KU LEUVEN

ARENBERG DOCTORAL SCHOOL
Faculty of Engineering Science

Low-Latency Hardware Secure Masking for Symmetric Key Cryptography

Zhenda Zhang

Dissertation presented in partial fulfillment
of the requirements for the degree of
Doctor of Engineering Science (PhD):
Electrical Engineering

August 2025

Supervisors:
Prof. dr. ir. Vincent Rijmen
(corresponding supervisor)
Dr. Svetla Nikova
(co-supervisor)

LOW-LATENCY HARDWARE SECURE MASKING FOR SYMMETRIC KEY CRYPTOGRAPHY

Zhenda ZHANG

Supervisors:
Prof. dr. ir. Vincent Rijmen
(corresponding supervisor)
Dr. Svetla Nikova
(co-supervisor)

Members of the
Examination Committee:
Prof. dr. ir. Jean Berlamont, chair
Dr. Benedikt Gierlichs (Secretary)
Prof. dr. ir. Ingrid Verbauwhede (Assessor)
Prof. dr. ir. Marian Verhelst (Assessor)
Prof. dr. Stefan Mangard
(Graz University of Technology, Austria)

Dissertation presented in partial fulfillment of the requirements for the degree of Doctor of Engineering Science (PhD): Electrical Engineering

August 2025

© 2025 Zhenda Zhang
Uitgegeven in eigen beheer, Zhenda Zhang, Leuven (Belgium)

Alle rechten voorbehouden. Niets uit deze uitgave mag worden vermenigvuldigd en/of openbaar gemaakt worden door middel van druk, fotokopie, microfilm, elektronisch of op welke andere wijze ook zonder voorafgaande schriftelijke toestemming van de uitgever.

All rights reserved. No part of the publication may be reproduced in any form by print, photoprint, microfilm, electronic or any other means without written permission from the publisher.

Preface

This dissertation is submitted in fulfillment of the requirements for the doctoral degree at the Faculty of Engineering Science, KU Leuven. Its purpose is to develop efficient countermeasures against side-channel attacks, which represent a significant threat to the security of real-world computing systems of cryptographic algorithms. This work contributes to the ongoing efforts in reducing latency and randomness costs in such hardware implementations of symmetric-key ciphers, concluding five years of doctoral study.

I would like to express my sincere gratitude to my supervisors, Prof. Vincent Rijmen and Dr. Svetla Nikova, for their expertise, constructive feedback, and continuous support throughout my doctoral studies. I am also thankful to my examination committee. Prof. Ingrid Verbauwhede and Prof. Marian Verhelst, who also served on my supervisory committee, provided valuable evaluations and feedback at each milestone, which significantly helped me to progress this work. I am grateful to Dr. Benedikt Gierlichs and Prof. Stefan Mangard for their review of this dissertation and for the insightful discussions we had during its writing. I also wish to thank the committee chair, Prof. Jean Berlamont, for coordinating the evaluation and defense sessions.

I want to extend my special thanks to the professors, researchers, and staff of the COSIC research group at KU Leuven for providing an excellent environment and the necessary technical facilities to conduct this research. Dr. Lennert Wouters, Dr. Arthur Becker, and Dr. Benedikt Gierlichs gave tremendous support in the COSIC hardware lab, especially for side-channel measurements and leakage evaluation platforms. Prof. Ingrid Verbauwhede welcomed me into the hardware group meetings, where I benefited greatly from high-level discussions in the field of hardware security. Dr. Siemen Dhooghe introduced me to COSIC and, together with Dr. Victor Arribas and Dr. Lauren De Meyer, guided me during the early stages of my doctoral journey. I also thank Dilara Toprakhisar, John Gaspoz, and Artemii Ovchinnikov for their collaboration and regular discussions within the TI Group. Dr. Ventzislav Nikov, Amund

Askeland, and Dr. Tim Beyne, as co-authors, offered critical support and inspiration that helped shape the research presented here. Michiel Verbauwhede helped to improve the abstract in het Nederlands in this thesis. Last but not least, I warmly thank the COSIC staff, especially Dana Brouckmans, Péla Noé, Saartje Verheyen, Vanessa Banze, and Elsy Vermoesen, for their kind welcome and outstanding administrative support throughout my time at KU Leuven.

Personally, I am deeply grateful to my family and friends for their unwavering motivation and support. Between 2020 and 2025, the world and our personal lives have undergone profound and often unexpected changes. Their presence helped me navigate the most difficult moments and reminded me of the more comprehensive purpose beyond this academic journey.

Completing this thesis has been one of the most demanding and rewarding challenges of my life. I hope it serves as a meaningful contribution to research in hardware security and cryptographic implementations, as well as a solid foundation for future work, both in my career and in collaboration with the research community.

Zhenda ZHANG

August 2025
Leuven, Belgium

Abstract

Cryptographic primitives are designed and evaluated against mathematical cryptanalysis under the assumption that an adversary does not know the internal states. However, practical hardware implementations require additional measures to ensure security against physical attacks, such as side-channel analysis. This thesis addresses passive side-channel attacks, such as the Differential Power Analysis by Kocher et al. in CRYPTO 1999, which exploits leakage by measuring power consumption and analyzing statistical properties of power traces to extract secret information.

This dissertation focuses on the logic-level countermeasure against side-channel attacks, which is known as masking. In masking, the secret internal states of a cryptographic algorithm are split into multiple random shares so that the power consumption depending on these shares leaks no information correlated to the original secret. This thesis primarily focuses on Boolean masking in the context of hardware-secure masking of symmetric-key cryptographic primitives, as these algorithms operate on Boolean functions and their hardware implementations consist of Boolean logic gates. Various Boolean masking schemes and formal proofs based on security models have been proposed.

However, the overhead caused by the Boolean masking schemes is high. Applying masking to a symmetric-key cipher either significantly increases the area or introduces additional latency in terms of time or clock cycles. In addition, masking requires a substantial amount of uniformly distributed random bits, which must be refreshed every clock cycle. As a result, an expensive pseudo-random number generator is another cost on top of the trade-off between area and latency. This dissertation thus addresses two key aspects of hardware secure masking in symmetric key ciphers, i.e., reducing latency in digital circuits in the presence of glitches and minimizing randomness costs.

