure 2.3, with the same logic functions as in Figure 2.2. On the left hand-side, Figure 2.3a, feeding a logic 0 at one of the inputs of the first NAND gates forces the output of the last OR gate to 1. Conversely, in Figure 2.3b, the final AND gate does not allow the locking value to propagate from its inputs further down the netlist. This is coherent with what has been said in Section 2.1.2. The propagation of a locking value is shown in thick red in Figure 2.3a.

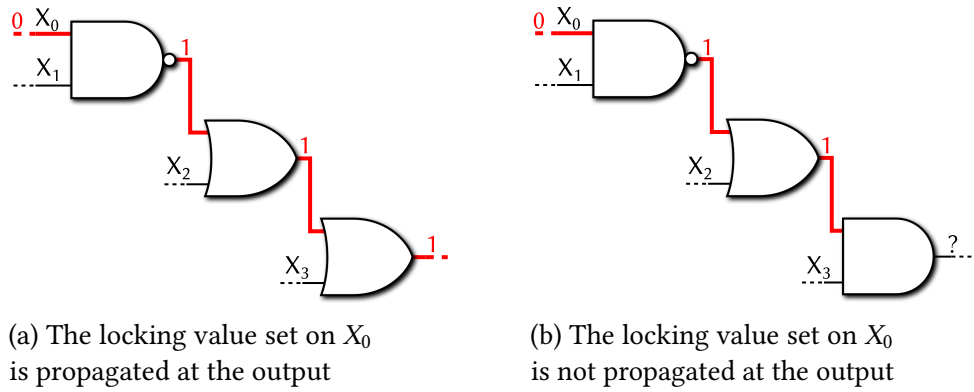


Figure 2.3 – Propagation of a locking value through a sequence of nodes (in thick red)

For a sequence of gate to behave like this, the logic value at which the output of each gate is forced must be the controlling value of the subsequent gate. For each gate, it is then necessary to own two values: the controlling value and the forced output value (see Table 2.1). Therefore, there are also two values for every node in the netlist: the value at which the preceding gate forces it and the controlling value of the subsequent gate. For a node to propagate a locking value, those two values must match. We call  $V_{\text{forced}}$  the value at which a node is forced by the preceding gate. We call  $V_{\text{locks}}$  the value at which the node should be forced to control the subsequent gate. This is the controlling value of this gate. Thus a node can propagate a locking value if it satisfies the following locking criterion.

**Criterion 1** *A netlist node can propagate a locking value if and only if its  $V_{\text{forced}}$  value is included in the set of its  $V_{\text{locks}}$  values called  $I_{V_{\text{locks}}}$ :*

$$V_{\text{forced}} \in I_{V_{\text{locks}}}$$

For example, if a node is the output of an OR gate and the input of an AND gate, then  $V_{\text{forced}} = 1$  and  $V_{\text{locks}} = 0$ . Since, in this case,  $V_{\text{forced}} \notin I_{V_{\text{locks}}}$ , this node cannot propagate a locking value. This is the case for the output of the OR gate in Figure 2.3b.

It can occur that a node is the input of logic gates that have a different controlling value. For example, a node can a fan-out of 2 and be the input of a NAND gate and an OR gate. In

this case,  $V_{locks}$  is set to  $\{0, 1\}$ . This is why Criterion 1 uses a membership relation instead of an equality between  $V_{forced}$  and  $V_{locks}$ .

## 2.2 Selection of the place of insertion

In order to select the best locations of insertion for the locking gates, the representation of the netlist as a graph is leveraged. This is detailed in the following sections.

### 2.2.1 Conversion from netlist to graph

The netlist is converted to a directed acyclic graph according to the following rule. Netlist nodes are converted to vertices, which are then connected to one another using directed edges. These edges are labelled after the logic function found in the original netlist. A toy example of netlist conversion is shown in Figure 2.4.

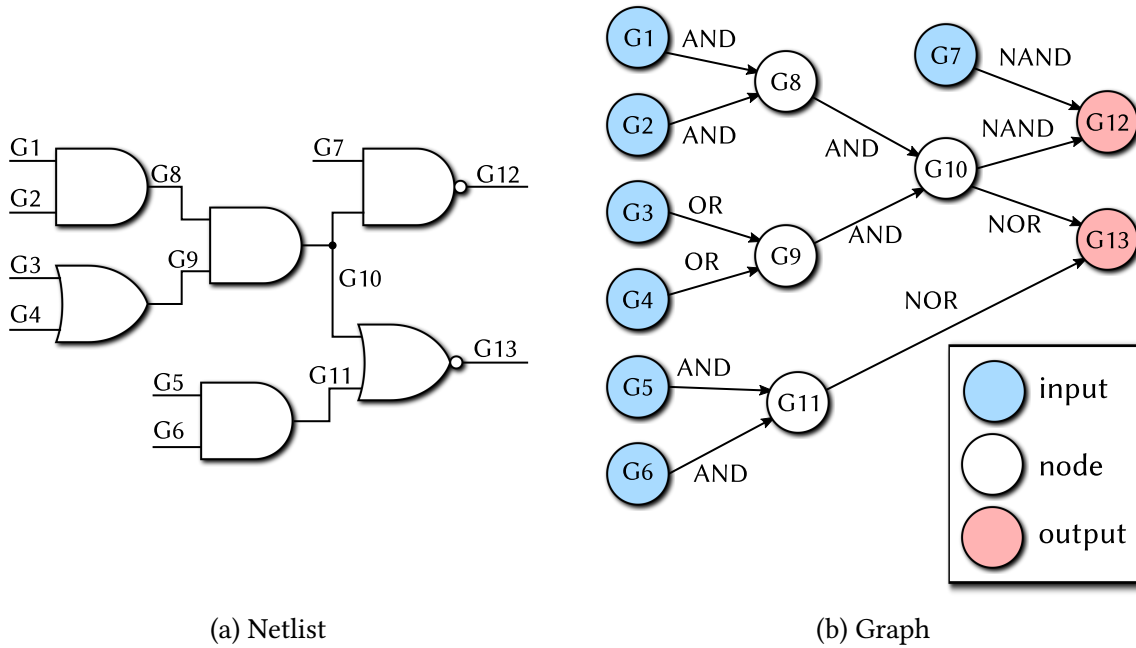


Figure 2.4 – Conversion of a netlist to a directed acyclic graph

### 2.2.2 Graph labelling

Once the graph has been built, a copy of the original graph is stored for later. The  $V_{forced}$  and  $V_{locks}$  values are computed for every vertex of the graph.  $V_{forced}$  depends on the incoming edges, while  $V_{locks}$  depends on the outgoing edges. Only internal nodes are considered.  $V_{forced}$  and  $V_{locks}$  values for the nodes of the netlist in Figure 2.4a are given in Table 2.2.

Node	$V_{\text{forced}}$	$V_{\text{locks}}$	Fulfil Criterion 1 ?
G8	0	0	✓
G9	1	0	×
G10	0	{0, 1}	✓
G11	0	1	×

Table 2.2 –  $V_{\text{forced}}$  and  $V_{\text{locks}}$  values for the internal nodes of the netlist in Figure 2.4a

### 2.2.3 Identification of the nodes that propagate a locking value to an output

*Incoming edges* of vertices for which Criterion 1 is not satisfied are deleted. Indeed, these vertices correspond to nodes that are not able to propagate a locking value. This is shown in Figure 2.5a, in which the incoming edges of G9 and G11 have been deleted.

For usual netlists, most of the nodes do not satisfy Criterion 1. Therefore, after this deletion, the graph is highly disconnected and comprises multiple connected components. Connected components of the graph that do not contain any output are not useful to implement logic locking, since only the outputs must be set to a fixed logic value. Therefore, those connected components are discarded and removed from the graph (see Figure 2.5b).

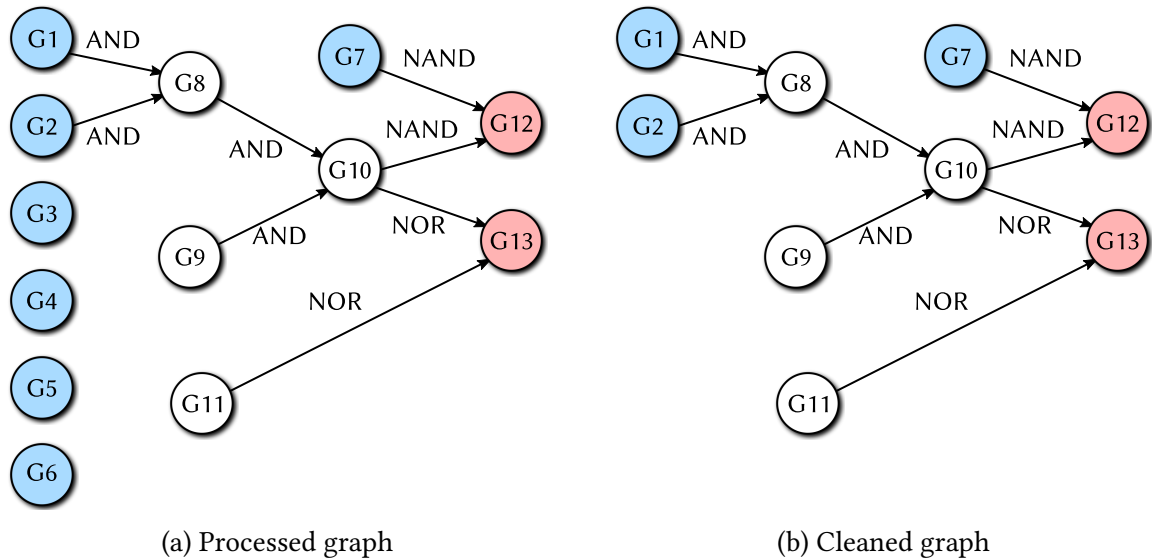


Figure 2.5 – Deletion of the incoming edges of vertices that do not satisfy  $V_{\text{forced}} \in V_{\text{locks}}$  and removal of connected components that do not contain any output.

Eventually, the graph comprises several connected components. They all contain at least one output and nodes that are all able to propagate a locking value. However, some of these nodes are more interesting than others for logic locking.



### 2.2.4 Selection of the best nodes to modify

Since all the vertices found in the final graph correspond to nodes that can propagate a locking value, the ones which are the *furthest* from the outputs must be picked. Therefore, only source vertices<sup>1</sup> are considered. Indeed, if a vertex is not a source vertex, then it has incoming edges. It is then the child of a least one other vertex that is further from the outputs. Going up the edges one eventually reaches one or more source vertices, which are the furthest from the outputs. In the final graph, four types of connected components can be found, according to the number and properties of the source vertices in them. These are shown in Figure 2.6.

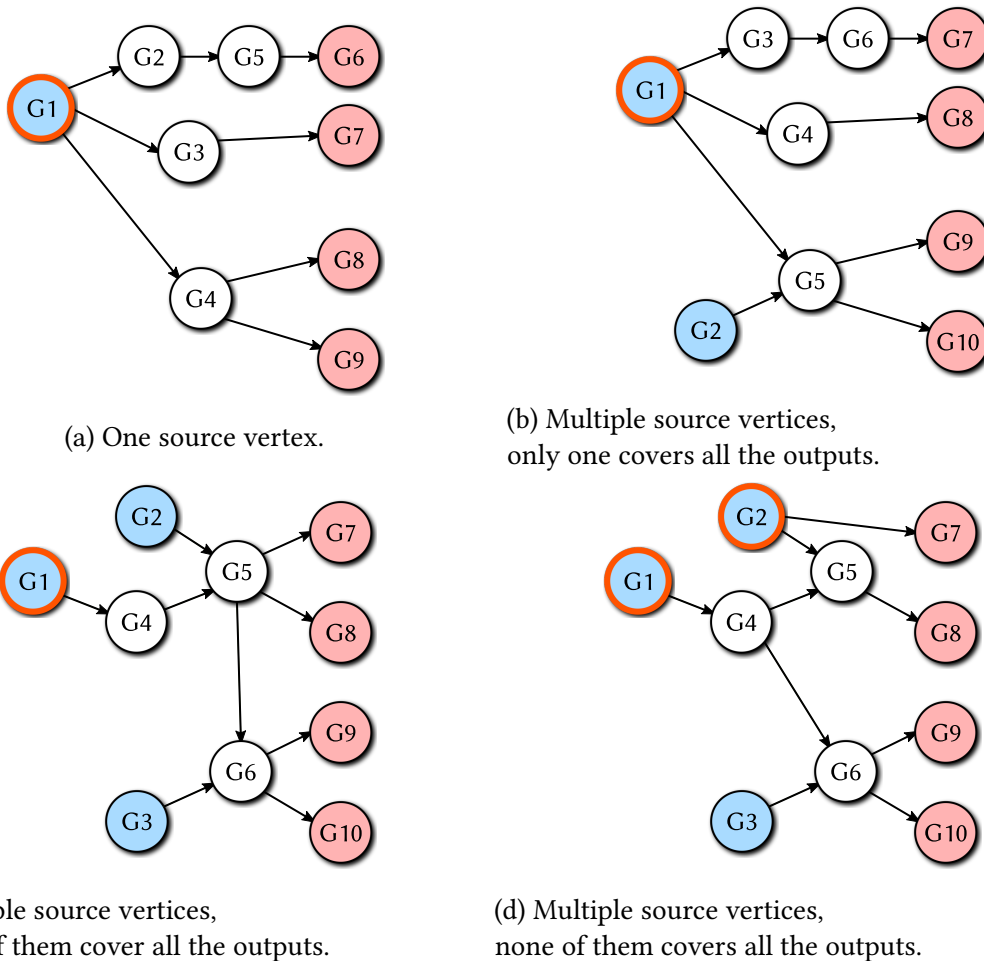


Figure 2.6 – Different types of connected components that are found in the final graph. The node(s) select to be modified for logic locking are highlighted in orange.

In the first case, the connected component has only one source vertex (see Figure 2.6a). It is selected for logic locking since it covers all the outputs and is as far as possible from them.

In the second case, there are several source vertices but only one of them covers all the outputs (see Figure 2.6b). Therefore, even though it might not be the furthest source vertex from the outputs, it is selected for locking. Indeed, since it covers all the outputs, it will result

<sup>1</sup>Source vertices are vertices that have no incoming edges.

in the most lightweight implementation since it requires only one locking gate.

The third type of connected component comprises multiple source vertices, and several of them cover all the outputs (see Figure 2.6c, where both G1 and G2 cover all the outputs). Since we want the locking gates to be as far from the outputs as possible, then the selected vertex is the one that maximises the sum of distances from it to the outputs, given in Equation (2.11).

$$m_d(v) = \sum_{o \in \text{outputs}} d(v, o) \quad (2.11)$$

Computing this sum of distances requires to start at the source vertex and search for the outputs. Using simple breadth-first search or depth-first search algorithms is the chosen solution since the connected components are of small size. Therefore, the execution time of these algorithms is manageable.

Finally, in the last type of connected component, there are multiple source vertices that cover one or several outputs, but none of them covers them all (see Figure 2.6d). In this case, the first step is to sort the vertices according to the number of outputs they cover. This is done by using one of the search algorithms mentioned above. Then, the nodes are greedily selected until all the outputs are covered. In the considered netlist (see Figure 2.4a), nodes G1 and G11 are selected. This is shown in Figure 2.7.

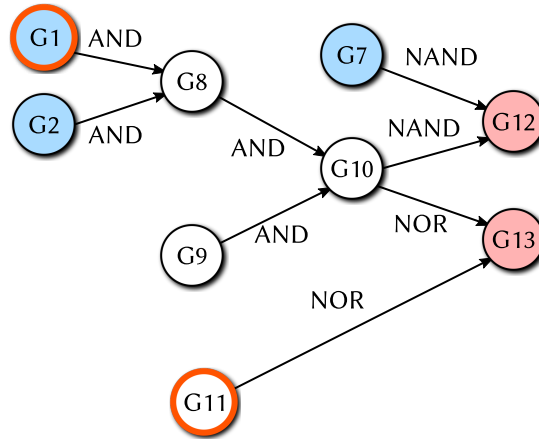


Figure 2.7 – Vertices selected for logic locking.

### 2.2.5 Locking gates insertion

Once the nodes to lock are selected, the locking gates can be inserted. The type of locking gate is determined by the  $V_{\text{locks}}$  value of the corresponding vertex. If  $V_{\text{locks}} = 0$ , the node associated to the vertex must be forced to 0 to start propagating the locking value. Therefore, an AND gate is inserted. Conversely, if  $V_{\text{locks}} = 1$ , the node associated to the vertex must be forced to 1 to start propagating the locking value. Therefore, an OR gate is inserted. These modifications, done on the *original* graph which had been saved previously, are depicted in Figure 2.8, while Figure 2.9 shows the graph with added vertices and edges for logic locking.

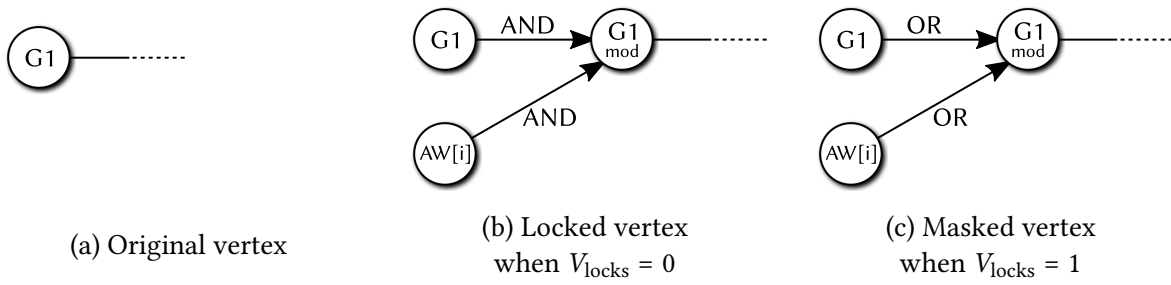


Figure 2.8 – Original and locked vertices depending on the associated activation bit

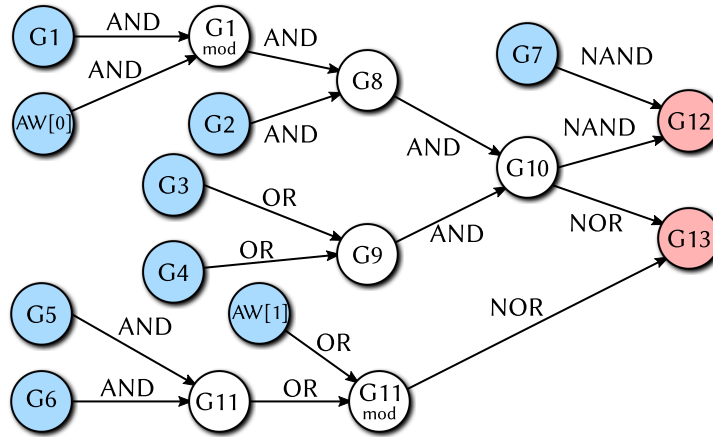


Figure 2.9 – Locking vertices and edges added to the graph

## 2.2.6 Conversion from graph to netlist

Once the original graph has been modified, it must be converted back into a netlist. This is done by following the inverse rule as previously described. Namely, vertices are converted to nodes, while edges are converted to logic gates. Figure 2.10 shows the lockable version of the netlist. Added gates are in dark grey.  $AW[0]$  allows to force the node  $G1$  to the logic value 0. It propagates to the output  $G12$ , forcing it to 1.  $AW[1]$  allows to force the node  $G11$  to the logic value 1. It propagates to the output  $G13$ , forcing it to 0.

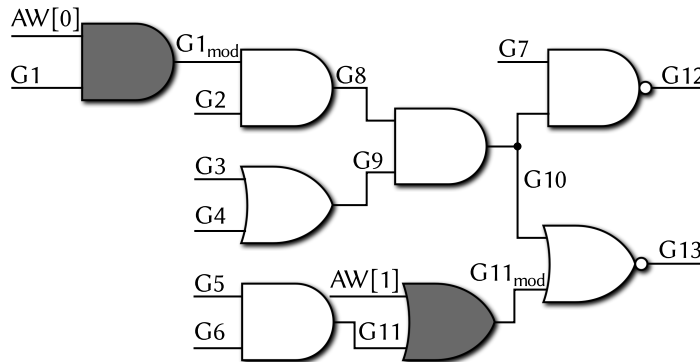


Figure 2.10 – Lockable version of the netlist

## 2.3 Experimental results

Combinational logic locking is now evaluated with respect to different metrics. The first one is the area overhead induced by the extra locking gates added. The second one is the computation time required by the logic locking process. This is divided into two parts: the time taken to build the graph from the netlist file and the time required to analyse the graph and convert it back into a netlist. The third metric is the average distance from the inserted locking gates to the outputs of the netlist. It gives an indication about how deep inside the netlist the locking gates are inserted. This is a criterion against reverse-engineering. Finally, the ratio between the number of outputs and the number of inserted gates is given. This is called the locking ratio. It quantifies how many locking gates affect each output, so this also gives how many bits of the AW affect each output.

We implemented the logic locking algorithm in Python, making use of the *igraph* package [CN06] to handle graphs. The computation times are obtained with a workstation embedding an Intel Core i5-4570 processor operating at 3.20GHz and 16GB of RAM. We used ITC'99 combinational benchmarks [Dav99], but only the ones with more than 1 000 logic gates.

Experimental results are mostly given in the form of plots, but an exhaustive list of values for all the benchmarks is given in Table 2.3.

Appendix 5.7 gives an example of the graphs that were obtained when applying the logic locking algorithm described above. Figure 16 shows the graph right after it has been built from the netlist file. The netlist has around 1 000 logic gates. Figure 18 shows the graph after it has been analysed and processed for logic locking. Thus only the paths that propagate a locking value are drawn.

### 2.3.1 Logic resources overhead

The first metric used to evaluate an IP protection scheme is the area overhead it induces. In order to remain as generic as possible, we measure it as the percentage of logic gates that must be added to the netlist to make it lockable. The added gates being of AND or OR type, the associated area for an ASIC implementation is rather low. For an FPGA implementation, the performance depends on the synthesiser. However, one can expect the overhead to be similar.

The area overhead observed when applying the previously described logic locking process to the considered netlists is shown in Figure 2.11. The area overhead required to achieve logic locking ranges from roughly 1 to 5%, with an average value of 2.89%. Detailed values can be found in the “Minimum overhead (%)” column of Table 2.3. The value for each benchmark is not related to its size. At first sight, this is coherent. Indeed, logic locking targets the outputs and the number of outputs broadly grows linearly with the benchmark size.

The overhead given here corresponds to total logic locking. Namely, all the outputs of a design can be locked to a fixed logic value. On a per-design basis, this could be adapted. Indeed,

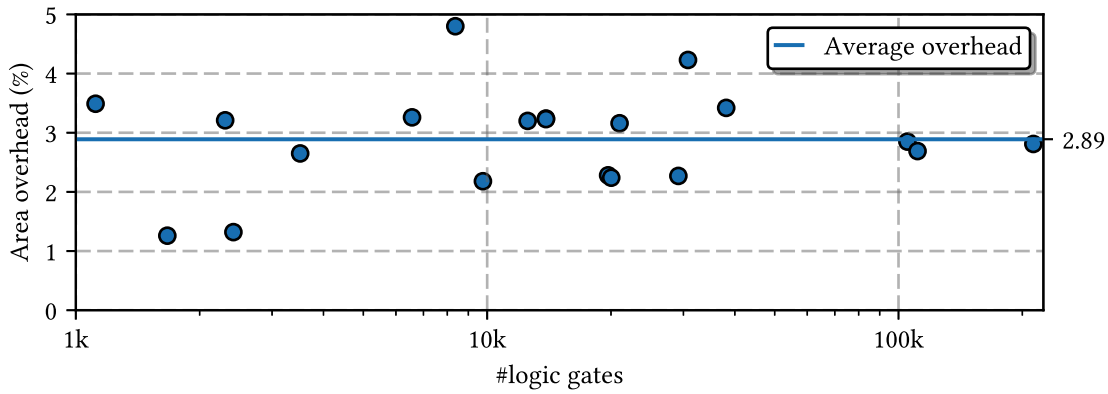


Figure 2.11 – Area overhead as the percentage of extra logic gates required to implement logic locking

for some designs, locking only a fraction of the outputs could be sufficient to ensure sufficiently erratic behaviour. For instance, only the outputs of the controller may be locked, effectively disabling the whole system. This requires an intervention of the designer to guide the logic locking method to the potential nodes to lock. On the other hand, if the designer can afford a larger area overhead, logic locking could be strengthened. This is detailed in Section 2.4 of this chapter.

### 2.3.2 Computation time

Another crucial evaluation criterion for IP protection schemes is their computational complexity. Although it is usually neglected, some works focus on reducing the execution of the heuristics used to select the nodes to modify [GGY15]. Nevertheless, computational complexity becomes a crucial characteristic when the protection scheme is meant to be integrated into EDA tools.

We compare our graph-based algorithm for total combinational logic locking with the state-of-the-art heuristic used for logic masking, which is based on fault-analysis [Raj+15].

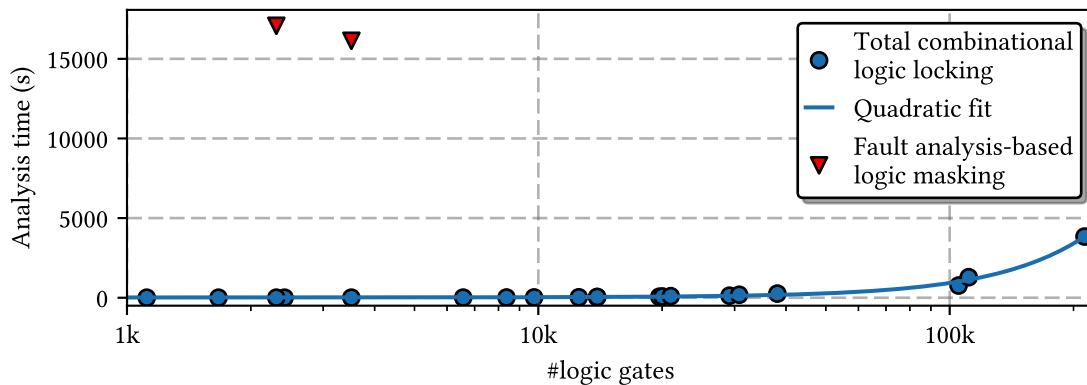


Figure 2.12 – Computation time required to process a netlist for logic locking and for fault analysis-based logic masking

Benchmark	# Outputs	# Gates	Graph building time (s)		Graph processing time (s)		# Gates at minimum overhead		Minimum Overhead (%)		Average distance to outputs		Locking ratio at minimum overhead		# Gates at maximum overhead		Maximum overhead (%)		Locking ratio at maximum overhead	
c2670	64	1117	0.22	0.77	1156	3.49	5.26	1.64	1406	25.87	4.52									
c3540	22	1669	0.28	1.18	1690	1.26	1.90	1.05	1828	9.53	7.23									
c5315	123	2307	0.39	2.55	2381	3.21	3.70	1.66	2863	24.10	4.52									
c6288	32	2416	0.4	1.77	2448	1.32	1.00	1.00	2512	3.97	3.00									
c7552	107	3511	0.58	3.68	3604	2.65	1.82	1.15	3834	9.20	3.02									
b14_1_C	245	6567	1.12	6.7	6781	3.26	1.70	1.14	7731	17.72	4.75									
b15_C	449	8367	1.48	12.64	8769	4.80	1.89	1.12	10961	31.00	5.78									
b14_C	245	9765	1.66	17.07	9978	2.18	1.64	1.15	11014	12.79	5.10									
b15_1_C	449	12543	2.18	25.16	12945	3.20	2.30	1.12	15358	22.44	6.27									
b21_1_C	512	13898	2.44	35.28	14348	3.24	1.51	1.14	16207	16.61	4.51									
b20_1_C	512	13899	2.42	45.25	14348	3.23	1.51	1.14	16152	16.21	4.40									
b20_C	512	19682	3.44	59.29	20130	2.28	1.56	1.14	22130	12.44	4.78									
b21_C	512	20027	3.49	73.41	20476	2.24	1.47	1.14	22554	12.62	4.94									
b22_1_C	757	20983	3.77	94.05	21646	3.16	1.54	1.14	24378	16.18	4.48									
b22_C	757	29162	5.27	122.22	29824	2.27	1.56	1.14	32850	12.65	4.87									
b17_C	1445	30777	6.02	180.25	32079	4.23	1.96	1.11	39363	27.90	5.94									
b17_1_C	1445	38116	7.21	252.01	39418	3.42	2.18	1.11	46870	22.97	6.06									
b18_1_C	3342	105102	22.41	742.71	108096	2.85	1.94	1.12	124199	18.17	5.71									
b18_C	3342	111241	23.61	1265.64	114233	2.69	1.95	1.12	130478	17.29	5.76									
b19_1_C	6669	212728	53.35	3787.02	218701	2.81	1.97	1.12	250943	17.96	5.73									
Average values:						2.89	2.02	1.17		17.38	5.07									

Table 2.3 – Experimental results obtained when applying combinational logic locking on ITC'99 benchmarks.

Figure 2.12 shows a comparison of the analysis times. It shows that our algorithm can handle very large combinational netlists. A netlist of 200 000 nodes takes around one hour to be analysed and made lockable. Detailed values can be found in the “Graph building time (s)” and “Graph processing time (s)” columns of Table 2.3. On the other hand, fault analysis-based logic masking cannot cope with large netlists. As said in the original article [Raj+15], “*This method took two hours to encrypt the C7552 circuit.*”, which is a benchmark of 3,500 gates. Graph analysis-based logic locking is then a very computationally efficient method compared to other heuristics used for logic modification of combinational aiming at IP protection.

### 2.3.3 Distance to outputs

Another metric that can be used to assess the efficiency of IP protection schemes based on logic locking is the distance from these gates to the primary outputs of the netlist. Indeed, one wants the inserted gates to lock the outputs while being as far from them as possible. This is to make their isolation and identification by reverse-engineering harder. The average distance from the inserted locking gates to the outputs that are reachable from them is given in Table 2.3. The definition of distance is the one used for graphs. Namely, it is the average number of edges found between the node considered and the nodes corresponding to the outputs. Detailed values can be found in the “Average distance to outputs” column of Table 2.3. The average value of 2.02 highlights the fact that the inserted locking gates are quite close to the outputs. This is because the sequences of nodes leading to the outputs that are capable of propagating a locking value are rare. Section 2.4 discusses possible improvements to increase this distance and obfuscate the locking gates.

### 2.3.4 Number of outputs affected

Finally, the last criterion that can evaluate the efficiency of IP protection schemes based on logic gates insertion is the average number of outputs that are affected by each extra locking gate inserted. For logic locking specifically, the effect of the locking gates is maximal and completely locks the output. Therefore, this is not required that multiple locking gates affect each output. This could be the case though, and is discussed in the following section.

We define the locking ratio as the number of inserted locking gates divided by the number of outputs of the netlist (see Equation (2.12)). Thus this ratio gives the average number of outputs affected by each inserted locking gate.

$$\text{locking ratio} = \frac{\text{\#inserted locking gates}}{\text{\#outputs}} \quad (2.12)$$

Detailed values can be found in the “Locking ratio at minimum overhead” column of Table 2.3. One can observe that the locking ratio is usually very close to 1. This indicates that the connected components found in the final graph after cleaning it usually contain only one

output. Every output has then its own locking gate. This has benefits and drawbacks. The benefit is that the connected components found in the final graph are very easy to analyse using the method presented in Section 2.2.4, since they contain only one output on average. Therefore, for every source node, only one distance from it to the output must be computed. The furthest one is then selected for logic locking. The drawback of having one locking gate assigned to each output is that the associated AW bit can be easily recovered by observing the input-output patterns. Indeed, flipping the AW bits one after the other allows to recover the whole AW easily [PM14], as discussed in Section 2.4.4.1. An  $n$ -bit AW is recovered after  $n/2$  trials on average. Section 2.4 proposes several ways to avoid this direct relation between locking gates and outputs.

## 2.4 Discussion

All the modifications suggested in this section consist in inserting additional logic after the locking gates have been inserted. In some cases, this extra combinational logic can be detected as redundant and simplified by a synthesiser. Therefore, they must be protected from such simplification, by specifying tool-specific constraints.

### 2.4.1 Locking strengthening

The aim of the methods described here is to tend toward an implementation of total logic locking given in Equation (2.1). The point is then to have as many AWs as possible for which as many outputs as possible are locked.

#### 2.4.1.1 Adding more locking gates to control one locking value

The logic resources overhead values given in Section 2.3.1 are the minimum required to be able to lock all the outputs. However, the final graph after cleaning contains a lot of other nodes that can propagate a locking value and are not selected because they are sub-optimal. Nevertheless, these nodes can be exploited to strengthen logic locking.

This is illustrated in Figure 2.13. The netlist portion in Figure 2.13a could potentially be locked by forcing  $X_0$  or  $X_1$  to 0. This is the optimal choice, requiring the minimum overhead and selecting the furthest nodes from the outputs. However, since all the other nodes can propagate a locking value, they could potentially all be forced. This is illustrated in Figure 2.13b, in which five locking gates are inserted. The output of such a netlist is then locked if and only if all the AW bits are set to the correct value. This effectively increases the brute force complexity in the average case from 1 to  $2^5/2 = 16$ . Indeed, only the correct AW value would allow the output to be correct. All the other combinations lock it.



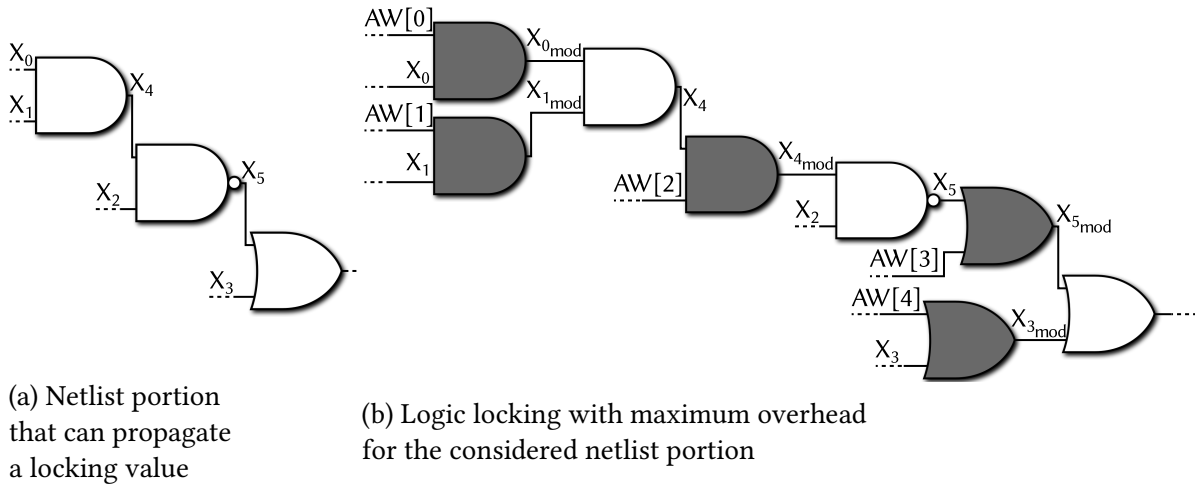


Figure 2.13 – Maximum logic locking of a netlist portion that can propagate a locking value

Figure 2.14 shows what the minimum and maximum overhead values for all the benchmarks we considered. Detailed maximum values can be found in the “Maximum overhead (%)” column of Table 2.3. The associated locking ratio values are given in the “Locking ratio at maximum overhead” column. One can observe that it differs greatly between benchmarks. However, for most of the cases, the designer has an interesting design margin, and can select the best trade-off between area overhead and locking strength.

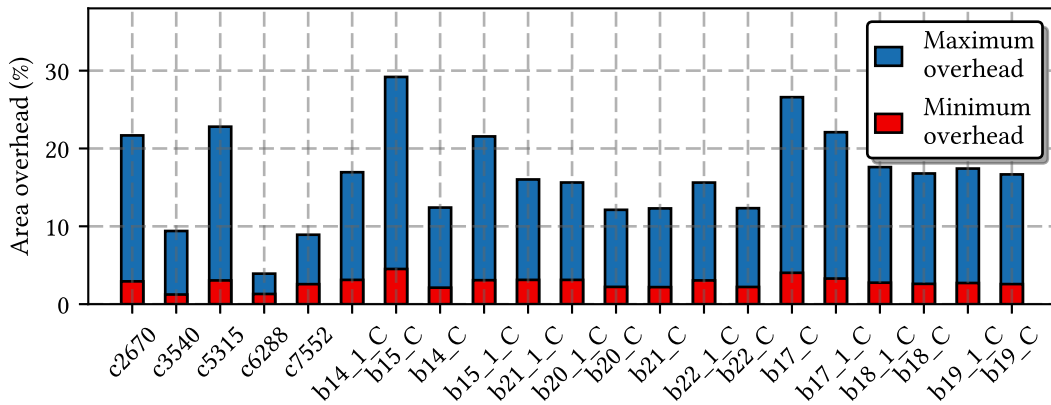


Figure 2.14 – Minimum and maximum overhead values for logic locking strength tuning

The solution proposed here tends to increase the length of the AW. To avoid this, AW bits can be interleaved and used to lock multiple outputs.

#### 2.4.1.2 AW bits interleaving

AW bits can be shared among connected components of the final graph in order to reduce the size of the AW and strengthen logic locking. Figure 2.15a shows three portions that belong to the same netlist that can be locked. Figure 2.15b shows these three portions with logic locking gates inserted. In the first portion, the locking gates inserted are the three OR gates at the

top. This corresponds to a high locking strength since multiple gates participate in locking one output, as described above.  $AW_{\text{valid}}$  is “000”. The second portion can also be locked by inserting three gates. However, some AW bits must be inverted to cope with the different types of locking gates that are picked. Namely, AW0 and AW2 are inverted to be reused. Finally, this can happen that some other netlist portions do not contain enough nodes that propagate a locking value to make use of all the available AW bits. This is the case in the last netlist portion of Figure 2.15a, in which forcing  $X_8$  or  $X_9$  would not lock the output. In this case, locking gates can be cascaded as in the bottom of Figure 2.15b where the locking gates associated to AW1 and AW2 are cascaded to lock  $X_{10}$ . Alternatively, fewer locking gates can be used.

In the example given in Figure 2.15b, if the 3-bit AW is different from “000”, the three outputs are locked. Even though it is limited to a 3-bit AW, this example shows an implementation of total logic locking as described in Equation (2.1).

Extending it to larger AW would of course induce a higher area overhead. A totally interleaved implementation with an  $n$ -bit AW requires to add  $n^2$  locking gates. This might not be affordable by the designer in practise. Instead, partial interleaving is possible, in which only a fraction of the AW bits are shared. It would make the set  $I_{y_{\text{unlocked}}}$  from Equation (2.4) smaller. This also has the side-benefit to allow to select the width of the AW, to adapt it to the output of a block cipher for instance. Again, this is up to the designer to pick the most appropriate trade-off.

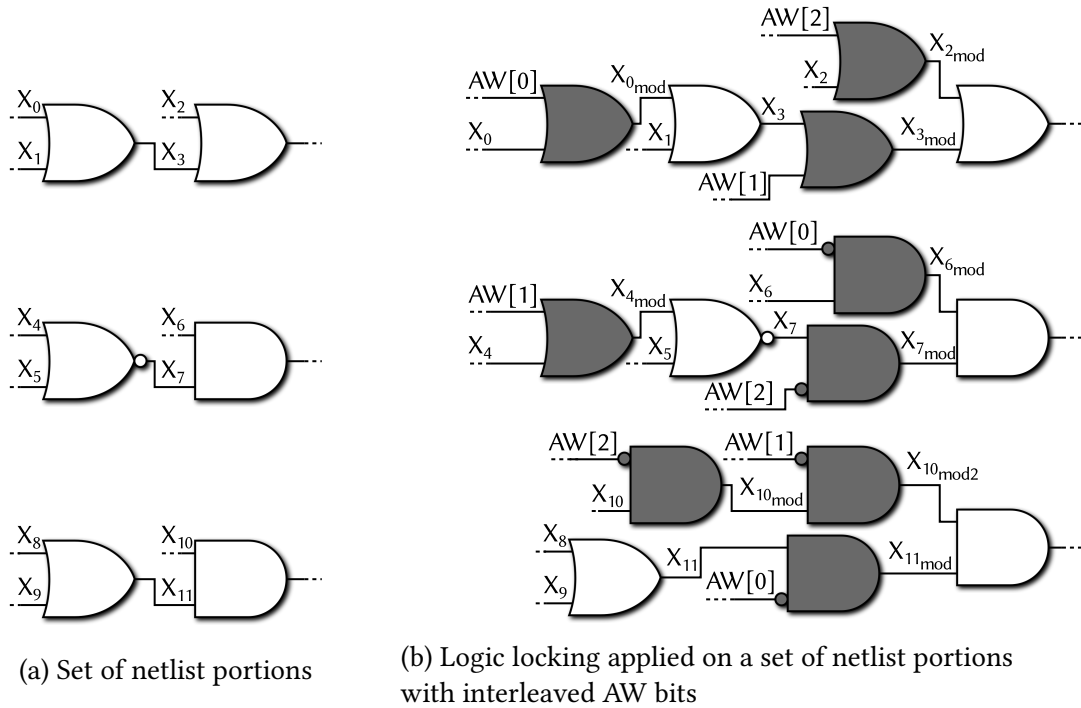


Figure 2.15 – Interleaving the AW bits to strengthen logic locking

### 2.4.1.3 Hardware point function

Finally, the last option is to implement a “hardware point function”. This is described in Equation (2.13). The output of this function is equal to the correct AW if it is fed at the input. Otherwise, it is equal to the complement of the correct AW.

$$F(x) = \begin{cases} AW_{\text{valid}} & \text{if } x = AW_{\text{valid}} \\ \overline{AW_{\text{valid}}} & \text{if } x \neq AW_{\text{valid}} \end{cases} \quad (2.13)$$

This function can also be used to adapt the width of the AW. Moreover, it can also adapt the logic value of the AW bits. This can be useful if the AW is combined internally with an instance-specific identifier such as the response of a PUF to make each instance uniquely unlockable. This requires to map the PUF response to the AW and can be done by this function.