To reduce the latency introduced by register stages that prevent glitch propagation, this thesis introduces Low-Latency Threshold Implementations

(LLTI). This novel masking approach eliminates intermediate registers between decomposed Boolean functions. LLTI achieves a reduction in area and an increase in maximum frequency compared to the direct sharing of Threshold Implementations. LLTI reduces cycle count without adding randomness overhead compared to low-latency $d + 1$ masking, which requires more registers and randomness. Additionally, this thesis presents Glitch-Stopping Circuits, which replace conventional registers with logic gates or latches to stop glitch propagation. Unlike Dual-Rail Pre-charge circuits, which suffer from early propagation effects and require a dedicated pre-charging phase, this method maintains the overall synchronous operation of the cipher with a modified clock configuration to drive Glitch-Stopping Circuits. Moreover, it is applicable to designs targeting high-order glitch-extended probing security.

Another contribution of this thesis is the reduction of randomness cost in hardware masking. This work demonstrates how randomness can be reused across multiple register stages by generalizing the changing-of-the-guards technique for $td + 1$ and $d + 1$ masking. This enables first-order masked implementations to rely entirely on randomness generated prior to the execution, which is subsequently extended to second-order masking. As a result, the need for randomness at each stage, which requires on-the-fly generation, is effectively eliminated.

This thesis advances the state-of-the-art hardware secure masking by introducing low-latency masking schemes, glitch-stopping circuit implementations, and randomness-reusing methods, thereby making secure cryptographic implementations more practical and efficient.

Beknopte samenvatting

Cryptografische primitieven worden ontworpen en geëvalueerd met behulp van wiskundige cryptanalyse, in de veronderstelling dat er geen interne toestanden bekend zijn. Praktische hardware-implementaties vereisen echter aanvullende maatregelen om beveiliging tegen fysieke aanvallen, zoals side-channel analyse, waarop dit proefschrift zich richt, te garanderen. Deze passieve aanvalstechniek, zoals differentiële vermogensanalyse van de eerste of hogere orde van Kocher et al. in CRYPTO 1999, maakt gebruik van fysieke eigenschappen die informatie lekken door het vermogen te meten en statistische momenten van de vermogenssporen te analyseren om geheime gegevens te extraheren.

Dit proefschrift is ontwikkeld op basis van de algoritmische tegenmaatregel tegen side-channel aanvallen, gekend als masking. Geheime waarden, waaronder invoer, uitvoer en de interne toestanden van een cryptografisch algoritme, worden opgesplitst in meerdere willekeurige delen, zodat het vermogen dat afhankelijk is van deze delen geen informatie lekt die gecorreleerd is met het geheim. Met name Booleaanse masking, specifiek in de context van hardwarematige, veilige masking op cryptografische primitieven met symmetrische sleutels, staat centraal in dit proefschrift. De reden hiervoor is dat algoritmen met symmetrische sleutels voornamelijk werken op Booleaanse functies, en de geïmplementeerde digitale circuits Booleaanse logische poorten zijn. Verschillende Booleaanse maskingsschema's en formele bewijzen gebaseerd op beveiligingsmodellen worden voorgesteld.

Echter, de overhead, die door verschillende Booleaanse maskingsschema's wordt veroorzaakt is hoog. Het toepassen van masking op een symmetrisch blockcijfer vergroot ofwel het oppervlakte aanzienlijk ofwel introduceert het extra vertraging in termen van tijd- of klokcycli. Bovendien vereist masking een groot aantal willekeurige bits, waarvan wordt aangenomen dat ze uniform verdeeld zijn en elke cyclus worden ververst. Een dure pseudo-random number generator is dus een extra kost bovenop de afweging tussen oppervlak en latentie. Dit proefschrift behandelt daarom deze twee belangrijke aspecten van hardwarematige veilige

masking in symmetrische blockcijfers, namelijk het verminderen van de latentie in digitale circuits en het minimaliseren van willekeurigheidskosten.

Om de latentie, die wordt veroorzaakt door registerstappen die glitch-propagation voorkomen, aan te pakken, introduceert dit proefschrift Low-Latency Threshold Implementations (LLTI), een nieuwe maskingsmethode die registers tussen Threshold Implementations van ontbonden Booleaanse functies elimineert. LLTI bereikt een oppervlakteverkleining en een verhoging van de maximale frequentie in vergelijking met directe Threshold Implementations. LLTI vermindert het aantal cycli zonder extra willekeurigheidskosten in vergelijking met low-latency $d+1$ masking, waarvoor meer registers en willekeur nodig zijn. Daarnaast onderzoekt dit proefschrift Glitch-Stopping Circuits, die conventionele registers vervangen door logische poorten of latches om glitch-propagation te stoppen. In tegenstelling tot Dual-Rail Pre-charge circuits, die last hebben van vroege propagatie-effecten en een pre-charge fase vereisen, behoudt deze methode de synchrone aard met een aangepaste klokconfiguratie om glitch-stopping circuits aan te sturen, en kan deze worden toegepast op high-order glitch-extended probing security.

Een andere bijdrage van dit proefschrift is de reductie van de willekeurigheids-kosten bij hardwaremasking.

Dit werk demonstreert hoe willekeur kan worden hergebruikt over meerdere registertrappen door de wisseling-van-de-wacht-techniek voor $td+1$ en $d+1$ masking te generaliseren.

Dit maakt het mogelijk dat gemaskeerde implementaties van de eerste orde volledig vertrouwen op willekeur die vóór de uitvoering wordt gegenereerd, en die later wordt uitgebreid naar masking van de tweede orde. Als gevolg hiervan wordt de noodzaak van willekeur in elke stap, die on-the-fly generatie vereist, geëlimineerd.