A hardware implementation of such a function is trivial. Each logic 0 of the AW, found at the output of the function, is driven by the sum of all the logic 0s found at the input. An AND gate with the appropriate fan-in and fan-out is then used. Similarly, each logic 1 of the AW is driven by the product of all the logic 1s found at the input. An OR gate with the appropriate fan-in and fan-out is then used.

The hardware point function is a lightweight structure, that does not require much logic resources. The experimental results obtained after implementing it on FPGA are given in Table 2.4. Only the input width matters, while the output width can be very large without affecting the number of LUTs used. This is because increasing the output width only requires more wiring to drive the individual AW bits, which does not require additional logic resources on FPGA. Consequently, implementing a hardware point function to turn a weak logic locking implementation into total logic locking is not costly and easily achievable.

Input width (bits)	Output width (bits)	# 4-LUTs required	# 6-LUTs required
64	64	17	12
64	128	16	13
64	256	17	14
64	512	17	14
64	1024	16	14
64	2048	16	14
128	64	33	22
128	128	33	22
128	256	33	22
128	512	33	22
128	1024	33	22
128	2048	33	22

Table 2.4 – Logic resources required to implement a hardware point function for different input and output widths

### 2.4.2 Obfuscation using extra logic layers

The main issue with the current description of combinational logic locking is the fact that the inserted gates are very close to the outputs. In order to conceal them more, adding dummy logic layers between them and the outputs is a solution. Those logic layers should have no effect on the functionality of the netlist portion. That is, when the correct AW bit is provided, the output must be valid. When the wrong AW bit is provided, the output must be locked.

Figure 2.16 depicts an obfuscated locking OR gate. The original locking gate is in dark grey. The two additional logic gates in light grey add an extra logic layer between the locking gate and the output. Moreover, one of these obfuscation gates is fed with a value taken randomly in the netlist ( $X_j$  in Figure 2.16). This connection could be obtained from a very different location in the netlist.

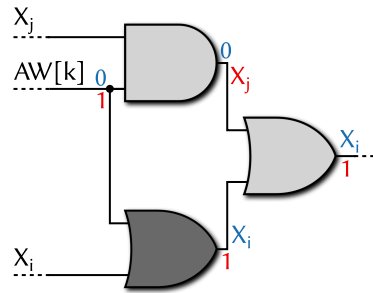


Figure 2.16 – OR locking gate (in dark grey) obfuscated by two extra gates (in light grey) with logic values shown in red and blue depending on the value of the AW bit

The overhead brought by this additional obfuscation method is not negligible though. Indeed, the area overhead brought by logic locking is increased again by the obfuscation gates. Adding  $n$  extra logic layers for obfuscation increases the locking overhead  $2n$  times. For example, if the locking overhead is originally 3%, obfuscating with one extra logic layer brings it to 9%. Therefore, this solution might not be suited to all the cases, especially if the designer has strong area constraints.

### 2.4.3 Exploiting connected components that contain no output

In Section 2.2.3, the connected components that contain no output are deleted from the final graph. Indeed, they do not participate directly in logic locking since they do not force any output to a fixed logic value. However, they can force an internal node. Although the effect of this internal locking is hard to estimate, it could still be studied and leveraged on a per-design basis.

## 2.4.4 Security considerations

### 2.4.4.1 Hill-climbing attack

Due to the fact that AW bits are directly related to the output they lock, the Hamming distance between the AW that is fed and the correct one  $AW_{\text{valid}}$  is proportional to the number of outputs that are locked (see Equation (2.14)).

$$HD(AW, AW_{\text{valid}}) \propto \# \text{outputs locked} \quad (2.14)$$

Therefore, there is a gradient toward  $AW_{\text{valid}}$ . By successively flipping the AW bits, the output bits can be unlocked one after the other. This is called the hill-climbing attack and has been described in [PM14], originally against logic masking. Algorithm 1 shows how this attack applies to weak logic locking.

---

**Algorithm 1:** Hill climbing attack on weak logic locking

---

**Input:** Locked IP core with an  $n$ -bit AW

**Output:** Unlocked IP core

```

1 Randomly pick one AW
2 for  $i$  ranging from 0 to  $n - 1$  do
3   Feed random input values to the netlist
4    $n_{\text{locked\_1}} \leftarrow \# \text{outputs locked}$ 
5   Flip the  $i^{\text{th}}$  bit of AW.
6   Feed random input values to the netlist
7    $n_{\text{locked\_2}} \leftarrow \# \text{outputs locked}$ 
8   if  $n_{\text{locked\_2}} > n_{\text{locked\_1}}$  then
9     Flip back the  $i^{\text{th}}$  bit of AW.
10 Return:  $AW_{\text{valid}}$ 

```

---

Although originally proposed against logic masking, the hill-climbing attack affects weak logic locking just as well. However, in the case of total logic locking, the outputs are all fixed until the correct AW is fed. Therefore, the comparison done at line 8 of Algorithm 1 cannot be carried out. Thus total logic locking is not subject to the hill-climbing attack.

### 2.4.4.2 SAT attack

In 2015, a so-called SAT attack has been proposed [SRM15] which applies logic locking/masking algorithms. The attacker has access to a netlist and a functional circuit which operates normally. The attack works by applying iteratively input patterns that have a distinguishing property. They are called distinguishing input patterns (DIPs). An input pattern is a DIP if, when two

different AWs are fed to the dedicated inputs, the outputs are different. When carefully chosen, DIPs can rule out multiple AWs at a time, reducing the search space rapidly.

Weak logic locking is affected by this attack. However, for strong logic locking, the outputs are all fixed for all wrong AWs. Therefore, one cannot find DIP in this case, since the output is always the same. Thus total logic locking is also not subject to the SAT attack.

## **2.5 Conclusion**

Total combinational logic locking is a new way to controllably lock the combinational part of a netlist. Based on the propagation of a locking value through specific sequences of nodes, it has the advantage to be very efficient to compute by using graph analysis. It can cope with very large netlists in a reasonable amount of time.

Hardware implementations on a wide range of benchmarks show that the area overhead to implement logic locking is limited, since it requires on average a 2.89% increase of the number of logic gates.

However, the direct relation between the inserted locking gates and the output(s) they lock makes it trivial to recover the correct AW if the AW inputs are directly exposed. We propose several solutions that allow a designer to strengthen the logic locking scheme intrinsically and make the AW bits interdependent. This highlights another interesting feature of logic locking from an industrial point of view, which is its great flexibility. Indeed, it offers a wide trade-off between area overhead and locking strength, leaving up to the designer the final tradeoff between cost and security. Another way to make the system more secure is to instantiate a lightweight cipher besides the logic locking module, with the output of the cipher driving the AW inputs. This solution is explored in the last chapter of this thesis in which a complete IP protection scheme architecture is detailed.



## Chapter 3

# Centrality indicators for efficient and scalable combinational logic masking

---

In the previous chapter, a degraded mode of operation called logic locking has been presented. However, the first degraded mode of operation based on modifications of combinational logic published in literature [RKM08a] is logic masking, sometimes referred to as “logic encryption”. It consists in altering the internal state of an IP core unless the correct AW is fed. To this end, XOR or XNOR logic gates are inserted at specific locations in the netlist. The aim is to controllably disturb the internal state as much as possible, while keeping the logic resources overhead induced by the extra gates as low as possible.

Based on the article presenting the principle of logic masking in 2008 [RKM08a], several heuristics have been proposed to select the best locations of insertion for the extra masking gates in the netlist. A closer look reveals, however, that these heuristics are either easy to compute or efficient at disrupting the internal state, but cannot meet both requirements. For industrial feasibility, one needs a selection heuristic that can cope with large netlists while offering efficient disruption of the outputs when the wrong AW is fed. In order to bridge the gap and offer a balance between computational efficiency and masking efficiency, we propose to use centrality indicators. Originating from graph theory, they allow to rank the nodes of a graph according to their relative *significance*.

We start by giving an overview of common centrality indicators before comparing them for application to logic locking. We show that they disturb the outputs of the netlist efficiently, effectively reducing the correlation between normal and masked outputs to low values. At the same time, they are efficient to compute, approximately one thousand times faster than the heuristic with the highest making efficiency, based on fault analysis [Raj+15]. This allows to handle netlists of up to 100 000 nodes, paving the way for integration into EDA tools.

---

The code associated with this chapter is available at:  
<https://gitlab.univ-st-etienne.fr/b.colombier/centrality-based-logic-masking/tree/master>



# Indicateurs de centralité pour le masquage logique combinatoire efficace et adaptable

---

Dans le chapitre précédent, un mode de fonctionnement appelé verrouillage logique a été présenté. Néanmoins, le premier mode de fonctionnement dégradé basé sur une modification de la logique combinatoire, publié en 2008 [RKM08a], est le masquage logique. Cela consiste à perturber l'état interne du composant virtuel à moins que le bon mot d'activation ne soit fourni. Pour ceci, des portes XOR ou XNOR sont insérées à des positions spécifiques dans le composant virtuel. L'objectif est de perturber l'état interne autant que possible tout en limitant le surcoût en ressources logiques induit par les portes supplémentaires.

Se basant sur le premier article sur le sujet publié en 2008 [RKM08a], plusieurs heuristiques ont été proposées pour sélectionner le meilleurs lieux d'insertion pour les portes de masquage à ajouter au composant virtuel. Une étude plus approfondie révèle, néanmoins, que ces heuristiques sont soit faciles à calculer soit efficaces pour perturber l'état interne, mais ne satisfont jamais ces deux critères simultanément. Dans un contexte d'utilisation industriel, l'heuristique de sélection doit être facile à calculer pour pouvoir gérer des composants virtuels de grande taille tout en offrant une perturbation efficace des sorties si le mauvais mot d'activation est appliqué. Pour un compromis entre ces deux objectifs, nous proposons d'utiliser les indicateurs de centralité. Venant de la théorie des graphes, ils permettent de classer les sommets d'un graphe en fonction de leur *importance*.

Nous commençons par donner une vue d'ensemble des indicateurs de centralité communs avant de les comparer pour une utilisation dans le cadre du masquage logique. Nous montrons qu'ils permettent de perturber efficacement les sorties du composant virtuel, réduisant la corrélation entre les sorties normale et masquée à des valeurs faibles. Dans le même temps, leur complexité est limitée, et ils sont mille fois plus rapides à calculer que l'heuristique la plus efficace de l'état de l'art basée sur l'analyse de fautes [Raj+15]. Cela permet de gérer des composants virtuels incluant jusqu'à 100 000 nœuds, ouvrant la voie à une intégration dans les outils de conception électronique.

---

Le code associé à ce chapitre est disponible à :  
<https://gitlab.univ-st-etienne.fr/b.colombier/centrality-based-logic-masking/tree/master>

### 3.1 Definition

Logic masking consists in inserting linear logic gates (XOR or XNOR) at well-chosen locations inside the netlist so that the outputs of the netlist are maximally corrupted if the wrong AW is fed to the dedicated activation inputs [RKM08a; RKM10]. These activation inputs are connected to one of the inputs of the inserted masking gates while their other input is connected to the internal node to mask (see Section 1.5.4.1, Figure 1.16) We call normal output values the ones obtained with the original netlist or with the masked one when the correct AW is fed to the activation input. We call masked output values the ones obtained with the masked netlist when the wrong AW is fed to the activation input. The aim is to alter the internal state of the netlist so that the similarity between the normal and masked output values is as low as possible.

### 3.2 A proposal for a masking efficiency evaluation metric

#### 3.2.1 Existing metrics for masking efficiency and their weaknesses

As detailed in Chapter 1, the first metric which is used to evaluate logic masking was corruptibility [RKM08a; RKM10]. Given in Equation (1.7), it makes sure that the output is valid only when the correct AW is applied. However, it does not qualify the masking efficiency. Indeed, inverting only one output bit is sufficient to ensure that the corruptibility requirement is satisfied. Later on, a requirement on the Hamming distance between the normal and masked output was derived [Raj+12a; Raj+13] (see Equation (1.8)). This Hamming distance should be of 50% on average.

However, this requirement alone is still not sufficient. Indeed, just as inverting one output permanently satisfies the corruptibility criterion, inverting half the outputs permanently satisfies the Hamming distance criterion. Thus there is a need for a stronger, more restrictive metric that could evaluate the masking efficiency. In [Raj+15], it is said that efficient masking “*can be done by minimizing the correlation between the corrupted and the original outputs*”. Therefore, we propose to develop a metric based on correlation to measure the masking efficiency.

#### 3.2.2 A new metric based on correlation

The two previous approaches fail at handling the cases described above since they deal with the output bits as a whole instead of considering them separately. Instead of considering an output vector, we will then deal with output bits as binary variables. The correlation between two binary variables can be computed in its simplest form by the Phi coefficient. Table 3.1 is the contingency table of the two binary variables  $y[i]$  and  $y_{\text{masked}}[i]$ .  $P_{00}$  and  $P_{11}$  represent the proportion of positions where the variables are identical. Conversely,  $P_{01}$  and  $P_{10}$  represent the proportion of positions where the variables are different.

		y[i]		Totals
		0	1	
y <sub>masked</sub> [i]	0	P <sub>00</sub>	P <sub>01</sub>	p <sub>1</sub>
	1	P <sub>10</sub>	P <sub>11</sub>	q <sub>1</sub>
Totals		p <sub>2</sub>	q <sub>2</sub>	1

$$\begin{aligned}
 \text{where : } p_1 &= P_{00} + P_{01} \\
 q_1 &= P_{10} + P_{11} \\
 p_2 &= P_{00} + P_{10} \\
 q_2 &= P_{01} + P_{11}
 \end{aligned}$$

 Table 3.1 – Contingency table of the binary variables y[i] and y<sub>masked</sub>[i].

The Phi coefficient is then given by Equation (3.1).

$$\phi = \frac{P_{00}P_{11} - P_{01}P_{10}}{\sqrt{p_1 q_1 p_2 q_2}} \quad (3.1)$$

In order to account for all the output bits and get a global metric, we propose to compute the quadratic mean of the Phi coefficients obtained for all the outputs. This way, Phi coefficients with an opposite sign for different outputs cannot compensate themselves. The masking efficiency metric is given in Equation (3.2).

$$E_m = 1 - \sqrt{\frac{1}{n} \sum_{i=0}^{\text{\#outputs}-1} \phi^2(y[i], y_{\text{masked}}[i])} \quad (3.2)$$

If the outputs are not masked,  $P_{00} = P_{11} = 1$ , so  $\phi = 1$  for all the outputs. Thus  $E_m = 0$ . If the outputs are perfectly masked, then for each output there is a 50% probability that it is inverted. Therefore, for every output,  $P_{00} = P_{11} = P_{01} = P_{10} = 0.25$  and  $\phi = 0$ . Thus  $E_m = 1$ . The masking efficiency evaluation metric  $E_m$  is then more constraining than the ones that were previously used [RKM08a; Raj+12a], based on corruptibility or Hamming distance.

Table 3.2 summarises how these metric perform at evaluating the masking efficiency. The first column correspond to the case where one output is inverted, as described above. The second columns correspond to the case where half the outputs are inverted. The last column shows the case where one XOR gate is added on every output. When measuring the masking efficiency with corruptibility, Hamming distance or bitwise correlation with  $E_m$ , this architecture is optimal. Indeed, it implements a kind of one-time pad on the outputs. Therefore, randomly picking an AW makes the correlation drop to 0.

### 3.2.3 Further requirements for a logic masking scheme

Even though inserting one XOR gate on every output achieves good masking efficiency according to  $E_m$ , looking at other criteria makes this architecture unusable. The first drawback is the fact that one bit of the AW is responsible for masking only one outputs bit. Therefore, the hill climbing attack presented in Section 2.4.4.1 and Algorithm 1 is very much applicable in this case too. Instead of observing which outputs are fixed, comparing with test vectors is sufficient to detect the wrong output bits, as it is done in the original article [PM14].

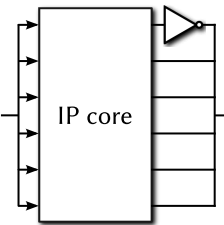
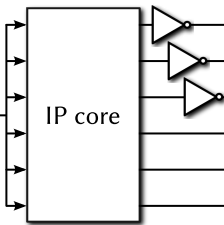
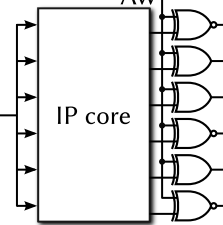
Metric	Masked IP cores		
			
Corruptibility [RKM08a]	✓	✓	✓
Hamming distance [Raj+12a]	×	✓	✓
Proposed metric $E_m$	×	×	✓

Table 3.2 – Masking efficiency evaluation by different metrics. ✓ stands for the masking efficiency being evaluated as good by the metric. × stands for the masking efficiency being evaluated as bad by the metric.

Moreover, another greater drawback is that a much simpler attack can be carried out if the attacker has access to a functional copy of a circuit that implements the IP core. By comparing a correct input-output pair  $(x, y)$  obtained from the functional circuit with an input-output pair  $(x, y_{\text{masked}})$  obtained from the masked one then the correct AW can be trivially computed (see Equation (3.3)).

$$AW_{\text{valid}} = y \oplus y_{\text{masked}} \quad (3.3)$$

In order to avoid this, the masking gates must be inserted deeper inside the netlist, so that each output is affected by multiple masking gates. To this end, various heuristics have been proposed to select the nodes to modify (see Section 1.5.4.1 for details). The following sections describe two heuristics that we investigated, based on controllability/observability and centrality indicators.

### 3.3 Selection of the place of insertion

#### 3.3.1 Combinational controllability and observability

The first metric we investigated is based on the concepts of combinational controllability and observability. They were first described in [Gol79], along with their sequential counterparts. They are very useful for testing a circuit, because they characterise how easy it is to set the value of a node from the primary inputs and observe this value at the primary outputs. Since we only deal with combinational logic masking here, we consider only combinational controllability and combinational observability.

### 3.3.1.1 Description

The combinational controllability of a netlist node measures how hard it is to set this particular node to a given logic value. Combinational 0 controllability (CC0) (respectively combinational 1 controllability (CC1)) measures how hard it is to set the node to 0 (respectively to 1). For a node  $N$ ,  $CC0(N)$  (respectively  $CC1(N)$ ) is then related to the number of primary inputs that must be set to a fixed logic value to set  $N$  to 0 (respectively to 1).

For example, in order to set the output of a 2-input AND gate to 1, both its inputs must be set to 1. Therefore, the hardness to set the output to 1 is the sum of hardnesses to set each input to 1. Conversely, setting the output to 0 only requires to set one input to 0. Let  $Y = A \cdot B$  be the equation of this logic gate, then the values of CC0 and CC1 for the output are given in Equations (3.4) and (3.5).

$$CC1(Y) = CC1(A) + CC1(B) + 1 \quad (3.4)$$

$$CC0(Y) = \min(CC0(A), CC0(B)) + 1 \quad (3.5)$$

By convention, the controllability of the primary inputs of the netlist is 0. Therefore, a high controllability value corresponds to a node that is hard to control. Table 3.3 gives the formulas to compute the controllability for the output of usual 1 and 2-input logic gates. For each of them, their logic equation is of the form  $Y = F(A)$  if  $F$  is a unary boolean function or  $Y = F(A, B)$  if  $F$  is a binary boolean function.

Logic gate	CC0(Y)	CC1(Y)
NOT	$CC1(A) + 1$	$CC0(A) + 1$
AND	$\min(CC0(A), CC0(B)) + 1$	$CC1(A) + CC1(B) + 1$
NAND	$CC0(A) + CC0(B) + 1$	$\min(CC1(A), CC1(B)) + 1$
OR	$\min(CC1(A), CC1(B)) + 1$	$CC0(A) + CC0(B) + 1$
NOR	$CC1(A) + CC1(B) + 1$	$\min(CC0(A), CC0(B)) + 1$
XOR	$\min(CC0(A) + CC0(B), CC1(A) + CC1(B)) + 1$	$\min(CC0(A) + CC1(B), CC1(A) + CC0(B)) + 1$
XNOR	$\min(CC0(A) + CC1(B), CC1(A) + CC0(B)) + 1$	$\min(CC0(A) + CC0(B), CC1(A) + CC1(B)) + 1$

Table 3.3 – Controllability values of the output of usual 1 and 2-input logic gates. Their logic equation is of the form  $Y = F(A)$  if  $F$  is a unary boolean function or  $Y = F(A, B)$  if  $F$  is a binary boolean function.

In addition to controllability, we also considered observability. The observability of a netlist node measures how hard it is to observe its value at the primary outputs of the netlist. For example, observing the value of one of the inputs of a 2-input OR gate requires to propagate it at the output by setting the other node to 0. Therefore, the observability of this input node depends on the combinational observability (CO) of the output and the CC0 value of the other input. Let  $Y = A + B$  be the equation of this logic gate, then the CO value for input  $A$  is given in Equation (3.6).

$$CO(A) = CO(Y) + CC0(B) + 1 \quad (3.6)$$

By convention, the observability of the primary outputs of the netlist is 0. Therefore, a high observability value corresponds to a node that is hard to observe. Table 3.4 gives the formulas to compute the observability for the input(s) of usual 1 and 2-input logic gates. For the logic gates that implement a binary boolean function, we consider only the  $A$  input. Since those inputs are identical, simply replacing  $B$  by  $A$  gives the formulas for the  $B$  input.

Logic gate	CO(A)
NOT	$\text{CO}(Y) + 1$
AND	$\text{CO}(Y) + \text{CC1}(B) + 1$
NAND	$\text{CO}(Y) + \text{CC1}(B) + 1$
OR	$\text{CO}(Y) + \text{CC0}(B) + 1$
NOR	$\text{CO}(Y) + \text{CC0}(B) + 1$
XOR	$\text{CO}(Y) + \min(\text{CC0}(B), \text{CC1}(B)) + 1$
XNOR	$\text{CO}(Y) + \min(\text{CC0}(B), \text{CC1}(B)) + 1$

Table 3.4 – Observability values of the input(s) of usual 1 and 2-input logic gates.

### 3.3.1.2 Selection heuristic for logic masking

Ideally for logic masking, our first approach was to select the nodes with high controllability (*i.e.* nodes that are hard to control) as well as low observability (*i.e.* nodes that are visible at the outputs). Unfortunately, this is exactly the definition of the primary outputs of the netlist. Using this metric, we ended up selecting the primary outputs, which is not a good option as described above.

The nodes selected for logic masking should be located deeper inside the netlist. This led us to define a metric for the selection heuristic given in Equation (3.7).

$$M(e) = \sqrt{\text{CC0}(e)^2 + \text{CC1}(e)^2 + \text{CO}(e)^2} \quad (3.7)$$

We then selected for logic masking the nodes for which this metric is maximised. However, using this selection heuristic turned out to be unsuccessful. The nodes that are selected have a low impact on the outputs.

We managed to obtain good results individually for some benchmarks by assigning different weights to controllability and observability values:  $w_{\text{CC}}$  and  $w_{\text{CO}}$ , see Equation (3.8).

$$M(e) = \sqrt{w_{\text{CC}}\text{CC0}(e)^2 + w_{\text{CC}}\text{CC1}(e)^2 + w_{\text{CO}}\text{CO}(e)^2} \quad (3.8)$$

However, this requires to tune the coefficients for each benchmark specifically. The  $E_m$  values that we observed were still considerably high, indicating that logic masking was not very efficient. The trade-off between inserting the masking gates deep inside the netlist and having them to disturb the outputs efficiently is hard to balance.

In order to insert the masking gates more efficiently and have a greater impact on the outputs, we investigated the use of centrality indicators. This is detailed in the following section.

### 3.3.2 Centrality indicators

Centrality indicators originate from graph theory. As their name suggests, they measure how *central* or *significant* a particular node is inside a given graph. Of course, the notion of centrality or significance is very broad. Therefore, a large range of centrality indicators have been proposed in literature. For some applications, some centrality indicators are more suited than others. For example, the PageRank indicator, used by Google to measure the popularity of web pages, is a centrality indicator that has been specifically designed for this usage.

Centrality indicators, depending on how they are defined, can give a centrality value that belongs to very different ranges. We chose to normalise it by dividing the raw centrality value for the vertex of interest by the maximum value obtained for the vertices of the graph (see Equation (3.9) where  $v$  is the considered vertex and  $V$  is the set of all the vertices of the graph). The centrality values then range from 0 to 1.

$$C(v) = \frac{C_{\text{raw}}(v)}{\max(C(i))}, i \in V \quad (3.9)$$

For some centrality indicators, the literal formulas given in the original articles include a normalising factor. We chose to not take them into account, since we are only interested in the relative values for the centrality.

#### 3.3.2.1 Conversion from netlist to graph

Converting the netlist into a graph is done as described in Section 2.2.1. The nodes of the netlist are converted to vertices and connected by directed edges labelled after the logic function.

#### 3.3.2.2 Degree centrality

Degree centrality measures the significance of a vertex by its number of incoming and outgoing edges. The in-degree  $\deg^-(v)$  is computed by counting incoming edges only. The out-degree  $\deg^+(v)$  is computed by counting outgoing edges only. The centrality value is the degree  $\deg(v)$ , computed by summing the two previous values (see Equation (3.10)). Figure 3.1 illustrates the degree centrality values of the vertices of a random graph.

$$C_D(v) = \deg(v) = \deg^-(v) + \deg^+(v) \quad (3.10)$$

This is not a good indicator for logic masking though. Indeed, by synthesising the netlist in different ways, some vertices can have their degree centrality changed even if the original logic



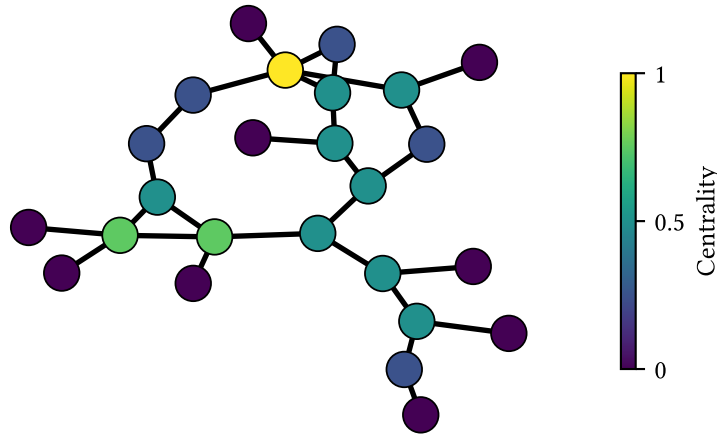


Figure 3.1 – Degree centrality values for the vertices of a random graph

function is identical. An example is given in Figure 3.2. The logic function  $G5 = G1 \cdot G2 \cdot G3 \cdot G4$  can be synthesised into two different forms, using one 4-input AND gate (see Figure 3.2a) or three 2-input AND gates (see Figure 3.2b). In the resulting associated graphs, shown in Figures 3.2c and 3.2d, the same vertex  $G5$  has a different degree centrality. In Figure 3.2c,  $C_D(G5) = 4$ , while in Figure 3.2d,  $C_D(G5) = 2$ . This discrepancy makes the centrality indicator dependent on the implementation, when it should only depend on the overall structure of the logic function.

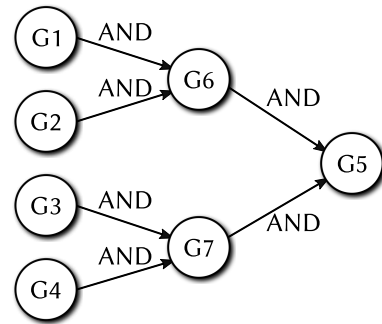
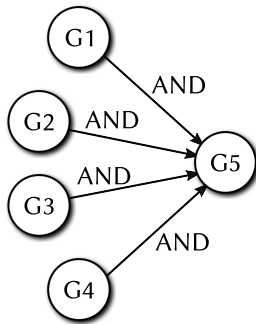
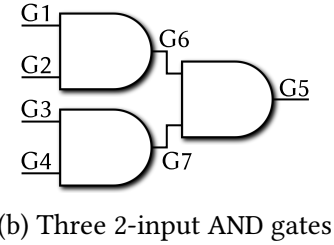
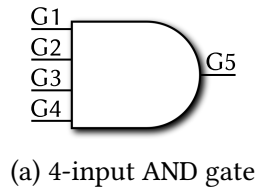


Figure 3.2 – Boolean function  $G5 = G1 \cdot G2 \cdot G3 \cdot G4$  synthesised using a 4-input AND gate (a) or three 2-input AND gates (b). The resulting graphs (c) and (d) lead to different degree centrality values for the vertex  $G5$ .

Moreover, the degree centrality is a *local* indicator. Thus it is only influenced by the direct neighbours of the vertex. Instead, *global* indicators should be used, because they take the whole graph into account.



### 3.3.2.3 Closeness centrality

Closeness centrality [Sab66] is the inverse of farness. The farness of a vertex  $v$  is the sum of distances from this vertex to all the other vertices of the graph. Closeness centrality of a vertex  $v$  is given in Equation (3.11), where  $V$  is the set of all the vertices of the graph and  $d(v, y)$  stands for the distance between vertices  $v$  and  $y$ .

$$C_C(v) = \frac{1}{\sum_{y \in V} d(v, y)} \quad (3.11)$$

A vertex is considered as important by the closeness centrality indicator if it is close to most of the other vertices of the graph. The vertices with the highest closeness centrality correspond to the nodes that are “in the middle” of the netlist. For logic masking, it is a more interesting indicator than degree centrality because it is global. Therefore, it is influenced by the graph structure and identifies the important nodes efficiently. Figure 3.3 shows the values of closeness centrality on an example graph. Note that a very efficient algorithm for approximating closeness centrality was proposed in [EW01] and runs in near-linear time.

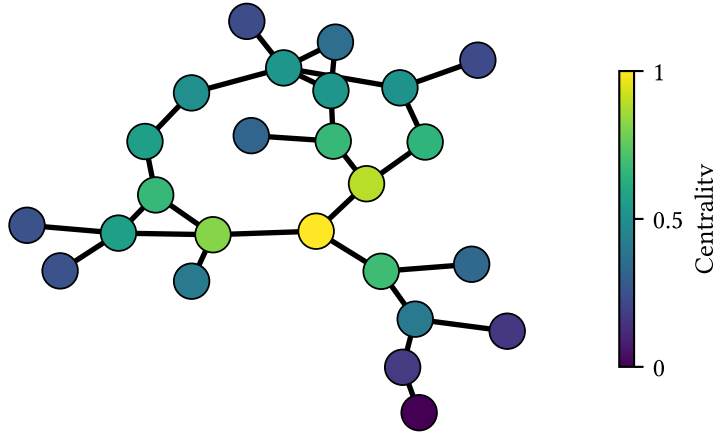


Figure 3.3 – Closeness centrality values for the vertices of a random graph

### 3.3.2.4 Betweenness centrality

Proposed in [Ant71; Fre77], betweenness centrality is the ratio of shortest paths between all the other pairs of vertices of the graph that go through the vertex of interest. Equation (3.12) shows the expression of betweenness centrality, in which  $\sigma_{st}$  stands for the number of shortest paths from  $s$  to  $t$ , and  $\sigma_{svt}$  stands for the number of shortest paths that go from  $s$  to  $t$  through  $v$ .

$$C_B(v) = \sum_{s \neq t, \{s, t\} \in V} \frac{\sigma_{svt}}{\sigma_{st}} \quad (3.12)$$

For a netlist, betweenness centrality is the highest for the nodes that are on the shortest paths from the inputs to the outputs. This is depicted in Figure 3.4 on an example graph.

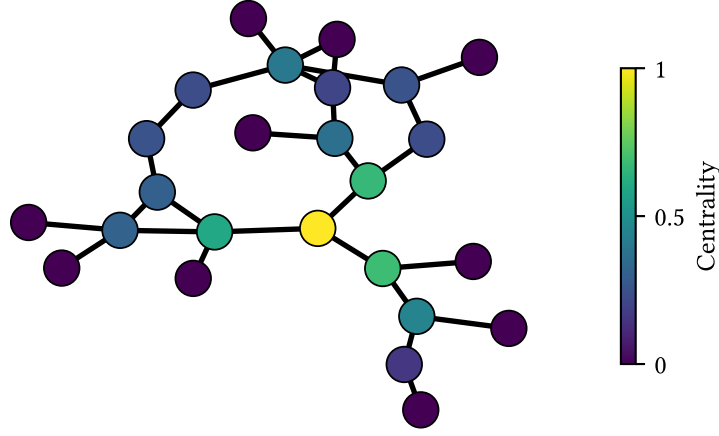


Figure 3.4 – Betweenness centrality values for the vertices of a random graph

This indicator, however, has the drawback to only take shortest paths, also referred to as *geodesic* paths, into account. This restriction is pointed out in [SZ89], implying that the information transits mostly on the shortest paths, which is not always the case. Instead of taking into account the shortest paths only, authors of [BF05] propose to assign a weight to paths according to their length. This is done by considering the graph as a network of unit resistors and measuring the current flowing through the nodes. This accounts for the fact that information, just like current, can split and spread in the network. These centrality indicators, based on current flow, are detailed below.

### 3.3.2.5 Current-flow betweenness centrality

In order to compute current-flow betweenness centrality [New05], the graph is considered as an electrical network. Vertices are converted to nodes. If two vertices are connected in the original graph, a unit resistor is added between the corresponding nodes in the electrical network.

Once the network is built, pairs of vertices are picked one after the other and set as current inputs and outputs. The current flowing through the node of interest for which the centrality is computed is added for all the possible pairs of vertices. An example is given in Figure 3.5. On the left-hand side, Figure 3.5a, an example graph is shown for which current-flow betweenness centrality is computed for the vertex G3. On the right-hand side, Figure 3.5b, the equivalent electrical network of the graph is shown. An example of current input/output selected is given although all pairs of nodes are selected iteratively for the centrality computation.

The expression for the current-flow betweenness centrality of vertex  $v$  is given in Equation (3.13), where  $I_v^{(st)}$  is the current flowing through node  $v$  when  $s$  is the current input and  $t$  is the current output. The current flowing through a node is computed using Kirchhoff's current law.

$$C_{CFB}(v) = \sum_{s \neq t: \{s, t\} \in V} I_v^{(st)} \quad (3.13)$$

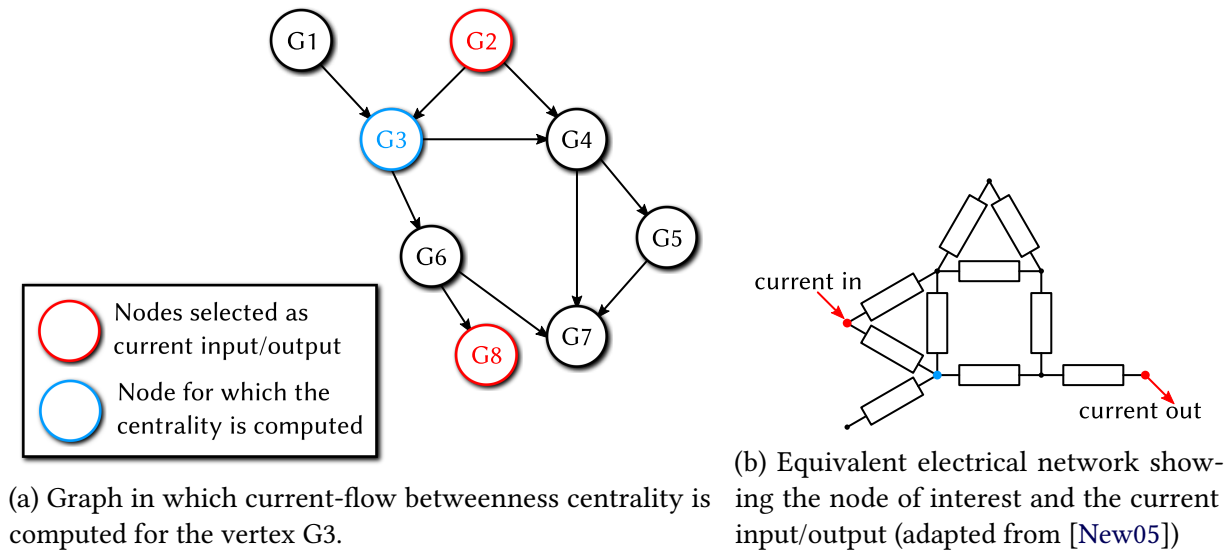


Figure 3.5 – Current-flow betweenness centrality computation on a graph and equivalent electrical network

Figure 3.6 shows the current-flow betweenness centrality values obtained for the previously considered random graph.

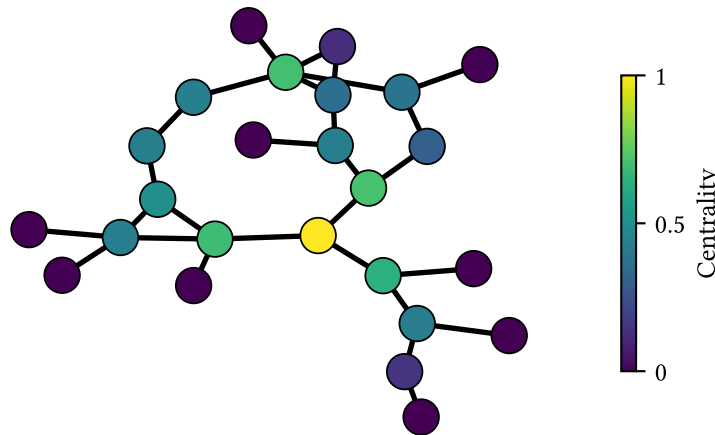


Figure 3.6 – Current-flow betweenness centrality values for the vertices of a random graph

**Approximated current-flow betweenness centrality** The running time and space for computing current-flow betweenness centrality become rapidly impractical. In [BF05], authors show that instead of selecting all the possible nodes pairs, a subset of them can be used. This comes at the cost of a loss in the precision of the centrality indicator. In the use case we consider here, we are only interested in the relative centrality of the nodes in order to select the most important ones. Therefore, a lack of precision is not strictly prohibitive.

### 3.3.2.6 Current-flow closeness centrality

A second centrality indicator that leverages the transformation of a graph into an electrical network of unit resistors is current-flow closeness centrality [BF05]. This has been shown to be equivalent to information centrality, originally proposed in [SZ89].

The expression of current-flow closeness centrality of a vertex  $v$  is given in Equation (3.14), in which  $R_{\text{eff}}(v, y)$  stands for the effective resistance between the nodes  $v$  and  $y$ . The notion of effective resistance intuitively conveys the notion of “distance” between the nodes which is necessary to measure the closeness. Just like the current accounted for non-geodesic paths in current-flow betweenness centrality, the effective resistance accounts for non-geodesic paths in current-flow closeness centrality. An example of current-flow closeness centrality values is shown in Figure 3.7.

$$C_{\text{FC}}(v) = \frac{1}{\sum_{y \in V} R_{\text{eff}}(v, y)} \quad (3.14)$$

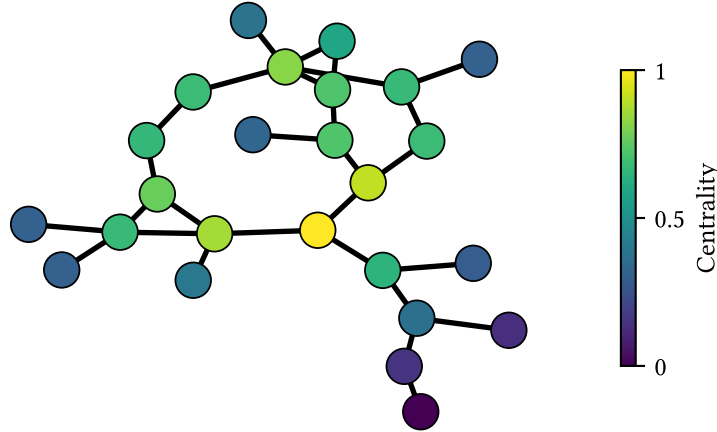


Figure 3.7 – Current-flow closeness centrality values for the vertices of a random graph

### 3.3.3 Masking gates insertion

Once the centrality value has been computed for all the nodes of the netlist, they are sorted according to their value. The nodes with the highest centrality are selected to be modified by logic masking. The number of nodes to modify is a parameter of the logic masking algorithm and is chosen by the designer, since it is directly related to the logic resources overhead.