In dit proefschrift worden de nieuwste technieken voor veilige hardwaremasking verder ontwikkeld door de introductie van maskingsschema's met een lage latentie, nieuwe implementaties van glitch-stopping circuits en methoden voor hergebruik van willekeur. Hierdoor worden veilige cryptografische implementaties praktischer en efficiënter.

List of Abbreviations

(S)NI (Strong) Non-Interference. 24, 45, 65

ANF Algebraic Normal Form. 8, 41–43, 50, 54, 59

ASIC Application-Specific Integrated Circuit. 9, 17, 23, 68, 69, 76

CMOS Complementary Metal-Oxide Semiconductor. 29, 38

CMS Consolidating Masking Scheme. 25, 26, 42, 82

CPA Correlation Power Analysis. 10, 11, 16

CPU Central Processing Unit. 2, 4

DAG Directed Acyclic Graph. 7

DOM Domain-Oriented Masking. 25, 26, 37, 42, 46, 47, 65, 70, 84, 92

DPA Differential Power Analysis. 4, 10, 11, 29, 31, 32, 91

DRP Dual-Rail Precharge. xv, xvii, 13, 27–31, 36, 37, 63, 64, 66, 67, 70, 90

FPGA Field Programmable Gate Array. 2–4, 16, 17, 22, 23, 67–70, 76, 77, 83, 91, 92

GE Gate Equivalent. 23, 38

GFC Glitch-Free Circuit. 36

GLM Generic Low-Latency Masking. 47, 48

GPU Graphics Processing Unit. 2, 4

GSC Glitch-Stopping Circuit. xvii, 5, 35, 67, 69–72, 90, 92

IoT Internet of Things. 2, 3, 36

ISW Ishai-Sahai-Wagner. 14, 23, 25

LFSR Linear Feedback Shift Register. 77

LLTI Low-Latency Threshold Implementations. xv, xvii, 5, 35, 46, 53–62, 72, 89, 90, 92

LMDPL LUT-based Masked Dual-Rail with Pre-charge Logic. xv, 30, 31, 64–67, 70, 72, 90

LSB Least Significant Bit. 39

LUT Look-Up Table. 22, 64, 69

MDPL Masked Dual-Rail with Pre-charge Logic. 30

MIA Mutual Information Analysis. 10, 11

MSB Most Significant Bit. 39

NI Non-Interference. 45

NIST National Institute of Standards and Technology. 2, 76

NLFSR Non-Linear Feedback Shift Register. 78

NN Neural Network. 92, 93

NPU Neural Processing Unit. 4

P&R Place and Route. 15–17, 29, 30, 32, 92

PINI Probe Isolating Non-Interference. 24, 45, 47

PRNG Psuedo-Random Number Generator. 76–78, 80

RTL Register Transfer Level. 12, 17

SCA Side-Channel Attacks. 3, 4, 6, 10, 12–15, 17, 25, 28–31, 35, 69, 91–93

SESYM Self-Synchronized Masking. xv, 30, 65–67, 72, 90

SoC System-on-Chip. 2–4

SPA Simple Power Analysis. 10, 11

TEE Trusted Execution Environment. 4

TI Threshold Implementations. xii, xvii, 5, 24–26, 35, 37, 43, 44, 46, 48, 50–54, 56, 58, 59, 61–63, 70, 72, 76, 80, 82–84, 89, 90, 92, 93

TRNG True-Random Number Generator. 76, 77

TVLA Test Vector Leakage Assessment. 31–33, 61, 69, 77

WDDL Wave Dynamic Differential Logic. 28–30, 65, 66

Contents

Abstract iii

List of Abbreviations ix

Contents xi

List of Figures xv

List of Tables xvii

I Part I. Side-Channel Attacks and Hardware Secure Masking 1

1 Introduction 2
- 1.1 Research Objectives and Contributions . . . 5
  - 1.1.1 Reduce Latency Cost . . . 5
  - 1.1.2 Reduce Randomness Cost . . . 5
- 1.2 Outline . . . 6

2 Background 7
- 2.1 Notations . . . 7
- 2.2 Threat Model . . . 8
- 2.3 Physical Attacks . . . 9
  - 2.3.1 Passive and Active Physical Attacks . . . 9
  - 2.3.2 Side-Channel Attacks . . . 10
  - 2.3.3 Differential Power Analysis . . . 11
- 2.4 The Adversary Model and Countermeasures . . . 12
  - 2.4.1 The Probing Model . . . 12
  - 2.4.2 Side-Channel Countermeasures . . . 12
  - 2.4.3 Theory vs. Practice . . . 14

2.4.4 Robust Probing Model 16
2.5 Hardware Secure Masking 17
2.5.1 From Boolean Functions to Digital Circuits 17
2.5.2 Implementations of Hardware Secure Masking 22
2.5.3 Glitch-Immune Masking 23
2.5.4 Glitch-Free Circuits 27
2.6 Statistical Leakage Assessments 31

3 Low-Latency Masking in Hardware 35
3.1 Latency in Hardware Secure Masking 35
3.1.1 Timing Overhead of Glitch-Free Circuits 36
3.1.2 Timing Overhead of Glitch-Immune Masking 37
3.2 Glitch-Immune Masking of Boolean Functions 37
3.2.1 Complexity of Boolean Circuits 38
3.2.2 Multiplicative Depth 38
3.2.3 Function Decompositions 39
3.2.4 Constructing Glitch-Immune Masking 41
3.2.5 From Masked Gadgets to Masked Circuits 44
3.2.6 Trade-Offs in Glitch-Immune Masking 46
3.3 Low-Latency Glitch-Immune Masking 46
3.3.1 Low-Latency Masking with $d + 1$ Shares 47
3.3.2 Unrolled Threshold Implementations 48
3.3.3 Low-Latency Threshold Implementations 53
3.3.4 Universal First-Order $t + 2$ TI for Permutations 61
3.3.5 Conclusion 63
3.4 Low-Latency Masking without Registers 63
3.4.1 Low-Latency Glitch-Free Circuits 63
3.4.2 Glitch-Stopping Circuits 67
3.4.3 Conclusion 70
3.5 My Contributions 72
3.5.1 Extend Threshold Implementations to Low-Latency Schemes 72
3.5.2 Glitch-Stopping Circuits 73

4 Reduce the Randomness Cost in Hardware Secure Masking 75
4.1 Randomness Requirement in Hardware Secure Masking 75
4.1.1 Practical Randomness Sources 76
4.1.2 Efficient Randomness Generation 78
4.2 Reuse Randomness in Hardware Secure Masking 80
4.2.1 Guarding the First-Order Masked AES Implementations 80
4.2.2 Amortize Randomness in First-Order Two-Share Masking 84
4.2.3 Second-Order Three-Share Masking 85
4.3 My Contributions 86

4.3.1 Apply the Changing-of-the-Guards Technique to the First-Order Masked AES Implementation 87

5 Conclusions **89**
5.1 Summary 89
5.2 Future Works 91
5.2.1 Succinct Security Models for Efficient Masking 92
5.2.2 Hardware Secure Masking for Trustworthy AI 92
5.2.3 Apply Masking to Hardware-Software Co-design 93

Bibliography **95**

II Part II. Publications **108**

6 LLTI: Low Latency Threshold Implementations **109**
Publication Report 109

7 Glitch-Stopping Circuits: Hardware Secure Masking without Registers **141**
Publication Report 141

8 Guarding the First Order: The Rise of AES Maskings **173**
Publication Report 173

Curriculum Vitae **195**

List of Figures

2.1 Registers in hardware secure masking 18
2.2 Glitches and leakages in masking with the Trichina gate 20
2.3 Synchronous digital circuits and their performance metrics 21
2.4 Hardware secure masking with registers and its timing and area overhead 21
2.5 Categories of hardware secure masking 23
2.6 The changing-of-the-guards of the shared S-box $\bar{S}$ with three shares 27
2.7 Wave Dynamic Differential Logic [60] 29

3.1 DRP logic and its timing metric 37
3.2 The tower field decomposition of the AES S-box 41
3.3 The cubic decomposition of the AES S-box 41
3.4 LLTI non-completeness requirements for a cubic function [4] . . 56
3.5 LLTI non-completeness requirements for composite ($t = 6$) and prime ($t = 7$) degree examples [4] 57
3.6 The LMDPL structure [97] 64
3.7 The SESYM structure [84] 65
3.8 The glitch-stopping circuits with `AND` or `MUX` gates, and their clock configurations [120] 67
3.9 `AND` and `MUX` instances of the glitch-stopping circuits [120] 71
3.10 Clocked logic-gate-based glitch-stopping circuit with clocks with phase-shift 71

4.1 The structure of a key-stream generator and its unroll implementation [26] 79
4.2 The changing-of-the-guards of the shared S-box $\bar{S}$ with two shares uses the randomness $\bar{r}$ [6] 81
4.3 Application of the changing-of-the-guards in the serialized architecture [6] 81

4.4 The bounded-query probing model consisting of a challenger $\mathcal{C}$, an adversary $\mathcal{A}$, a left-right oracle $\mathcal{O}^b$, two inputs k_0, k_1, a set of probes $\mathcal{P}$, and a set of probed wire values $(v_1^b, ..., v_t^b)$ of the circuit $C(k_b)$ [15] 86
4.5 The pairing of two masked S-boxes [104] 87

List of Tables

2.1 Dual-rail encoding of one-bit value in DRP logics 28

3.1 Implementation costs of hardware composable gadgets 46
3.2 The function composition of a unrolled implementation of $f = f_n \circ \ldots \circ f_1$ in n layers 49
3.3 Estimate the area cost of the unrolled implementations on quartic monomials 50
3.4 The reformulation of the non-complete set covering 52
3.5 The elements $S_i \in \widehat{\mathcal{C}}^{\mathrm{nc}}(6,2,2)$ and their complements 53
3.6 The LLTI function decomposition on a function with degree t . 55
3.7 Non-complete set coverings present TI and LLTI in format $\langle \mathcal{C}^{\mathrm{nc}}(s,t,d), s_o \rangle$ for algebraic degree t, non-completeness order d, number of shares at input s and output s_o 58
3.8 Summary of final first-order LLTI coverings for $t = \{3,6,7\}$ applied to a single monomial [4] 61
3.9 AES S-box implementations using low-latency glitch-immune masking schemes 62
3.10 Physical leakage tests of GSC using the FPGA primitives on Xilinx Sakura-G [120] 69
3.11 AES S-box with tower field decomposition with low-latency masking without registers 72

Part I

Part I. Side-Channel Attacks and Hardware Secure Masking

Chapter 1

Introduction

Digital technologies increasingly permeate various aspects of our lives, from personal devices to large-scale computations. Technologies, such as the Internet of Things (IoT), personal smart devices, and cloud computing, have become ubiquitous, and the security of the underlying hardware platforms is essential. These platforms are often connected to the internet and accessible to multiple users, running secure applications and processing sensitive data. This makes them vulnerable to security threats and requires protection from adversaries. Thus, protecting the hardware platforms from various security threats is crucial for ensuring the confidentiality and integrity of the data processed.

Cryptographic Devices

Instead of using dedicated devices for secure applications, modern computing systems like IoT, personal smart devices, and cloud computing are highly complex and integrated. Cryptographic accelerators and secure elements are integrated into System on Chip (SoC), interfacing with memories, sensors, and networks. Cloud computing, according to the National Institute of Standards and Technology (NIST) [76], provides users with on-demand and ubiquitous access to a shared pool of configurable computing resources. This model allows multiple untrusted tenants to share computing resources, which raises security concerns. Service providers offering heterogeneous computing resources, such as CPU, GPU, and FPGA, together with memory and storage, must ensure the isolation and security of the tenants' data and applications. Thus, there are cryptographic processors integrated into SoCs for secure data storage and memory access.