The masking gates of type XOR or XNOR are inserted in the same way AND and OR gates are inserted in Section 2.2.5 but taking the AW bits into account as shown in Figure 1.16. If the AW bit is a 0, an XOR gate is inserted. If the AW bit is a 1, an XNOR gate is inserted. Then the resulting graph is converted back into a netlist as described in Section 2.2.6.

### 3.3.4 Time complexity of centrality indicators

Before giving the performance of the centrality indicators at logic masking, we consider their time complexity. This is a good indicator of the scalability of these indicators to real-world netlists. Let  $n$  be the number of edges and  $m$  the number of vertices in the graph. We recall that the single-source shortest paths problem can be solved in linear time  $\mathcal{O}(m + n)$  on graphs with unit edge weights.

For betweenness centrality computation, the time complexity per node is  $\mathcal{O}(n^2)$ , since it is required to compute both the shortest paths from  $s$  to  $v$  and from  $v$  to  $t$  in order to find those that go through  $v$ . Naively, computing it for all the vertices of the graph leads to a time complexity of  $\mathcal{O}(n^3)$ . An improved betweenness centrality computation algorithm is given in [Bra01] and runs in  $\mathcal{O}(nm)$  time. For the graphs derived from netlists that we consider here, the number of edges is approximately two times larger than the number of vertices, since most of the gates that are used have two inputs. Therefore, the actual time complexity of computing the betweenness centrality is close to  $\mathcal{O}(n^2)$ .

For closeness centrality, only one instance of the single-source shortest paths problem must be solved for every vertex of the graph. Therefore, the time complexity of the closeness centrality computation is  $\mathcal{O}(n^2)$ .

Although these complexities are polynomial, they remain expensive to compute for large graphs. The authors of [EW01] showed that closeness centrality can be approximated in  $\mathcal{O}(\frac{\log n}{\epsilon^2}(n \log n + m))$  time, with an additive error of at most  $\epsilon \Delta_G$  where  $\epsilon$  is a fixed constant and  $\Delta_G$  is the diameter of the graph. This was extended to betweenness centrality in [BE05], leading to the same time complexity with an additive error of  $(n - 2)\epsilon$ .

Centrality indicators based on current-flow are more complex to compute. As shown in [BF05], the algorithms for computing current-flow betweenness centrality runs in  $\mathcal{O}(I(n - 1) + mn \log n)$  time with  $I(n) \in \mathcal{O}(n^3)$  while current-flow closeness centrality has a time complexity of  $\mathcal{O}(I(n) + n)$ . This is because computing these centrality indicators requires to invert a matrix. Matrix inversion using Gaussian elimination runs in  $\mathcal{O}(n^3)$  time. However, since the matrices we are dealing with here are sparse, specific methods can be used to invert them leading to a computation time of  $\mathcal{O}(mn^{1.5})$ . More details can be found in [BF05]. The approximated version of current-flow betweenness centrality [BF05], taking only a subset of the vertices into account, runs in  $\mathcal{O}(\frac{1}{\epsilon} m \sqrt{k} \log n)$  time with an absolute error of  $\epsilon$ . The time complexities of computing the different centrality indicators are summarised in Table 3.5.

## 3.4 Experimental results

We implemented the logic masking algorithm in Python, making use of the *igraph* package [CN06] to handle graphs. The computation times are obtained with a workstation embedding an Intel Core i5-4570 processor operating at 3.20GHz and 16GB of RAM. We used ITC'99

Centrality indicator	Time complexity
Betweenness	$\mathcal{O}(nm)$ [Bra01]
Closeness	$\mathcal{O}(n^2)$ [BE05]
Current-flow betweenness	$\mathcal{O}(I(n-1) + mn \log n)$ with $I(n) \in \mathcal{O}(n^3)$ [BF05]
Approximated current-flow betweenness with absolute error $\varepsilon$ by picking $k$ pairs	$\mathcal{O}(\frac{1}{\varepsilon^2} m \sqrt{k} \log n)$ [BF05]
Current-flow closeness	$\mathcal{O}(I(n) + n)$ with $I(n) \in \mathcal{O}(n^3)$ [BF05]

Table 3.5 – Time complexity of centrality indicators

combinational benchmarks [Dav99], but only the ones with more than 1,000 logic gates. In addition, we also considered some more recent benchmarks from EPFL [AGM15], released in 2015. Although they include benchmarks of up to 23 million gates, we restricted to the ones of up to 100 000 gates for run-time considerations.

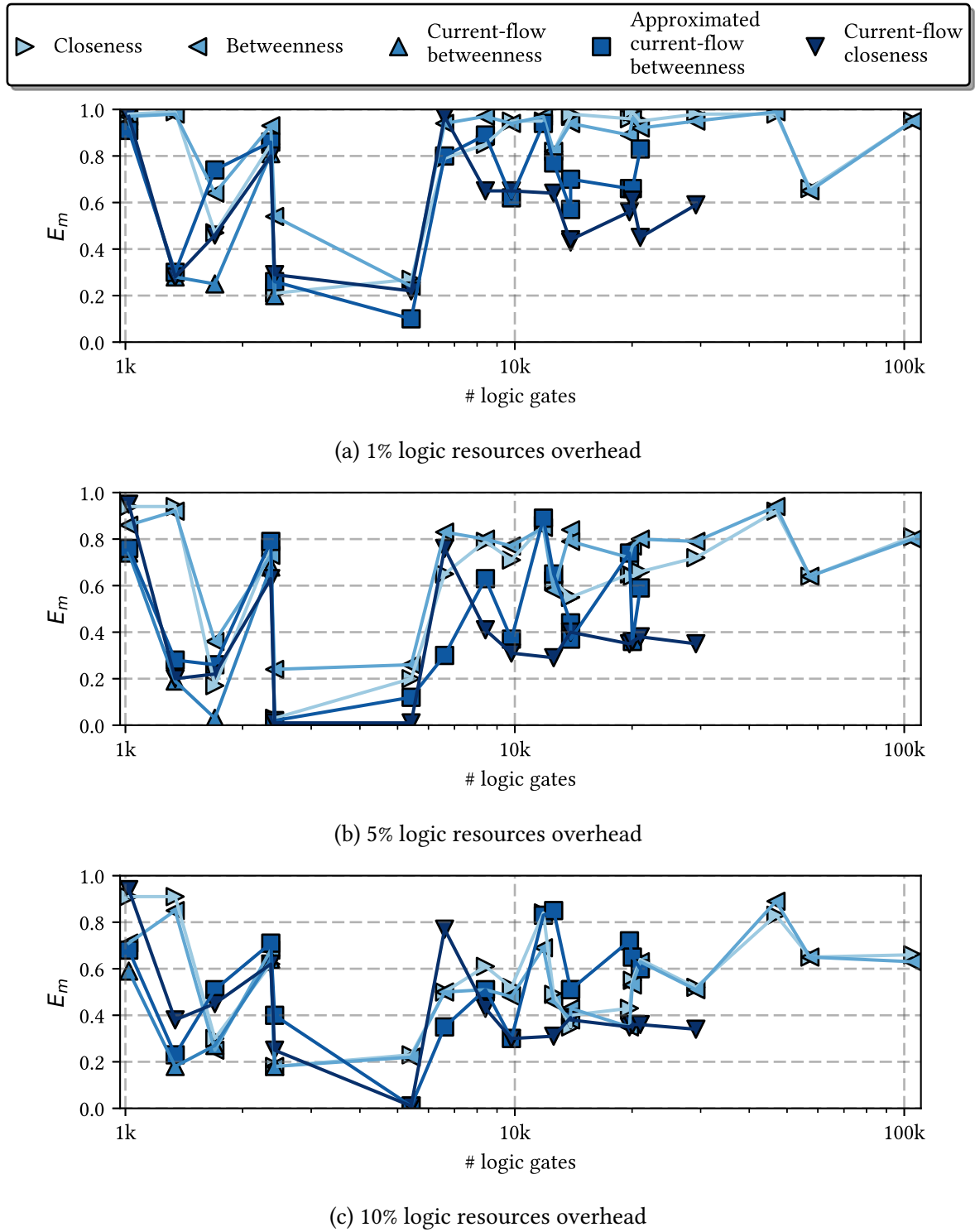
Experimental results are mostly given in the form of plots, but an exhaustive list of values for all the benchmarks is given in Table 3.7. In this table, a “—” symbol means that the centrality value could not be computed by the workstation we used.

### 3.4.1 Masking efficiency $E_m$ based on bitwise correlation

In order to estimate the masking efficiency of the different centrality indicators, we consider three logic resources overheads: 1%, 5% and 10%. For each of them, one hundred random AW were fed to the netlist, with one hundred random input patterns fed at the primary inputs for each of them. Thus ten thousand random test patterns are fed in total to each netlist.

Figure 3.8 shows a plot of the  $E_m$  values (see Equation (3.2)) obtained for the benchmarks of different sizes with the three logic resources overhead considered. Overall, increasing the logic resources decreases the  $E_m$  value in general. We can also see that the masking efficiency differs greatly from one benchmark to another. For instance, the sin benchmark, that implements the sine function, is very easy to mask. Even at 1% overhead, masking it using approximated current-flow betweenness centrality as the node selection heuristic makes the  $E_m$  value drop to 0.10 (see Table 3.7d). Conversely, the mem\_ctrl benchmark is hard to mask. Using betweenness centrality as the node selection heuristic only reduces  $E_m$  down to 0.89 at 10% overhead.

Among centrality indicators, some perform better than others. They lead to lower  $E_m$  at the same overhead. The ones that account for geodesic paths only, namely closeness and betweenness centrality, exhibit the highest  $E_m$  values on average, 0.64 and 0.71 respectively at 5% overhead. This is because, in a netlist, the information transits on non-geodesic paths as well. It follows that centrality indicators based on current flow perform much better. The average  $E_m$  values obtained for current-flow betweenness, approximate current-flow betweenness and current-flow closeness are 0.33, 0.47 and 0.38 respectively at 5% overhead. These low values indicate a good masking efficiency. For comparison, the  $E_m$  values obtained with other selection heuristics at 5% logic resources are given in Table 3.6.


 Figure 3.8 –  $E_m$  values obtained for several logic resources overhead

Heuristic	$E_m$ value at 5% logic resources overhead
Random [RKM08a]	0.74
Fan-in/out [CB09]	0.83
Fault analysis [Raj+15]	0.185

 Table 3.6 –  $E_m$  values obtained with other selection heuristics at 5% logic resources overhead

Benchmark	Outputs	# Gates	Graph building time (s)		Graph processing time (s)		
					$E_m$ at 1% overhead	$E_m$ at 5% overhead	$E_m$ at 10% overhead
adder	129	1020	0.67	0.38	0.98	0.94	0.91
i2c	142	1342	0.81	0.42	0.99	0.94	0.91
c3540	22	1669	0.28	0.51	0.47	0.17	0.3
c5315	123	2307	0.39	0.95	0.85	0.76	0.62
c6288	32	2416	0.40	0.91	0.21	0.03	0.18
sin	25	5416	3.39	6.05	0.27	0.2	0.23
b14_1_C	245	6567	1.12	6.89	0.79	0.65	0.5
b15_C	449	8367	1.48	12.12	0.85	0.79	0.61
b14_C	245	9765	1.66	16.59	0.95	0.71	0.52
b15_1_C	449	12543	2.18	23.64	0.82	0.62	0.49
arbiter	129	11839	8.36	52.56	0.95	0.86	0.84
b21_1_C	512	13898	2.44	36.60	0.97	0.55	0.35
b20_1_C	512	13899	2.42	34.40	0.98	0.55	0.4
b20_C	512	19682	3.44	71.96	0.96	0.65	0.43
b21_C	512	20027	3.49	78.30	0.99	0.64	0.55
b22_1_C	757	20983	3.77	74.21	0.95	0.66	0.64
b22_C	757	29162	5.27	170.33	0.98	0.72	0.52
mem_ctrl	1231	46836	43.84	768.46	0.98	0.92	0.83
div	128	57247	66.66	1068.01	0.66	0.64	0.65
b18_1_C	3342	105102	22.41	1653.32	0.95	0.81	0.66
<b>Average values:</b>					0.83	0.64	0.56

(a) Closeness centrality

Benchmark	Graph processing time (s)		$E_m$ at 1% overhead		$E_m$ at 5% overhead	$E_m$ at 10% overhead
adder	0.69	0.97	0.86	0.71		
i2c	0.71	0.98	0.92	0.85		
c3540	0.79	0.64	0.36	0.25		
c5315	1.34	0.93	0.73	0.68		
c6288	1.40	0.54	0.24	0.18		
sin	10.67	0.24	0.26	0.22		
b14_1_C	12.57	0.94	0.83	0.50		
b15_C	18.28	0.97	0.80	0.51		
b14_C	25.11	0.94	0.77	0.48		
b15_1_C	37.16	0.82	0.58	0.46		
arbiter	51.84	0.97	0.85	0.69		
b21_1_C	54.76	0.94	0.84	0.38		
b20_1_C	99.63	0.94	0.79	0.43		
b20_C	108.41	0.89	0.72	0.35		
b21_C	114.13	0.96	0.77	0.53		
b22_1_C	119.25	0.92	0.80	0.63		
b22_C	246.37	0.95	0.79	0.51		
mem_ctrl	1866.95	0.99	0.94	0.89		
div	2731.64	0.65	0.64	0.65		
b18_1_C	3258.28	0.95	0.80	0.63		
<b>Average values:</b>		0.86	0.71	0.53		

(b) Betweenness centrality

Table 3.7 – Experimental results obtained when applying logic masking on ITC'99 and EPFL benchmarks for different centrality indicators.



Benchmark	Graph processing time (s)	$E_m$ at 1% overhead	$E_m$ at 5% overhead	$E_m$ at 10% overhead
adder	15.57	0.95	0.74	0.59
i2c	23.51	0.28	0.19	0.18
c3540	13.43	0.25	0.03	0.27
c5315	27.15	0.81	0.68	0.64
c6288	39.31	0.20	0.03	0.18
sin	—	—	—	—
b14_1_C	—	—	—	—
b15_C	—	—	—	—
b14_C	—	—	—	—
b15_1_C	—	—	—	—
arbiter	—	—	—	—
b21_1_C	—	—	—	—
b20_1_C	—	—	—	—
b20_C	—	—	—	—
b21_C	—	—	—	—
b22_1_C	—	—	—	—
b22_C	—	—	—	—
mem_ctrl	—	—	—	—
div	—	—	—	—
b18_1_C	—	—	—	—
<b>Average values:</b>	0.50	0.33	0.37	

(c) Current-flow  
betweenness centrality

Benchmark	Graph processing time (s)	$E_m$ at 1% overhead	$E_m$ at 5% overhead	$E_m$ at 10% overhead
adder	1.67	0.91	0.76	0.68
i2c	1.65	0.30	0.28	0.23
c3540	1.38	0.74	0.26	0.51
c5315	2.43	0.86	0.79	0.71
c6288	2.31	0.26	0.02	0.40
sin	9.01	0.10	0.12	0.01
b14_1_C	7.39	0.80	0.30	0.35
b15_C	20.19	0.89	0.63	0.51
b14_C	30.32	0.62	0.37	0.30
b15_1_C	32.70	0.77	0.65	0.85
arbiter	42.89	0.94	0.89	0.83
b21_1_C	45.42	0.57	0.44	0.51
b20_1_C	50.24	0.70	0.37	0.51
b20_C	75.02	0.66	0.74	0.72
b21_C	79.26	0.66	0.36	0.65
b22_1_C	93.17	0.83	0.59	0.60
b22_C	—	—	—	—
mem_ctrl	—	—	—	—
div	—	—	—	—
b18_1_C	—	—	—	—
<b>Average values:</b>	0.66	0.47	0.52	

(d) Approximated current-flow  
betweenness centrality

Benchmark	Graph processing time (s)	$E_m$ at 1% overhead	$E_m$ at 5% overhead	$E_m$ at 10% overhead
adder	21.84	0.97	0.95	0.94
i2c	20.71	0.28	0.20	0.38
c3540	11.38	0.46	0.22	0.45
c5315	21.67	0.80	0.63	0.62
c6288	22.75	0.29	0.01	0.25
sin	334.16	0.22	0.01	0.01
b14_1_C	166.29	0.97	0.76	0.77
b15_C	337.45	0.65	0.41	0.43
b14_C	444.58	0.65	0.31	0.30
b15_1_C	781.16	0.64	0.29	0.31
arbiter	—	—	—	—
b21_1_C	957.94	0.43	0.40	0.39
b20_1_C	949.88	0.44	0.40	0.38
b20_C	1702.18	0.56	0.35	0.35
b21_C	1780.25	0.61	0.36	0.36
b22_1_C	2056.71	0.45	0.38	0.36
b22_C	4068.99	0.59	0.35	0.34
mem_ctrl	—	—	—	—
div	—	—	—	—
b18_1_C	—	—	—	—
<b>Average values:</b>	0.56	0.38	0.42	

(e) Current-flow  
closeness centrality

Table 3.7 – Experimental results obtained when applying logic masking on ITC'99 and EPFL benchmarks for different centrality indicators.

### 3.4.2 Computation time

The second metric that we used to evaluate the proposed node selection heuristic based on centrality indicators is the computation time. Indeed, this criterion is essential for a smooth integration into EDA tools. Figure 3.9 shows a plot of the computation time required for each benchmark, as well as a baseline that accounts for the time taken to build the graph from the netlist file. The outliers on this baseline, that appear as small peaks, are the EPFL benchmarks. Indeed, they are provided in the BLIF description format, which is more time-consuming to parse than the BENCH format of ITC-99 benchmarks. Detailed computation time values for each benchmark and centrality indicator considered are given in Table 3.7, in the “Graph processing time (s)” column.

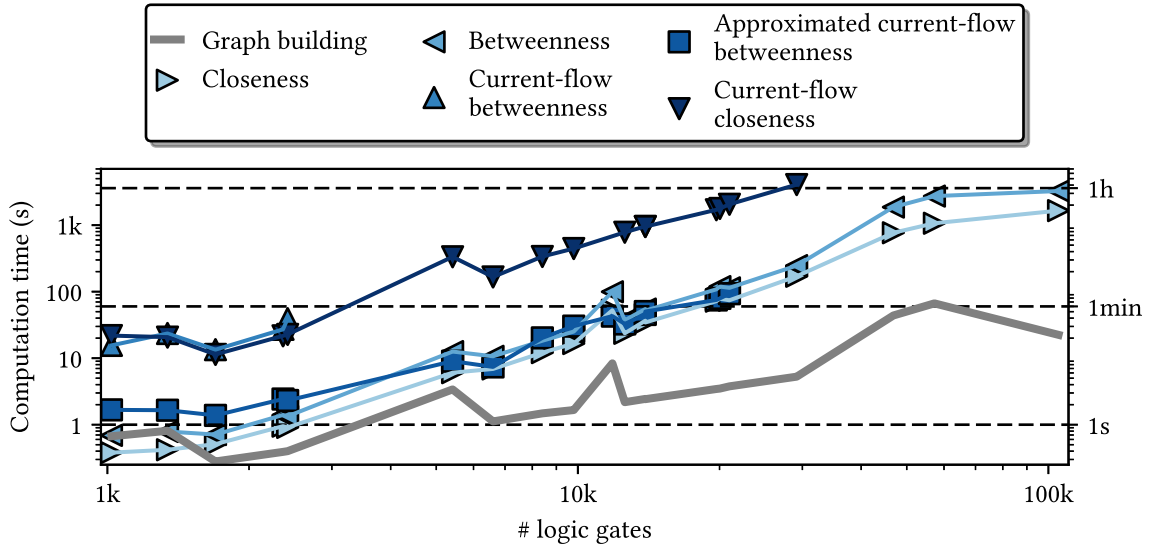


Figure 3.9 – Computation time required for the centrality indicators considered for different benchmark sizes.

The plots shown in Figure 3.9 are coherent with the quadratic time complexities described in Section 3.3.4. Closeness and betweenness are quite efficient to compute, allowing large netlists of up to 100 000 gates to be processed. The first centrality indicator to become impractical to compute with our workstation is current-flow betweenness. However, the approximated version can be used to handle larger designs. Current-flow closeness is almost equivalent in computation time to current-flow betweenness centrality, but can be used for netlists of up to 30 000 gates.

#### 3.4.2.1 Parallel computation

In order to speed-up the centrality computations, parallel algorithms can be used. For example, in [BM06], parallel approaches for betweenness and closeness centrality are described. Implementing these methods would allow to speed up the computations. However, this might not

allow to handle larger netlists, due to the space complexity requirements. This aspect should be further evaluated.

### 3.4.3 Trade-off between masking efficiency and computation time

To allow for a better comparison between the existing node selection heuristics and the ones that use centrality indicators, it is interesting to plot the computation time ratio against the average  $E_m$  value for each. The computation time ratio is defined as the time taken to compute the heuristic of interest divided by the time to perform random selection. The result is shown in Figure 3.10. It is important to consider that the  $E_m$  value (see Equation 3.2) obtained for the node selection heuristic based on fault-analysis is only averaged on benchmarks of up to 3 500 gates, after the results provided by the authors of [Raj+15]. This limitation for the size of the considered benchmarks could potentially lead to an underestimation of  $E_m$ .

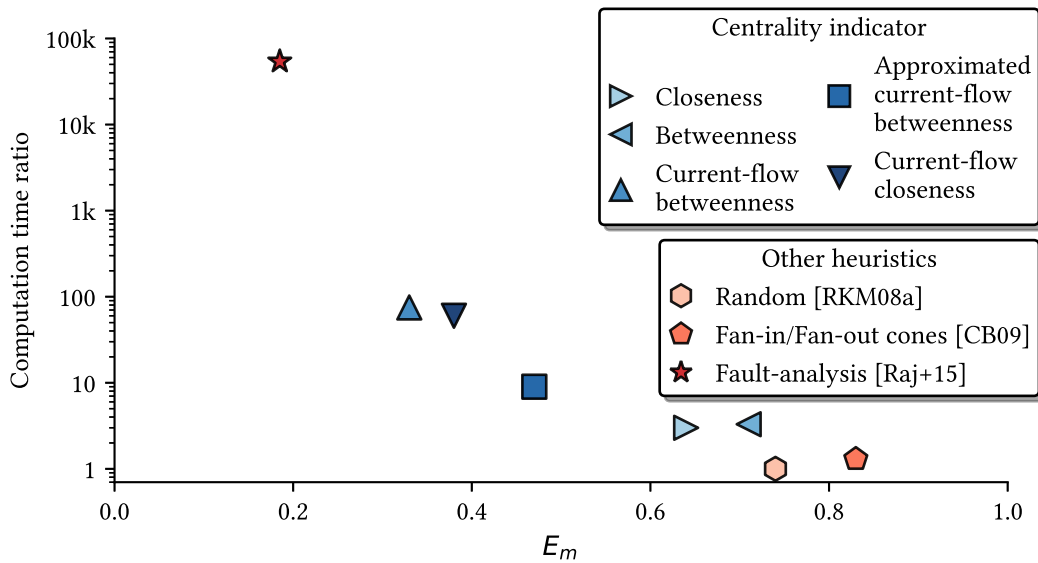


Figure 3.10 – Trade-off between masking efficiency and computation time for different node selection heuristics at 5% logic resources overhead.

This plot clearly shows that existing heuristics are either easy to compute or efficient at masking. Conversely, using centrality indicators allows for a nice trade-off between those two criteria. Even though current-flow betweenness centrality seems to be the best performing heuristic, the results presented in Table 3.7 show that it can not be used for large netlists. Current-flow closeness centrality exhibits a similar masking efficiency and computational complexity, while being able to handle larger netlists. Therefore, among centrality indicators, current-flow closeness centrality is the most usable one for efficient logic masking.

### 3.4.4 Distance to inputs/outputs

Finally, as discussed in Section 3.2.3, the masking gates must be inserted as deep as possible in the netlist to avoid bitwise dependencies between the AW bits and the outputs. Table 3.8 shows the average distance from the masking gates to the inputs/outputs of the netlist. 0% means that the masking gates are inserted at the inputs, 100% means that the masking gates are inserted at the outputs and 50% means that the masking gates are inserted as far from the inputs as from the outputs.

Centrality indicator	Average distance from the masking gates to the inputs/outputs
Betweenness	56%
Closeness	57%
Current-flow betweenness	59%
Approximated current-flow betweenness	53%
Current-flow closeness	54%

Table 3.8 – Distance from the inserted logic masking gates to the inputs/outputs when using different centrality indicators. 0% means that the masking gates are inserted at the inputs, 100% means that the masking gates are inserted at the outputs and 50% means that the masking gates are inserted as far from the inputs as from the outputs.

These results indicate that the inserted masking gates are approximately as far from the inputs as from the outputs. Therefore, they are in the middle of the netlist and can affect multiple output bits. A more strict evaluation of the impact of each masking gate could be developed by exploiting the avalanche criterion. A good masking scheme should then get half the output bits to flip on average when the AW bits are flipped consecutively.

## 3.5 Possible improvements

### 3.5.1 Deleting selected nodes from the graph

For some graphs, selecting the vertices with the highest centrality for logic masking does not alter the outputs as much as it could if the selection process was carried out differently. An example of such a graph is shown in Figure 3.11.

In this example graph, we can observe that the three vertices with the highest centrality are adjacent. Therefore, two problems arise when selecting them for logic masking. First of all, since the masking gates are inserted in a row, their efficiency will be reduced. Indeed, if two masking gates are inserted one after the other, then both AW bits combinations “00” and “11” make the design operate normally. This increases the number of valid AWs. The second concern is that if there are some outputs outside the output logic cone of the masking gates, then they are not affected by logic masking. Therefore, the masking efficiency is reduced.

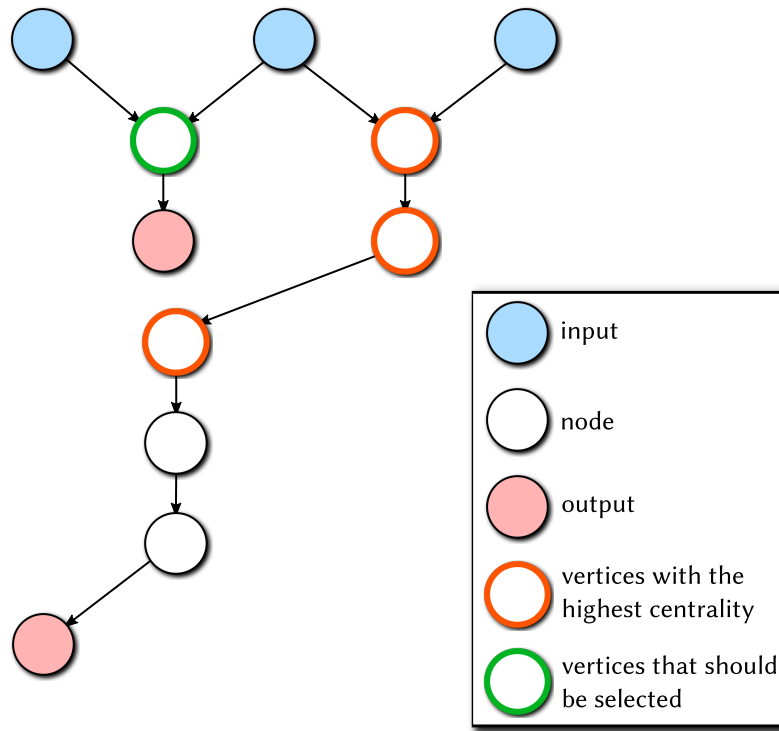


Figure 3.11 – Graph for which selecting the vertices with the highest centrality does not alter the outputs optimally

In order to avoid this phenomenon, a modification could be applied to the node selection process. The vertices that have been selected because they have the highest centrality could be removed from the graph. This way, they do not participate anymore in the measurement of path lengths or current-flow that are used by centrality indicators. Thus this allows other vertices, that are far from the selected ones, to be reconsidered by recomputing the centrality indicator.

In [Raj+15], the fault-impact, used as the selection heuristic, is recomputed every time a node is selected for logic masking. Instead, in order to reduce processing time, a larger number of nodes could be selected every time the heuristic is computed. Fine tuning this number should be done for each heuristic, after considering the computing power and time available.

### 3.5.2 Vitality indicators

Following the idea of removing high-importance vertices from the graph, vitality indicators could be considered in the development of future node selection heuristics. As defined in [BE05, p. 36], for a graph  $G$ : “Given an arbitrary real-valued function on  $G$  a vitality measure quantifies the difference between the value on  $G$  with and without the vertex or edge”. By defining the mentioned real-valued function as a measure of the correct operation of the netlist, the vitality measure allows to target specific nodes that alter the operation as much as possible. This could be investigated in future works.

### 3.5.3 Very-low overhead logic masking

Another interesting criterion that could also be exploited to evaluate the nodes selection heuristics is their masking efficiency at low overhead. Indeed, as shown by the results in Table 3.7, increasing the logic resources overhead from 5 to 10% does not necessarily lead to better logic masking (*i.e.* lower  $E_m$  value). At low overhead of 1%, current-flow betweenness centrality already allows to reach  $E_m = 0.50$  on average, indicating quite efficient masking. By considering how fast the masking efficiency varies when the logic resources overhead increases, the designer's choice about the affordable overhead for efficient masking could be better guided.

## 3.6 A priori evaluation of the masking potential

As illustrated by the plots in Figure 3.8, the masking efficiency varies a lot from one benchmark to another. The benchmarks for which the  $E_m$  value drops the fastest when the logic resources overhead increases are the multiplier (c6288) and the sine benchmarks. Intuitively, this can be explained by the fact that the output can take a lot of different values. The output space is very large. Moreover, when one output changes, it is very likely that the others change as well. Conversely, for the arbiter for example, the outputs can take much less different values. Since the aim of such IP core is to grant access to peripherals, it can only take  $m$  output values if it has  $m$  outputs, since it cannot grant access to two peripherals at the same time. In addition, if one output changes, only one other output changes.

Moreover, the multiplier and sine benchmarks have the property that changing one input bit changes the output bits a lot. This property is related to the avalanche criterion used to assess the *diffusion* property of ciphers. This criterion states that when one input bit flips, half the output bits should flip on average.

These properties of some benchmarks should be formalised in order to evaluate *a priori* how well a benchmark can be masked. The two main paths that could be investigated are the following. First, the avalanche criterion could be evaluated on the nodes of the benchmark. This would allow to evaluate the efficiency at propagating the disturbance from the nodes of interest to the outputs. The other option is, for every output, to evaluate how many outputs change when the output of interest changes. This could highlight the relation between the outputs. These methods, allowing to assess the masking potential of a benchmark *a priori*, could be very helpful to IP core designers.

## 3.7 Attacks aiming at recovering the activation word

The logic masking schemes have been subject to a variety of attacks aiming at recovering the AW from a masked IP core. If the attacker has access to the gate-level netlist, he can determine paths inside the netlist that can sensitise the AW bits to the outputs. This is called

the sensitisation attack [Raj+12b]. However, it requires the attacker to have full access to the design file of the gate-level netlist, which is quite a restrictive constraint.

Later on, attacks that do not require access to a gate-level netlist were proposed. In [PM15], a hill-climbing attack leverages the bitwise dependency between AW bits and output bits. This is detailed in Section 2.4.4.1.

The state-of-the-art attack on logic masking schemes is the SAT attack [SRM15]. The principle of this attack is given in Section 2.4.4.2.

To thwart this attack, various additions to the masking gates were proposed. The first observation is that the inputs associated to the AW bits should not be exposed directly, but a one-way random function could be inserted before them. To this end, [Yas+15] proposed to use an AES block cipher with a fixed secret key, since it performs as a pseudo-random function.

To reduce the logic resources overhead, the AES core can be replaced by several structures that are known to be hard to handle by a SAT solver. These structures tend toward a point function behaviour [Yas+16a; XS16; Yas+17c], and alter the outputs only for a few number of input patterns. An example of such structure is an AND tree, which can be detected inside a netlist and exploited to harden the logic masking scheme [Li+16].

However, as pointed out in [Yas+16a], there is a dichotomy between SAT resistance and corruptibility. Indeed, the SAT attack is very efficient because it exploits the fact that the masked outputs are altered a lot. By reducing the Hamming distance between the normal and masked outputs, the attack becomes harder. However, the masking efficiency drops considerably in this case. The extreme case is TTLock [Yas+17c], in which the outputs are altered for only one input pattern. In such case, we do not believe that the logic modification can be labelled “masking” anymore, considering its extremely poor efficiency at disturbing the outputs.

Several attacks have also been published against anti-SAT blocks. The signal skew towards 0 or 1 can help in identifying functions that tend to behave like point functions [Yas+17b]. These functions can then be removed from the netlist [Yas+17a] so that it operates normally. Of course, these attacks imply that an attacker has access to the netlist.

Finally, it is our feeling that security should not be the primary concern of a logic masking scheme, as highlighted by this whole chain of attack-defense articles. We believe that security can only be guaranteed by a cryptographic core. Making a cryptographic core secure is already a complex, challenging task. Trying to obtain security in a cryptographic sense from a few masking gates inserted inside an IP core with its own functional purpose seems impossible.

## 3.8 Conclusion

This chapter proposes a new set of heuristics based on centrality indicators to select the nodes to modify by logic masking. We first reviewed existing centrality indicators before highlighting which ones perform the best in the frame of logic masking. When compared to existing selection heuristics, it offers a nice trade-off between masking efficiency and computational complexity.

Thus, heuristics based on centrality indicators, particularly current-flow closeness, are the only ones to date that can mask large netlists efficiently. This makes them suitable candidates for integration into EDA tools.





# Chapter 4

## Key reconciliation protocols for error correction of silicon PUF responses

---

PUFs, presented in Section 1.5.2.3, are now a widely known root of trust and bring features such as hardware identification, authentication and key generation to electronic systems. Their main drawback, however, is that the response that is generated by querying the PUF with a fixed challenge varies from time to time. This is due to the intrinsic properties of the PUF, that extracts manufacturing process variations. In order to obtain a reliable response, an error correction module must then be integrated as well.

Correction is currently performed by a classical decoder, BCH, Reed-Muller or convolutional for instance. The first time the PUF is challenged, helper data is generated. This helper data, which should leak a limited amount of information about the PUF response, is later used by the decoder to regenerate the original response if the same challenge is fed. Some encoding methods were proposed as well, to take into account the specific properties of PUF responses. All these methods, however, require a significant amount of logic resources.

In this chapter, we show that the CASCADE key reconciliation protocol, originating from quantum key distribution, can be successfully used to reconcile two slightly different PUF responses obtained at different times. We give several sets of parameters for the protocol that can be used depending on the error rate observed at the PUF output. The amount of information leaked when executing the protocol is manageable and is evaluated for several use cases. Finally, implementation results on the device side show that this is the most lightweight solution for error correction, with at least a three times improvement in logic resources occupation at least over state-of-the-art error correction codes.

---

The code associated with this chapter is available at:  
<https://gitlab.univ-st-etienne.fr/b.colombier/cascade/tree/master>

# Protocoles de réconciliation de clés pour la correction des erreurs dans les réponses des PUFs

---

Les PUFs sont aujourd'hui des primitives matérielles bien connues et permettent l'identification matérielle, l'authentification ou encore la génération de clés. Leur inconvénient principal, néanmoins, est le fait que la réponse générée en envoyant un challenge fixe à la PUF change d'une fois à l'autre. Ceci est dû aux propriétés intrinsèques de la PUF, qui extrait les variations de processus de fabrication. Afin d'obtenir une réponse fiable, un module de correction des erreurs doit donc être ajouté également.

Actuellement, ceci est réalisé en implantant un décodeur classique, de type BCH, Reed-Muller ou convolutif par exemple. Lorsqu'un challenge est envoyé à la PUF pour la première fois, des données auxiliaires sont générées. Ces dernières, qui doivent fuiter le moins d'information possible sur la réponse de la PUF, sont utilisées plus tard par le décodeur pour régénérer la réponse originale si le même challenge est envoyé. Des méthodes d'encodage ont également été proposées, qui prennent en compte les propriétés spécifiques des réponses des PUFs. Toutes ces méthodes, néanmoins, ont un coût important en ressources logiques.

Dans ce chapitre, nous montrons que le protocole de réconciliation de clés CASCADE, utilisé en distribution quantique de clés, peut être utilisé pour réconcilier deux réponses de PUF légèrement différentes obtenues à deux moments distincts. Nous donnons plusieurs jeux de paramètres pour le protocole qui peuvent être utilisés en fonction du taux d'erreur observé à la sortie de la PUF. La quantité d'information fuitée pendant l'exécution du protocole est gérable et évaluée pour différents cas d'usage. Finalement, les résultats d'implémentation côté composant virtuel montrent que c'est la solution la plus légère à ce jour pour la correction des erreurs, avec un coût en ressources logiques au moins trois fois moindre par rapport aux codes correcteurs d'erreurs les plus adaptés.

## 4.1 Similarities between key reconciliation in quantum key distribution and reliable shared key generation from a PUF response

Originally proposed in the context of quantum key distribution, key reconciliation protocols allow two parties who exchanged a stream of bits through a quantum channel to reconcile their respective information [BS93]. Indeed, because the quantum channel is noisy and can be eavesdropped, the message that is received is slightly different from the one that was sent. In order to make these messages identical, the two parties involved carry out a key reconciliation protocol. This key reconciliation consists in a public discussion. Obviously, since the discussion is public, some information is leaked in the process. Depending on the actual amount of information that is leaked, an appropriate privacy amplification method is applied to obtain a shared secret with a sufficient amount of entropy per bit. The protocol is shown in Figure 4.1a.

This use case is very similar to the one of shared key generation between a circuit embedding a PUF and a server. At enrolment, the circuit generates  $r_0$  and sends it to the server. Thus both the circuit and the server own  $r_0$ . However, later on, when the circuit must be identified, the response  $r_t$  generated by the PUF is noisy. Error correction is carried out on the server side, like in [Her+12], so that the server owns  $r_t$  as well. The PUF response is then turned into a cryptographic key. This is illustrated in Figure 4.1b.

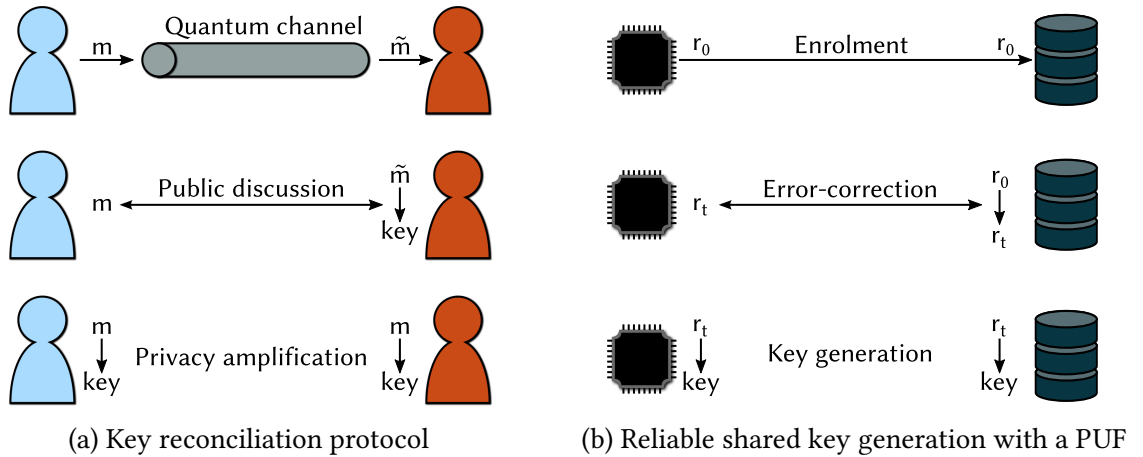


Figure 4.1 – Illustration of the similarities between key reconciliation and reliable shared key generation from a PUF response

In order to understand how the CASCADE key reconciliation protocol can be applied to correct the errors in PUF responses, we first present the foundations of the protocol, namely parity checks and binary search. We then show how they are extended to make the full CASCADE protocol.

## 4.2 Error correction based on multiple parity checks and binary searches

### 4.2.1 Method

Given two responses  $r_0$  and  $r_t$  of length  $n$ , identifying, isolating and correcting errors between them can be done by multiple parity checks followed by binary searches. We consider that  $n$  is a power of two in the rest of the chapter. First, both strings are split into blocks of size  $m$ , which is a power of two as well. A block is a list of indexes, like [12, 13, 14, 15] for example, that are the indexes of the bits of interest in the PUF response. From the parity of both associated blocks  $B_0$  and  $B_t$  from  $r_0$  and  $r_t$ , the relative parity,  $P_r$ , is computed (see Equation (4.1)).

$$P_r(B_0, B_t) = \underbrace{\left( \bigoplus_{i=0}^{m-1} r_0[B_0[i]] \right)}_{\text{Parity of } B_0} \oplus \underbrace{\left( \bigoplus_{i=0}^{m-1} r_t[B_t[i]] \right)}_{\text{Parity of } B_t} \quad (4.1)$$

If the relative parity is even, then no error is detected. If the relative parity is odd, then the CONFIRM method [BS93] is applied on both blocks  $B_0$  and  $B_t$  from  $r_0$  and  $r_t$ . This method consists in splitting the blocks in two and computing the relative parity of the first half. If it is even, then the error is in the second half. If it is odd, then the error is in the first half. The half for which the parities differ is then subsequently split in two. The process is repeated until the block size is two bits. The first bit from  $B_t$  is then transmitted. If this bit is the same as the corresponding bit in  $B_0$  then the other bit is flipped. If this bit is different from the corresponding bit in  $B_0$ , then this bit is flipped. Algorithm 2 summarises the CONFIRM method, while Figure 4.2 illustrates it on 16-bit blocks.

---

**Algorithm 2: CONFIRM**


---

**Input:**  $B_0, B_t$

```

1 while  $\text{size}(B_0) > 1$  do
2   Split  $B_0$  into two parts  $B_{0,0}$  and  $B_{0,1}$ 
3   Split  $B_t$  into two parts  $B_{t,0}$  and  $B_{t,1}$ 
4   if  $P_r(B_{0,0}, B_{t,0}) = 1$  then
5      $B_0 = B_{0,0}$ 
6      $B_t = B_{t,0}$ 
7   else
8      $B_0 = B_{0,1}$ 
9      $B_t = B_{t,1}$ 
10 return  $B_0$ 

```

---

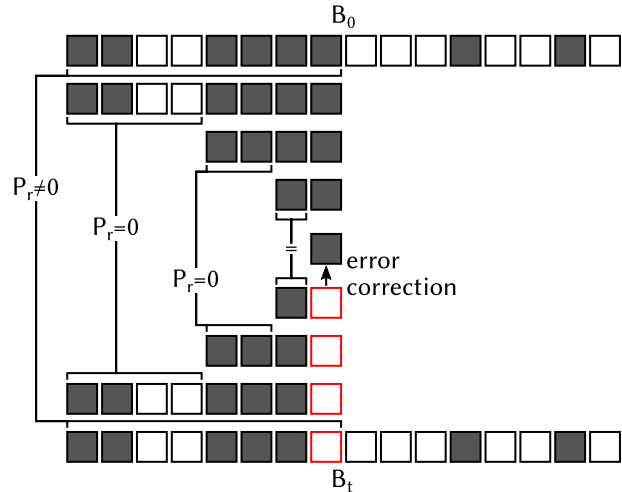


Figure 4.2 – CONFIRM applied on 16-bit blocks

### 4.2.2 Failure rate

The failure rate of the CONFIRM method depends on the location of the faulty bits in the PUF response. The failure rate is defined as the ratio of responses in which some errors are left uncorrected. If two faulty bits end up in the same block, then they are not detected by the parity check and cannot be corrected. To maximise the probability to isolate faulty bits, the block size must be reduced. Therefore, the *smaller* the block size, the *lower* the failure rate is.

### 4.2.3 Associated leakage

#### 4.2.3.1 Initial parity checks

Every time the parity is computed on a block, one bit of information is leaked. Therefore, when an  $n$ -bit response is split into blocks of size  $m$ , performing parity checks on every block leaks  $n/m$  bits, which is the number of blocks. Therefore, the *smaller* the block size is, the *higher* the information leakage associated to the initial parity checks is.

#### 4.2.3.2 Error isolation and correction

When a block exhibits a different parity in  $r_0$  and  $r_t$ , the CONFIRM method is applied on it. Since the blocks are of size  $m$ , which is a power of two, then successively splitting in two and computing the parity of the first half leaks  $\log_2(m)$  bits. Therefore, the *smaller* the block size is, the *lower* the information leakage associated with binary search and error correction is.

### 4.2.4 Drawback

The drawback of this method for error correction is that if two errors are found in the same block, then they are undetected. This is solved in the BINARY protocol.

## 4.3 BINARY protocol

The BINARY protocol improves on CONFIRM by repeating it multiple times. Moreover, responses are shuffled randomly between two passes, spreading the errors across and preventing two originally adjacent errors to always end up in the same block for parity checks.

### 4.3.1 Method

Given two responses  $r_0$  and  $r_t$ , the BINARY protocol starts by shuffling them identically using a public random permutation  $\sigma_0$ . Indeed, in a quantum channel, errors usually occur in burst. Therefore, these errors must be spread among the blocks so that they are detected by the parity checks and corrected (see Figure 4.3). When using the protocol with PUF responses, however, this initial shuffling step can be omitted since the errors do not occur in burst.

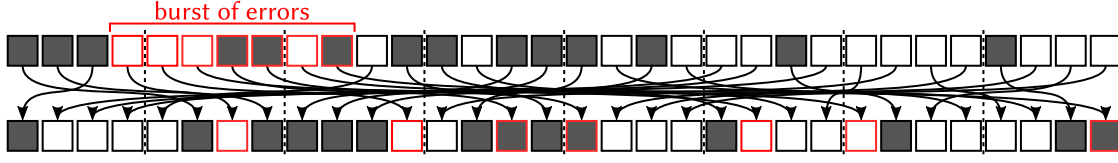


Figure 4.3 – Spreading a burst of errors among multiple blocks

The initial block size is determined from the error rate  $\varepsilon$ . In the original protocol,  $\varepsilon$  is estimated by transmitting a dummy public frame through the quantum channel. In the case of PUF responses, the error rate can be estimated by characterisation of the PUF of interest.

For every pass, the parity checks and binary search-based error correction is done. This ends by applying the CONFIRM method on the blocks that have a relative parity of 1 (lines 7 and 8 of Algorithm 3). After this, the block size is doubled to reduce the leakage brought by the parity checks in subsequent passes. Although doubling the block size increases the probability to find an even number of errors in a block, most of the errors are corrected in the first passes since the blocks are then smaller. Therefore, a small block size is no longer necessary. Afterwards, the responses are scrambled again with another public random permutation.

After all the passes have been carried out, the responses must be unscrambled by using the inverse permutations  $\sigma_0^{-1}, \sigma_1^{-1}, \dots, \sigma_{n_{passes}}^{-1}$ . If the number of passes is sufficiently high, then the responses  $r_0$  and  $r_t$  are correctly reconciled with a very high probability.

A toy example of applying the BINARY algorithm on 16-bit PUF responses is shown in Figure 4.4. In this example, an integrated circuit that embeds a PUF tries to authenticate to a server. To achieve this, one step is to have a shared secret. The communication goes both ways. The server sends the response indexes contained in the block on which the parity must be computed. The circuit then sends back the associated parity value.

---

**Algorithm 3: BINARY**


---

**Input:**  $r_0, r_t, \varepsilon, n_{passes}$

---

- 1 Scramble  $r_0$  and  $r_t$  using a public random permutation  $\sigma_0$
  - 2 Estimate the initial block size  $k_1$  from the error rate  $\varepsilon$
  - 3 **for**  $i = 1$  **to**  $n_{passes}$  **do**
  - 4     Split  $r_0$  and  $r_t$  into blocks of size  $k_i$
  - 5     **forall** *blocks* **do**
  - 6         Compute the relative parity  $P_r(B_{0,i}, B_{t,i})$
  - 7         **if**  $P_r(B_{0,i}, B_{t,i}) = 1$  **then**
  - 8             CONFIRM( $B_{0,i}, B_{t,i}$ )
  - 9     Double the block size  $k_{i+1} = 2 \times k_i$
  - 10    Scramble  $r_0$  and  $r_t$  using a public random permutation  $\sigma_i$
  - 11 Unscramble  $r_0$  and  $r_t$  with  $\sigma_0^{-1}, \sigma_1^{-1}, \dots, \sigma_{n_{passes}}^{-1}$
  - 12 **return**  $r_0, r_t$
-

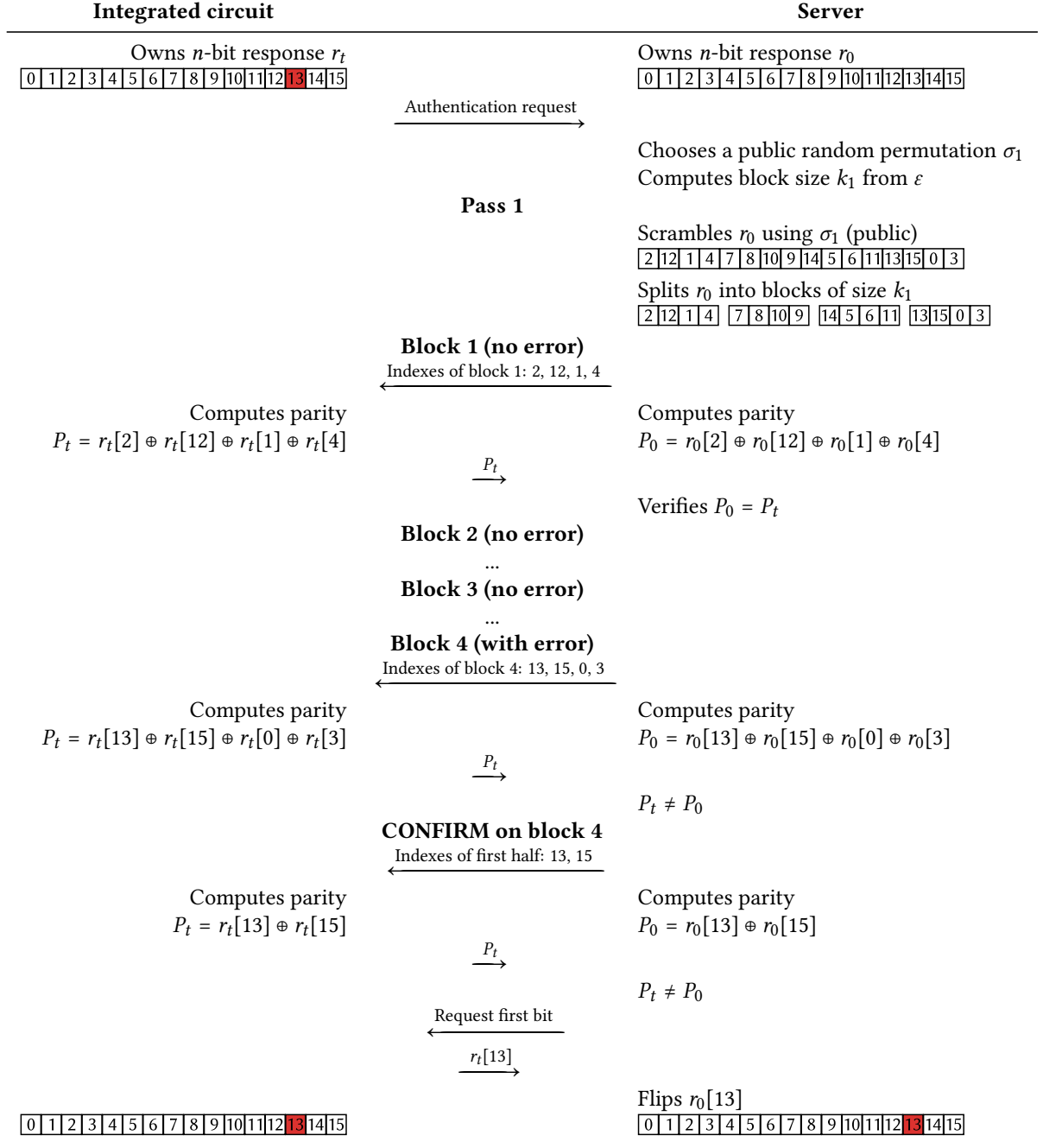


Figure 4.4 – Example of executing the BINARY protocol on 16-bit responses with one error.

### 4.3.2 Failure rate

#### 4.3.2.1 Influence of the block size

Starting with a small initial block size  $k_1$  decreases the failure rate. Indeed, the probability to isolate one error per block is higher. Therefore, the *smaller* the initial block size is, the *lower* the failure rate is.



### 4.3.2.2 Influence of the number of passes

Increasing the number of passes also reduces the failure rate. By performing more parity checks, more errors can be detected and corrected. Therefore, the *higher* the number of passes is, the *lower* the failure rate is.

## 4.3.3 Associated leakage

### 4.3.3.1 Influence of the block size

**Initial parity checks** Just as discussed before, computing the parity of  $m$ -bit blocks in an  $n$ -bit response leaks  $n/m$  bits. Therefore, the *smaller* the block size is, the *higher* the information leakage associated to the initial parity checks is.

**Error isolation and correction** When an error is detected by parity check, performing binary search on an  $m$ -bit block leaks  $\log_2(m)$  bits. Therefore, the *smaller* the block size is, the *lower* the information leakage associated with binary search and error correction is.

### 4.3.3.2 Influence of the number of passes

If more passes are carried out, more parity checks are performed. Even though final passes leak less, since the block size is greater, some bits are still leaked. Therefore, the *higher* the number of passes is, the *higher* the leakage is.

## 4.3.4 Improvement

The BINARY protocol can be improved by noticing the following. If, in a pass, two blocks have a even relative parity, then if in a subsequent pass an error is corrected at an index that was in these blocks, then the blocks now have an odd relative parity. Thus these blocks can be processed by CONFIRM again to isolate the error and correct it.

## 4.4 CASCADE protocol

### 4.4.1 Method

The CASCADE protocol consists in adding a backtracking step at the end of each pass of the protocol. After each pass, since all detected errors have been corrected, all the blocks have an even relative parity. Therefore, if an error is detected and corrected at index  $i$  in a pass, then all the blocks from previous passes that contain index  $i$  are now of odd relative parity. Therefore, they contain an error that can be located and corrected using CONFIRM.

The extra requirement compared to BINARY is to have two lists holding the blocks depending on their relative parity:  $L_{even}$  and  $L_{odd}$ . The backtracking step starts by applying CONFIRM

on the smallest block of  $L_{odd}$ , minimising the associated leakage. This corrects an error at position  $j$ . All the blocks from  $L_{even}$  and  $L_{odd}$  that contain  $j$  are now moved from one list to the other. This process is repeated until  $L_{odd}$  is empty, meaning that no more erroneous blocks are known. Another pass can then start. Overall, since it corrects more errors than BINARY for the same number of passes, the CASCADE protocol is more efficient. The CASCADE protocol is detailed in Algorithm 4.

---

**Algorithm 4: CASCADE**


---

**Input:**  $r_0, r_t, \varepsilon, n_{passes}$

---

```

1 Scramble  $r_0$  and  $r_t$  using a public permutation  $\sigma_0$ 
2 Estimate the initial block size  $k_1$  from the error rate  $\varepsilon$ 
3 Create two list of blocks of even and odd relative parity:  $L_{even}$  and  $L_{odd}$ 
4 for  $i = 1$  to  $n_{passes}$  do
5     Split  $r_0$  and  $r_t$  into blocks of size  $k_i$ 
6     forall  $blocks$  do
7         Compute the relative parity  $P_r(B_{0,i}, B_{t,i})$ 
8         if  $P_r(B_{0,i}, B_{t,i}) = 1$  then
9             CONFIRM( $B_{0,i}, B_{t,i}$ ): correct an error at index  $j$ 
10        Move all blocks containing  $j$  from  $L_{even}$  to  $L_{odd}$  or from  $L_{odd}$  to  $L_{even}$ 
11    Add all blocks to  $L_{even}$ 
12    while  $L_{odd}$  is not empty do
13        // Backtracking step
14        Find the smallest block  $B$  from  $L_{odd}$ 
15        CONFIRM( $B_0, B_t$ ): correct an error at index  $j$ 
16        Move all blocks containing  $j$  from  $L_{even}$  to  $L_{odd}$  or from  $L_{odd}$  to  $L_{even}$ 
17    Double the block size  $k_{i+1} = 2 \times k_i$ 
18    Scramble  $r_0$  and  $r_t$  using a public random permutation  $\sigma_i$ 
19 Unscramble  $r_0$  and  $r_t$  with  $\sigma_0^{-1}, \sigma_1^{-1}, \dots, \sigma_{n_{passes}}^{-1}$ 
20 return  $r_0, r_t$ 

```

---

A toy example of running the CASCADE protocol on a 16-bit response with five errors is shown in Figure 4.5. Only the extra features found in the CASCADE protocol compared to BINARY are shown. For example, the indexes and parities exchanges between the server and the device are hidden. The backtracking step, on the other hand, is detailed.

## 4.5 Parameters of the CASCADE protocol

Computing the exact number of bits leaked during an execution of the CASCADE protocol remains an open question [SNK13; Mar+15]. However, the leakage can still be analysed by considering its lower and upper bounds.

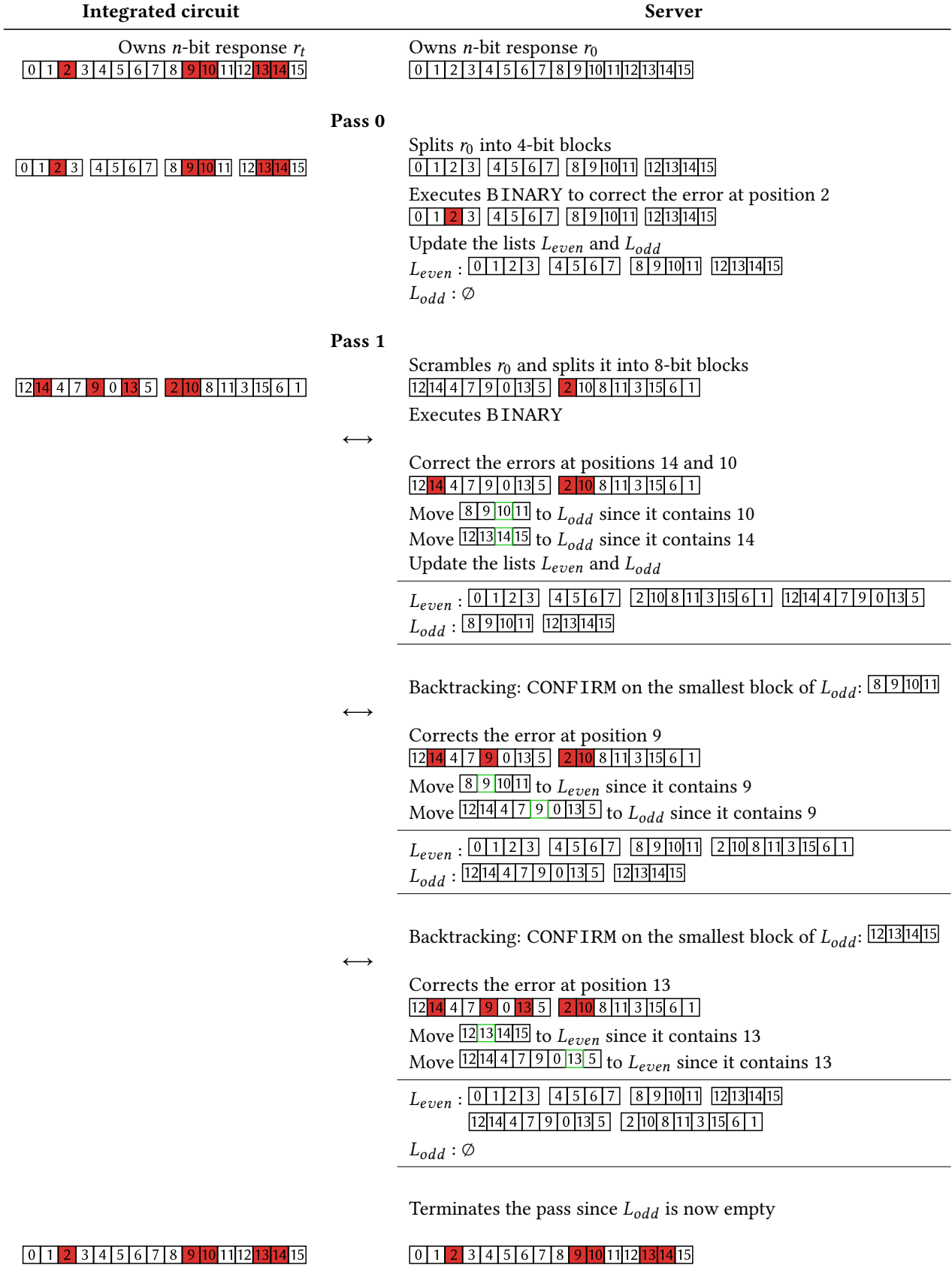


Figure 4.5 – Example of executing the CASCADE protocol on 16-bit responses with five errors.

### 4.5.1 Upper and lower bound on the information leakage

The information needed to recover a variable  $X$  from a noisy version  $Y$  is given by the conditional entropy  $H(X|Y)$ , as highlighted in [Mar+15]. The conditional entropy is related to the error rate  $\varepsilon$ . The minimum amount of information that must be exchanged between the two parties to reconcile their respective responses is given in Equation (4.2), where  $n$  is the size of the response and  $h(\varepsilon)$  is the Shannon entropy.

$$nh(\varepsilon) = n(-\varepsilon \log_2(\varepsilon) - (1 - \varepsilon) \log_2(1 - \varepsilon)) \quad (4.2)$$

This then gives a lower bound on the leakage value. Because information is leaked, the maximum number of PUF bits that can be expected to remain secret is given in Equation (4.3)

$$n - nh(\varepsilon) = n(1 - h(\varepsilon)) \quad (4.3)$$

For instance, if the error rate is 5%, one cannot expect to keep secret more than 182 bits from an initial 256-bit response. Of course, if the error rate is lower, 1% for example, then up to 235 bits can be kept secret. In practise, since there is no exact literal formula for the leakage, one can find a higher bound on the leakage value by considering that one bit is leaked every time one parity value is sent over the channel. This is an overestimation of the leakage and tighter bounds can be found in literature [Ng14]. In order to limit the leakage, the CASCADE protocol parameters must be carefully chosen. This is presented in the next section.

### 4.5.2 Choice of parameters

There are three parameters for the CASCADE protocol. The first one is the initial block size and the second one is the number of passes. The third one, not present in the original article, is the multiplication factor for the block size between two successive passes. These parameters are not set in stone but can be changed on the field when the protocol starts. This could be useful to adapt to a higher error rate if the operating conditions of the PUF have changed.

#### 4.5.2.1 Initial block size

The initial block size should be set so that, after the initial scrambling step, there is one error per block on average. This would make the error detectable by the initial parity checks. Therefore, the initial block size  $k_1$  depends on the error rate  $\varepsilon$ . In the original article [BS93],  $k_1 \approx 0.73/\varepsilon$ . Optimised versions of the protocol presented in [Mar+15], however, tend to increase it up to  $1/\varepsilon$ . Moreover, [Mar+15] emphasises that  $k_1$  should be a power of two to reach the best reconciliation efficiency. Finally, the initial block size given in [Pac+15] is shown in Equation (4.4).

$$k_1 = \min(2^{\lceil \log_2(\frac{1}{\varepsilon}) \rceil}, \frac{n}{2}) \quad (4.4)$$

This initial block size, however, is only valid for very long frames, typically found in quantum key distribution. Using the value obtained from Equation (4.4) for PUF responses leaves errors in them most of the time. Next,  $k_1$  values from 4 to 32 bits are investigated.

#### 4.5.2.2 Number of passes

Performing more passes corrects more errors, but increases the leakage. The number of passes is limited by the fact that the block size cannot exceed half the response size  $n/2$ . This limitation is already present for the frames of  $2^{14}$  bits found in quantum key distribution, but is much more problematic for PUF responses, that are much shorter. For example, the passes must stop when  $k_i$  reaches 128 bits if the response has 256 bits. One solution [Mar+15; Pac+15] is to add passes with a block size of  $n/2$  to reduce the failure rate. Each extra pass requires only two parity checks, leaking two bits.

#### 4.5.2.3 Multiplication factor for the block size

As detailed in [Mar+15], the block size can be multiplied by another factor than two, but the best efficiency is achieved when the block size is a power of two. Therefore, we investigated multiplication factors of values two, four and eight, leading to the block sizes given in Table 4.1.

Table 4.1 – Block sizes used for the first passes and after

(a) 256-bit responses						(b) 1024-bit responses						
$k_1$	$k_2$	$k_3$	...	$k_i$		$k_1$	$k_2$	$k_3$	$k_4$	$k_5$	...	$k_i$
4	32	128	...	128		4	8	32	128	512	...	512
8	32	128	...	128		8	32	128	512	512	...	512
16	64	128	...	128		16	32	128	512	512	...	512
32	64	128	...	128		32	128	512	512	512	...	512

### 4.5.3 Design flow

Setting the parameters of the CASCADE protocol requires to know the error rate and the target failure rate. The PUF can be characterised to know the error rate. The target application characteristic defines the failure rate. From the simulation results, the initial block size and the number of passes can then be obtained. This also gives the leakage. If the leakage is too high for the application, more bits from the PUF can be requested to obtain a secret of sufficient length. Table 4.2 shows which parameters can be chosen for the CASCADE protocol in real-life examples to achieve a failure rate of  $10^{-4}$ ,  $10^{-6}$  or  $10^{-8}$  and to keep at least 128 bits secret. Three PUF architectures are considered: TERO-PUF, RO-PUF and SRAM-PUF (see Section 1.5.2.3 for detailed descriptions). The error rates for these PUFs provided in the original articles are used to obtain the initial block size  $k_1$ , the number of passes and the number of bits required from the PUF.

Table 4.2 – Examples of parameters to achieve failure-rates of  $10^{-4}$ ,  $10^{-6}$  and  $10^{-8}$  for different PUF architectures, aiming at keeping at least 128 bits secret.

PUF	Article	Target	Technology node	Error rate $\varepsilon$	Target failure rate								
					$10^{-4}$			$10^{-6}$			$10^{-8}$		
					$k_1$ [bits]	#passes	PUF bits required	$k_1$ [bits]	#passes	PUF bits required	$k_1$ [bits]	#passes	PUF bits required
RO	[Mai+10]	FPGA	90 nm	0.9%	8	10	256	8	20	256	8	30	256
	[Mae+12]	ASIC	65 nm	2.8%	8	15	256	8	25	256	8	30	256
TERO	[Bos+14]	FPGA	90 nm	1.7%	8	15	256	8	25	256	8	30	256
	[MBC16]	FPGA	28 nm	1.8%	8	15	256	8	25	256	8	30	256
	[CBM16]	ASIC	350 nm	0.6%	8	10	256	8	20	256	8	30	256
SRAM	[Gua+07]	FPGA	—	4%	8	15	256	8	20	512	8	30	512
	[Ays+15]	FPGA	—	10%	4	15	512	8	25	512	4	44	512
	[MTV09a]	FPGA	65 nm	15%	4	15	1024	4	20	1024	4	50	1024
	[CLB11]	ASIC	65 nm	5.5%	8	18	256	8	20	512	8	30	512

## 4.6 Implementation

The implementation of the CASCADE protocol in the context of error correction of silicon PUF responses is done both on the device side and on the server side. The server is assumed to have high computational capabilities, while the device-side implementation should be as lightweight as possible.

The only feature that must be implemented on the device is the parity computation. Upon receiving a list of indexes, the device computes the parity of the block composed of the PUF response bits found at these indexes. This parity value is then sent back to the server. All the other operations required by the protocol, namely the block size computation, the choice of random permutations and the error detection and correction, are done on the server. This distribution of operations between the device and the server is summarised in Table 4.3.

Table 4.3 – Distribution of operations between device and server.

Feature	Device side	Server side
Block-size computation		✓
Parity computations	✓	✓
Permutations		✓
Error detection		✓
Error correction		✓

There are several possibilities to implement the parity computation module on the device. They are detailed below.

### 4.6.1 Large multiplexer

The first option to implement the parity computation module is shown in Figure 4.6. This architecture computes the parity of a block, given the indexes, by multiplexing the associated response bits one after the other to an XOR gate. The parity value is sampled by a DFF.

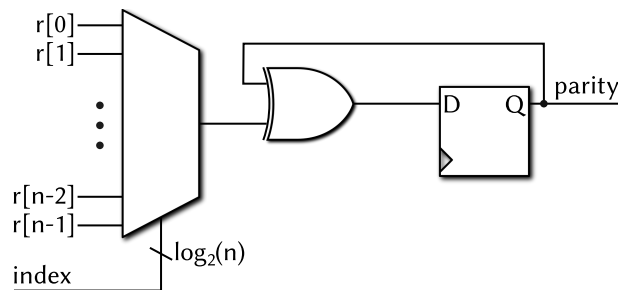


Figure 4.6 – Implementation of the parity computation module using one large multiplexer

In this first implementation, we assume that the response obtained from the PUF is stored in an  $n$ -bit shift register. This shift register can be made circular to individually select the response bits.

### 4.6.2 Circular shift register

Among the classical PUF architectures, the ones based on ring oscillators have the characteristic to not directly generate the whole response. For example, the RO-PUF compares the frequencies of two ring-oscillators, generating the response bit by identifying the fastest one. Individual bits are generated one after the other, and must be stored in a shift register that will eventually hold the full response. Such shift register can be made circular by connecting its output to its input. It reduces the amount of logic resources required to implement the parity computation module. The architecture is shown in Figure 4.7, where  $r[k]$  is the response bit generated by the PUF that is going to be stored in the shift register.

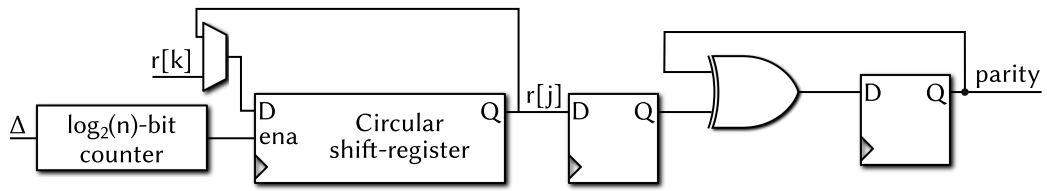


Figure 4.7 – Implementation of the parity computation module by making an existing shift register circular

In order to select the individual response bits, a  $\log_2(n)$ -bit counter is required. It holds the number of positions of which the register must be shifted to obtain the response bit. In order to pre-load this number, the counter has a  $\Delta$  input (see Figure 4.7). The value fed to the  $\Delta$  input is computed in the following manner. Let two response bits that must be selected for the parity computation be called  $r[i]$  and  $r[j]$ .  $r[j]$  must then be selected after  $r[i]$ . There are two possible cases when selecting these response bits:

- If  $j > i$ , the counter must be preloaded to  $j - i$ , which is the number of positions that must be shifted to go from  $r[i]$  to  $r[j]$ .
- If  $j < i$ , the counter must be preloaded to  $n + j - i$ , which is the number of positions that must be shifted to go from  $r[i]$  to  $r[j]$  when wrapping beyond the response length  $n$ .

The counter must then be preloaded to the  $\Delta$  value shown in Equation (4.5).

$$\Delta = (j - i) \bmod n \quad (4.5)$$

Therefore, the counter must be  $\log_2(n)$ -bit wide to index all the response bits. A list of  $\Delta$  values is computed by the server and sent to the device, instead of the list of indexes.

### 4.6.3 RAM

The last implementation option is to have the PUF response stored in RAM. In order to store 256, 512 or 1024-bit responses,  $32 \times 8$ ,  $64 \times 8$  and  $128 \times 8$  RAM blocks are used respectively, and



the response is split into bytes. Since the RAM has an intrinsic multiplexing capability for the bytes, only one 8:1 multiplexer is needed to access the response bits individually. The index input is split into two parts. The three least significant bits drive the selection input of the multiplexer, while the other bits are sent to the address input of the RAM.

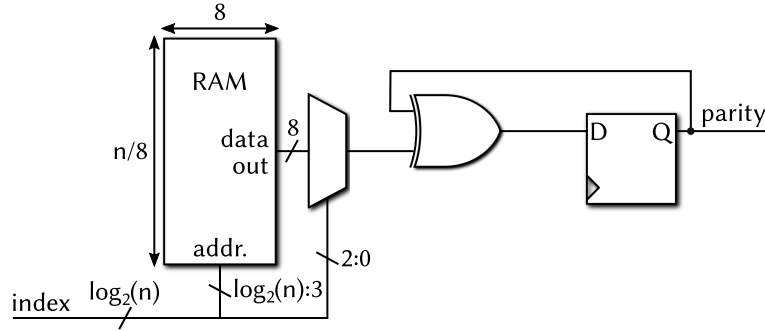


Figure 4.8 – Implementation of the parity computation module when the response is stored in RAM

## 4.7 Experimental results

In this section, we observe how the leakage, the failure rate and the execution time change with respect to the CASCADE parameters: the initial block size and the number of passes. These results were obtained after simulating one hundred million executions of the protocol in parallel on a computing server that embeds two Intel Xeon E5-2667 CPUs. Each CPU has eight cores, operating at 3.20GHz. The PUF response  $r_0$  and  $r_t$  were randomly generated with the error rate of interest. The added errors were assumed to be independent and identically distributed. This might not be the case for real PUF implementations and will be discussed in Section 4.8.1.1.

### 4.7.1 Leakage

When considering the leakage induced by the CASCADE protocol execution, we arbitrarily define a security threshold at 128 bits. This means that the objective is to keep secret at least 128 bits of the response. In case the PUF response is then processed to generate a symmetric cryptographic key, this value of 128 bits is in accordance with the recommendations made by known agencies and institutes<sup>1</sup>. The leakage values obtained for different sets of parameters are shown in Figure 4.9, while detailed values can be found in Table 4.4.

As mentioned before, increasing the number of passes leads to leaking more bits. For some cases, the security threshold of 128 bits is crossed. For example, for a 15% error rate, 30 passes with 8-bit initial blocks leaks the whole response. Conversely, starting with smaller

<sup>1</sup><https://www.keylength.com/>

Passes	Error rate and initial block sizes															
	1%				3%				5%				15%			
	4	8	16	32	4	8	16	32	4	8	16	32	4	8	16	32
1	68	39	24	17	78	50	36	25	86	59	42	27	—	—	—	—
3	79	49	32	26	91	67	55	50	104	86	73	63	—	—	—	—
5	83	54	26	31	95	72	61	59	109	92	86	81	—	—	—	—
10	93	64	47	41	105	82	72	72	119	103	102	104	811	685	565	330
15	103	74	57	51	115	92	82	82	129	113	113	116	—	—	—	—
20	113	84	67	61	125	102	92	92	139	123	123	126	831	878	763	527
30	—	—	—	—	—	—	—	—	—	—	—	—	851	1024	958	724
40	—	—	—	—	—	—	—	—	—	—	—	—	872	1024	1024	920

Table 4.4 – Leakage values (in bits) obtained with different error rates, initial block sizes and number of passes

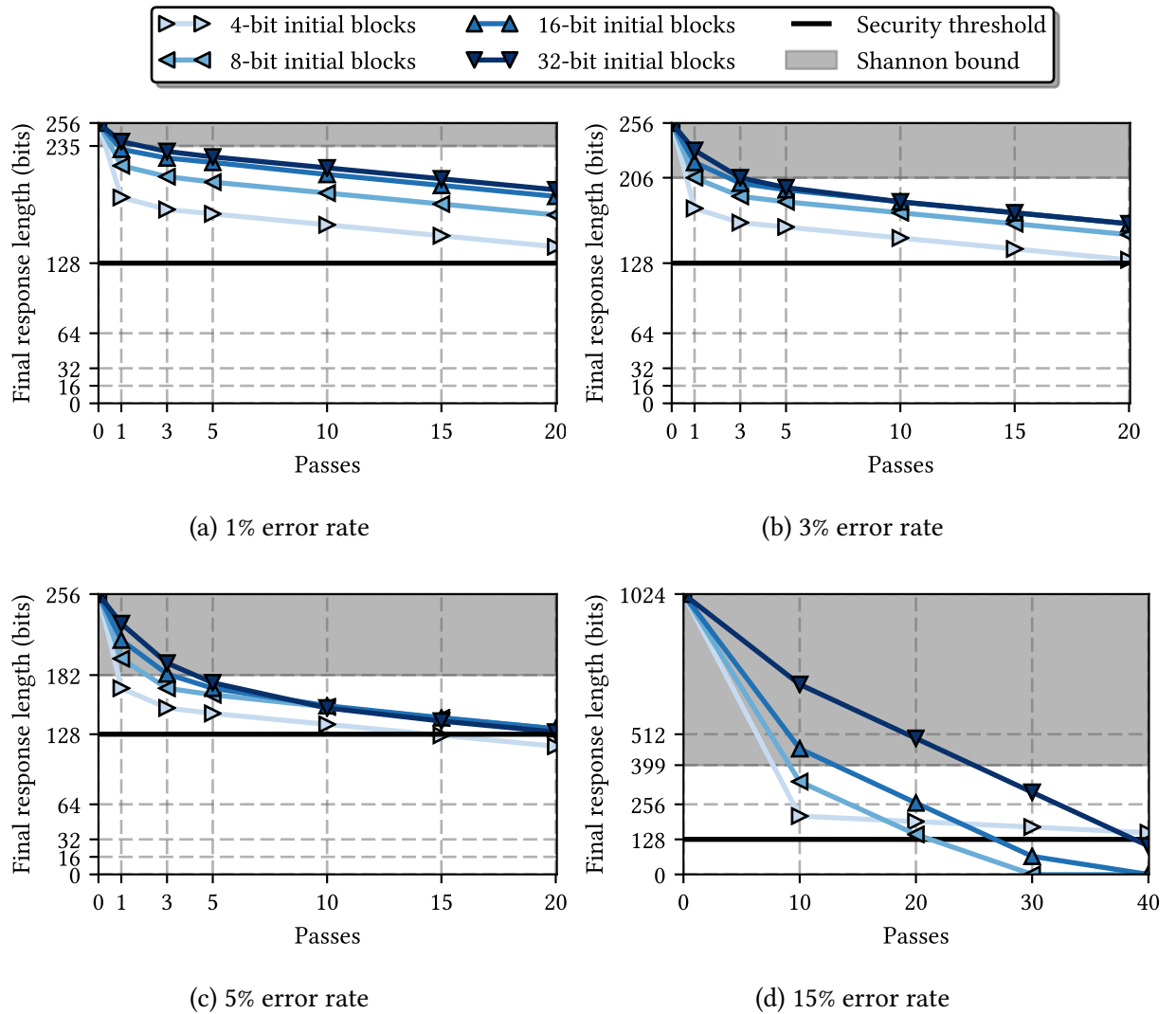


Figure 4.9 – Leakage values (in bits) obtained with different error rates, initial block sizes and number of passes.

blocks of 4 bits keeps 128 bits secret. Up to a 5% error rate, which is typically observed for RO-PUFs and TERO-PUF responses, stopping at 20 passes keeps 128 bits secret in all the cases.

An interesting phenomenon occurs for  $\varepsilon = 15\%$ . Starting with small blocks leaks less. This is because, when the initial blocks are larger, the amount of blocks in the first passes is not sufficient to detect all the errors. Therefore, they are corrected in later passes, when the blocks are even larger. Then, the binary search carried out in the CONFIRM method leaks more information to isolate the error than when it is carried on smaller blocks.

The second criterion that must be taken into account is the failure rate. Indeed, keeping 128 bits secret is of no use if some errors are left uncorrected. This is detailed in the following section.

### 4.7.2 Failure rate

The failure rate values obtained for different sets of parameters are shown in Figure 4.10, while detailed values can be found in Table 4.5. Increasing the number of passes makes it possible to detect and correct more errors, reducing the failure rate. Additionally, starting with smaller blocks also detects and corrects more errors, reducing the failure rate even further. These results show that for all the considered error rates, a failure rate below  $10^{-6}$  can be reached. This is in accordance with the failure rates typically achieved with classical error correction codes used for PUFs [MTV09b; Hil+12; HYS16]. Figure 4.10d shows the failure rate pattern observed for a 15% error rate. It clearly shows that the only solution when the error rate is so high is to start with small blocks of four bits. All other configurations starting with larger blocks cannot reach satisfactory failure rates.

### 4.7.3 Logic resources

We implemented the three proposed architectures given in Figures 4.6, 4.7 and 4.8, based on a large multiplexer, a circular shift register or a RAM block. The implementation is done on cost-optimised FPGAs Xilinx Spartan and Intel Cyclone, since those are typically used for applications that require low cost in logic resources. We only report the implementation cost of the parity computation module itself. The controller is not taken into account, as it is done for the majority of existing works. We give the implementation results in Table 4.6 with low level metrics: number of LUTs, number of DFFs and number of RAM bits. This allows for a fair comparison between FPGAs from different vendors. For comparison with existing work, we also provide the implementation results in number of Slices/ALMs<sup>2</sup>/LCs<sup>3</sup>.

As one can see by comparing these implementation results with the ones obtained with classical error correction codes, given in Table 1.4, the CASCADE protocol has a very lightweight device-side implementation.