The IoT paradigm connects embedded sensors and systems to the Internet. Due to the large number of devices and the distributed nature of the IoT, physical access to the devices is often difficult to control. The protection of sensitive data is crucial for devices running cryptographic applications, such as smart cards, smart home devices, and industrial control systems. Data from IoT devices is often transmitted to the cloud, where larger-scale computations are performed.

Personal devices, such as smartphones and smartwatches, have become essential to daily life and are equipped with a complex SoC that senses and stores sensitive information. Modern smart devices incorporate dedicated cryptographic hardware components to perform essential cryptographic operations, ensuring encrypted communications, secure boot, and authentication. Thus, the security from silicon to data center becomes the central design goal of modern, trustworthy computing systems. Cryptographic devices are increasingly integrated into complex computing systems and SoCs rather than being operated as a dedicated module.

Side-Channel Attacks

Side-channel attacks (SCA) use execution (cache) timing, power consumption, and electromagnetic changes or emanations when processors execute cryptographic operations as sources of unintentional information leakage. This thesis focuses on the leakages through power consumption. The power consumption side channel is particularly vulnerable to attackers if the target devices are remotely or physically accessed and measured. For example, embedded devices in the IoT paradigm are often deployed in various locations where the owner may be unable to ensure their physical security. Academic research uses measurement setups on dedicated hardware or software platforms running only cryptographic algorithms. Successful attacks require signal processing to extract partial information from the noisy leakage signals. A practical full key-recovery attack on the Google Titan Security Key by Lomné and Roche [96] uses electromagnetic radiation during ECDSA signatures and a non-supervised machine learning method to recover the partial ephemeral key. Schellenberg et al. [101] develop sensors that can be synthesized with FPGA primitives and share the power distribution network of an FPGA with the cryptographic core. The sensors, which measure the dynamic power consumption of the module under attack and perform key recovery attacks, can be placed next to the module or can be remotely inserted into the FPGA. These attacks reveal vulnerabilities in security applications of embedded devices and FPGAs, which can be physically accessed or shared between multiple users. Thus, users cannot guarantee the security of computing systems unless cryptographic countermeasures are developed to protect them from side-channel attacks.

Applications of the Side-Channel Countermeasures

Modern processors used in cloud computing and data centers are required to perform secure computations that ensure the confidentiality and integrity of the data being processed. Although there are cryptographic algorithms, such as Fully Homomorphic Encryption and Secure Multi-Party Computation, that can compute on encrypted data without leaking any information, these algorithms are computationally expensive and not practical for current applications. Alternatively, the Trusted Execution Environment (TEE) provides a securely isolated execution environment for running software applications and protecting secret data from various software and hardware attacks, which results in a more efficient solution than pure software. At the time of writing this thesis, prominent SoC designers for cloud service and smart devices provide their commercial TEE solutions, e.g., Apple Secure Enclave [117], ARM TrustZone [2], Google Titan M2 [21], Intel SGX/TDX [62], Amazon Nitro [1], etc.

The TEE relies on a secure root of trust to verify its integrity. Typically, a root of trust is responsible for secure boot, secure key generation, and cryptographic operations. Such a root of trust is often implemented as a secure hardware module to protect sensitive cryptographic keys. For highly secure or government applications, the hardware root of trust includes countermeasures against physical SCA, e.g., Differential Power Analysis. For example, Rambus provides DPA resistant cryptographic accelerators [92]. Apple, in its Secure Enclave, includes the AES engine with side-channel countermeasures starting with the A9 SoC[61]. The open-source silicon root of trust project, OpenTitan[50], also includes an AES core with DPA countermeasures.

For embedded devices, a full implementation of the root of trust may not be feasible due to limited resources, and it may also be overkill for the application. In such scenarios, cryptographic hardware accelerators are integrated with state machines and connected to the rest of the system through a bus. These cryptographic functions are vulnerable to side-channel attacks and require countermeasures against such attacks.

With the recent trend of applying FPGA accelerators, GPUs, and NPUs in high-performance computing and machine-learning applications, the security of these hardware platforms has become a growing concern. The sharing of computing resources extends from the CPUs to heterogeneous systems [114], i.e., multi-tenancy on using FPGAs, which opens new attack surfaces to physical side-channel attacks.

1.1 Research Objectives and Contributions

This thesis explores methods to reduce the overhead of secure hardware implementations against side-channel attacks. It focuses on innovative masking schemes that minimize the need for randomness and novel hardware implementation techniques that reduce cycle-wise latency to one. Note that in this dissertation, latency is defined both in terms of clock cycles and time, as detailed in Section 2.5.1.

1.1.1 Reduce Latency Cost

The first part of the thesis tackles the issue of latency in masking schemes. Hardware secure masking under the glitch-extended probing model requires register stages, which introduce extra latency in clock cycles into the implementation. Techniques to reduce latency in hardware secure masking are built around removing register stages from the circuit. This thesis continues to focus on reducing latency costs and exploring the trade-offs among latency, area, and randomness costs.

My Contributions: My first attempt to construct a secured masking scheme without registers is called the Low-Latency Threshold Implementation (LLTI) [4], published in the journal IEEE TIFS 2022. This work studies the method for dividing high algebraic degree Boolean functions and masking each stage under the properties of Threshold Implementations (TI), i.e., non-completeness and uniformity, to construct a secure masked circuit without registers. In addition to working on masking schemes that are secured in the presence of glitches, another line of work is to construct circuits that prevent glitches from happening or propagating. My contribution following this line is *Glitch-Stopping Circuits: Hardware Secure Masking without Registers* [120], presented at the conference ACM CCS 2024. GSC is a group of digital logic circuits that can replace the registers in hardware secure masking, which stops glitch propagation and reduces the latency of the masking scheme to one clock cycle with its dedicated clock signal configurations.

1.1.2 Reduce Randomness Cost

Masking schemes that accommodate the glitch-extended probing model require randomness that updates every clock cycle. These random bits are used to refresh the masking value of intermediate computing variables, so the quality

of the randomness affects the side-channel security. At the same time, a high-quality random number generator is often expensive in terms of area and power consumption. This thesis concerns methods to reduce the dependency on randomness in the masking scheme.