---

<sup>2</sup>ALM: Adaptive Logic Module

<sup>3</sup>LC: Logic cell

Passes	Error rate and initial block sizes															
	1%				3%				5%				15%			
	4	8	16	32	4	8	16	32	4	8	16	32	4	8	16	32
1	$10^{-1}$	$10^{-1}$	1	1	1	1	1	1	1	1	1	1	—	—	—	—
3	$10^{-2}$	$10^{-2}$	$10^{-1}$	$10^{-1}$	$10^{-1}$	$10^{-1}$	1	1	$10^{-1}$	1	1	1	—	—	—	—
5	$10^{-3}$	$10^{-2}$	$10^{-2}$	$10^{-1}$	$10^{-2}$	$10^{-1}$	$10^{-1}$	1	$10^{-1}$	$10^{-1}$	1	1	—	—	—	—
10	$10^{-4}$	$10^{-4}$	$10^{-3}$	$10^{-3}$	$10^{-3}$	$10^{-3}$	$10^{-2}$	$10^{-1}$	$10^{-3}$	$10^{-2}$	$10^{-1}$	$10^{-1}$	1	1	1	1
15	$10^{-6}$	$<10^{-6}$	$10^{-5}$	$10^{-4}$	$10^{-5}$	$10^{-4}$	$10^{-3}$	$10^{-3}$	$10^{-4}$	$10^{-3}$	$10^{-2}$	$10^{-2}$	—	—	—	—
20	$<10^{-6}$	$<10^{-6}$	$10^{-6}$	$10^{-6}$	$<10^{-6}$	$10^{-5}$	$10^{-5}$	$10^{-4}$	$<10^{-6}$	$10^{-5}$	$10^{-3}$	$10^{-3}$	$10^{-4}$	1	1	1
30	—	—	—	—	—	—	—	—	—	—	—	—	$<10^{-6}$	1	1	1
40	—	—	—	—	—	—	—	—	—	—	—	—	$<10^{-6}$	1	1	1

Table 4.5 – Order of magnitude of the failure rate values obtained with different error rates, initial block sizes and number of passes

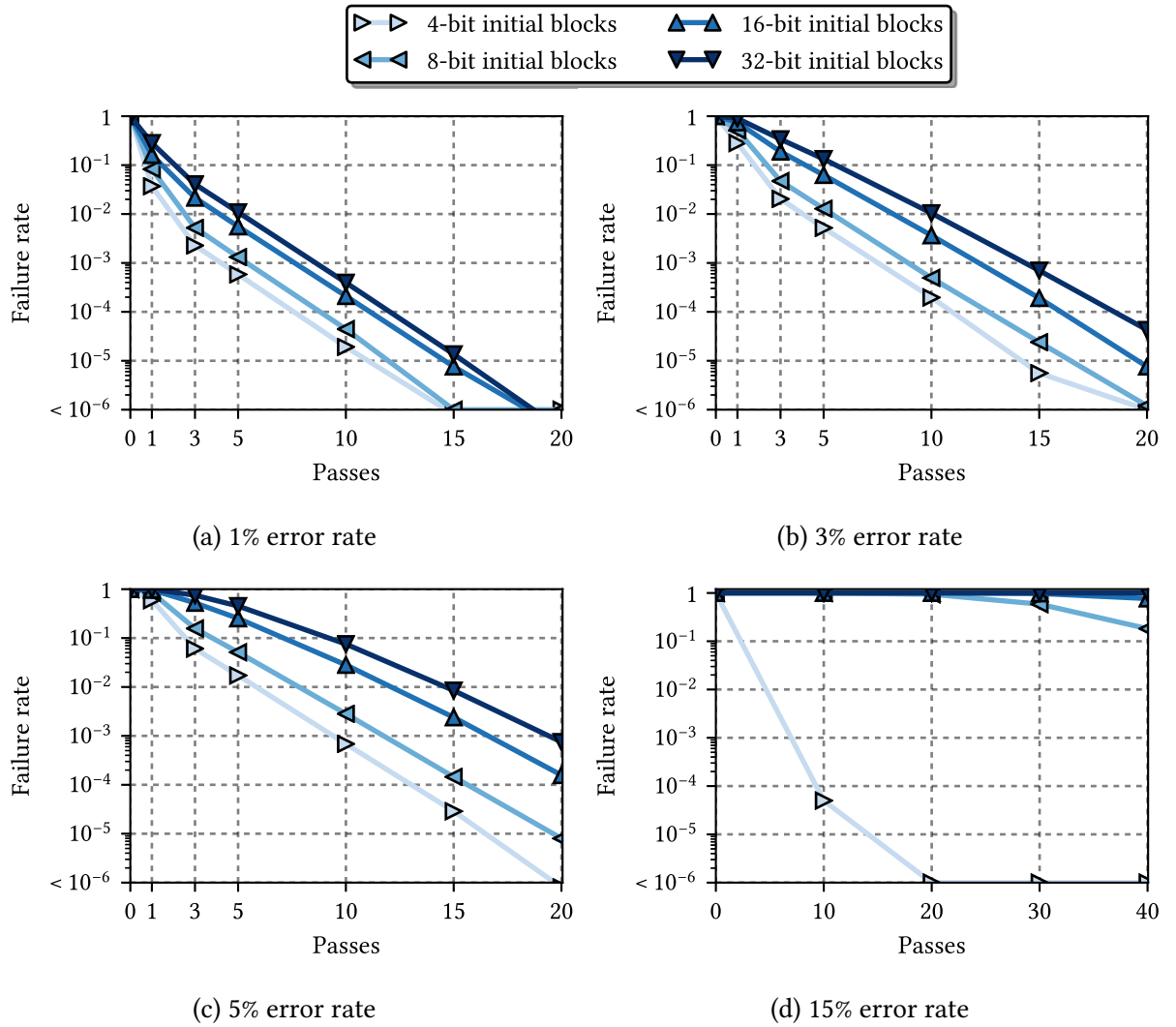


Figure 4.10 – Failure rate values obtained with different error rates, initial block sizes and number of passes.

256-bit response													
Target device	Option 1: Large multiplexer				Option 2: Circular shift register				Option 3: RAM				Logic
	LUTs	DFFs	RAM bits	Logic	LUTs	DFFs	RAM bits	Logic	LUTs	DFFs	RAM bits	Logic	
Xilinx Spartan 3 <sup>a</sup>	133	1	0	67 Slices	26	12	0	17 Slices	5	1	256	3 Slices	
Xilinx Spartan 6 <sup>b</sup>	67	1	0	19 Slices	17	12	0	7 Slices	3	1	256	1 Slice	
Intel Cyclone III <sup>a</sup>	170	1	0	170 LCs	25	20	0	26 LCs	6	1	256	6 LCs	
Intel Cyclone V <sup>c</sup>	86	1	0	46 ALMs	23	20	0	13 ALMs	4	1	256	3 ALMs	
512-bit response													
Target device	Option 1: Large multiplexer				Option 2: Circular shift register				Option 3: RAM				Logic
	LUTs	DFFs	RAM bits	Logic	LUTs	DFFs	RAM bits	Logic	LUTs	DFFs	RAM bits	Logic	
Xilinx Spartan 3 <sup>a</sup>	265	1	0	133 Slices	26	13	0	18 Slices	5	1	512	3 Slices	
Xilinx Spartan 6 <sup>b</sup>	171	1	0	92 Slices	25	13	0	11 Slices	3	1	512	1 Slice	
Intel Cyclone III <sup>a</sup>	342	1	0	342 LCs	28	22	0	29 LCs	6	1	512	6 LCs	
Intel Cyclone V <sup>c</sup>	171	1	0	87 ALMs	26	22	0	14 ALMs	4	1	512	3 ALMs	
1024-bit response													
Target device	Option 1: Large multiplexer				Option 2: Circular shift register				Option 3: RAM				Logic
	LUTs	DFFs	RAM bits	Logic	LUTs	DFFs	RAM bits	Logic	LUTs	DFFs	RAM bits	Logic	
Xilinx Spartan 3 <sup>a</sup>	529	1	0	265 Slices	28	14	0	18 Slices	5	1	1024	3 Slices	
Xilinx Spartan 6 <sup>b</sup>	341	1	0	182 Slices	27	14	0	10 Slices	3	1	1024	1 Slice	
Intel Cyclone III <sup>a</sup>	683	1	0	683 LCs	30	24	0	31 LCs	6	1	1024	6 LCs	
Intel Cyclone V <sup>c</sup>	342	1	0	176 ALMs	28	24	0	15 ALMs	4	1	1024	3 ALMs	

<sup>a</sup> 4-input LUTs<sup>b</sup> 6-input LUTs<sup>c</sup> 7-input LUTs

Table 4.6 – Logic resources required for three implementation options of the parity computation module and three response sizes.

When choosing the first implementation option, most of the resources are occupied by the large  $n$  to 1 multiplexer. The number of LUTs required to implement it grows linearly with the response length. Such implementation option is better suited for ASIC. Indeed, a large multiplexer is costly to implement using LUTs, while an ASIC implementation is more compact.

The second implementation option, that consists in reusing an existing shift register and make it circular, is much more lightweight. The size of the counter that must be added to index the response bits grows logarithmically with respect to the number of bits in the PUF response. When the response size is doubled, only one extra DFF is required. This option is suited for both ASICs and FPGAs.

Finally, the third option is clearly better suited for FPGAs. On such devices, distributed or block RAM is available and easily usable. Since the RAM has an intrinsic capability to multiplex bytes, the logic resources required is much lower than for other implementation options. The extra 8:1 multiplexer that selects the response bits individually has a constant size, no matter the response length. The number of RAM bits required to store the response grows linearly with the response length. The implementation results show that this implementation option takes between 3 and 6 LUTs and only one DFF. This makes it the most lightweight error correction module to date.

#### 4.7.4 Execution time

The last criterion is the execution time of the protocol. In order to remain independent on the target device, the execution times are given in clock cycles. The first and third implementation options, based on a large multiplexer or a RAM, have an identical way to select the PUF bits. Therefore, their execution time is identical. The second implementation option, based on a circular shift register, has a longer execution time though. Indeed, it requires to shift the register to select the response bit of interest.

The protocol has both a fixed and a variable execution time parts. The fixed part corresponds to the initial parity checks. The variable part corresponds to the execution of the CONFIRM method. This is variable because the block size influences the time taken by the CONFIRM method. If the errors are detected when the blocks are small, the binary search is faster. Therefore, the sooner the errors are detected, the faster the overall protocol.

##### 4.7.4.1 Implementation options based on a large multiplexer or a RAM

For these two implementation options, the response bits are multiplexed to the XOR gate in one clock cycle, no matter how long the response is. Accessing the response bits has then  $\mathcal{O}(1)$  time complexity. For the initial parity checks, it then takes  $n$  clock cycles to compute the parity of all the blocks for an  $n$ -bit response. Applying the CONFIRM method on  $t$ -bit blocks takes  $t - 1$  clock cycles. This is the run time of the binary search, as given in Equation (4.6). This

corresponds to computing parities on blocks of size from  $t/2$  bits down to 1 bit.

$$\sum_{i=1}^{\log_2(t)} \frac{t}{2^i} = t - 1 \quad (4.6)$$

We now consider the previous case of a 256-bit response with an error-rate of 2%. This means that, on average, five bits are faulty. Choosing the best CASCADE parameters for this situation leads to pick  $k_1 = 32$  and 15 passes.

As mentioned before, the execution time depends on when the errors are detected by the parity checks. Therefore, we must distinguish an upper and a lower bound for the execution time. In the best case, giving the lower bound for the execution time, the errors are corrected as soon as possible in the execution of the protocol. The binary search is then done on smaller blocks. We consider in this case that the five errors are corrected in the first pass of the protocol. The device-side execution time is then:

$$256 \times 15 + 5 \times (32 - 1) = 3\,995 \text{ clock cycles}$$

If we take the worst case, the number of errors can be higher. For example, we consider here that 14 bits are faulty, which can occur with a probability of  $5 \cdot 10^{-4}$ . Since we consider the worst case scenario, the errors are corrected as late as possible. Therefore, CONFIRM is applied on larger blocks and takes longer. In this case, that is the upper bound, since the errors are corrected in the last passes, the execution time is:

$$256 \times 14 + 14 \times (128 - 1) = 5\,362 \text{ clock cycles}$$

#### 4.7.4.2 Implementation option based on a circular shift register

In order to select an individual response bit, the circular shift register must be shifted by an amount  $\Delta \in [1; n - 1]$ . On average, reaching the next response bit then takes  $n/2$  shifts. It follows that accessing the response bits has an  $\mathcal{O}(n)$  time complexity in this case, for an  $n$ -bit response.

For a  $t$ -bit block, computing its parity then takes  $nt/2$  clock cycles on average. Since carrying out the initial parity checks requires to compute the parity of the  $n/t$  blocks found in the response, then it takes  $n^2/2$  clock cycles on average. This is much longer than for the previously considered implementation options, that take only  $n$  clock cycles.

The number of clock cycles required to apply the CONFIRM method on a  $t$ -bit block is given in Equation (4.7)

$$\sum_{i=1}^{\log_2(t)} \frac{t \cdot \frac{n}{2}}{2^i} = \frac{n \cdot (t - 1)}{2} \quad (4.7)$$

Again, we consider the best and worst cases here, with a 256-bit response and a 2% error rate. The protocol starts with 32-bit blocks and runs for 15 passes.

In the best case, the errors are corrected as early as possible, in the first pass and on 32-bit blocks. The execution time is then:

$$\frac{256^2}{2} \times 15 + 5 \times \frac{256 \times (32 - 1)}{2} = 511\,360 \text{ clock cycles}$$

When the errors are corrected as late as possible, the block size is 128 bits. If there are 15 errors, the execution time is:

$$\frac{256^2}{2} \times 15 + 15 \times \frac{256 \times (128 - 1)}{2} = 735\,360 \text{ clock cycles}$$

#### 4.7.4.3 Comparison with the execution time of existing codes

In order to compare the execution time of the CASCADE protocol with existing codes, we consider two corner cases. First, the protocol is carried out with 256-bit response, an error rate of 1% and errors that are corrected as early as possible. In the other case, the protocol is carried out with 1024-bit responses, an error rate of 15% and errors that are corrected as late as possible. Table 4.7 shows the execution times for these cases as well as other previously considered codes.

Table 4.7 – Device-side execution time in clock cycles of different codes with different constructions.

Article	Construction and code(s)	Execution time (clock cycles)
[MHV12]	Concatenated: Repetition (7, 1, 3) and BCH (318, 174, 17)	50 831
[Hil+15]	Reed-Muller (4, 7)	108 000
[MTV09b]	Reed-Muller (2, 6)	10 298
[Bös+08]	Concatenated: Repetition (5, 1, 5) and Reed-Muller (1, 6)	6 505
[Bös+08]	Concatenated: Repetition (11, 1, 11) and Golay (24, 13, 7)	1 210
[HYS16]	Differential Sequence Coding	29 243
CASCADE with MUX or RAM	on 256-bit responses and $\varepsilon = 1\%$ , 15 passes starting with 32-bit blocks (errors corrected as early as possible)	3 933
CASCADE with MUX or RAM	on 1024-bit responses and $\varepsilon = 15\%$ , 45 passes, starting with 4-bit blocks (errors corrected as late as possible)	203 622
CASCADE with circular SR	on 256-bit responses and $\varepsilon = 1\%$ , 15 passes, starting with 32-bit blocks (errors corrected as early as possible)	503 424

As the results show, the execution time of the CASCADE protocol is very dependent on the size of the response to correct as well as the error rate. Implementation options based on a large multiplexer or a RAM have execution times between 4 000 and 200 000 clock cycles approximately. This is in the same order of magnitude as the other codes that are considered,



that range from 1 210 to 108 000 clock cycles.

When the second implementation option is picked, the execution time grows dramatically. This is because of the PUF response bit selection that has an  $\mathcal{O}(n)$  time complexity in this case. This option might then only be suitable for small responses and low error rates. Otherwise, the other implementation options should be preferred.

Depending on the target device on which the error correction module must be implemented, these results could be improved. Indeed, the logic function is very simple here and has a very short critical path. A higher clock frequency could then be used for this module specifically, reducing the overall latency of the protocol.

Nevertheless, due to the great interactivity of the CASCADE protocol, the main execution time bottleneck is the communication between the device and the server. Depending on the target platform, this could be an order of magnitude slower than intra-device communication. Therefore, the actual time taken to execute the whole protocol is very dependent on the final hardware target.

## 4.8 Security: attacks and countermeasures

We investigate three types of attacks against the CASCADE protocol: server impersonation, device impersonation and eavesdropping. We then make some propositions for countermeasures to thwart these attacks.

### 4.8.1 Server impersonation: chosen indexes scenario

In the case of server impersonation, the objective of the attacker is to recover the generated PUF response. This can be done by sending chosen indexes and observing the resulting parity value sent back by the device. Thus is a chosen indexes scenario. If done for a sufficient amount of times, the attacker can build a system of linear equations that is sufficiently determined to be solved by Gaussian elimination.

#### 4.8.1.1 Countermeasure: deterministic shuffling

Instead of picking a random permutation at the beginning of each pass in order to spread the errors, a deterministic set of permutations could be predefined to maximise the probability to separate faulty bits into different blocks. It prevents the attacker from adding new independent equations to the system that would need to be solved to recover the response. Another interesting point of choosing a deterministic set of permutations is to account for the error rate of each response bit individually. The stability of each PUF response bit can be obtained by characterisation [Mae13; MBC16]. Also, in the case of TERO-PUF for example for which multiple response bits are obtained for each challenge, some response bits are known to be less stable than others. Consequently, choosing a set of permutations would separate in different

blocks the bits that are known to be the most unstable. The method of choosing the best set of permutations could be studied in future works.

#### **4.8.1.2 Countermeasure: limitation to only one execution of the protocol**

In the use case of remote activation of integrated circuits that we consider in the SALWARE project, the circuit must be activated only once and remains active afterwards. Therefore, allowing for only one execution the protocol could be a countermeasure to server impersonation. However, this would require to hold one permanent bit of state on the device to know if the protocol has already been executed or not. A fuse could be blown to implement this, but it may not be possible to have this on the device depending on the technology used.

#### **4.8.1.3 Countermeasure: limitation of the number of parity values sent out**

The attacker must obtain a sufficient amount of parity information to build a system of equation that can be solved. Therefore, a hard limit could be set on the number of parity values that could be obtained from the device. By setting this limit at the security requirement of the application, the designer can make sure that a sufficient number of bits are kept secret. However, storing the number of parity bits extracted is problematic. Indeed, if an attacker resets the system, this information is lost and the protocol can be carried-out again to obtain more parity values. Moreover, nothing stops an attacker to execute the protocol multiple times. The number of parity values could be stored in NVM so that it cannot be reset. However, it might not be technologically feasible to add non-volatile memory to the IP core. In addition, an attacker could reset the circuit at the end of the protocol, before the number of parity values is written to the NVM.

#### **4.8.1.4 Countermeasure: generation of a response at each protocol execution**

The last countermeasure that we propose to thwart server impersonation is to force the generation of a new PUF response every time the protocol is initiated. This way, the parity values that an attacker would obtain correspond to different responses and cannot be merged into a system that is sufficiently determined to be solved.

Of course, in order for this countermeasure to work, two responses generated one after the other should always be different. This could be checked by always storing the previous response on the device, in an electrically-erasable programmable read-only memory (EEPROM) for instance, and comparing it to the newly generated one. Moreover, a potential attacker has no way of knowing if two consecutive responses are indeed identical or not.

The attack that consists in recovering the response from contradictory parity values is similar to the Learning parities with noise (LPN) problem, which is considered a hard problem and that has been used as the hardness assumption for some cryptographic constructs [Pie12]. Solving the LPN problem has an equivalent complexity to decoding from a random linear code

[BMT78], which is known to be NP-hard. Proving rigorously the equivalence between the LPN problem and the attack we described on the CASCADE protocol would require further investigation.

## 4.8.2 Device impersonation: chosen parities scenario

Another attack consists in impersonating the device with the aim of setting the reference response stored on the server to a chosen value. This could be achieved by sending specific parity values to the server. We propose to implement the following countermeasure on the server side against this threat.

### 4.8.2.1 Countermeasure: limitation of the number of server-side modifications

Device impersonation is prevented by limiting the number of bits that can be modified in the reference response stored on the server. Since the error rate is  $\varepsilon$ , the number of bits that are flipped in an  $n$ -bit response follows the binomial distribution  $\mathcal{B}(n, \varepsilon)$ . This sets a hard limit on the number of bits that flip, so that the probability that so many bits are flipped is lower than the failure rate of the protocol. For example, if 256-bit responses are used and exhibit a 2% error rate, if a failure rate of  $10^{-6}$  is required, then the limit is set to  $m$  so that  $P(X = m) < 10^{-6}$ . Therefore, in this case, up to 20 bits can be modified on the server side, but not more.

The maximum number of bits  $m$  that are allowed to be modified on the server side is given in Equation (4.8), in which  $f$  is the failure rate and  $X$  is the number of bits modified during one execution of the CASCADE protocol.

$$m : P(X = m) < f \quad (4.8)$$

Beyond the threshold  $m$ , the probability that an attacker is trying to impersonate a device and force the reference response is higher than the failure rate of the protocol. Therefore, no further modifications are done to the reference response stored on the server and the protocol is aborted.

## 4.9 Discussion

### 4.9.1 Privacy amplification

The number of bits leaked during one execution of the CASCADE protocol is known. The remaining entropy is then not only located on specific bits, but is spread over the PUF response bits. Therefore, the individual bits cannot be selectively discarded. Moreover, the initial response can exhibit poor statistical properties, and the response bits may not be independent. The next step consists then in processing the PUF response to have a higher entropy per bit.

This is called privacy amplification. Since we place ourselves in the random oracle model, a hash function can be used to this end. Figure 4.11 illustrates how the number of bits changes at different stages.

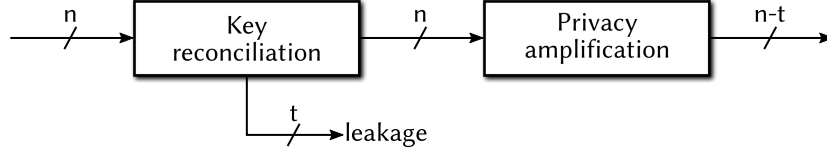


Figure 4.11 – Changes in the number of bits in the response at different steps.

During the key reconciliation protocol execution,  $t$  bits from the PUF response are leaked because of the parity checks. Consequently, the hash function that is used for privacy amplification should have an output of size  $n - t$  at most, so that all the output bits have maximum entropy. In order to limit the amount of logic resources required to implement the privacy amplification step, a lightweight hash function can be selected. SPONGENT [Bog+11] was chosen in [MHV12] and takes only 22 Slices on a Xilinx Spartan 6 FPGA, with an output block size of 128 bits. In [Mae+09], Toeplitz hashing [Kra94] was used. It occupies 59 Slices on Xilinx Spartan 3. The SHA-3 webpage provides other options for this use case in the “low area implementations” section<sup>4</sup>.

### 4.9.2 Replacing parity checks with hashing

In some works, it is suggested to replace the simple parity checks with hashing [BBR88; YI01]. This is sufficient to detect if errors occurred and has the advantage to detect an even number of errors in a block. However, this idea cannot be applied to our use case because of the small block size. Indeed, an attacker would only need to precompute the  $2^{k_1}$  possible values of the hash during the first parity check step. Since in our case  $k_1$  ranges from 4 to 32, this is computationally feasible. By observing the hash values sent by the device to the server, the attacker could then look up the associated response values and recover the whole response.

## 4.10 Conclusion

This chapter proposes a new way of correcting the errors found in silicon PUF responses, by using the existing key reconciliation protocol CASCADE. Originally proposed in the frame of quantum key distribution, we show that this protocol can be successfully applied to reconcile two slightly different PUF responses obtained from the same challenge but at different times. A server and a device then own a shared secret, that can later be processed to generate a cryptographic key.

<sup>4</sup>[http://ehash.iaik.tugraz.at/wiki/SHA-3\\_Hardware\\_Implementations](http://ehash.iaik.tugraz.at/wiki/SHA-3_Hardware_Implementations)

When using the CASCADE protocol for PUF responses though, some adaptations are necessary. We show by simulation that, by tuning the protocol parameters, it can cope with the short response sizes and typical error rates found in usual PUF architectures. We propose several sets of parameters that account for common error rates, response length and failure rates.

From a practical point of view, implementation results show that the device-side implementation of the CASCADE protocol is very lightweight in logic resources. We propose three architectures to implement the parity computation module, all leading to implementations that occupy at least three times less logic resources than existing ones that use classical error correction codes. The most lightweight implementation, when the PUF response is stored in RAM, takes less than six LUTs and one DFF.

Finally, we give a thorough security analysis of the use case of the protocol for PUF responses. We propose countermeasures against the described attacks, that do not hamper the area performance of the scheme.

In the use case of remote activation of IP cores, the CASCADE protocol is then a lightweight solution to correct the errors found in PUF responses. The tunable parameters allow to accommodate common PUF error rates and comply with the failure rates found in common applications.

## Chapter 5

# Complete hardware/software infrastructure IP for design protection

The final chapter of this thesis presents the integration of previously described individual components into a complete IP protection module. Besides the three contributions of this thesis, namely logic locking, logic masking and error correction based on key reconciliation protocols, it also presents other required primitives (such as the PUF and the lightweight block cipher, see Figure 5.1). It details the different implementation choices that can be made, as well as extra components that may be integrated to extend the features or the security of the IP protection module. A typical use case is then detailed, along with an illustrative example, giving the different steps that a designer should follow to protect an IP core at design time and activate it remotely later on.

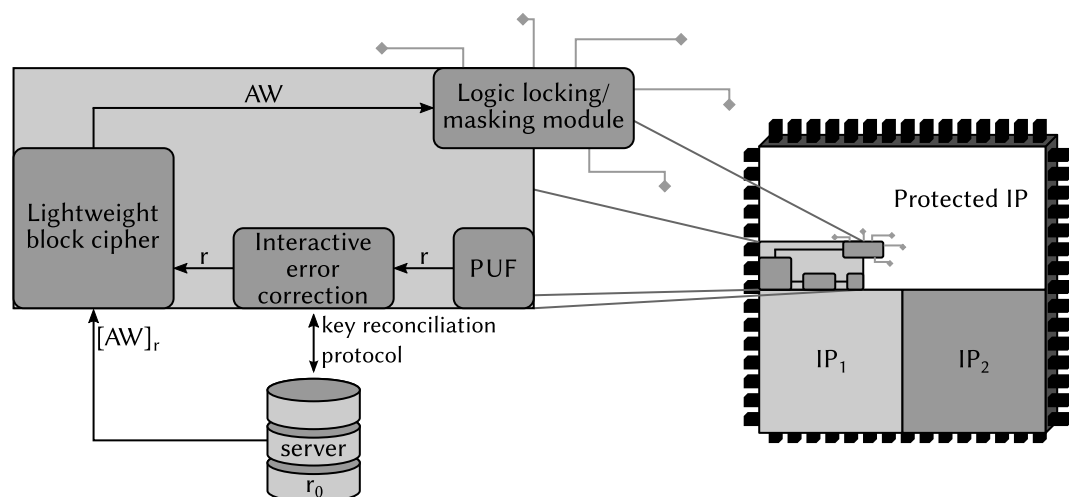


Figure 5.1 – IP protection module

# Infrastructure matérielle/logicielle pour la protection des données de conception

Le dernier chapitre de cette thèse présente l'intégration des modules individuels décrits précédemment dans un système de protection des données de conception. Au delà des trois contributions de cette thèse, le verrouillage logique, le masquage logique et la correction d'erreurs utilisant les protocoles de réconciliation de clés, il présente également d'autres primitives nécessaires tels que la PUF et le chiffreur léger (voir Figure 5.2). Ce chapitre présente les différents choix d'implémentation qui peuvent être faits, ainsi que les modules supplémentaires qui peuvent être intégrés pour étendre les possibilités ou la sécurité du module de protection. Un cas d'utilisation typique est ensuite détaillé, ainsi qu'un exemple illustratif, donnant les différentes étapes à suivre par un concepteur pour protéger un composant virtuel lors de sa conception et l'activer à distance plus tard.

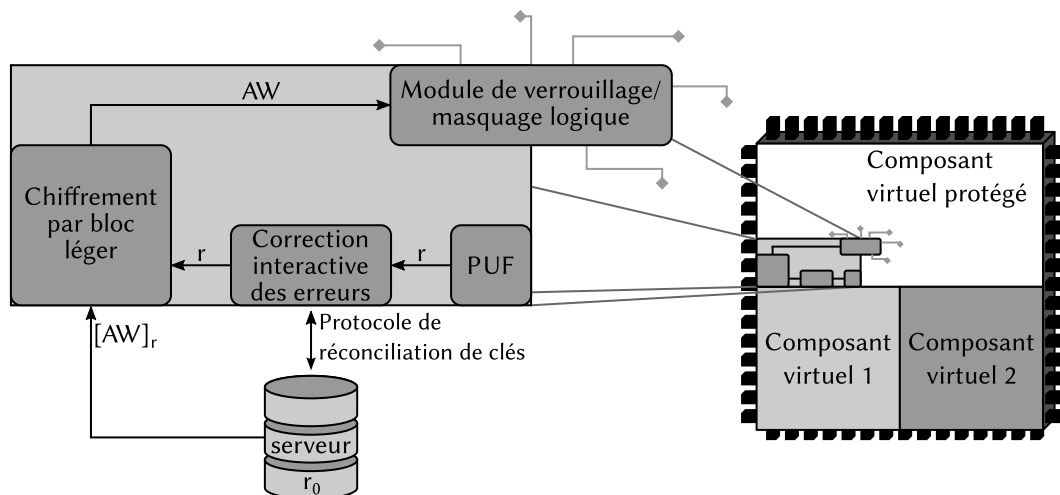


FIGURE 5.2 – Module de protection des données de conception

## 5.1 Integration into EDA tools

The IP protection module depicted in Figure 5.1 must be integrated into an existing design. Therefore, since the original IP core is modified to incorporate it, the design flow must be adapted. First the combinational logic is modified to incorporate extra logic gates that implement logic locking (see Chapter 2) or logic masking (see Chapter 3). Then, the extra modules like the lightweight cipher, the PUF, the parity computation module (see Chapter 4), etc. are added.

### 5.1.1 Modifications of combinational logic

The first step is to modify the combinational part of the design by logic locking or logic masking to ensure that, when not activated, the design does not operate correctly. First, the netlist is converted into a directed acyclic graph, following the conversion rules given in Section 2.2.1, Figure 2.4. When integrated into EDA tools, this conversion should handle several netlist formats: EDIF, BLIF, SLIF, gate-level VHDL or gate-level Verilog. The input netlist must be described at the gate-level. This is necessary for logic locking to identify the paths that propagate a locking value, and for logic masking to identify the best nodes to modify. The netlist to protect can be a description made by the designer directly, but in the most likely scenario a post-synthesis netlist is used. These netlists are typically found in various formats. A dedicated parser has been developed and is used to convert these different formats into a graph.

The graph can then be processed by the logic locking algorithm presented in Chapter 2. This step is optional, in case a designer only wants to implement logic masking, not locking. It is not recommended to implement both logic modifications on the same netlist. The compatibility and interaction between those two techniques could be studied in future works. The algorithm is driven by the number of outputs to lock. The designer does not choose the associated overhead, although a threshold could be set to limit it. In this case however, if the acceptable overhead is not sufficient, some outputs are left unlocked. Conversely, the designer could choose to increase the overhead in order to obtain a stronger locking, as described in Section 2.4.1. The type of locking gates that are inserted depends on the value that must be forced to propagate the locking value to the outputs. Therefore, the associated AW bits are not chosen by the designer in this case.

Then, the logic masking algorithm presented in Chapter 3 is applied to the graph. This step is optional, in case a designer only wants to implement logic locking, not masking. This algorithm is driven by the logic resources overhead the designer can afford. The higher the overhead, the more efficient the masking is. A typical EDA interface for the logic masking scheme will then let the designer pick the overhead as well as the selection heuristic used to select the nodes to mask. The choice of logic gates to insert, either XOR or XNOR, depends on the associated AW bit (see Section 1.5.4.1, Figure 1.16). Therefore, a random AW should first be



generated inside the EDA tool. The width of the AW depends on the logic resources overhead picked by the designer. The greater the overhead, the more masking gates are inserted, the longer the AW is. Obviously, the AW value should be truly random and not manipulable.

The designer can finally save the AW associated to the modified design. Afterwards, the final graph is converted back into a gate-level netlist as detailed in Section 2.2.6. The overall process is shown in Figure 5.3.

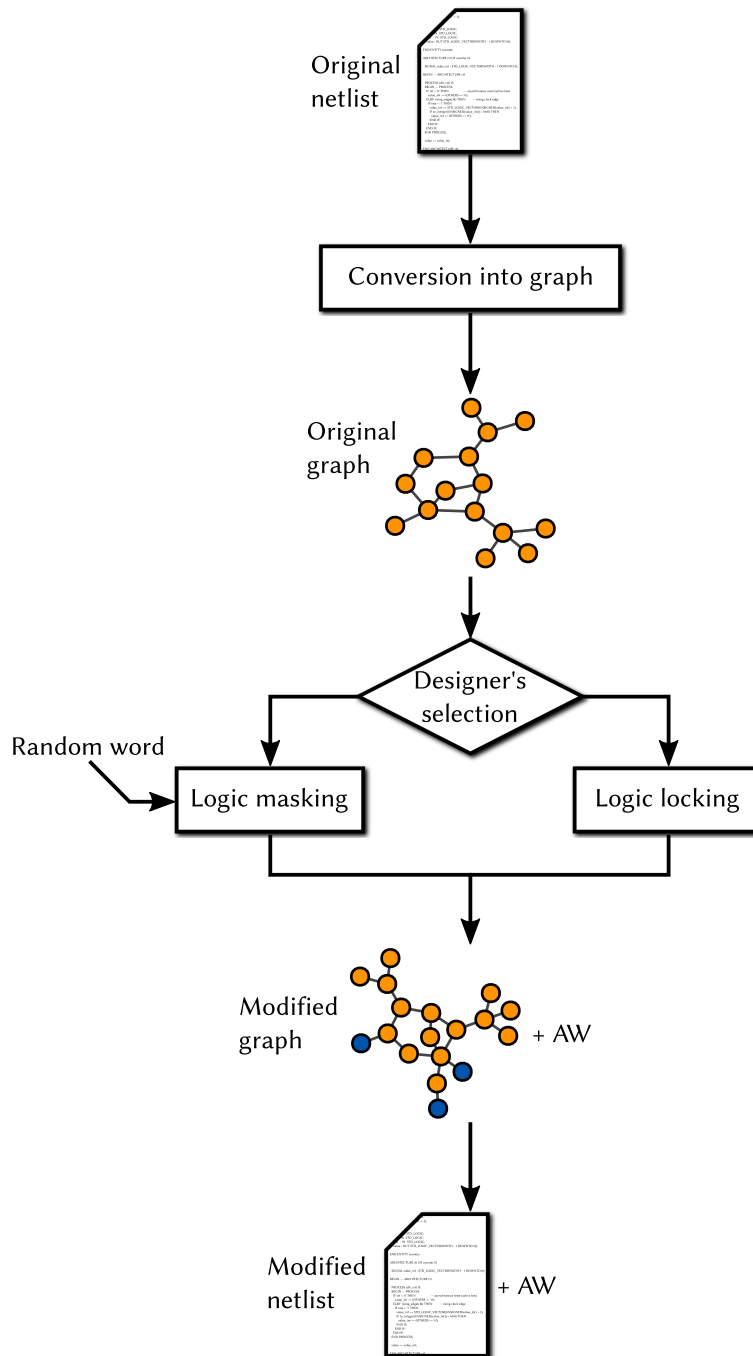


Figure 5.3 – Part of the design flow augmented for logic locking or logic masking

## 5.1.2 Additions to the original design

### 5.1.2.1 Lightweight block cipher

The modified design has extra inputs that must be driven by the correct AW. However, for security reasons these inputs are not directly exposed. As suggested in [Yas+15], a one way random function should be inserted before the AW inputs. Even though they propose to implement an AES encryption core with a fixed key to this end, we focus here on lightweight block cipher alternatives in order to limit the logic resources overhead. Implementations of lightweight block ciphers were done by Cédric Marchand who was a PhD student working in the framework of the SALWARE project as well [Mar16; MBG17]. The selected algorithms are recent and have a key size of 80 bits (KLEIN [GNL11], LILLIPUT [Ber+16] and KTANTAN[CDK09]) or 128 bits (LED [Guo+11]). In the threat model we use, 80-bit security is sufficient. The hardware implementation results are provided in Table 5.1. According to these, the most suited block cipher is KTANTAN [CDK09] since it takes less resources.

The EDA tool could give the possibility to the designer to pick the block cipher. The width of the key input should be identical to the one of the PUF response, since it is used as a symmetric encryption key. Another option is to hash the PUF response before using it as a key. This is detailed in Section 5.1.3.1.

For simplicity, in our demonstrator, we implemented only a one-time pad between the AW and the PUF response.

Cipher	4-input LUTs	DFFs
KLEIN [GNL11]	633	194
LED [Guo+11]	555	218
LILLIPUT [Ber+16]	558	205
KTANTAN [CDK09]	222	153

Table 5.1 – Logic resources required to implement a lightweight block cipher (from [Mar16; MBG17])

### 5.1.2.2 AW storage options during operation

The AW must be stored inside the IP core once it has been received in order to drive the activation inputs and make the IP core operate properly. The way the AW is stored depends on the mode of operation used for the lightweight block cipher. If used in Cipher Block Chaining, feedback (Output Feedback Mode or Cipher Feedback Mode) or stream cipher-like (Counter Mode or Galois Counter Mode [PP09]) modes, the AW is decrypted and then stored in a large register. The other option, if the cipher is used in Electronic Codebook Mode [PP09], is to use a decoder to adapt the AW size to the outputs of the block cipher.

**Large register** If the AW is larger than the output size of the cipher, the latter can be used in Cipher Block Chaining, feedback (Output Feedback Mode or Cipher Feedback Mode) or stream cipher-like (Counter Mode or Galois Counter Mode [PP09]) mode. The decrypted plaintext is then stored in a large register, as large as the AW. The output of this register drives the activation inputs of the IP core, activating it only if the correct AW encrypted with the reconciled PUF response is provided.

**Decoder** The cipher could also be used in Electronic Codebook Mode [PP09]. In this case, only one block is decrypted and stored in a register of the same size as the output of the cipher. However, this is usually the case that the block cipher has an output block size that is different from the width of the AW. Therefore, a decoder is required to map the  $n$ -bit output of the block cipher to the  $m$ -bit AW, as depicted in Figure 5.4.



Figure 5.4 – Position of the AW decoder

We can distinguish three cases when implementing the AW decoder. In the first case, there are less 0s (respectively 1s) at the output of the cipher than in the AW. For logic masking and for logic locking, each 0 (respectively 1) found at the cipher output drives multiple 0s (respectively 1s) found at the activation input. The decoder then implements an injective function  $\{0, 1\}^n \rightarrow \{0, 1\}^m$ .

In the second case, there are as many 0s (respectively 1s) at the output of the cipher as in the AW. For logic masking and for logic locking, each 0 (respectively 1) found at the cipher output is connected to a 0 (respectively 1) found at the activation input. The decoder then implements a bijective function  $\{0, 1\}^n \rightarrow \{0, 1\}^m$ .

In the last case, there are more 0s (respectively 1s) at the output of the cipher than in the AW. For logic masking and for logic locking, each 0 (respectively 1) found in the AW is driven by the disjunction (logical OR) of multiple 0s (respectively the conjunction (logical AND) of multiple 1s) found at the cipher output. The decoder then implements a surjective function  $\{0, 1\}^n \rightarrow \{0, 1\}^m$ .

In the case of total logic locking, the AW decoder implements the mapping from  $\{0, 1\}^n$  to the AW that is required to force *all* the outputs to a fixed logic value unless the correct AW is provided. Thus each 0 (respectively 1) found in the AW is driven by the disjunction of *all* the 0s (respectively the conjunction of multiple 1s) found at the cipher output. The decoder then implements a surjective function  $\{0, 1\}^n \rightarrow \text{AW}$ .

All these possibilities for the AW decoder architecture are illustrated in Table 5.2, while associated implementation results are given in Table 5.3. The logic resources required to implement the AW decoder for total logic locking are the same as the ones given for the hardware point function in Chapter 2, Table 2.4, but are provided for comparison.

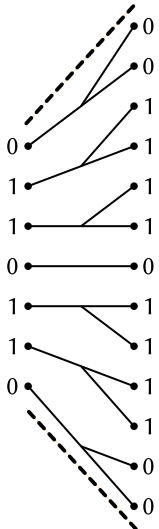
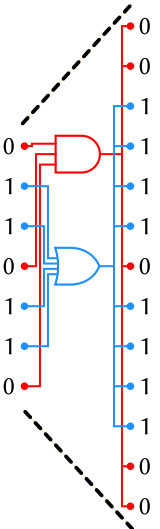
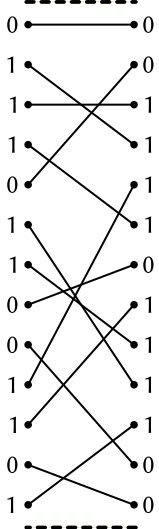
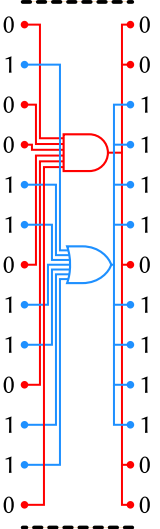
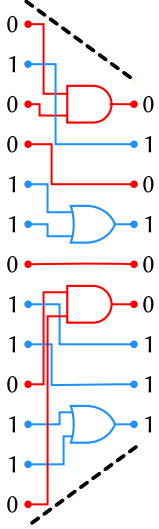
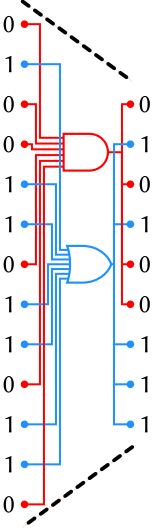
Case	Logic locking/Logic masking	Total logic locking
Less 0s or 1s at the cipher output than in the AW.		
Same number of 0s or 1s at the cipher output as in the AW.		
More 0s or 1s at the cipher output than in the AW.		

Table 5.2 – AW decoder architectures

These results show that, in the case where the number of 0s or 1s is lower or the same at the cipher output than in the AW, the AW decoder does not occupy logic resources. This is because the decoder is then just made of connections, that are already present in the FPGA. Since no logic function is implemented, no LUTs are occupied. In the other case, there are more more 0s or 1s at the output of the cipher than in the AW. Due to the fact that the logic function is very simple but has a lot of inputs/outputs, it prevents grouping inside the LUTs. In the example given in Table 5.3, a mapping from 126 to 64 bits requires to implement on average 64 2-input AND/OR logic functions. Since all the inputs of these functions are different, 64 LUTs are required, but only two inputs out of four or six are then used. In most real-life cases, however, the AW is wider than the output of the block cipher. For instance, a small benchmark of 5 000 gates locked or masked at 3% overhead leads to an AW of 150 bits. The output of the block cipher is usually 64 bits. Therefore, the decoder is implemented at zero cost most of the time. This is only valid for FPGA implementation. On ASIC, such AW decoder consists in a lot of routing, which cost must be evaluated on a per-design basis.

Input width (bits)	Output width (bits)	Logic locking/masking		Total logic locking	
		# 4-LUTs required	# 6-LUTs required	# 4-LUTs required	# 6-LUTs required
64	64	0	0	17	12
64	128	0	0	16	13
64	256	0	0	17	14
64	512	0	0	17	14
64	1024	0	0	16	14
64	2048	0	0	16	14
128	64	64	64	33	22
128	128	0	0	33	22
128	256	0	0	33	22
128	512	0	0	33	22
128	1024	0	0	33	22
128	2048	0	0	33	22

Table 5.3 – Logic resources required to implement the AW decoder

### 5.1.2.3 TERO-PUF

Once the block cipher has been picked, the PUF that generates the symmetric key is implemented. We implemented a TERO-PUF by combining TERO cells (see Figure 1.10) in the architecture shown in Section 1.5.2.3, Figure 1.9b. With this architecture, we extract two bits per comparison of the number of oscillations of two TERO cells. Therefore, if an  $n$ -bit response is required by the block cipher, two banks of  $n/2$  TERO cells are required. This is in case the PUF response is directly used as a symmetric key (see Section 5.1.3.1 if a hash function is added to derive the key).

The final PUF response that we use here is 128-bit long. In our implementation, we implemented two banks of 64 TERO cells. Pairs of cells are selected from those banks and compared. Two response bits are generated per comparison. The final response is then obtained after 64 comparison.

There are eight delay elements per TERO cell branch in our implementation (see Figure 5.5): seven inverters and one NAND gate.

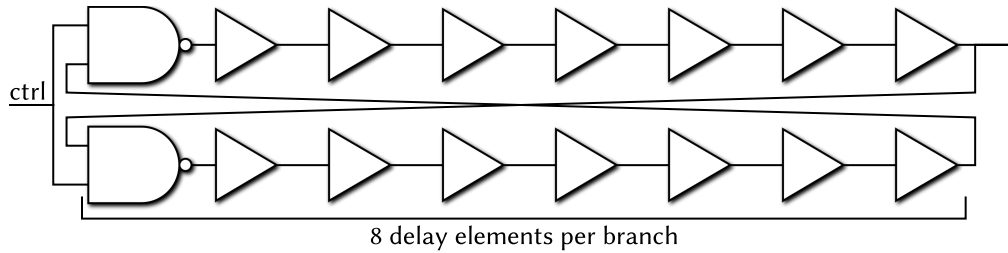


Figure 5.5 – TERO cell with 8 delay elements per branch (7 buffers and 1 NAND gate)

#### 5.1.2.4 CASCADE module

The CASCADE module can then be implemented on the device side. Depending on the chosen hardware target, an architecture based on RAM (see Figure 4.8) or a large multiplexer (see Figure 4.6) can be chosen by the designer. As said before, the parameters of the protocol are not fixed but are chosen by the server when the protocol starts. Therefore, the device-side implementation is generic. The only constraint is the size of the RAM or the size of the multiplexer, which should be the same as the size of the PUF response. For our implementation, since we deal with 128-bit responses, we allow for initial block sizes of 4, 8, 16, 32 or 64 bits, with a number of passes from 1 to 40.

#### 5.1.2.5 Controller and communication interface

Finally, a controller must also be added to the system to sequence the operations, as well as a communication interface. In order to minimise the communication time, as many parities as possible are computed on the device before sending them to the server. In our case, the smallest initial block size we consider is 4 bits. Since the PUF response we use is 128-bit long, the initial parity checks result in at most 32 parity bits. These parity values are accumulated and sent out all together. The controller and communication interface could be further optimised to reduce the logic resources overhead.

### 5.1.3 Optional additions

#### 5.1.3.1 Hash function

When assuming to be in the random oracle model, a hash function can be used to achieve the privacy amplification done at the end of the key reconciliation protocol (see Section 4.9.1). In

that case, the output block size of the hash function should be of the same size as the key input of the block cipher. The PUF response, which is then fed to the hash function to generate the key, can be of any size. To limit the logic resources required, the PUF response should be of the size of the smallest possible message that can be hashed without padding. The designer could then pick the hash function of his choice.

### 5.1.3.2 Watermark

The PUF described above allows to identify individual instances so that the key used to encrypt the AW is unique to each device. However, it may be necessary to identify the IP core itself in the first place. This can be easily achieved for example by inserting a small transmitter as proposed in [BBF15]. This transmitter, shown in Figure 5.6, can fit in only two 4-input LUTs on FPGA or less than 5 gate-equivalent in ASIC.

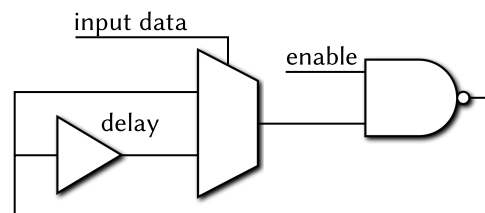


Figure 5.6 – BFSK transmitter from [BBF15]

## 5.2 Hardware platform: HECTOR board

The HECTOR board is composed of one motherboard, on which different daughterboards can be plugged. These boards have been developed in the framework of the European Union H2020 HECTOR project<sup>1</sup>.

### Motherboard

The HECTOR motherboard (see Figure 5.7) embeds a Microsemi SmartFusion 2 System on Chip (SoC) FPGA. The microcontroller subsystem allows to communicate easily with the PC by using Tcl scripts. This is interfaced with the FPGA fabric, which can then communicate with the daughterboard. The daughterboard is plugged directly on the motherboard using a SATA connector. Thus the motherboard is typically used for communication while the design to test is implemented on the daughterboard.

---

<sup>1</sup><https://hector-project.eu/>



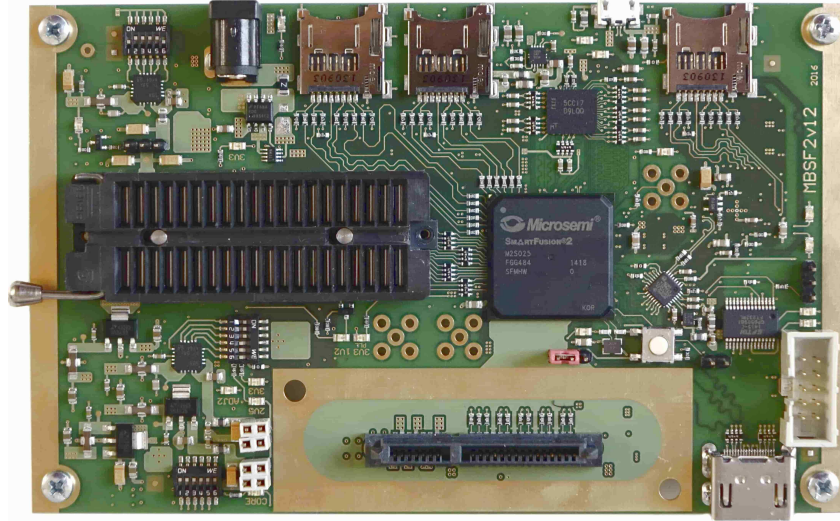
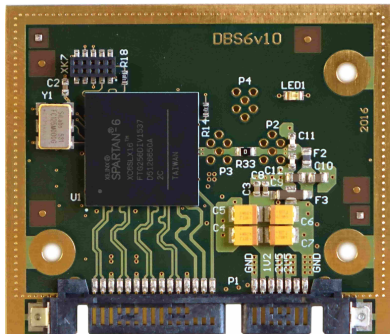


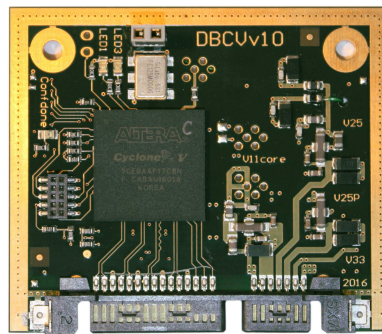
Figure 5.7 – HECTOR motherboard

## Daughterboards

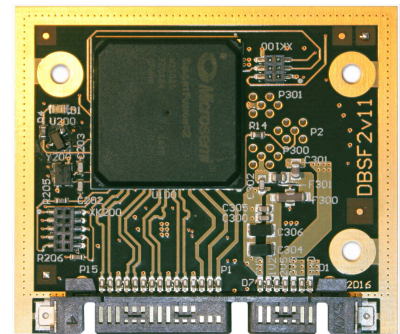
The HECTOR daughterboards embed three different FPGAs: Xilinx Spartan 6 (see Figure 5.8a), Intel Cyclone V (see Figure 5.8b) and Microsemi Smartfusion 2 (see Figure 5.8c).



(a) Xilinx Spartan 6



(b) Intel Cyclone V



(c) Microsemi SmartFusion 2

Figure 5.8 – HECTOR daughterboards

## 5.3 Overall hardware implementation results

The implementations results for the overall IP protection module are shown in Table 5.4. We implemented it on two FPGA families, Intel Cyclone V and Microsemi SmartFusion 2. For the implementation of the parity computation module used by the CASCADE protocol, we chose the option of using a large multiplexer. Using RAM would reduce the logic resources requirements. In our implementation, the communication between the server and the device is done with frames of up to 1024 bits, that contain the indexes or the parity values. Therefore, two 1024-bit registers are used as input and output registers for the communication. This implementation choice requires a large multiplexer to select the received indexes individually



for the CASCADE protocol execution. This could be adapted depending on the requirements and limitations of the target application.

The results presented in Table 5.4 are obtained from the synthesis tools Intel Quartus II 13.1 and Microsemi Libero SoC 11.7. The logic resources individually occupied by each entity can be obtained. However, for complex designs such as this one, separating the logic resources between the entities does not always give meaningful results. Indeed, the synthesis performs a lot of merging of logic to save logic resources. As a consequence, the values provided in Table 5.4 should be analysed while maintaining a critical perspective. The absolute values do not have much intrinsic value. Conversely, the relative implementation cost of each entity is more interesting. On the one hand, as mentioned before, the CASCADE module is extremely lightweight. On the other hand, the large multiplexer used to select the PUF response indexes occupies a lot of LUTs.

Entities	Intel Cyclone V		Microsemi SF2	
	6-LUTs	DFFs	4-LUTs	DFFs
PUF	4841	160	2258	158
Response shift register	0	128	0	128
Communication	321	2560	2664	2478
IP protection module	<b>444</b>	<b>357</b>	<b>1030</b>	<b>376</b>
MUX indexes 128x7:7	301	0	595	0
MUX response bits 128:1	37	0	85	0
One time pad	128	0	128	0
AW storage	0	128	0	128
CASCADE module	1	1	1	1
Controller	104	90	101	69
Parities shift register	0	35	0	32
<b>Total</b>	5606	2949	5746	2803

Table 5.4 – Device-side implementation results for the whole design data protection module

The logic resources overhead brought by the logic locking scheme is dependent on the design to protect and is not shown here. As said in the associated chapter, the overhead is 2.9% on average. The logic resources overhead associated to logic masking is also not shown, since it is up to the designer to choose it depending on the required masking efficiency.

Overall, these results show that the IP protection module is lightweight. Further optimisations could be carried out to reduce the cost. Vendor-specific FPGA resources can be used to implement specific functions. For example, Xilinx SRL16 can be exploited to implement the shift registers. We chose to make our implementation as generic as possible and did not use them.

## 5.4 Software interface

The graphical user interface described here is what could be integrated into EDA tools to allow a designer to protect an IP core from counterfeiting and illegal copying. The interface is split into four tabs, described below and meant to be used at different stages of the design process.

### Logic modifier

The *Logic modifier* tab, shown in Figure 5.9, performs the actions described in Section 5.1.1.

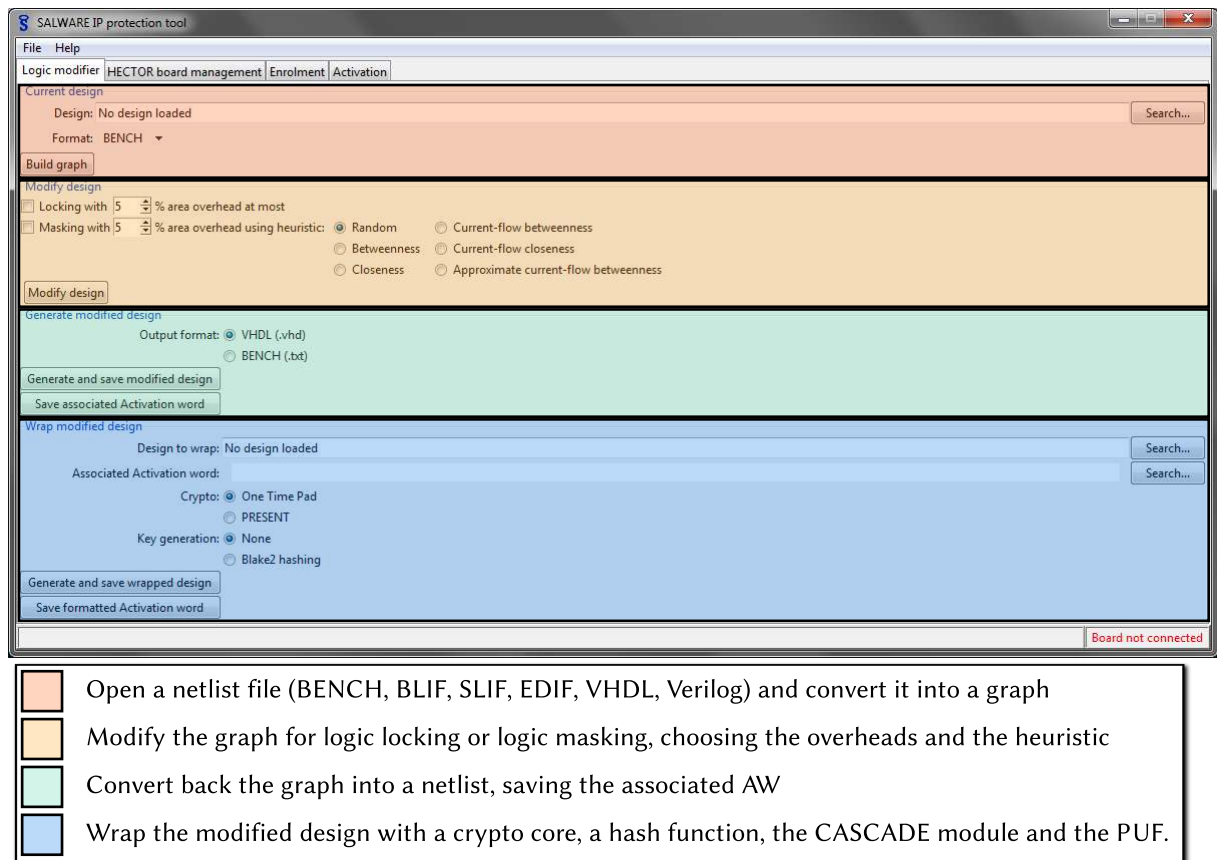


Figure 5.9 – Logic modifier tab of the graphical user interface

In the *Current design* frame, a design is loaded and converted into a directed acyclic graph from different netlist formats. In the *Modify design* frame, the designer can choose to lock or mask the design, setting the associated area overheads and the selection heuristic for logic masking. The modified netlist is then generated using the *Generate modified design* frame, along with the associated AW, that is stored in a dedicated file. Finally, the modified design is wrapped and associated with other building blocks such as the lightweight cipher, the parity computation module, the AW decoder, etc. This is done in the *Wrap modified design* frame. The formatted activation word that can be saved at this stage is the one that must be encrypted by the reconciled PUF response at activation time. It is the input of the AW decoder.

## HECTOR board management

The *HECTOR board management* tab, shown in Figure 5.10, allows to connect to the HECTOR board. This is necessary to perform the enrolment and activation phases.

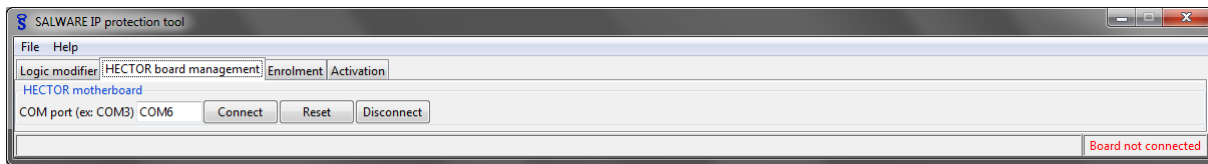


Figure 5.10 – HECTOR board management tab of the graphical user interface

## Enrolment

This tab (see Figure 5.11) allows the designer to perform the enrolment phase: obtaining the reference PUF response before storing it on the server. It is later used in the CASCADE protocol.

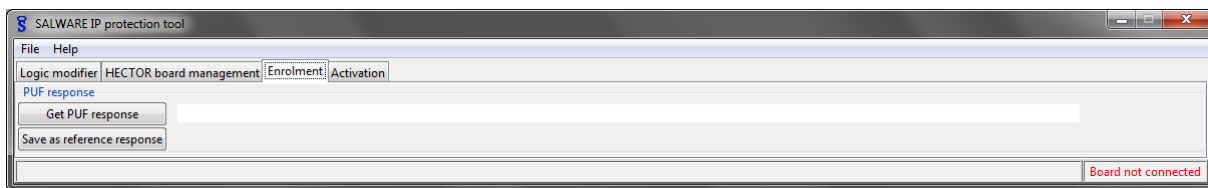


Figure 5.11 – Enrolment tab of the graphical user interface

## Activation

The last tab is dedicated to the activation phase (see Figure 5.12).

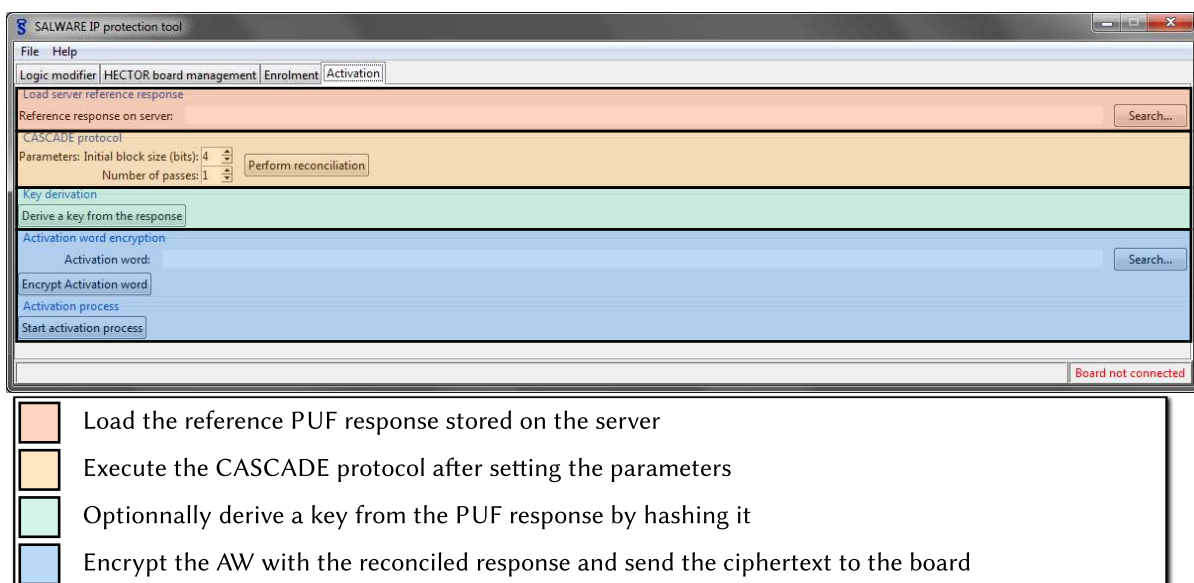


Figure 5.12 – Activation tab of the graphical user interface

This phase starts with the CASCADE protocol. The interface allows to load the reference response stored on the server. The parameters of the CASCADE protocol can then be set: the initial block size and the number of passes. After performing the protocol, the interface shows how many bits were leaked during its execution. Then, the reconciled PUF response is stored as is as a key, or optionnally hashed before. The designer can then load the AW, encrypt it with the PUF response and send the obtained ciphertext to the HECTOR board. This is decrypted internally and the design implemented on the board is activated.

## 5.5 Illustrative example

To illustrate the use of the IP protection scheme, we applied it on a test benchmark. It is a  $64 \times 64$  bits combinational multiplier, entirely implemented in LUTs. We also designed a graphical user interface to allow for easy tests for different inputs. The different cases obtained are depicted in Figures 5.13 and 5.14. In the first pictures on the left, Figures 5.13a and 5.14a, an example input is shown. The results obtained when the IP core is locked are shown in Figures 5.13b and 5.14b. Whatever the input operands are, when this particular design is locked, the output is always 0. The results obtained when the IP core is masked are shown in Figures 5.13c and 5.14c. The output is different for each input, but is always wrong. Finally, after activation has been carried out, the correct result is obtained. This is shown in Figures 5.13d and 5.14d.

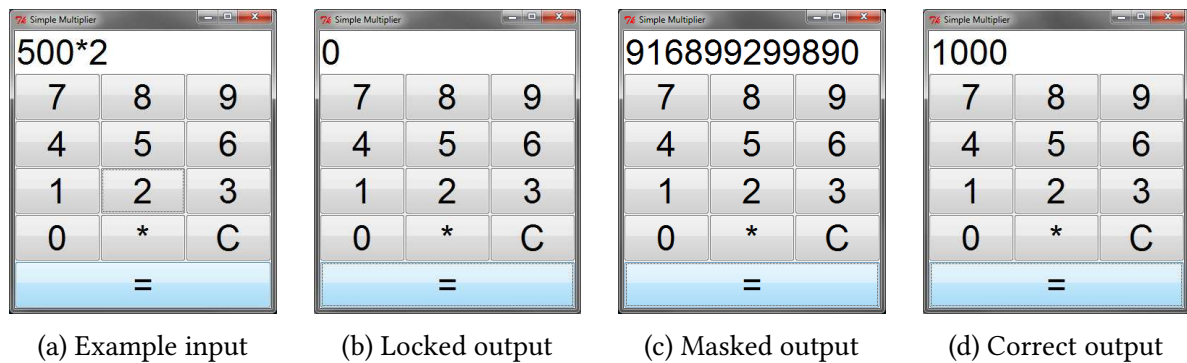


Figure 5.13 – Graphical user interface to the hardware multiplier with input  $500 \times 2$

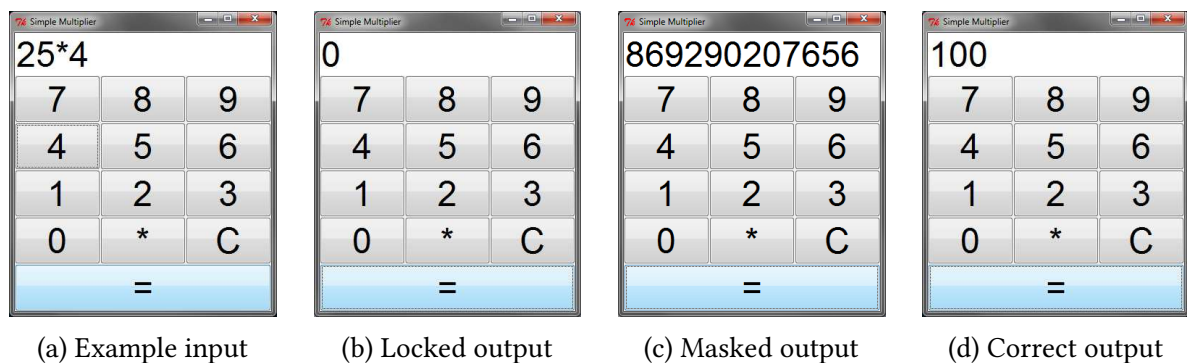


Figure 5.14 – Graphical user interface to the hardware multiplier with input  $25 \times 4$

## 5.6 Use case

The typical use case of the software/hardware infrastructure is the following, shown in Figure 5.15. For each step of the design process, the evolution of the IP core is depicted. Designer's constraints are also shown for several steps.

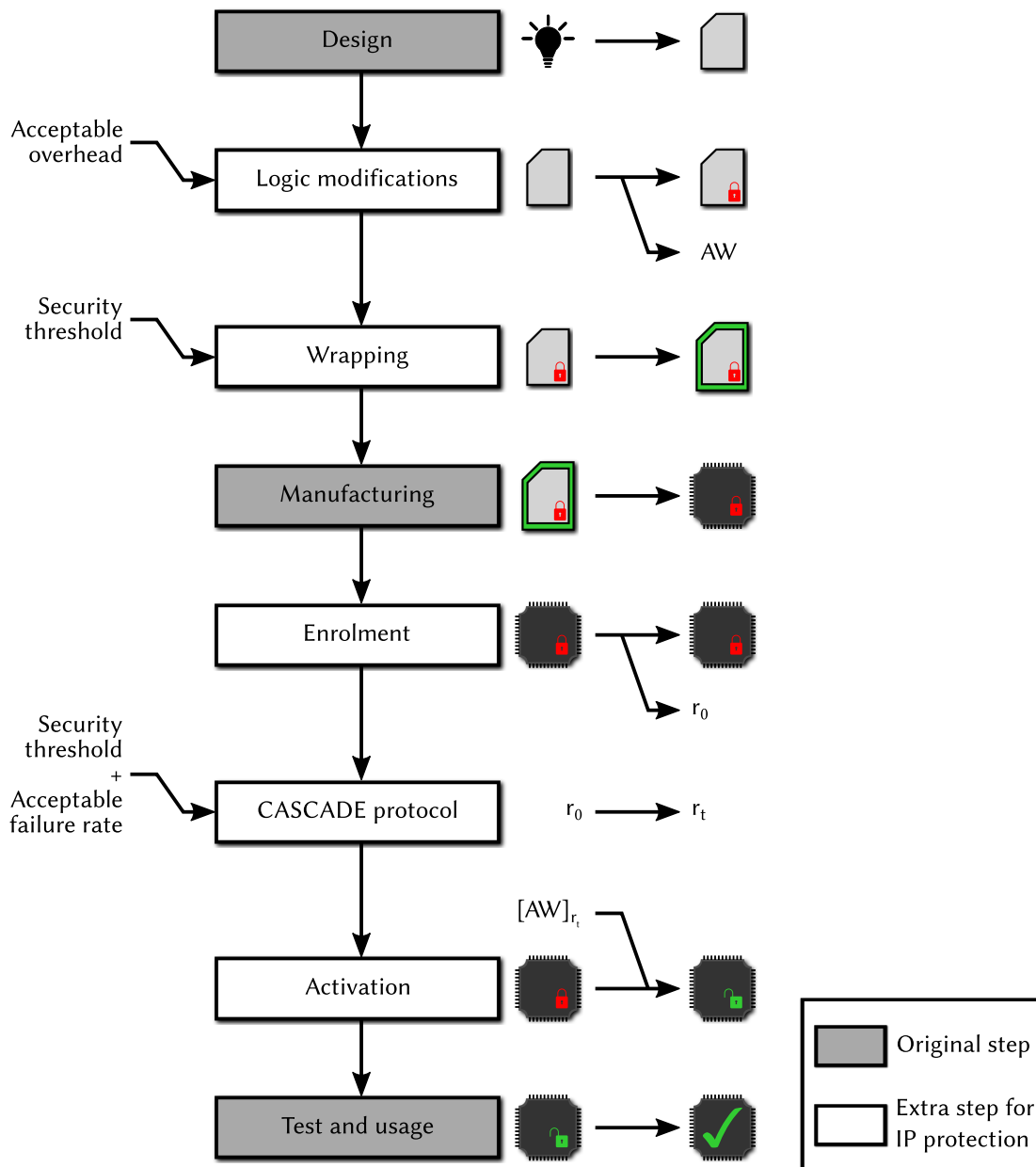


Figure 5.15 – Simplified design flow with steps implementing secure remote activation highlighted.

First, the designer opens the netlist with the EDA tool, and chooses to modify it with logic locking or logic masking, selecting the associated overheads. The associated AW is then stored on the server. The overall wrapper for the modified design is generated, comprising the submodules described above. The security threshold is defined at this step, setting the

PUF response size and the cipher key size. The design can then be instantiated by a system integrator, before being manufactured.

Afterwards, it is sent to a facility trusted by the original designer for enrolment. The reference PUF response is obtained and stored. PUF characterisation can be done at this stage for some of the devices to estimate the error rate. Then, the PUF response must be made inaccessible, typically by blowing a fuse inside the circuit. The device must later be activated.

The activation phase starts by challenging the PUF in the circuit to regenerate a response. The CASCADE key reconciliation protocol is then carried out to reconcile the PUF found in the circuit and on the server. The AW encrypted with this response is fed to the circuit to activate it. The circuit can then be used for its original purpose.

One aspect worth noting is the fact that, in this simplified design flow, testing of the chip is done after the activation. The implications of activating the device before or after test are discussed in [Yas+16b].

## 5.7 Conclusion

The implementation of the overall IP protection module is presented in this chapter. Implementation details are discussed as well as extra modules such as the AW decoder that are required. The results of implementation on Intel Cyclone V and Microsemi SmartFusion 2 are given, demonstrating that the hardware resources occupied are limited. We present the implementation of a demonstrator that illustrates the concepts discussed in this thesis. This demonstrator comprises a software interface and an hardware implementation done on the HECTOR board. For illustration purposes, a test design was modified following the proposed methodology. Finally, we present the typical use case for the overall scheme, showing how it can be integrated into the design flow. This demonstrates the practical usability and relevance of the IP protection scheme described in this thesis.



# Conclusion

Due to the ever-increasing complexity of integrated circuits, core-based design is now the main paradigm but comes with new threats for design data. Reported cases of illegal copying and counterfeiting have risen in recent years. The aim of this thesis was to propose an industrially relevant solution to actively prevent those illegal actions. The solution should provide hardware licensing capabilities, allowing for a remote and secure activation of the electronic system.

## Summary of contributions

The second and third chapter of this thesis propose to modify the combinational logic of a design to allow for IP protection. Combinational logic locking, presented in Chapter 2, proposes a new method to achieve logic locking at the combinational level. By detecting the sequences of logic gates that can propagate a locking value, it allows to controllably force the outputs of a design to a fixed logic value. AND or OR gates are inserted to controllably force these sequences of gates to the desired value. The algorithm that detects such sequences of gates, that leverages the representation of a netlist as a graph, is very efficient and can handle large netlists. So far, this is the only method for IP protection based on modifications of combinational logic that can deal with very large netlists of hundreds of thousands of logic gates. Moreover, we showed that the extra locking gates that must be inserted result in a low logic resources overhead of 2.9% on average. This is when all the outputs can be locked. The overhead brought by combinational logic locking can be reduced by locking only a subset of the outputs, or increased to make logic locking stronger.

In Chapter 3, another method for IP protection based on modifications of the combinational logic is studied. Logic masking, proposed in 2008 [RKM08a], consists in inserting XOR or XNOR gates at specific locations inside the netlist to controllably alter the internal state, disturbing the outputs. Current heuristics used to determine the place of insertion, however, could not handle large netlists while providing sufficiently low correlation at the outputs. We proposed to bridge the gap between computational complexity and masking efficiency by using centrality indicators. They allow to detect the most relevant nodes in a netlist, namely the ones through



which the information flow is the greatest. We give an overview of existing centrality indicators before showing that the ones based on current-flow can be efficiently used as the node selection heuristic for logic masking. Experimental results show that netlists of up to 30 000 nodes can be processed in around one hour, reducing the correlation at the outputs to low levels. This makes this selection heuristic the only one to be efficient and usable in a real-world context with medium-sized netlists. The designer can again pick the acceptable logic resources overhead for the target level of correlation.

The fourth chapter presents the CASCADE key reconciliation before showing how it can be successfully implemented alongside a PUF to correct the errors found at its output. Compared to existing error correcting codes, the device-side implementation can be an order of magnitude less costly in logic resources. This makes it very usable in a resource-constrained context, which is typically the case when a PUF is employed. Experimental results show that the protocol can accommodate the error rates observed for usual PUF architectures. The extensive simulation performed allowed us to provide several sets of parameters for error rates ranging from 1 to 15%, while maintaining very low failure rates down to  $10^{-8}$ . The protocol is very flexible, since the parameters can be changed at each execution. Therefore, the error correction can meet the required failure rate even if the error rate increases due to poor operating conditions. Again, this is up to the designer to choose the most suitable compromise.

All those propositions have in common to be very adaptable to the target application. For logic locking, the designer can balance the locking strength and the logic resources overhead. Similarly, for logic masking, the designer can balance the masking efficiency and the logic resources overhead. For the CASCADE protocol as well, the parameters can be easily tuned to deal with various error rates and target failure rates. These trade-offs allow the designer to balance the cost of implementation with the target security level, ensuring feasibility in an industrial context.

The final chapter of this thesis presents the integration of all the contributions of the SALWARE project in a complete IP protection module. Implementation results show that the scheme is suitable for industrial use, providing efficient protection of design data at reduced cost.

## Perspectives

Several perspectives can be identified that could extend the contributions of this thesis. For modifications of combinational logic targeting IP protection, the interaction between logic locking and logic masking could be studied. In particular, exploiting the sequences of nodes that propagate a locking value inside the netlist could be useful to interact with the masking gates. One could also design a two-step scheme that both locks and masks the outputs. Recovering the original behaviour would then require to deactivate both protections. Combining those two techniques would certainly lead to an efficient method to controllably disturb the outputs.

Another possibility to leverage logic locking is to adapt it to sequential systems. Locking the registers that store the current state of the system allows to force the system to a known, fixed state. Conversely, logic masking may not be used in this case since it could force the system into an unknown state.

Logic masking may be improved by analysing the design to protect before applying the node selection heuristic on it. It may be possible to determine a lower bound on the correlation that is achievable by applying logic masking on a particular design. Indeed, experimental results show that increasing the number of inserted masking gates does not necessarily reduce the bitwise output correlation. Analysing the netlist before modification could allow the designer to know the lowest level of correlation that is achievable and determine the associated logic resources overhead that would be optimal.

Regarding the CASCADE protocol, the sets of parameters that we give for different error rates and failure rates were obtained by simulation. However, those simulations took extensive time to perform, and were only done for the considered error rates and failure rates. A generic method to derive the parameters of the protocol given the error rate and failure rate could be developed. However, it should be specifically targeted at the application we consider here, namely correcting the errors in PUF responses. Indeed, the methods used in the context of quantum key distribution deal with very long bit frames, making them unsuitable for our use case. Specifically, some asymptotically valid approximations are not correct anymore, since PUF responses are much shorter. Integrated into the activation software, such method would allow the designer to enter the expected error rate and the required failure rate before executing the protocol.

Finally, the overall IP protection scheme should be evaluated as well. Even though the security and the leakage associated to the CASCADE protocol have been discussed, some weaknesses might be exploited. This would require further investigations, while keeping the same threat model as defined in Chapter 1. Considering other threat models could be interesting as well, while keeping in mind that the main objective of this work is industrial applicability.

In order to broaden the scope of this work, more fine-grained licensing could be investigated as well. Indeed, we only considered two modes of operation, activated or not. However, on a per design basis, some evaluation or premium modes are possible. For example, an Ethernet controller could be provided for evaluation with a throughput of 10Mbps, in a normal mode with a throughput of 100Mbps or in premium mode at 1Gbps. Similarly, a H.264 video decoder could decode in 720p in evaluation mode, in 1080p in normal mode and in 4K in premium mode. This type of feature-based licensing is very interesting from a marketing point of view, but is hard to make generic.

Some IP cores that are well suited for feature-based licensing are analog IP cores. Indeed, analog-to-digital converters or filters, for instance, must be calibrated to achieve the best performance. By acting on the calibration system, a wide range of performances can be obtained, paving the way for fine-grained performance-based licensing. The state-of-the-art in

IP protection for analog IP cores is scarce and specific protection schemes should be developed in the future.

Finally, one can also take the point of view of the system integrator, who wishes to integrate an IP core provided by an untrusted IP core designer. For example, an IP core designer could provide a cryptographic core with a hidden backdoor or with deliberately high side-channel leakage. Moreover, complex IP core like softcore microprocessors are meant to execute embedded code. How can a designer ensure that the IP core will remain harmless to the overall system if the code is malicious? How can a system integrator ensure that the IP cores integrated in the final system are connected and interacting with one another while being sufficiently isolated so that one malicious IP core cannot take down the whole system? These are important questions, that would require different threat models, and could also be studied in future works.

# Conclusion

Du fait de la complexité croissante des circuits intégrés, la conception modulaire est à présent le paradigme de conception dominant, mais est associé à de nouvelles menaces pour les données de conception. Les cas de copie illégale et de contrefaçon signalés ont considérablement augmentés ces dernières années. L’objectif de cette thèse était de proposer une solution applicable dans un contexte industriel afin d’empêcher ces actes illégaux. La solution proposée doit mettre en place un système de licence matérielle, permettant l’activation sécurisée et à distance du système électronique.

## Résumé des contributions

Les deuxième et troisième chapitres de cette thèse proposent de modifier la logique combinatoire d’un composant virtuel pour permettre la protection des données de conception. Le verrouillage combinatoire de la logique, présenté dans le chapitre 2, propose une nouvelle méthode pour permettre le verrouillage logique au niveau de la logique combinatoire. En identifiant des suites de portes logiques qui peuvent propager une valeur de verrouillage, cette méthode permet de forcer les sorties d’un composant virtuel à une valeur fixe. Des portes logiques ET ou OU sont insérées afin de pouvoir forcer ces suites de portes logiques à la valeur souhaitée. L’algorithme qui détecte ces suites de portes logiques, qui exploite la représentation d’une netlist sous forme de graphe, est très efficace et peut gérer des netlists de grande taille. Actuellement, c’est la seule méthode visant à protéger les données de conception basée sur une modification de la logique combinatoire qui puisse gérer des netlists de très grande taille, de l’ordre d’une centaine de milliers de portes logiques. De plus, nous avons montré que les portes logiques supplémentaires à insérer n’entraînent un surcoût que de 2,9% en moyenne, et ce dans le cas où toutes les sorties peuvent être verrouillées. Le coût en ressources logiques induit par le verrouillage combinatoire de la logique peut être réduit en ne verrouillant qu’une partie des sorties, ou augmenté pour renforcer le verrouillage.

Dans le chapitre 3, une autre méthode basée sur une modification de la logique combinatoire permettant la protection des données de conception est étudiée. Le masquage logique, proposé en 2008 [RKM08a], consiste à insérer des portes OU exclusif ou NON-OU exclusif à

des endroits spécifiques dans une netlist afin de pouvoir altérer son état interne de manière contrôlée, perturbant ainsi les sorties. Néanmoins, les heuristiques utilisées actuellement pour déterminer le lieu d'insertion ne permettaient pas de gérer des netlists de grande taille tout en obtenant une corrélation suffisamment basse aux sorties. Nous proposons de combler ce manque entre complexité algorithmique et efficacité de masquage en utilisant les indicateurs de centralité. Ces derniers permettent d'identifier les nœuds les plus importants d'une netlist, c'est à dire ceux à travers lesquels le flux d'information est le plus important. Nous donnons un aperçu des indicateurs de centralité existants avant de montrer que ceux basés sur le courant électrique peuvent être utilisés de manière efficace comme heuristique de sélection pour les nœuds à modifier par masquage logique. Les résultats expérimentaux montrent que des netlists contenant jusqu'à 30 000 nœuds peuvent être analysées en environ une heure, tout en réduisant la corrélation en sortie à des niveaux bas. Cela fait de cette heuristique de sélection la seule efficace et utilisable dans un contexte concret de protection de netlists de taille moyenne. Encore une fois, le concepteur peut choisir le surcoût en ressources logiques jugé acceptable pour le niveau de corrélation en sortie souhaité.