My Contributions: To reduce the overall randomness cost, potential methods include reusing random bits across multiple clock cycles, utilizing imperfect randomness, or developing uniform masking schemes that require no randomness. The work, which I jointly conducted with co-authors, develops the first-order AES masking schemes that generalize the changing-of-the-guard technique and reuse random bits, with the title *Guarding the First Order: The Rise of AES Maskings* [6], presented in the conference CARDIS 2022.

1.2 Outline

This thesis is publication-based and divided into two parts. The first part begins with this chapter, which provides the essential background and state-of-the-art secure implementations against physical side-channel attacks. Chapter 2 covers the basic security concepts of side-channel attacks and countermeasures, the masking schemes and security models against SCA, and the statistical testing methods for empirically evaluating secure implementations. Chapter 3 explains the timing overhead of hardware secure masking, which can be represented in cycles or time, and the low-latency development of hardware secure masking. Two aspects of state-of-the-art masking are discussed: low-latency masking in glitch-immune masking schemes and glitch-stopping circuits. Chapter 4 focuses on the randomness cost in hardware secure masking.

The second part of this thesis contains the peer-reviewed articles published throughout the doctoral program. The articles are compiled from the author's version and adapted to the dissertation format with minor editorial changes. Chapter 6 and Chapter 7 are two publications about low-latency masking and implementing symmetric-key ciphers in hardware. Chapter 8 is an article about reducing randomness cost in the first-order secured masking schemes.

Chapter 2

Background

This dissertation will focus on low-latency countermeasures against side-channel attacks, specifically hardware secure masking implementations. This chapter starts with the attack surface, which specifies how an attacker can interact with the system. Then, an overview of side-channel attacks and the corresponding security model is given, offering the theoretical basis for developing hardware countermeasures. What follows are the categories of hardware secure masking that effectively protect against side-channel attacks in practice and the statistical leakage assessment methods used widely in physical tests.

2.1 Notations

The finite field with two elements is denoted by $\mathbb{F}_2$, and the finite field with 2^n elements by $\mathbb{F}_{2^n}$, where n is a positive integer. $\mathbb{F}_{2^n}^s$ is a s-dimensional vector space over the finite field $\mathbb{F}_{2^n}$, where s is also a positive integer. Thus, a Boolean function $f : (0,1)^n \to (0,1)$ in n variables maps n binary variables, or length-n binary vector in $\mathbb{F}_2^n$, to one binary value in $\mathbb{F}_2$, or a single bit. This function $f : \mathbb{F}_2^n \to \mathbb{F}_2$ can represent the desired behavior of a digital circuit with n inputs to one output.

A deterministic Boolean circuit $\mathcal{C}$ can be modeled as a directed acyclic graph (DAG), where the vertices correspond to Boolean gates, and the edges represent wires that define the flow of computation, ensuring a well-ordered, non-cyclic structure. For consistency, given two binary values a and b, the commonly used Boolean gates in this context are `AND` ($a \cdot b$ or ab), `OR` ($a + b$), `INV` ($\bar{a}$), `NAND`

($\overline{a \cdot b}$ or $\overline{ab}$), `XOR` ($a \oplus b = \bar{a}b + a\bar{b}$), etc. Note that using the symbol + to denote the Boolean gate `OR` only applies to digital circuits. In the context of Boolean functions, the addition over $\mathbb{F}_2$ is equivalent to an `XOR` gate.

A Boolean function $f : \mathbb{F}_2^n \to \mathbb{F}_2$ in n variables $x_1, \ldots, x_n$ has a unique polynomial representation, where the monomials are products of input variables, called the algebraic normal form (ANF):

$$f(x_1, \ldots, x_n) = \sum_{u \in \mathbb{F}_2^n} a_j (\prod_{i=1}^{n} x_i^{j_i}), \ a_j \in \mathbb{F}_2 \tag{2.1}$$

The monomials are the product of input variables, where the exponent j_i of each x_i is either 0 or 1, depending on which variable is present in the monomial. The algebraic degree of f, denoted as $\deg(f)$, refers to the highest degree of any monomial in the ANF of f. The algebraic degree is a crucial property in cryptographic functions, as higher-degree functions generally provide better resistance against linear and differential cryptanalysis, making them desirable for block ciphers. A function $F : \mathbb{F}_2^n \to \mathbb{F}_2^m$ can be considered as a vectorial Boolean function, $F(x_1, \ldots, x_n) = (f_1(x_1, \ldots, x_n), \ldots, f_m(x_1, \ldots, x_n))$, where each of its coordinate functions is a Boolean function $f_i : \mathbb{F}_2^n \to \mathbb{F}_2, i \in \{1, \ldots, m\}$. A vectorial Boolean function is equivalent to a Boolean circuit with n inputs and m outputs.

2.2 Threat Model

The attack surface refers to the total set of points in a system where an attacker can interact with or compromise the system. The countermeasure developed to oppose an attacker is based on the attack surface and the adversary model employed.

Black-Box Model

The black-box model assumes that the attacker can only access the cryptographic system's inputs and outputs and cannot observe its internal state. Thus, this model evaluates the characteristics independent of the implementation details.

White-Box Model

The white-box model assumes that the attacker has full access to the implementation details of the cryptographic system. The attacker can observe

the internal states, like the computation's intermediate or hard-coded secret values.

Grey-Box Model

The grey-box model lies between the black-box and white-box models, assuming that the attacker can analyse partial knowledge of the system. In this model, an attacker can learn information about the platform on which the cryptographic algorithm is implemented, while the complete state of the system under test remains hidden. The cryptographic system can be implemented in software or hardware, which means it could be in the form of assembly instructions or digital circuits. The system's state in software or hardware is all the values of the implemented instructions or the structure of the digital circuits throughout the entire computation process. The attacker in this model can retrieve a function of the state that partially reveals the secret information. Thus, this thesis will focus on the grey-box model, specifically the countermeasures to physical side-channel attacks.