Le quatrième chapitre présente le protocole de réconciliation de clés CASCADE avant de montrer comment ce dernier peut être utilisé en présence d'une PUF pour corriger les erreurs observées à sa sortie. Comparée aux codes correcteurs d'erreurs existants, l'implantation coté circuit peut être plus légère d'un ordre de grandeur en terme de ressources logiques. Cela le rend particulièrement utilisable dans un contexte où les ressources disponibles sont limitées, ce qui est typiquement le cas lorsqu'une PUF est utilisée. Les résultats expérimentaux montrent que le protocole peut gérer les taux d'erreur observés avec les architectures de PUF courantes. Les simulations poussées que nous avons menées nous ont permis de fournir plusieurs jeux de paramètres pour des taux d'erreurs allant de 1 à 15%, tout en maintenant des taux d'échecs très bas jusqu'à  $10^{-8}$ . Le protocole est très adaptable, puisque les paramètres peuvent être modifiés à chaque exécution. Ainsi, la correction des erreurs peut atteindre des taux d'échecs très bas même si le taux d'erreur augmente à cause de conditions de fonctionnement mauvaises. Encore une fois, c'est au concepteur de choisir le meilleur compromis.

Toutes ces propositions ont en commun d'être très facilement adaptables à l'application ciblée. Pour le verrouillage combinatoire de la logique, le concepteur peut équilibrer la force du verrouillage et le coût en ressources logiques. De même, pour le masquage logique, le concepteur peut équilibrer l'efficacité de masquage et le coût en ressources logiques. Pour le protocole CASCADE, les paramètres peuvent également être facilement ajustés pour gérer différents taux d'erreur et taux d'échec. Ces compromis permettent d'équilibrer le coût l'implantation et le niveau de sécurité souhaité, assurant la faisabilité dans un contexte industriel.

Le chapitre final de cette thèse présente l'intégration de toutes les contributions du projet SALWARE dans un module complet de protection des données de conception. Les résultats l'implantation montrent que le système est adéquat pour une utilisation industrielle, fournissant une protection efficace des données de conception à un coût réduit.

## Perspectives

Plusieurs perspectives peuvent être envisagées pour étendre les contributions de cette thèse. Concernant les modifications de la logique visant à protéger les données de conception, l'interaction entre le verrouillage et le masquage logiques pourrait être étudié. En particulier, exploiter les suites de nœuds qui propagent une valeur de verrouillage à l'intérieur de la netlist pourrait être utile pour interagir avec les portes logiques de masquage. Un système en deux étapes qui assure à la fois le verrouillage et le masquage des sorties pourrait également être conçu. Combiner ces deux techniques résulterait sûrement en une méthode efficace pour altérer les sorties.

Une autre possibilité pour mettre à profit le verrouillage logique est de l'appliquer aux systèmes séquentiels. Verrouiller les registres qui stockent l'état courant du système permet de forcer le système dans un état fixe connu. À l'inverse, le masquage logique ne pourrait sûrement pas être utilisé dans ce cas car le système serait alors placé dans un état inconnu.

Le masquage logique pourrait être amélioré en analysant le design à protéger avant d'y appliquer l'heuristique de sélection des nœuds. Il serait peut-être possible d'identifier une borne inférieure pour le niveau de corrélation en sortie atteignable en appliquant la méthode de masquage logique à un design spécifique. En effet, les résultats expérimentaux montrent qu'augmenter le nombre de portes de masquage logique insérées ne réduit pas nécessairement le niveau de corrélation des sorties. Analyser la netlist avant modification pourrait permettre au concepteur de connaître le niveau minimal de corrélation atteignable et de déterminer le coût en ressources logiques associé, qui serait optimal.

En ce qui concerne le protocole CASCADE, les jeux de paramètres que nous donnons pour différents taux d'erreur et d'échec ont été obtenus par simulation. Néanmoins, réaliser ces simulations a pris beaucoup de temps, et ces dernières n'ont été faites que pour les taux d'erreur et d'échec considérés. Une méthode générique pour déduire les paramètres du protocole à partir des taux d'erreur et d'échec pourrait être mise au point. Néanmoins, elle devrait cibler particulièrement l'application que nous considérons ici, à savoir la correction des erreurs dans les réponses des PUFs. En effet, les méthodes utilisées dans le contexte de distribution quantique de clés utilisent des messages de très grande taille, ce qui les rend inapplicables dans notre cas. En particulier, des approximations valables asymptotiquement ne le sont plus, puisque les réponses des PUFs sont beaucoup plus courtes. Intégrée dans le logiciel d'activation, une telle méthode permettrait au concepteur d'entrer seulement le taux d'erreur attendu et le taux d'échec requis avant d'exécuter le protocole.

Enfin, le module complet de protection des données de conception devra être évalué. Même si la sécurité et la fuite d'information associées au protocole CASCADE ont été discutées, des faiblesses pourraient être exploitées. Cela nécessite une étude plus approfondie, tout en gardant un modèle de menace identique à celui défini au chapitre 1. Considérer d'autres modèles de menace pourrait également être intéressant, tout en gardant à l'esprit que l'objectif principal

de ces travaux est l'applicabilité industrielle.

Afin d'étendre la portée de ces travaux, un système de licence plus fin pourrait également être exploré. En effet, nous n'avons envisagé que deux modes de fonctionnement, activé ou non. En revanche, au cas par cas pour chaque design, des modes d'évaluation ou *premium* sont envisageables. Par exemple, un contrôleur Ethernet pourrait être proposé avec un débit de 10Mbps en mode évaluation, en mode normal avec un débit de 100Mbps ou en mode *premium* à 1Gbps. De la même façon, un décodeur vidéo H.264 pourrait décoder en 720p en mode évaluation, en 1080p en mode normal et en 4K en mode *premium*. Ce type de licence basé sur les fonctionnalités est très intéressant d'un point de vue commercial, mais est difficile à définir de manière générique.

Certains composants virtuels particulièrement adaptés à ce type de licence basé sur les fonctionnalités sont les composants virtuels analogiques. En effet, les convertisseurs analogique-numérique ou les filtres, par exemple, doivent être calibrés pour atteindre les meilleures performances. En agissant sur le système de calibration, une large gamme de performances peut être obtenue, jetant les bases d'un système de licence basé sur les performances. Les méthodes de protection des données de conception adaptées aux composants virtuels analogiques sont rares dans la littérature et des techniques de protection spécifiques pourront être mises au point à l'avenir.

Enfin, il est également possible de se placer du point de vue de l'intégrateur système, qui souhaite utiliser un composant virtuel fourni par un concepteur de composants virtuels qui n'est pas approuvé. Par exemple, un concepteur pourrait fournir un module cryptographique avec une *backdoor* cachée ou avec une fuite sur le canal auxiliaire délibérément élevée. De plus, des composants virtuels complexes tels que les processeurs doivent exécuter du code embarqué. Comment un concepteur peut-il s'assurer que le composant virtuel demeurera inoffensif vis à vis du système complet si le code exécuté est malveillant ? Comment un concepteur peut-il s'assurer que les composants virtuels intégrés dans le système final sont connectés et interagissent les uns avec les autres tout en étant suffisamment isolés de manière à ce que le fonctionnement du système complet ne puisse pas être compromis par un composant virtuel malveillant ? Toutes ces questions sont importantes, requièrent des modèles de menace différents et pourrait être étudiées à l'avenir.

# Publications and communications

## Peer-reviewed journals

- [Col+17a] Brice Colombier, Lilian Bossuet, David Hély and Viktor Fischer, “Key Reconciliation Protocols for Error Correction of Silicon PUF Responses”, *IEEE Transactions on Information Forensics and Security* 12.8 (Aug. 2017), pp. 1988–2002.
- [BC16] Lilian Bossuet and Brice Colombier, “Comments on ‘A PUF-FSM Binding Scheme for FPGA IP Protection and Pay-per-Device Licensing’”, *IEEE Transactions on Information Forensics and Security* 11.11 (Nov. 2016), pp. 2624–2625.
- [CBH16a] Brice Colombier, Lilian Bossuet and David Hély, “From Secured Logic to IP Protection”, *Elsevier Microprocessors and Microsystems* 47 (Nov. 2016), pp. 44–54 (cited on pp. 42, 46).
- [CB14] Brice Colombier and Lilian Bossuet, “Survey of Hardware Protection of Design Data for Integrated Circuits and Intellectual Properties”, *IET Computers & Digital Techniques* 8.6 (Nov. 2014), pp. 274–287.

## International peer-reviewed conferences with proceedings

- [CBH17a] Brice Colombier, Lilian Bossuet and David Hély, “Centrality Indicators For Efficient And Scalable Logic Masking”, *IEEE Computer Society Annual Symposium on VLSI*, Bochum, Germany, July 2017.
- [CBH15a] Brice Colombier, Lilian Bossuet and David Hély, “Reversible Denial-of-Service by Locking Gates Insertion for IP Cores Design Protection”, *IEEE Computer Society Annual Symposium on VLSI*, Montpellier, France, July 2015, pp. 210–215.



## Book chapters

- [CBH17c] Brice Colombier, Lilian Bossuet and David Hély, “Logic Modification-Based IP Protection Methods: An Overview and a Proposal”, *Foundations of Hardware IP Protection*, 2017, pp. 37–64.
- [CBH17d] Brice Colombier, Lilian Bossuet and David Hély, “Turning Electronic Circuits Features into On-Chip Locks”, *Foundations of Hardware IP Protection*, 2017, pp. 15–36.

## Workshops without proceedings

- [CBH17b] Brice Colombier, Lilian Bossuet and David Hély, “Centrality Indicators For Efficient And Scalable Logic Masking”, *Cryptarchi Workshop*, Smolenice, Slovakia, June 2017.
- [CBH16b] Brice Colombier, Lilian Bossuet and David Hély, “Key reconciliation protocol application to error correction in silicon PUF responses”, *TRUDEVICE Workshop, Design, Automation & Test in Europe Conference*, Dresden, Germany, Mar. 2016.
- [CBH16c] Brice Colombier, Lilian Bossuet and David Hély, “Key reconciliation protocol application to error correction in silicon PUF responses”, *Cryptarchi Workshop*, La Grande Motte, France, June 2016.
- [CBH15b] Brice Colombier, Lilian Bossuet and David Hély, “Reversible Denial-of-Service by Locking Gates Insertion for IP Cores Design Protection”, *Cryptarchi Workshop*, Leuven, Belgium, June 2015.

## Demonstrations

- [Col+17b] Brice Colombier, Ugo Mureddu, Marek Laban, Oto Petura, Lilian Bossuet and Viktor Fischer, “Hardware Demo: Complete Activation Scheme for IP Design Protection”, *International Symposium on Hardware Oriented Security and Trust*, McLean, VA, USA, May 2017.
- [Col+17c] Brice Colombier, Ugo Mureddu, Marek Laban, Oto Petura, Lilian Bossuet and Viktor Fischer, “Hardware Demo: Complete Activation Scheme for IP Design Protection”, *International Conference on Field-Programmable Logic and Applications*, Ghent, Belgium, Sept. 2017.

## Seminar

- [CBH16d] Brice Colombier, Lilian Bossuet and David Hély, “Key reconciliation protocol application to error correction in silicon PUF responses”, *Journé Sécurité Numérique du GDR SoC-SiP : 11ème édition, La génération d’aléa dans le matériel : TRNG & PUF*, Paris, France, May 2016.

## Posters

- [CBH16e] Brice Colombier, Lilian Bossuet and David Hély, “Key reconciliation protocol application to error correction in silicon PUF responses”, *Colloque national du GDR SoC/SiP*, Nantes, France, June 2016.
- [CBH16f] Brice Colombier, Lilian Bossuet and David Hély, “Secure remote activation scheme for integrated circuits”, *Journée de la recherche de l’École doctorale EDSIS*, Saint-Étienne, France, June 2016.
- [CB15a] Brice Colombier and Lilian Bossuet, “Functional Locking Modules for Design Protection of Intellectual Property Cores”, *TRUDEVICE Workshop, Design, Automation & Test in Europe Conference*, Grenoble, France, Mar. 2015.
- [CB15b] Brice Colombier and Lilian Bossuet, “Functional Locking Modules for Design Protection of Intellectual Property Cores”, *IEEE International Symposium on Field-Programmable Custom Computing Machines*, Vancouver, Canada, May 2015, p. 233.
- [CBH15c] Brice Colombier, Lilian Bossuet and David Hély, “Système sécurisé d’activation à distance de circuits intégrés et de composants virtuels”, *Journée scientifique de l’ARC6*, Grenoble, France, Nov. 2015.

## Popular science communications

- Science & You, mai-juin 2015, Université de Lorraine, Nancy.
- Fête de la Science, octobre 2016, Université Jean Monnet, Saint-Étienne.
- Ramène ta science, mai 2017, Université Jean Monnet, Saint-Étienne.



# Bibliography

- [ATA04] Amr T. Abdel-Hamid, Sofiène Tahar and El Mostapha Aboulhamid, “A survey on IP watermarking techniques”, *Design Automation for Embedded Systems* 9.3 (2004), pp. 211–227 (cited on p. 27).
- [AK07] Y. Alkabani and F. Koushanfar, “Active hardware metering for intellectual property protection and security”, *USENIX Security*, Boston MA, USA, Aug. 2007, pp. 291–306 (cited on pp. 49, 50).
- [Alt09] Altera, *Protecting the FPGA Design From Common Threats*, 2009, URL: [https://www.altera.com/content/dam/altera-www/global/en\\_US/pdfs/literature/wp/wp-01111-anti-tamper.pdf](https://www.altera.com/content/dam/altera-www/global/en_US/pdfs/literature/wp/wp-01111-anti-tamper.pdf) (cited on p. 45).
- [AGM15] Luca Amarú, Pierre-Emmanuel Gaillardon and Giovanni De Micheli, “The EPFL Combinational Benchmark Suite”, *International Workshop on Logic & Synthesis*, Mountain View, CA, USA, June 2015 (cited on p. 97).
- [Ant71] Jac M. Anthonisse, “The rush in a directed graph”, *Mathematische Besliskunde BN* 9/71 (1971), pp. 1–10 (cited on p. 92).
- [ARM17] ARM, *AMBA Specifications*, 2017, URL: <http://www.arm.com/products/system-ip/amba-specifications> (cited on p. 51).
- [Ays+15] Aydin Aysu, Ege Gulcan, Daisuke Moriyama, Patrick Schaumont and Moti Yung, “End-To-End Design of a PUF-Based Privacy Preserving Authentication Protocol”, *International Workshop on Cryptographic Hardware and Embedded Systems*, Saint-Malo, France, Sept. 2015 (cited on p. 121).
- [BM06] David A. Bader and Kamesh Madduri, “Parallel Algorithms for Evaluating Centrality Indices in Real-world Networks”, *International Conference on Parallel Processing*, Columbus, Ohio, USA, Aug. 2006, pp. 539–550 (cited on p. 101).
- [BFS16] Chongxi Bao, Domenic Forte and Ankur Srivastava, “On Reverse Engineering-Based Hardware Trojan Detection”, *IEEE Trans. on CAD of Integrated Circuits and Systems* 35.1 (2016), pp. 49–57 (cited on p. 20).
- [BCM16] Mario Barbareschi, Alessandro Cilardo and Antonino Mazzeo, “Partial FPGA bitstream encryption enabling hardware DRM in mobile environments”, *ACM International Conference on Computing Frontiers*, Como, Italy, May 2016, pp. 443–448 (cited on p. 45).
- [BZB14] Abhishek Basak, Yu Zheng and Swarup Bhunia, “Active defense against counterfeiting attacks through robust antifuse-based on-chip locks”, *IEEE 32<sup>nd</sup> VLSI Test Symposium*, Napa CA, USA, Apr. 2014, pp. 1–6 (cited on pp. 50, 55, 56).

- [BTZ10] A. Baumgarten, A. Tyagi and J. Zambreno, “Preventing IC Piracy Using Reconfigurable Logic Barriers”, *IEEE Design & Test of Computers* 27.1 (2010), pp. 66–75 (cited on p. 41).
- [Bay+12] Pierre Bayon, Lilian Bossuet, Alain Aubert, Viktor Fischer, François Poucheret, Bruno Robisson and Philippe Maurine, “Contactless Electromagnetic Active Attack on Ring Oscillator Based True Random Number Generator”, *International Workshop on Constructive Side-Channel Analysis and Secure Design*, vol. 7275, Darmstadt, Germany, May 2012, pp. 151–166 (cited on p. 35).
- [Bec15] Georg T. Becker, “The Gap Between Promise and Reality: On the Insecurity of XOR Arbiter PUFs”, *International Workshop on Cryptographic Hardware and Embedded Systems*, vol. 9293, Saint-Malo, France, Sept. 2015, pp. 535–555 (cited on p. 32).
- [BBR88] Charles H. Bennett, Gilles Brassard and Jean-Marc Robert, “Privacy Amplification by Public Discussion”, *SIAM Journal on Computing* 17.2 (1988), pp. 210–229 (cited on p. 135).
- [BSH12] Florian Benz, André Seffrin and Sorin A. Huss, “Bil: A tool-chain for bitstream reverse-engineering”, *International Conference on Field Programmable Logic and Applications*, Oslo, Norway, Aug. 2012, pp. 735–738 (cited on pp. 19, 45).
- [Ber+16] Thierry P. Berger, Julien Francq, Marine Minier and Gaël Thomas, “Extended Generalized Feistel Networks Using Matrix Representation to Propose a New Lightweight Block Cipher: Lilliput”, *IEEE Trans. Computers* 65.7 (2016), pp. 2074–2089 (cited on p. 141).
- [BMT78] Elwyn R. Berlekamp, Robert J. McEliece and Henk C. A. van Tilborg, “On the inherent intractability of certain coding problems”, *IEEE Transactions on Information Theory* 24.3 (1978), pp. 384–386 (cited on p. 134).
- [Boc+10] Nathalie Bochar, Florent Bernard, Viktor Fischer and Boyan Valtchanov, “True-Randomness and Pseudo-Randomness in Ring Oscillator-Based True Random Number Generators”, *Int. J. Reconfig. Comp.* 2010 (2010), pp. 1–13 (cited on pp. 35, 37).
- [Bog+11] Andrey Bogdanov, Miroslav Knezevic, Gregor Leander, Deniz Toz, Kerem Varici and Ingrid Verbauwhede, “SPONGENT: A Lightweight Hash Function”, *International Workshop on Cryptographic Hardware and Embedded Systems*, Nara, Japan, Sept. 2011, pp. 312–325 (cited on p. 135).
- [Boo54] George Boole, *An Investigation of the Laws of Thought: On which are Founded the Mathematical Theories of Logic and Probabilities*, 1854 (cited on p. 62).
- [Bös+08] Christoph Bösch, Jorge Guajardo, Ahmad-Reza Sadeghi, Jamshid Shokrollahi and Pim Tuyls, “Efficient Helper Data Key Extractor on FPGAs”, *International Workshop on Cryptographic Hardware and Embedded Systems*, Washington, D.C., USA, Aug. 2008, pp. 181–197 (cited on pp. 38, 39, 131).
- [BBF15] Lilian Bossuet, Pierre Bayon and Viktor Fischer, “An Ultra-Lightweight Transmitter for Contactless Rapid Identification of Embedded IP in FPGA”, *Embedded Systems Letters* 7.4 (2015), pp. 97–100 (cited on pp. 28, 146).
- [Bos+14] Lilian Bossuet, Xuan Thuy Ngo, Zouha Cherif and Viktor Fischer, “A PUF based on transient effect ring oscillator and insensitive to locking phenomenon”, *IEEE Transaction on Emerging Topics in Computing* 2.1 (2014), pp. 30–36 (cited on pp. 35, 121).

- [Bra01] Ulrik Brandes, “A faster algorithm for betweenness centrality”, *Journal of mathematical sociology* 25.2 (2001), pp. 163–177 (cited on pp. 96, 97).
- [BE05] Ulrik Brandes and Thomas Erlebach, eds., *Network Analysis: Methodological Foundations*, vol. 3418, Lecture Notes in Computer Science, Springer, 2005 (cited on pp. 96, 97, 104).
- [BF05] Ulrik Brandes and Daniel Fleischer, “Centrality Measures Based on Current Flow”, *Annual Symposium on Theoretical Aspects of Computer Science*, vol. 3404, Stuttgart, Germany, Feb. 2005, pp. 533–544 (cited on pp. 93–97).
- [BS93] Gilles Brassard and Louis Salvail, “Secret-Key Reconciliation by Public Discussion”, *EUROCRYPT*, Lofthus, Norway, May 1993, pp. 410–423 (cited on pp. 111, 112, 119).
- [Bru+09] M. Brutscheck, M. Franke, A. Th. Schwarzbacher and St. Becker, “Non-Invasive Reverse Engineering of CMOS Integrated Circuits”, *IEEE 17<sup>th</sup> Telecommunications Forum TELFOR*, Belgrade, Serbia, Nov. 2009 (cited on p. 19).
- [BY07] Maciej Brzozowski and Vyacheslav N. Yarmolik, “Obfuscation as Intellectual Rights Protection in VHDL Language”, *International Conference on Computer Information Systems and Industrial Management Applications*, Elk, Poland, June 2007, pp. 337–340 (cited on p. 44).
- [CDK09] Christophe De Cannière, Orr Dunkelman and Miroslav Knezevic, “KATAN and KTANTAN - A Family of Small and Efficient Hardware-Oriented Block Ciphers”, *International Workshop on Cryptographic Hardware and Embedded Systems*, Lausanne, Switzerland, Sept. 2009, pp. 272–288 (cited on p. 141).
- [CB09] Rajat Subhra Chakraborty and Swarup Bhunia, “HARPOON: An Obfuscation-Based SoC Design Methodology for Hardware Protection”, *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 28.10 (2009), pp. 1493–1502 (cited on pp. 48–50, 55, 56, 98).
- [CD00] Roy Chapman and Tariq S Durrani, “IP protection of DSP algorithms for system on chip implementation”, *IEEE Transactions on Signal Processing*. 48.3 (2000), pp. 854–861 (cited on p. 29).
- [Che+13] Zouha Cherif, Jean-Luc Danger, Florent Lozach, Yves Mathieu and Lilian Bossuet, “Evaluation of delay PUFs on CMOS 65 nm technology: ASIC vs FPGA”, *The Second Workshop on Hardware and Architectural Support for Security and Privacy*, ed. by Ruby B. Lee and Weidong Shi, Tel-Aviv, Israel: ACM, June 2013, p. 4 (cited on p. 34).
- [CBM16] Abdelkarim Cherkaoui, Lilian Bossuet and Cédric Marchand, “Design, Evaluation and Optimization of Physical Unclonable Functions based on Transient Effect Ring Oscillators”, *IEEE Transactions on Information Forensics and Security* 11.6 (2016), pp. 1291–1305 (cited on pp. 36, 121).
- [CLB11] Mathias Claes, Vincent van der Leest and An Braeken, “Comparison of SRAM and FF-PUF in 65nm Technology”, *Nordic Conference on Secure IT Systems*, vol. 7161, Tallinn, Estonia, Oct. 2011, pp. 47–64 (cited on pp. 36, 121).
- [Coc+14] Ronald P. Cocchi, James P. Baukus, Lap Wai Chow and Bryan J. Wang, “Circuit Camouflage Integration for Hardware IP Protection”, *Annual Design Automation Conference*, San Francisco, CA, USA, June 2014, 153:1–153:5 (cited on pp. 41, 44).

- [CBH17c] Brice Colombier, Lilian Bossuet and David Hély, “Logic Modification-Based IP Protection Methods: An Overview and a Proposal”, *Foundations of Hardware IP Protection*, 2017, pp. 37–64.
- [CBH17d] Brice Colombier, Lilian Bossuet and David Hély, “Turning Electronic Circuits Features into On-Chip Locks”, *Foundations of Hardware IP Protection*, 2017, pp. 15–36.
- [Col+17b] Brice Colombier, Ugo Mureddu, Marek Laban, Oto Petura, Lilian Bossuet and Viktor Fischer, “Hardware Demo: Complete Activation Scheme for IP Design Protection”, *International Symposium on Hardware Oriented Security and Trust*, McLean, VA, USA, May 2017.
- [Col+17c] Brice Colombier, Ugo Mureddu, Marek Laban, Oto Petura, Lilian Bossuet and Viktor Fischer, “Hardware Demo: Complete Activation Scheme for IP Design Protection”, *International Conference on Field-Programmable Logic and Applications*, Ghent, Belgium, Sept. 2017.
- [CK06] N. Couture and K. B. Kent, “Periodic Licensing of FPGA Based Intellectual Property”, *IEEE International Conference on Field Programmable Technology*, Bangkok, Thailand, Dec. 2006, pp. 357–360 (cited on p. 51).
- [CN06] Gabor Csardi and Tamas Nepusz, “The igraph software package for complex network research”, *InterJournal Complex Systems* 1695.5 (2006), pp. 1–9 (cited on pp. 71, 96).
- [Cui+11] Aijiao Cui, Chip-Hong Chang, Sofiéne Tahar and Amr T. Abdel-Hamid, “A Robust FSM Watermarking Scheme for IP Protection of Sequential Circuit Design”, *IEEE Transactions on CAD of Integrated Circuits and Systems* 30.5 (2011), pp. 678–690 (cited on p. 29).
- [CQZ15] Aijiao Cui, Gang Qu and Yan Zhang, “Ultra-Low Overhead Dynamic Watermarking on Scan Design for Hard IP Protection”, *IEEE Transactions on Information Forensics and Security* 10.11 (2015), pp. 2298–2313 (cited on p. 28).
- [Dav99] Scott Davidson, “ITC’99 Benchmark Circuits - Preliminary Results”, *IEEE International Test Conference*, Atlantic City, NJ, USA, Sept. 1999, p. 1125 (cited on pp. 71, 97).
- [Del+15] Jeroen Delvaux, Dawu Gu, Dries Schellekens and Ingrid Verbauwhede, “Helper Data Algorithms for PUF-Based Key Generation: Overview and Analysis”, *IEEE Transactions on CAD of Integrated Circuits and Systems* 34.6 (2015), pp. 889–902 (cited on p. 38).
- [Dod+08] Yevgeniy Dodis, Rafail Ostrovsky, Leonid Reyzin and Adam D. Smith, “Fuzzy Extractors: How to Generate Strong Keys from Biometrics and Other Noisy Data”, *SIAM J. Comput.* 38.1 (2008), pp. 97–139 (cited on p. 38).
- [Dru69] Peter F. Drucker, *The age of discontinuity: Guidelines to our changing society*, 1969 (cited on pp. 2, 9).
- [EW01] David Eppstein and Joseph Wang, “Fast approximation of centrality”, *Symposium on Discrete Algorithms*, Washington, DC, USA., Jan. 2001, pp. 228–229 (cited on pp. 92, 96).
- [Fab06] Fabless Semiconductor Association, *Understanding the Semiconductor Intellectual Property (SIP) Business Process*, tech. rep., 2006 (cited on p. 17).



- [FT03] Yu-Cheng Fan and Hen-Wai Tsao, “Watermarking for intellectual property protection”, *IET Electronics Letters* 39.18 (2003), pp. 1316–1318 (cited on p. 28).
- [Fre77] Linton C. Freeman, “A Set of Measures of Centrality Based on Betweenness”, *Sociometry* 40.1 (1977), pp. 35–41 (cited on p. 92).
- [Fro11] Frontier-Economics, *Estimating the global economic and social impacts of counterfeiting and piracy*, tech. rep., Business Action to Stop Counterfeiting and Piracy (BASCAP), 2011 (cited on p. 18).
- [Gas+12] L. Gaspar, V. Fischer, T. Guneyusu and Z. C. Jouini, “Two IP Protection Schemes for Multi-FPGA Systems”, *International Conference on Reconfigurable Computing and FPGAs*, Cancun, Mexico, Dec. 2012, pp. 1–6 (cited on p. 45).
- [Gas+02] Blaise Gassend, Dwaine E. Clarke, Marten van Dijk and Srinivas Devadas, “Silicon physical random functions”, *ACM Conference on Computer and Communications Security*, Washington, DS, USA, Nov. 2002, pp. 148–160 (cited on p. 33).
- [Gol79] Lawrence H. Goldstein, “Controllability/Observability analysis of digital circuits”, *IEEE Transactions on Circuits and Systems* 26.9 (Sept. 1979), pp. 685–693 (cited on p. 87).
- [GNL11] Zheng Gong, Svetla Nikova and Yee Wei Law, “KLEIN: A New Family of Lightweight Block Ciphers”, *International Workshop on RFID Security and Privacy*, Amherst, USA, June 2011, pp. 1–18 (cited on p. 141).
- [GGY15] Sezer Gören, Cemil Cem Gürsoy and Abdullah Yildiz, “Speeding Up Logic Locking via Fault Emulation and Dynamic Multiple Fault Injection”, *Journal of Electronic Testing* 31.5-6 (2015), pp. 525–536 (cited on pp. 49, 72).
- [Gua+09] J. Guajardo, T. Guneyusu, S. S. Kumar and C. Paar, “Secure IP-Block Distribution for Hardware Devices”, *IEEE International Workshop on Hardware-Oriented Security and Trust*, San Francisco CA, USA, July 2009, pp. 82–89 (cited on pp. 52, 53).
- [Gua+07] Jorge Guajardo, Sandeep S Kumar, Geert-Jan Schrijen and Pim Tuyls, “FPGA intrinsic PUFs and their use for IP protection”, *International Workshop on Cryptographic Hardware and Embedded Systems*, Vienna, Austria, Sept. 2007, pp. 63–80 (cited on p. 121).
- [GDT14] Ujjwal Guin, Daniel DiMase and Mohammad Tehranipoor, “Counterfeit integrated circuits: Detection, avoidance, and the challenges ahead”, *Journal of Electronic Testing* 30.1 (2014), pp. 9–23 (cited on pp. 18, 20).
- [GMP07] Tim Guneyusu, Bodo Moller and Christof Paar, “Dynamic intellectual property protection for reconfigurable devices”, *International Conference on Field-Programmable Technology*, Kitakyushu, Japan, Dec. 2007, pp. 169–176 (cited on p. 54).
- [Guo+11] Jian Guo, Thomas Peyrin, Axel Poschmann and Matthew J. B. Robshaw, “The LED Block Cipher”, *International Workshop on Cryptographic Hardware and Embedded Systems*, Nara, Japan, Sept. 2011, pp. 326–341 (cited on p. 141).
- [GZ97] Rajesh K. Gupta and Yervant Zorian, “Introducing Core-Based System Design”, *IEEE Design & Test of Computers* 14.4 (1997), pp. 15–25 (cited on pp. 2, 8).
- [Hac03] Gaël Hachez, “A Comparative Study of Software Protection Tools Suited for E-Commerce with Contributions to Software Watermarking and Smart Cards”, PhD thesis, Université Catholique de Louvain, Mar. 2003 (cited on pp. 41, 46).



- [Her+12] Anthony Van Herrewege, Stefan Katzenbeisser, Roel Maes, Roel Peeters, Ahmad-Reza Sadeghi, Ingrid Verbauwhede and Christian Wachsmann, “Reverse Fuzzy Extractors: Enabling Lightweight Mutual Authentication for PUF-Enabled RFIDs”, *International Conference on Financial Cryptography and Data Security*, Kralendijk, Bonaire, Feb. 2012, pp. 374–389 (cited on pp. 38, 39, 111).
- [Hil+13] Benjamin Hill, Robert Karmazin, Carlos Tadeo Ortega Otero, Jonathan Tse and Rajit Manohar, “A split-foundry asynchronous FPGA”, *CICC*, 2013, pp. 1–4 (cited on p. 45).
- [Hil+15] Matthias Hiller, Ludwig Kurzinger, Georg Sigl, Sven Muelich, Sven Puchinger and Martin Bossert, “Low-Area Reed Decoding in a Generalized Concatenated Code Construction for PUFs”, *IEEE Computer Society Annual Symposium on VLSI*, Montpellier, France, July 2015, pp. 143–148 (cited on pp. 38, 39, 131).
- [Hil+12] Matthias Hiller, Dominik Merli, Frederic Stumpf and Georg Sigl, “Complementary IBS: Application specific error correction for PUFs”, *IEEE International Symposium on Hardware-Oriented Security and Trust*, San Francisco, CA, USA, June 2012, pp. 1–6 (cited on pp. 38, 39, 126).
- [HYP15] Matthias Hiller, Meng-Day Yu and Michael Pehl, “Systematic Low Leakage Coding for Physical Unclonable Functions”, *ACM Symposium on Information, Computer and Communications Security*, Singapore, Apr. 2015, pp. 155–166 (cited on p. 38).
- [HYS16] Matthias Hiller, Meng-Day Yu and Georg Sigl, “Cherry-Picking Reliable PUF Bits With Differential Sequence Coding”, *IEEE Trans. Information Forensics and Security* 11.9 (2016), pp. 2065–2076 (cited on pp. 38, 39, 126, 131).
- [Hod11] David A. Hodges, “Building the Fabless/Foundry Business Model”, *IEEE Solid-State Circuits Magazine* 3.4 (2011), pp. 7–44 (cited on pp. 2, 8).
- [HL08] J. Huang and J. Lach, “IC Activation and User Authentication for Security-Sensitive Systems”, *IEEE International Workshop on Hardware-Oriented Security and Trust*, Anaheim CA, USA, June 2008, pp. 76–80 (cited on pp. 52, 55, 56).
- [Huf+08] Ted Huffmire, Jonathan Valamehr, Timothy Sherwood, Ryan Kastner, Timothy Levin, Thuy D Nguyen and Cynthia Irvine, “Trustworthy system security through 3-D integrated hardware”, *IEEE International Workshop on Hardware-Oriented Security and Trust*, Anaheim CA, USA, June 2008, pp. 91–92 (cited on p. 40).
- [Ime+13] Frank Imeson, Ariq Emtenan, Siddharth Garg and Mahesh V Tripunitara, “Securing Computer Hardware Using 3D Integrated Circuit (IC) Technology and Split Manufacturing for Obfuscation”, *USENIX Security Symposium*, Washington DC, USA, Aug. 2013, pp. 495–510 (cited on p. 40).
- [Int17] Intel, *Avalon® Interface Specifications*, tech. rep., 2017 (cited on p. 51).
- [Jai+03] Adarsh K Jain, Lin Yuan, Pushkin R Pari and Gang Qu, “Zero overhead watermarking technique for FPGA designs”, *13<sup>th</sup> Great Lakes symposium on VLSI*, Washington DC, USA, Apr. 2003, pp. 147–152 (cited on p. 28).
- [Kah+01] Andrew B. Kahng, John Lach, William H. Mangione-Smith, Stefanus Mantik, Igor L. Markov, Miodrag Potkonjak, Paul Tucker, Huijuan Wang and Gregory Wolfe, “Constraint-based watermarking techniques for design IP protection”, *IEEE Trans. on CAD of Integrated Circuits and Systems* 20.10 (2001), pp. 1236–1252 (cited on p. 27).

- [Kea02] T. Kean, “Cryptographic Rights Management of FPGA Intellectual Property Cores”, *ACM/SIGDA 10<sup>th</sup> International Symposium on Field-programmable gate arrays*, Monterey CA, USA, Feb. 2002, pp. 113–118 (cited on p. 52).
- [KL16] Filip Kodýtek and Róbert Lórencz, “Proposal and Properties of Ring Oscillator-Based PUF on FPGA”, *Journal of Circuits, Systems, and Computers* 25.3 (2016) (cited on p. 34).
- [Kou11] F. Koushanfar, “Integrated Circuits Metering for Piracy Protection and Digital Rights Management: An Overview”, *Great Lakes Symposium on VLSI*, Lausanne, Switzerland., May 2011, pp. 449–454 (cited on p. 22).
- [Kou12] F. Koushanfar, “Provably Secure Active IC Metering Techniques for Piracy Avoidance and Digital Rights Management”, *IEEE Transactions on Information Forensics and Security* 7.1 (2012), pp. 51–63 (cited on pp. 50, 55, 56).
- [Kra94] Hugo Krawczyk, “LFSR-based Hashing and Authentication”, *Annual International Cryptology Conference*, vol. 839, Santa Barbara, California, USA, Aug. 1994, pp. 129–139 (cited on p. 135).
- [LB12] Bertrand Le Gal and Lilian Bossuet, “Automatic low-cost IP watermarking technique based on output mark insertions”, *Design Automation for Embedded Systems* 16.2 (2012), pp. 71–92 (cited on p. 29).
- [LT15] Yu-Wei Lee and Nur A. Touba, “Improving logic obfuscation via logic cone analysis”, *16th Latin-American Test Symposium*, Puerto Vallarta, Mexico, Mar. 2015, pp. 1–6 (cited on p. 46).
- [LPS12] Vincent van der Leest, Bart Preneel and Erik van der Sluis, “Soft Decision Error Correction for Compact Memory-Based PUFs Using a Single Enrollment”, *International Workshop on Cryptographic Hardware and Embedded Systems*, vol. 7428, Leuven, Belgium, Sept. 2012, pp. 268–282 (cited on p. 38).
- [Lew+12] Matthew Lewandowski, Richard Meana, Matthew Morrison and Srinivas Katkoori, “A novel method for watermarking sequential circuits”, *IEEE International Symposium on Hardware-Oriented Security and Trust*, San Francisco, CA, USA, June 2012, pp. 21–24 (cited on p. 29).
- [Li+16] Meng Li, Kaveh Shamsi, Travis Meade, Zheng Zhao, Bei Yu, Yier Jin and David Z. Pan, “Provably secure camouflaging strategy for IC protection”, *International Conference on Computer-Aided Design*, Austin, TX, USA, Nov. 2016, p. 28 (cited on p. 106).
- [Mae+09] R. Maes, D. Schellekens, P. Tuyls and I. Verbauwhede, “Analysis and Design of Active IC Metering Schemes”, *IEEE International Workshop on Hardware-Oriented Security and Trust*, San Francisco CA, USA, July 2009, pp. 74–81 (cited on p. 135).
- [MSV12] R. Maes, D. Schellekens and I. Verbauwhede, “A Pay-per-Use Licensing Scheme for Hardware IP Cores in Recent SRAM-Based FPGAs”, *IEEE Transactions on Information Forensics and Security* 7.1 (2012), pp. 98–108 (cited on pp. 45, 52, 54).
- [Mae13] Roel Maes, “An Accurate Probabilistic Reliability Model for Silicon PUFs”, *International Workshop on Cryptographic Hardware and Embedded Systems*, vol. 8086, Santa Barbara, CA, USA, Aug. 2013, pp. 73–89 (cited on p. 132).

- [MHV12] Roel Maes, Anthony Van Herrewege and Ingrid Verbauwhede, “PUFKY: A Fully Functional PUF-Based Cryptographic Key Generator”, *International Workshop on Cryptographic Hardware and Embedded Systems*, vol. 7428, Leuven, Belgium, Sept. 2012, pp. 302–319 (cited on pp. 38, 39, 131, 135).
- [Mae+12] Roel Maes, Vladimir Rozic, Ingrid Verbauwhede, Patrick Koeberl, Erik van der Sluis and Vincent van der Leest, “Experimental evaluation of Physically Unclonable Functions in 65 nm CMOS”, *European Solid-State Circuit Conference*, Bordeaux, France, Sept. 2012, pp. 486–489 (cited on p. 121).
- [MTV09a] Roel Maes, Pim Tuyls and Ingrid Verbauwhede, “A soft decision helper data algorithm for SRAM PUFs”, *IEEE International Symposium on Information Theory*, Seoul, Korea, June 2009, pp. 2101–2105 (cited on pp. 38, 121).
- [MTV09b] Roel Maes, Pim Tuyls and Ingrid Verbauwhede, “Low-Overhead Implementation of a Soft Decision Helper Data Algorithm for SRAM PUFs”, *International Workshop on Cryptographic Hardware and Embedded Systems*, Lausanne, Switzerland, Sept. 2009, pp. 332–347 (cited on pp. 38, 39, 126, 131).
- [Mai+10] Abhranil Maiti, Jeff Casarona, Luke McHale and Patrick Schaumont, “A large scale characterization of RO-PUF”, *IEEE International Symposium on Hardware-Oriented Security and Trust*, Anaheim CA, USA, June 2010, pp. 94–99 (cited on pp. 34, 39, 121).
- [MGS13] Abhranil Maiti, Vikash Gunreddy and Patrick Schaumont, “A Systematic Method to Evaluate and Compare the Performance of Physical Unclonable Functions”, *IACR Cryptology ePrint Archive* (2013), pp. 245–267 (cited on p. 32).
- [Mal+15] Shweta Malik, Georg T. Becker, Christof Paar and Wayne P. Burleson, “Development of a Layout-Level Hardware Obfuscation Tool”, *IEEE Computer Society Annual Symposium on VLSI*, Montpellier, France, July 2015, pp. 204–209 (cited on pp. 41, 42).
- [Mar16] Cédric Marchand, “Conception de matériel salubre pour lutter contre la contrefaçon et le vol de circuits intégrés”, PhD thesis, Université Jean Monnet, 24th Nov. 2016 (cited on pp. 4, 5, 11, 12, 141).
- [MBC16] Cédric Marchand, Lilian Bossuet and Abdelkarim Cherkaoui, “Enhanced TERO-PUF Implementations and Characterization on FPGAs”, *International Symposium on Field-Programmable Gate Arrays*, Monterey, CA, USA, Feb. 2016, p. 282 (cited on pp. 36, 39, 121, 132).
- [MBG17] Cédric Marchand, Lilian Bossuet and Kris Gaj, “Area-oriented comparison of lightweight block ciphers implemented in hardware for the activation mechanism in the anti-counterfeiting schemes”, *International Journal of Circuit Theory and Applications* 45.2 (2017), CTA-16-0095.R3, pp. 274–291 (cited on p. 141).
- [MKM08] C. Marsh, T. Kean and D. McLaren, “Protecting Designs with a Passive Thermal Tag”, *15<sup>th</sup> IEEE International Conference on Electronics, Circuits and Systems*, St. Julians, Malta, Aug. 2008, pp. 218–221 (cited on p. 28).
- [Mar+15] Jesus Martinez-Mateo, Christoph Pacher, Momtchil Peev, Alex Ciurana and Vicente Martin, “Demystifying the Information Reconciliation Protocol CASCADE”, *Quantum Information & Computation* 15.5&6 (2015), pp. 453–477 (cited on pp. 117, 119, 120).