2.3 Physical Attacks

Physical attacks pose a significant threat to the security of computer hardware in the grey-box model. These attacks can be categorized as passive or active, depending on the attacker's method of interacting with the system.

2.3.1 Passive and Active Physical Attacks

The passive attacker does not affect the system and only observes its operation through physical side channels. The attacker can have physical access to the target devices, such as microprocessors or ASICs running cryptographic algorithms, and analyse the measurements to extract secret information. This access can be local in the case of attacking an embedded device or remote in the multi-tenant FPGA cloud environment [114, 74]. The targeted measurements include power consumption, electromagnetic radiation, or timing information [70, 90, 69]. The execution time of cryptographic algorithms can depend on secret values, making many implementations vulnerable to exploitation. Power consumption and electromagnetic emission during encryption or decryption have also been shown to correlate with intermediate secret values.

Active attacks, also known as fault attacks [64], involve attackers modifying the system's internal values, for example, by inserting clock glitches or voltage spikes to disrupt the circuit's operation. Electromagnetic signals and laser beams are also widely used to introduce faults, such as flipping bits in the memory elements or distorting computation results. These attacks can also be invasive, such as decapsulating the processor or memory package to access the chip's circuits. This makes digital circuit components more exposed to active physical attacks and, thus, easier to manipulate. Invasive techniques may include etching, drilling, and inspecting the circuits using optical microscopes and small metal probes. This group of attacks changes or interrupts the computing process, thus leading to a different line of research outside the scope of this dissertation.

2.3.2 Side-Channel Attacks

Successful Side-Channel Attacks (SCA) require physical measures in the grey-box model to observe the characteristics of the cryptographic device to extract secret information from various measurements through statistical analysis. Thus, SCAs are categorized as passive attacks in the grey-box model, and this thesis focuses on the protection against such attacks. The protections against SCA aim to remove the statistical relations between the secret and the observed characteristics or physically isolate the device from such passive analysis.

Power SCA measures the power consumption, for example, by using a shunt resistor mounted on the power rail of the device under test and detecting the voltage drop or current variation across the shunt resistor over time. The power traces are correlated with the computation's intermediate values or the operation of the cryptographic algorithm on-device. When the attacker can observe the difference in power consumption depending on secret keys or processed data from a single power trace, this attack is referred to as Simple Power Analysis (SPA) [70]. Template attacks [32] are more powerful alternatives based on SPA because the attacker can access and control the device under test, allowing them to combine statistical analysis with a preparation phase. In this phase, the attacker characterizes the device's leakage behavior and creates a template model of the power consumption's dependency on the secret data. Then, the attacker can incorporate advanced statistical methods or machine learning algorithms with a template model. When the attacker cannot perform SPA due to the noise in the power traces, the attacker can collect multiple power traces and apply various statistical methods, which results in Differential Power Analysis (DPA) [70], Correlation Power Analysis (CPA) [24], or Mutual Information Analysis (MIA) [11], etc.

2.3.3 Differential Power Analysis

Kocher et al. introduced the Differential Power Analysis (DPA) attack in 1999 [70]. SPA mainly analyses the shape of the power trace along the time axis. In contrast, DPA looks at the dependency of many power traces on a fixed time moment on the processed data of a cryptographic operation. Each such power trace corresponds to a known plaintext or ciphertext and an unknown secret key.

DPA begins by selecting a sensitive intermediate value, which is an internal variable computed during the execution of the cryptographic algorithm that depends on both a known input or output and a small portion of the secret key. These values are considered sensitive because they influence the device's instantaneous power consumption in a key-dependent way. Assuming the power traces are appropriately aligned with respect to a fixed point in time, the attacker predicts the power consumption of this intermediate value using a chosen leakage model, such as Hamming weight or Hamming distance. The predicted power consumptions are computed for each key By computing the statistical dependency between the hypothetical power consumption and the measured power traces, the attacker obtains a result trace for each key guess. A significant peak appears in the trace corresponding to the correct key hypothesis, whereas incorrect guesses yield no distinguishable peaks. The attacker can thus identify the correct key guess, which leads to partial key recovery. Such a process can be repeated for other key bytes and intermediate values, allowing the entire secret key to be recovered. There are power attacks that use other statistical methods, i.e., CPA by Brier et al. [24] and MIA by Batina et al. [11], that improve the efficiency, robustness, and reduce the number of traces required for a successful attack.

From a defender's perspective, the countermeasures against the statistical power analyses, i.e., DPA, CPA, and template attacks, are the same because they all aim to remove the dependence between the power consumption and sensitive intermediate values, in other words, to de-correlate observable side-channel measurement from secret-dependent computations.

Higher-order side-channel attacks increase the adversary's capability by combining multiple leakage points, often across different time samples, and exploiting higher-order statistical moments. For example, higher-order DPA aggregates several leakage samples and analyses their joint statistical moments to bypass masking or decrease noise. Template attacks can model multivariate leakage distributions and are inherently capable of operating at high-order joint statistics. As such, mitigating higher-order attacks generally requires more robust countermeasures, such as higher-order masking or physical-level isolation

techniques.

2.4 The Adversary Model and Countermeasures

The focus of this section shifts from practical attacks to the corresponding theoretical adversary model, which defines the side-channel attacker's capability and threat model in the context of cryptographic implementations. The goal is for the security proof of the masking scheme within the adversary model to imply its practical security against side-channel attacks.

2.4.1 The Probing Model

The probing model, also known as the d-threshold probing model, by Ishai et al. [63] works in the context of deterministic Boolean circuits, which is an ideal representation of Boolean functions with Boolean gates and wires. This ideal Boolean circuit assumes that the leakages from different intermediate values in separate calculations are independent, which is also known as the independent leakage assumption. It describes the attacker's ability to probe the values of d wires in the circuit, which leaks the absolute value of the internal wires.