- [MN08] G. Masalskis and R. Navickas, “Reverse Engineering of CMOS Integrated Circuits”, *Elektronika ir Elektrotechnika - Electronics and Electrical Engineering* 8.88 (2008), pp. 28–31 (cited on p. 19).
- [McD+16] Jeffrey Todd McDonald, Yong C. Kim, Todd R. Andel, Miles A. Forbes and James McVicar, “Functional polymorphism for intellectual property protection”, *IEEE International Symposium on Hardware Oriented Security and Trust*, McLean, VA, USA, May 2016, pp. 61–66 (cited on pp. 41, 42).
- [McL11] I. McLoughlin, “Reverse engineering of embedded consumer electronic systems”, *IEEE 15<sup>th</sup> International Symposium on Consumer Electronics*, Singapore, Singapore, June 2011, pp. 352–356 (cited on p. 19).
- [Mey+11] Uwe Meyer-Bäse, Encarni Castillo, Guillermo Botella, L. Parrilla and Antonio García, *Intellectual property protection (IPP) using obfuscation in C, VHDL, and Verilog coding*, 2011 (cited on p. 44).
- [Mic17a] Microsemi, *Antifuse FPGAs*, 2017, URL: <https://www.microsemi.com/products/fpga-soc/antifuse-fpgas> (cited on p. 30).
- [Mic17b] Microsemi, *UG0533 User Guide Libero SoC Secure IP Flow*, tech. rep., 2017 (cited on p. 55).
- [Moo65] Gordon E. Moore, “Cramming more components onto integrated circuits”, *Electronics* 38.8 (1965), pp. 114–117 (cited on pp. 1, 7).
- [Moo75] Gordon E. Moore, “Progress in Digital Integrated Electronics”, *Technical Digest of International Electron Devices Meeting*, 1975, pp. 11–13 (cited on pp. 1, 7).
- [Mor+11] Amir Moradi, Alessandro Barenghi, Timo Kasper and Christof Paar, “On the vulnerability of FPGA bitstream encryption against power analysis attacks: extracting keys from xilinx Virtex-II FPGAs”, *Conference on Computer and Communications Security*, Chicago, Illinois, USA, Oct. 2011, pp. 111–124 (cited on p. 45).
- [Mor+13] Amir Moradi, David Oswald, Christof Paar and Pawel Swierczynski, “Side-channel attacks on the bitstream encryption mechanism of Altera Stratix II: facilitating black-box analysis using software reverse-engineering”, *International Symposium on Field Programmable Gate Arrays*3, Monterey, CA, USA, Feb. 2013, pp. 91–100 (cited on p. 45).
- [MS16] Amir Moradi and Tobias Schneider, “Improved Side-Channel Analysis Attacks on Xilinx Bitstream Encryption of 5, 6, and 7 Series”, *International Workshop on Constructive Side-Channel Analysis and Secure Design*, vol. 9689, Graz, Austria, Apr. 2016, pp. 71–87 (cited on p. 45).
- [New05] M. E. J. Newman, “A measure of betweenness centrality based on random walks”, *Social Networks* 27.1 (2005), pp. 39–54 (cited on pp. 93, 94).
- [Ng14] Ruth Li-Yung Ng, “A Probabilistic Analysis of CASCADE”, *International conference on quantum cryptography*, 2014 (cited on p. 119).
- [NR08] Jean-Baptiste Note and Éric Rannaud, “From the bitstream to the netlist”, *International Symposium on Field Programmable Gate Arrays*, Monterey, CA, USA, Feb. 2008, p. 264 (cited on pp. 19, 45).
- [OM95] Kevin O’Brien and Serge Maginot, “KRYPTON: Portable, Non-Reversible Encryption for VHDL”, *Model Generation in Electronic Design*, Boston, MA, 1995, pp. 127–151 (cited on p. 44).



- [Oli01] Arlindo L Oliveira, “Techniques for the creation of digital watermarks in sequential circuit designs”, *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 20.9 (2001), pp. 1101–1117 (cited on p. 29).
- [Ope10] Opencores, *Wishbone B4 : Wishbone System-on-Chip (SoC) Interconnection Architecture for Portable IP Cores*, tech. rep., 2010 (cited on p. 51).
- [PP09] Christof Paar and Jan Pelzl, *Understanding Cryptography*, Springer, 2009 (cited on pp. 141, 142).
- [Pac+15] Christoph Pacher, Philipp Grabenweger, Jesus Martinez-Mateo and Vicente Martin, “An information reconciliation protocol for secret-key agreement with small leakage”, *IEEE International Symposium on Information Theory*, Hong Kong, Hong Kong, June 2015, pp. 730–734 (cited on pp. 119, 120).
- [Par+10] James D. Parham, J. Todd McDonald, Yong C. Kim and Michael R. Grimaila, “Hiding Circuit Components Using Boundary Blurring Techniques”, *IEEE Annual Symposium on VLSI*, 2010 (cited on p. 44).
- [Par+09] L. Parrilla, E. Castillo, A. García, E. Todorovich, D. González and A. Lloris, “Intellectual Property Protection of  $\mu$ P cores”, *Design of Circuits and Integrated Systems*, Saragossa, Spain, Nov. 2009 (cited on p. 51).
- [PVK16] Vinay C. Patil, Arunkumar Vijayakumar and Sandip Kundu, “On meta-obfuscation of physical layouts to conceal design characteristics”, *IEEE International Symposium on Defect and Fault Tolerance in VLSI and Nanotechnology Systems*, Storrs, CT, USA, Sept. 2016, pp. 147–152 (cited on p. 44).
- [Pie12] Krzysztof Pietrzak, “Cryptography from Learning Parity with Noise”, *38th Conference on Current Trends in Theory and Practice of Computer Science*, Špindlerův Mlýn, Czech Republic, Jan. 2012, pp. 99–114 (cited on p. 133).
- [PM14] Stephen M. Plaza and Igor L. Markov, “Protecting Integrated Circuits from Piracy with Test-aware Logic Locking”, *International Conference on Computer Aided Design*, San Jose, CA, USA, Nov. 2014 (cited on pp. 75, 80, 86).
- [PM15] Stephen M. Plaza and Igor L. Markov, “Solving the Third-Shift Problem in IC Piracy With Test-Aware Logic Locking”, *IEEE Trans. on CAD of Integrated Circuits and Systems* 34.6 (2015), pp. 961–971 (cited on p. 106).
- [Qua+16] Shahed E. Quadir, Junlin Chen, Domenic Forte, Navid Asadizanjani, Sina Shahbazmohamadi, Lei Wang, John Chandy and Mark Tehranipoor, “A Survey on Chip to System Reverse Engineering”, *JETC* 13.1 (2016), 6:1–6:34 (cited on pp. 19, 20).
- [Raj+12a] Jeyavijayan Rajendran, Youngok Pino, Ozgur Sinanoglu and Ramesh Karri, “Logic encryption: A fault analysis perspective”, *Design, Automation & Test in Europe Conference*, Dresden, Germany, Mar. 2012, pp. 953–958 (cited on pp. 47, 48, 85–87).
- [Raj+12b] Jeyavijayan Rajendran, Youngok Pino, Ozgur Sinanoglu and Ramesh Karri, “Security analysis of logic obfuscation”, *Annual Design Automation Conference*, San Francisco CA, USA, June 2012, pp. 83–89 (cited on pp. 48, 49, 106).
- [Raj+13] Jeyavijayan Rajendran, Michael Sam, Ozgur Sinanoglu and Ramesh Karri, “Security analysis of integrated circuit camouflaging”, *ACM Conference on Computer & communications security*, Berlin, Germany, Nov. 2013, pp. 709–720 (cited on pp. 41, 47–49, 85).

- [RSK13] Jeyavijayan Rajendran, Ozgur Sinanoglu and Ramesh Karri, “Is split manufacturing secure?”, *Design, Automation and Test in Europe*, Grenoble, France, Mar. 2013, pp. 1259–1264 (cited on p. 40).
- [Raj+15] Jeyavijayan Rajendran, Huan Zhang, Chi Zhang, Garrett S. Rose, Youngok Pino, Ozgur Sinanoglu and Ramesh Karri, “Fault Analysis-Based Logic Encryption”, *IEEE Transactions on Computers* 64.2 (2015), pp. 410–424 (cited on pp. 46, 48, 49, 72, 74, 83–85, 98, 102, 104).
- [RKM08a] J. A. Roy, F. Koushanfar and I. Markov, “EPIC: Ending Piracy of Integrated Circuits”, *Design, Automation and Test in Europe*, 2008, pp. 1069–1074 (cited on pp. 46–49, 51, 52, 83–87, 98, 155, 159).
- [RKM08b] J. A. Roy, F. Koushanfar and I. Markov, “Protecting Bus-based Hardware IP by Secret Sharing”, *45<sup>th</sup> Design Automation Conference*, Anaheim CA, USA, June 2008, pp. 846–851 (cited on p. 51).
- [RKM10] J. A. Roy, F. Koushanfar and I. Markov, “Ending Piracy of Integrated Circuits”, *Computer* 43.10 (2010), pp. 30–38 (cited on pp. 46, 47, 85).
- [Rüh+10] Ulrich Rührmair, Frank Sehnke, Jan Sölter, Gideon Dror, Srinivas Devadas and Jürgen Schmidhuber, “Modeling attacks on physical unclonable functions”, *ACM Conference on Computer and Communications Security*, Chicago, IL, USA, Oct. 2010, pp. 237–249 (cited on p. 32).
- [Sab66] Gert Sabidussi, “The centrality index of a graph”, *Psychometrika* 31.4 (1966), pp. 581–603 (cited on p. 92).
- [SZT08] Moritz Schmid, Daniel Ziener and Jürgen Teich, “Netlist-level IP protection by watermarking for LUT-based FPGAs”, *International Conference on Field-Programmable Technology*, Taipei, Taiwan, Dec. 2008, pp. 209–216 (cited on p. 28).
- [SNK13] Sean Seet, Ruth Li-Yung Ng and Khoongming Khoo, “An Accurate Analysis of the BINARY Information Reconciliation Protocol by Generating Functions”, *International Conference on Quantum Cryptography*, Waterloo, Canada, Aug. 2013 (cited on p. 117).
- [SEM06] SEMI, *Intellectual Property (IP) Challenges and Concerns of the Semiconductor Equipment and Materials Industry*, tech. rep., SEMI, 2006 (cited on p. 18).
- [SHF14] Mitsuru Shiozaki, Ryohei Hori and Takeshi Fujino, “Diffusion Programmable Device : The device to prevent reverse engineering”, *IACR Cryptology ePrint Archive* 2014 (2014), p. 109 (cited on p. 41).
- [SS06] E. Simpson and P. Schaumont, “Offline Hardware/Software Authentication for Reconfigurable Platforms”, *International Workshop on Cryptographic Hardware and Embedded Systems*, Yokohama, Japan, Oct. 2006 (cited on p. 52).
- [Sko05] Sergei P. Skorobogatov, *Semi-invasive attacks - A new approach to hardware security analysis*, tech. rep., University of Cambridge, Apr. 2005 (cited on p. 30).
- [Smi+17] Jessica Smith, Kiri Oler, Carl Miller and David Manz, “Reverse Engineering Integrated Circuits Using Finite State Machine Analysis”, *Hawaii International Conference on System Sciences*, Koloa, HI, USA, Jan. 2017 (cited on p. 19).
- [Soc14] IEEE Computer Society, ed., *IEEE Recommended Practice for Encryption and Management of Electronic Design Intellectual Property (IP)*, 2014 (cited on p. 55).

- [SZ89] Karen Stephenson and Marvin Zelen, “Rethinking centrality: Methods and examples”, *Social Networks* 11.1 (1989), pp. 1–37 (cited on pp. 93, 95).
- [SRM15] Pramod Subramanyan, Sayak Ray and Sharad Malik, “Evaluating the security of logic encryption algorithms”, *IEEE International Symposium on Hardware Oriented Security and Trust*, Washington, DC, USA, May 2015, pp. 137–143 (cited on pp. 80, 106).
- [SD07] G. Edward Suh and Srinivas Devadas, “Physical Unclonable Functions for Device Authentication and Secret Key Generation”, *Design Automation Conference*, San Diego, CA, USA, June 2007, pp. 9–14 (cited on pp. 34, 38).
- [TJ11] Randy Torrance and Dick James, “The state-of-the-art in semiconductor reverse engineering”, *Proceedings of the Design Automation Conference, DAC 2011, San Diego, California, USA, June 5-10, 2011*, 2011, pp. 333–338 (cited on pp. 19, 22, 27).
- [Tuz+12] Nicholas Tuzzio, Kan Xiao, Xuehui Zhang and Mohammad Tehranipoor, “A zero-overhead IC identification technique using clock sweeping and path delay analysis”, *Great Lakes Symposium on VLSI*, Salt Lake Cit, UT, USA, May 2012, pp. 95–98 (cited on p. 31).
- [VDF13] Michal Varchola, Milos Drutarovský and Viktor Fischer, “New universal element with integrated PUF and TRNG capability”, *International Conference on Reconfigurable Computing and FPGAs*, Cancun, Mexico, Dec. 2013, pp. 1–6 (cited on p. 35).
- [VMV13] Jo Vliegen, Nele Mentens and Ingrid Verbauwhede, “A Single-chip Solution for the Secure Remote Configuration of FPGAs using Bitstream Compression”, *International Conference on Reconfigurable Computing and FPGAs*, Cancun, Mexico, Dec. 2013 (cited on p. 45).
- [WKP11] S. Wei, F. Koushanfar and M. Potkojnak, “Integrated Circuit Digital Rights Management Techniques Using Physical Level Characterization”, *Annual ACM workshop on Digital rights management*, Chicago, USA, Oct. 2011, pp. 3–14 (cited on p. 31).
- [Wil15] Kyle Wilkinson, *Using Encryption to Secure a 7 Series FPGA Bitstream*, tech. rep., Xilinx, 2015 (cited on p. 45).
- [Xia+16] Kan Xiao, Domenic Forte, Yier Jin, Ramesh Karri, Swarup Bhunia and Mohammad Tehranipoor, “Hardware Trojans: Lessons Learned After One Decade of Research”, *ACM Transactions on Design Automation of Electronic Systems* 22.1 (May 2016), pp. 1–23 (cited on p. 20).
- [XS16] Yang Xie and Ankur Srivastava, “Mitigating SAT Attack on Logic Locking”, *International Conference on Cryptographic Hardware and Embedded Systems*, Santa Barbara, CA, USA, Aug. 2016, pp. 127–146 (cited on p. 106).
- [Xil13] Xilinx, *Xilinx Plug-and-Play IP: Accelerating Productivity and Design Reuse*, tech. rep., 2013 (cited on p. 55).
- [YI01] Akihiro Yamamura and Hirokazu Ishizuka, “Error Detection and Authentication in Quantum Key Distribution”, *Australasian Conference on Information Security and Privacy*, vol. 2119, Sydney, Australia, July 2001, pp. 260–273 (cited on p. 135).
- [Yas+16a] Muhammad Yasin, Bodhisatwa Mazumdar, Jeyavijayan J. V. Rajendran and Ozgur Sinanoglu, “SARLock: SAT attack resistant logic locking”, *IEEE International Symposium on Hardware Oriented Security and Trust*, McLean, VA, USA, May 2016, pp. 236–241 (cited on p. 106).

- [Yas+17a] Muhammad Yasin, Bodhisatwa Mazumdar, Ozgur Sinanoglu and Jeyavijayan Rajendran, “Removal Attacks on Logic Locking and Camouflaging Techniques”, *IACR Cryptology ePrint Archive 2017* (2017), p. 348 (cited on p. 106).
- [Yas+17b] Muhammad Yasin, Bodhisatwa Mazumdar, Ozgur Sinanoglu and Jeyavijayan Rajendran, “Security analysis of Anti-SAT”, *Asia and South Pacific Design Automation Conference*, Chiba, Japan, Jan. 2017, pp. 342–347 (cited on p. 106).
- [Yas+15] Muhammad Yasin, Jeyavijayan Rajendran, Ozgur Sinanoglu and Ramesh Karri, “On Improving the Security of Logic Locking”, *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 35.9 (2015), pp. 1411–1424 (cited on pp. 106, 141).
- [Yas+16b] Muhammad Yasin, Samah Mohamed Saeed, Jeyavijayan Rajendran and Ozgur Sinanoglu, “Activation of logic encrypted chips: Pre-test or post-test?”, *Design, Automation & Test in Europe Conference*, Dresden, Germany, Mar. 2016, pp. 139–144 (cited on p. 153).
- [Yas+17c] Muhammad Yasin, Abhrajit Sengupta, Benjamin Carrión Schäfer, Yiorgos Makris, Ozgur Sinanoglu and Jeyavijayan Rajendran, “What to Lock?: Functional and Parametric Locking”, *Great Lakes Symposium on VLSI*, Banff, AB, Canada, May 2017, pp. 351–356 (cited on p. 106).
- [Yen14] Bülent Yener, *Hardware Reverse-engineering*, Rensselaer Polytechnic Institute, 2014 (cited on p. 26).
- [YD10] Meng-Day (Mandel) Yu and Srinivas Devadas, “Secure and Robust Error Correction for Physical Unclonable Functions”, *IEEE Design & Test of Computers* 27.1 (2010), pp. 48–65 (cited on p. 38).
- [ZC12] Li Zhang and Chip-Hong Chang, “State encoding watermarking for field authentication of sequential circuit intellectual property”, *IEEE International Symposium on Circuits and Systems*, Seoul, South Korea, May 2012, pp. 3013–3016 (cited on p. 29).
- [ZT08] Daniel Ziener and Jürgen Teich, “Power Signature Watermarking of IP Cores for FPGAs”, *Journal of Signal Processing Systems* 51.1 (2008), pp. 123–136 (cited on p. 28).



## BIBLIOGRAPHY

---

# **Appendices**



# Examples of graphs found in graph analysis for combinational logic locking

Figure 16 depicts the resulting graph obtained after converting the c2670 benchmark netlist, which comprises 1117 logic gates.

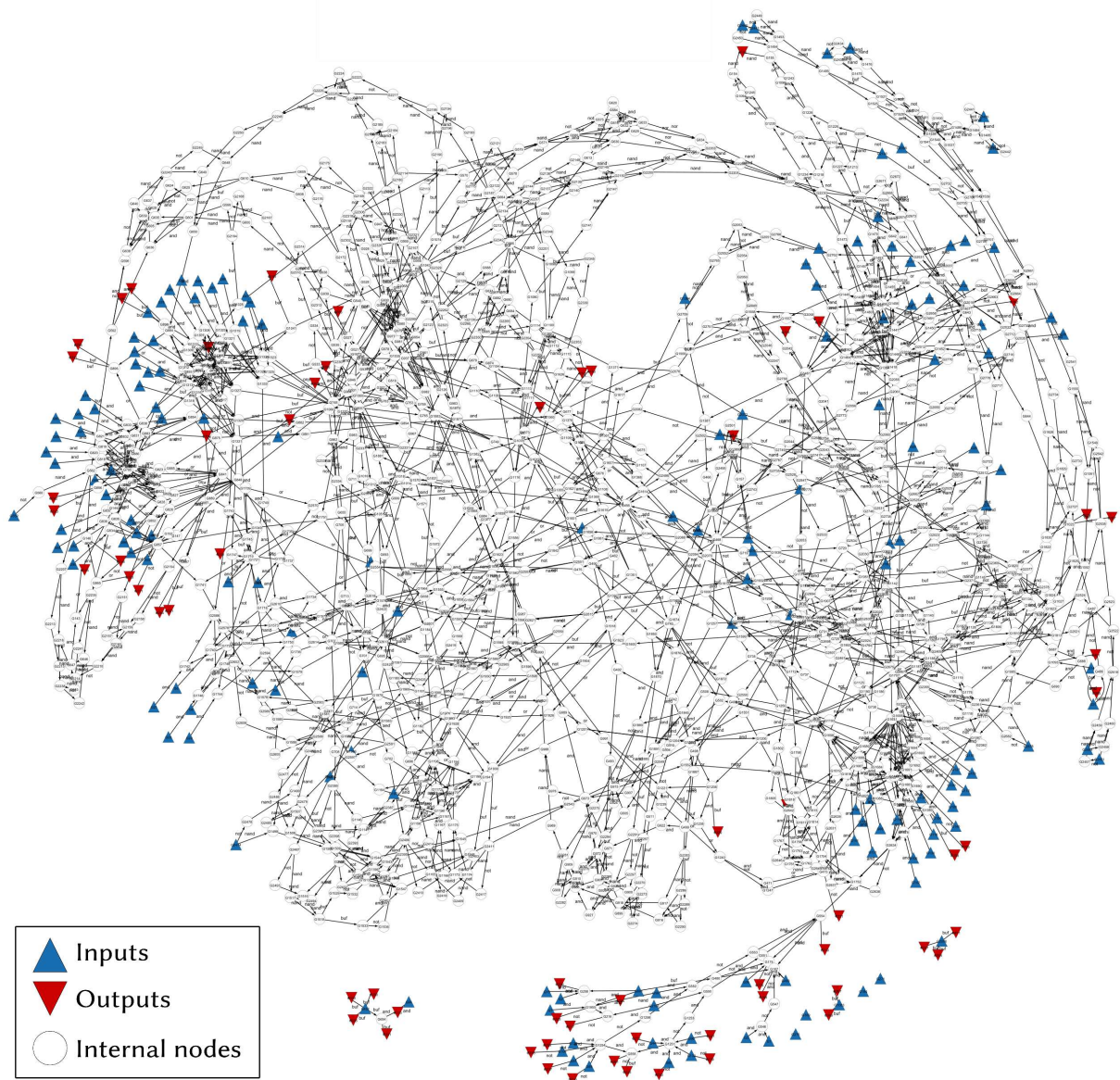


Figure 16 – Example of the c2670 benchmark, which comprises 1117 logic gates, converted into a directed acyclic graph.

After processing the graph for combinational logic locking (see Chapter 2), the remaining paths that can propagate a locking value are shown in Figure 18.

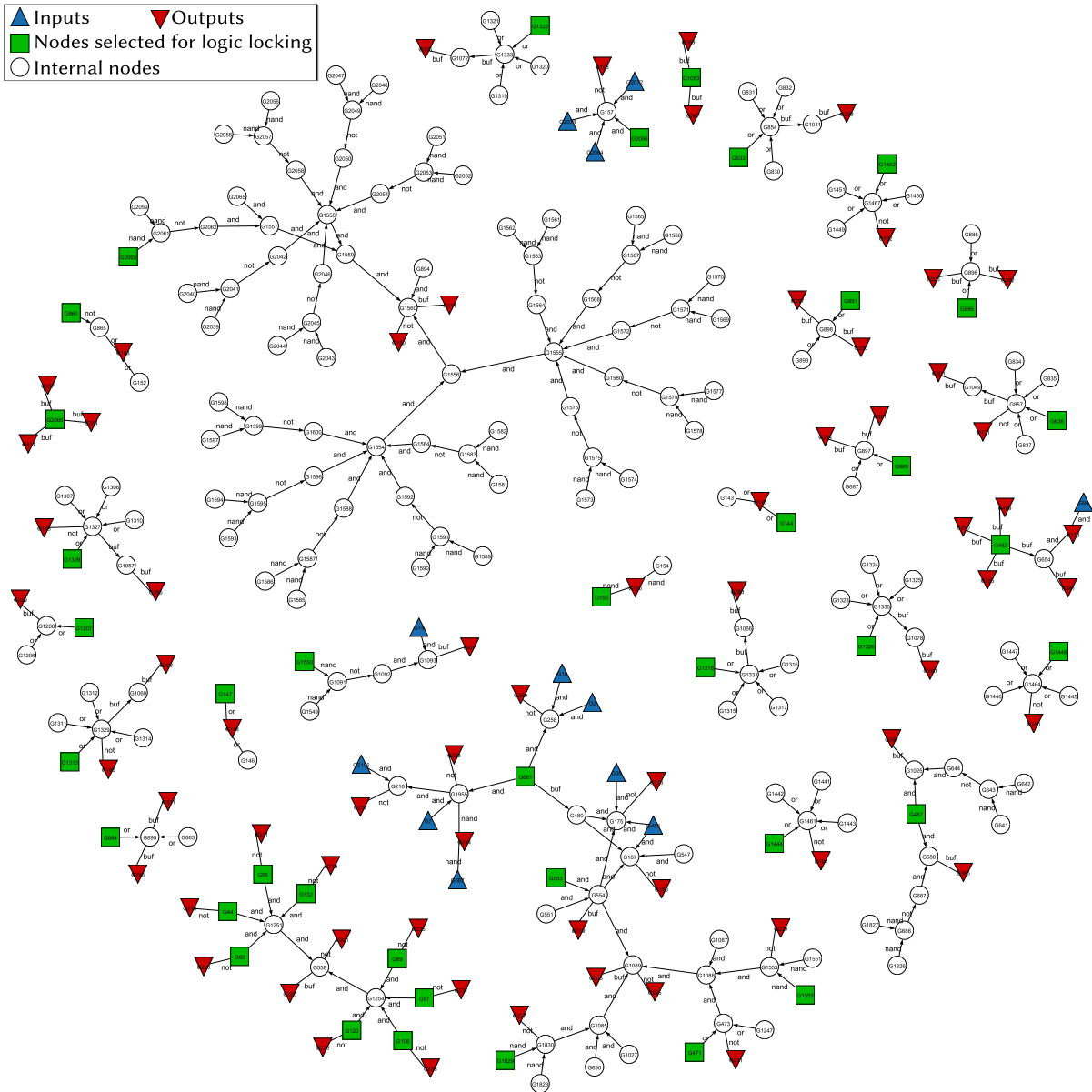
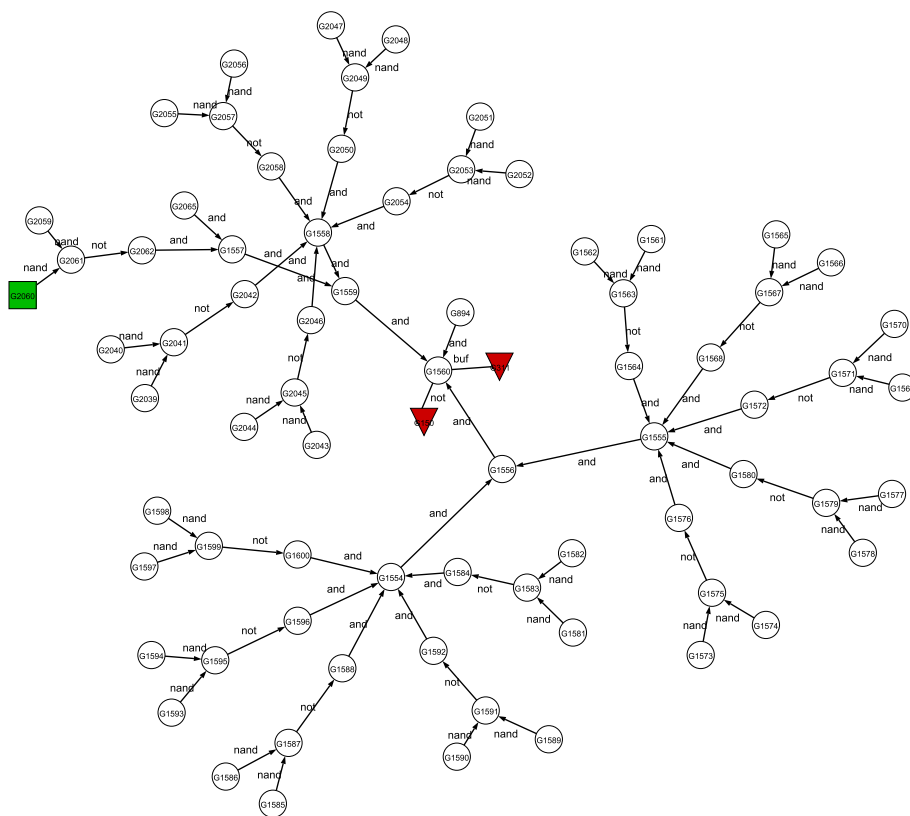
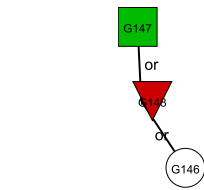


Figure 17 – Example of graph after analysis for combinational logic locking.

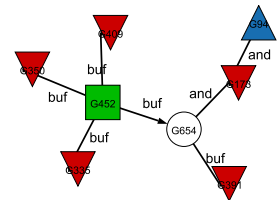
This graph shows a wide variety of connected components. Some examples are given in Figure 18. The very large connected component depicted in Figure 18a comprises 75 vertices. The node to modify for logic locking, in green, is six logic levels away from the two outputs it locks. Conversely, the connected component shown in Figure 18b has only 3 vertices. This is a final OR gate before the output, where none of its inputs could propagate a locking value. In that case, the locking gate inserted locks only one output and is very close to it. The last connected component shown in Figure 18c shows a connected component with five outputs. Only one locking gate is necessary to lock them all, which is interesting from a logic resources overhead perspective. However, the inserted locking gate is very close to the outputs it locks.



(a) Large connected component with a locking gate inserted very far from the outputs.



(b) Small connected component



(c) Locking gate spanning five outputs but very close to them.

Figure 18 – Example of connected components found in the final graph after analysis for combinational logic locking.

---

# List of Figures

1	Semiconductor companies and their respective positions in the integrated circuit design process. . . . .	3
2	Entreprises de la micro-électronique et leur position respective dans le processus de conception d'un circuit intégré. . . . .	9
1.1	Design data transfer in the semiconductor IP business . . . . .	17
1.2	Specific threats to design data in the semiconductor IP business. Trusted and untrusted parties are from the IP designer point of view . . . . .	21
1.3	Hierarchy of design data protection methods classified according to their efficiency at protecting design data. . . . .	25
1.4	Methods for identifying an IP core itself or the individual instances. . . . .	26
1.5	Different types of mask ROM . . . . .	26
1.6	Examples of integrated circuits embedding an electrically erasable ROM that can be erased by shining UV light through the quartz window . . . . .	30
1.7	Basic protocol for IP authentication using a PUF . . . . .	32
1.8	Arbiter PUF with challenge "011 . . . 1" applied, comparing the blue and red path . . . . .	34
1.9	Ring oscillator cell and PUF. . . . .	35
1.10	Transient effect ring oscillator cell. . . . .	36
1.11	Typical initialisation pattern observed in an SRAM array . . . . .	36
1.12	Front end of line and back end of line layers in the CMOS manufacturing process <sup>2</sup> . . . . .	40
1.13	Active layer of Syphermedia gates [Coc+14]. . . . .	41
1.14	Logic obfuscation of a boolean function . . . . .	42
1.14	Logic obfuscation of a boolean function. . . . .	43
1.15	An example of VHDL design files obfuscation. . . . .	44
1.16	Original and masked nodes depending on the associated activation bit. . . . .	46
1.17	Boosted FSM . . . . .	50
1.18	Example of public-key cryptography usage in the EDA tool for a secure key exchange and IP block transmission (adapted from [Gua+09]). . . . .	53
1.19	Example of the implication of a trusted third party (MA) in the transactions between an FPGA vendor (FV), a system integrator (SYS) and two IP core designers (CV) (from [MSV12]). . . . .	54



---

1.20	Overview of the IP protection module designed in the framework of the SAL-WARE project . . . . .	58
2.1	Modification of an output logic gate . . . . .	62
2.2	Two examples of logic functions and the inputs that can lock their output . . .	64
2.3	Propagation of a locking value through a sequence of nodes (in thick red) . . .	65
2.4	Conversion of a netlist to a directed acyclic graph . . . . .	66
2.5	Deletion of the incoming edges of vertices that do not satisfy $V_{\text{forced}} \in V_{\text{locks}}$ and removal of connected components that do not contain any output. . . . .	67
2.6	Different types of connected components that are found in the final graph. The node(s) select to be modified for logic locking are highlighted in orange. . . .	68
2.7	Vertices selected for logic locking. . . . .	69
2.8	Original and locked vertices depending on the associated activation bit . . . .	70
2.9	Locking vertices and edges added to the graph . . . . .	70
2.10	Lockable version of the netlist . . . . .	70
2.11	Area overhead as the percentage of extra logic gates required to implement logic locking . . . . .	72
2.12	Computation time required to process a netlist for logic locking and for fault analysis-based logic masking . . . . .	72
2.13	Maximum logic locking of a netlist portion that can propagate a locking value	76
2.14	Minimum and maximum overhead values for logic locking strength tuning . .	76
2.15	Interleaving the AW bits to strengthen logic locking . . . . .	77
2.16	OR locking gate (in dark grey) obfuscated by two extra gates (in light grey) with logic values shown in red and blue depending on the value of the AW bit	79
3.1	Degree centrality values for the vertices of a random graph . . . . .	91
3.2	Boolean function $G5 = G1 \cdot G2 \cdot G3 \cdot G4$ synthesised using a 4-input AND gate (a) or three 2-input AND gates (b). The resulting graphs (c) and (d) lead to different degree centrality values for the vertex $G5$ . . . . .	91
3.3	Closeness centrality values for the vertices of a random graph . . . . .	92
3.4	Betweenness centrality values for the vertices of a random graph . . . . .	93
3.5	Current-flow betweenness centrality computation on a graph and equivalent electrical network . . . . .	94
3.6	Current-flow betweenness centrality values for the vertices of a random graph	94
3.7	Current-flow closeness centrality values for the vertices of a random graph . .	95
3.8	$E_m$ values obtained for several logic resources overhead . . . . .	98
3.9	Computation time required for the centrality indicators considered for different benchmark sizes. . . . .	101
3.10	Trade-off between masking efficiency and computation time for different node selection heuristics at 5% logic resources overhead. . . . .	102

---

3.11	Graph for which selecting the vertices with the highest centrality does not alter the outputs optimally . . . . .	104
4.1	Illustration of the similarities between key reconciliation and reliable shared key generation from a PUF response . . . . .	111
4.2	CONFIRM applied on 16-bit blocks . . . . .	112
4.3	Spreading a burst of errors among multiple blocks . . . . .	114
4.4	Example of executing the BINARY protocol on 16-bit responses with one error.	115
4.5	Example of executing the CASCADE protocol on 16-bit responses with five errors. . . . .	118
4.6	Implementation of the parity computation module using one large multiplexer	122
4.7	Implementation of the parity computation module by making an existing shift register circular . . . . .	123
4.8	Implementation of the parity computation module when the response is stored in RAM . . . . .	124
4.9	Leakage values (in bits) obtained with different error rates, initial block sizes and number of passes. . . . .	125
4.10	Failure rate values obtained with different error rates, initial block sizes and number of passes. . . . .	127
4.11	Changes in the number of bits in the response at different steps. . . . .	135
5.1	IP protection module . . . . .	137
5.2	Module de protection des données de conception . . . . .	138
5.3	Part of the design flow augmented for logic locking or logic masking . . . . .	140
5.4	Position of the AW decoder . . . . .	142
5.5	TERO cell with 8 delay elements per branch (7 buffers and 1 NAND gate) . . .	145
5.6	BFSK transmitter from [BBF15] . . . . .	146
5.7	HECTOR motherboard . . . . .	147
5.8	HECTOR daughterboards . . . . .	147
5.9	Logic modifier tab of the graphical user interface . . . . .	149
5.10	HECTOR board management tab of the graphical user interface . . . . .	150
5.11	Enrolment tab of the graphical user interface . . . . .	150
5.12	Activation tab of the graphical user interface . . . . .	150
5.13	Graphical user interface to the hardware multiplier with input $500 \times 2$ . . . . .	151
5.14	Graphical user interface to the hardware multiplier with input $25 \times 4$ . . . . .	151
5.15	Simplified design flow with steps implementing secure remote activation highlighted. . . . .	152
16	Example of the c2670 benchmark, which comprises 1117 logic gates, converted into a directed acyclic graph. . . . .	183

---

17	Example of graph after analysis for combinational logic locking. . . . .	184
18	Example of connected components found in the final graph after analysis for combinational logic locking. . . . .	185

# List of Tables

1.1	Threats on design data. . . . .	24
1.2	Advantages and drawbacks of the considered PUF architectures . . . . .	37
1.3	Sketch (SS) and recover (Rec) procedures for code-offset and syndrome constructions of secure sketches. . . . .	38
1.4	Logic resources required by the presented error-correction schemes on FPGA. . . . .	39
1.5	Masking efficiency opposed to computational complexity for existing nodes selection heuristics. The symbol $\times$ means that the property is not fulfilled, the symbol $\checkmark$ means that the property is fulfilled. . . . .	49
1.6	Knowledge of the keys and encrypted data among parties ( $\checkmark$ : known, $\times$ : unknown). . . . .	54
1.7	Association of solutions to achieve complete IP protection . . . . .	56
1.8	Suitability of IP protection solutions at addressing different threats . . . . .	56
2.1	Controlling value of non-linear logic gates and the associated forced output value . . . . .	62
2.2	$V_{\text{forced}}$ and $V_{\text{locks}}$ values for the internal nodes of the netlist in Figure 2.4a . . . . .	67
2.3	Experimental results obtained when applying combinational logic locking on ITC'99 benchmarks. . . . .	73
2.4	Logic resources required to implement a hardware point function for different input and output widths . . . . .	78
3.1	Contingency table of the binary variables $y[i]$ and $y_{\text{masked}}[i]$ . . . . .	86
3.2	Masking efficiency evaluation by different metrics. $\checkmark$ stands for the masking efficiency being evaluated as good by the metric. $\times$ stands for the masking efficiency being evaluated as bad by the metric. . . . .	87
3.3	Controllability values of the output of usual 1 and 2-input logic gates. Their logic equation is of the form $Y = F(A)$ if $F$ is a unary boolean function or $Y = F(A, B)$ if $F$ is a binary boolean function. . . . .	88
3.4	Observability values of the input(s) of usual 1 and 2-input logic gates. . . . .	89
3.5	Time complexity of centrality indicators . . . . .	97
3.6	$E_m$ values obtained with other selection heuristics at 5% logic resources overhead . . . . .	98
3.7	Experimental results obtained when applying logic masking on ITC'99 and EPFL benchmarks for different centrality indicators. . . . .	99

---

3.7	Experimental results obtained when applying logic masking on ITC'99 and EPFL benchmarks for different centrality indicators. . . . .	100
3.8	Distance from the inserted logic masking gates to the inputs/outputs when using different centrality indicators . . . . .	103
4.1	Block sizes used for the first passes and after . . . . .	120
4.2	Examples of parameters to achieve failure-rates of $10^{-4}$ , $10^{-6}$ and $10^{-8}$ for different PUF architectures, aiming at keeping at least 128 bits secret. . . . .	121
4.3	Distribution of operations between device and server. . . . .	122
4.4	Leakage values (in bits) obtained with different error rates, initial block sizes and number of passes . . . . .	125
4.5	Order of magnitude of the failure rate values obtained with different error rates, initial block sizes and number of passes . . . . .	127
4.6	Logic resources required for three implementation options of the parity computation module and three response sizes. . . . .	128
4.7	Device-side execution time in clock cycles of different codes with different constructions. . . . .	131
5.1	Logic resources required to implement a lightweight block cipher (from [Mar16; MBG17]) . . . . .	141
5.2	AW decoder architectures . . . . .	143
5.3	Logic resources required to implement the AW decoder . . . . .	144
5.4	Device-side implementation results for the whole design data protection module	148