2.4.2 Side-Channel Countermeasures

SCA countermeasures are design methods in digital circuits that prevent side-channel attacks. Thus, the design of the countermeasures follows the abstraction level that is used in general digital circuit design [91]. In the context of hardware secure masking, the physical-level and the logic-level are the two abstraction levels concerned. The logic-level digital design describes the circuit using Boolean logic gates in combinational or sequential logic, also known as Register Transfer Level (RTL). This level specifies the circuits' functionality and data flow, which is independent of physical layout and behaviour. The physical-level design maps the logic into a layout that can be fabricated on silicon, which is defined by the physical placement of transistors and interconnections. At this level, practical timing, power, and reliability metrics can be calculated that depend on placement, routing, and design rule checks.

Physical-Level Countermeasures

Physical-level countermeasures [103] add additional components that act to flatten the instantaneous current or randomize the power consumption, which increases the signal-to-noise ratio to reduce side-channel leakage. In contrast, this thesis focuses on countermeasures against SCA at the logic level. The logic-level countermeasure transfers the logic of a cryptographic operation to a side-channel resistant one, which retains the same functionality but does not leak secrets through the power side channel. Note that a precise power consumption of the physical circuit is not available at the logic level, so countermeasures at this level aim to be secure under various physical implementations and hardware platforms. The logic level countermeasures are categorized into two main strategies: hiding and masking, which can be combined to achieve a secure implementation.

Hiding

Hiding the leakage means eliminating variations caused by computations or introducing artificial noise into the circuit to obscure leakage patterns. In this approach, the power consumption will be either random or constant regardless of the operation or data processing. In hardware, the logic gates are implemented according to the Boolean functions of the cryptographic algorithm, which cannot be changed during the operation. Thus, the hiding techniques should be applied to the power consumption of digital circuits during all clock cycles. Dual-Rail Pre-charge (DRP) logics are the logic-level hiding techniques with such properties.

Masking

Masking is a technique that amplifies noise by randomizing the intermediate values of the cryptographic device's computation. Thus, the computation and its power consumption become noise-like and independent of the secret values. The first adversary model to provide provable security with order d is the probing model by Ishai et al. [63] The Boolean masking (also called sharing) is achieved by bit-wise XOR ($\oplus$), which is equivalent to addition in $\mathbb{F}_2$. A first-order Boolean masking scheme splits the secret variable x into two random variables x_1 and x_2, such that $x = x_1 \oplus x_2$. Similarly, a dth-order masking divides the secret x into $d+1$ uniformly distributed variables $x_1, \ldots, x_d, x_{d+1}$, so that $x = x_1 \oplus \ldots \oplus x_d \oplus x_{d+1}$. With the probing model, a single `AND` gate can be masked with $d+1$ shares and proven to be d^{th}-order secure against SCA. The proof can be summarized as follows: knowing any subset of up to d probed

values on the masked circuits leaks no information about the original secret inputs.

Consider a dth-order secure `AND` gate in the probing model, $z = xy$, where each variable is masked with m shares to $x_1, \ldots, x_m$, $y_1, \ldots, y_m$, and $z_1, \ldots, z_m$. The challenge lies in computing the shares of z, $z_1, \ldots, z_m$, by appropriately grouping elements from the sum of partial products $x_i y_j$, ensuring that no set of d intermediate values leaks information to the adversary. For example, the Trichina gate [112], as shown in Figure 2.2a, was an early solution of a masked multiplication over $\mathbb{F}_2$ with two shares and a random bit r. It adopted a similar idea that carefully groups the random mask r and partial products $x_i y_j$ to ensure no intermediate value is revealed, which can be expressed as:

$$z_0 = r,\ r \leftarrow \$$$

$$z_1 = ((((r \oplus x_0 y_0) \oplus x_0 y_1) \oplus x_1 y_0) \oplus x_1 y_1) \tag{2.2}$$

The construction of a dth-order secure `AND` gate $z = xy$, also called the ISW multiplication [63], with m shares follows three steps:

1. For each $1 \leq i < j \leq m + 1$, introduce a random bit $r_{i,j} \leftarrow \$$.
2. Compute $r_{j,i} = (r_{i,j} \oplus x_i y_j) \oplus x_j y_i$.
3. For every $1 \leq i \leq m + 1$, compute $c_i = x_i y_i \oplus (\bigoplus_{j \neq i} r_{i,j})$.

Rivain and Prouff [95] proved that the minimum number of shares required for d^{th}-order security is $d + 1$, i.e., $m \geq d + 1$.

This masking technique and the theoretical foundation of provable security against SCA laid the basis for secure hardware and software implementations. However, this model overlooked the physical imperfections of practical digital circuits, which Section 2.4.3 explains. Other adversary models, such as the bounded moment model [10], noisy leakage model [44], and random probing model [13], are not the focus of this dissertation. This is because the dissertation primarily addresses hardware implementations, where security in the probing model can imply security in these other models [44].

2.4.3 Theory vs. Practice

Hardware implementations of masking schemes introduce physical effects that relate to the imperfections of digital signals in practice, which is explained in detail in Section 2.5.1. The prominent signal imperfection in the context of side-channel leakages is the glitch effect. The glitches occur when the combinational

circuits are in metastability, during which dynamic power consumption is present. Transition on a memory element is not only dependent on its present value, but also on the last value kept in the previous clock cycle. The coupling effect can break the probing model's assumption of independent leakage, which states that the probed wires are independent of the rest of the circuit. These effects emerge as the Boolean functions are implemented on real-world digital circuits, which in turn should be considered in the theory of countermeasures to SCA.

Glitches

In digital logic circuits, Boolean functions are implemented as combinational logic cells connected in a large circuit block. Different data paths in such a circuit have different propagation delays, so the inputs of combinational cells arrive at various times. Thus, the output of the combinational cells has temporary state transitions until all inputs become stable. This temporary state is metastability of the circuit. Such glitches also propagate through all the digital logic gates in the circuit block, resulting in various glitching behaviours related to the circuit's input values. This data-dependent dynamic switch of values on the wire contributes significantly to the power consumption and causes side-channel leakage. Mangard et al. [75] first exposed this critical weakness in the masking schemes implemented in hardware, showing that unintentional glitches caused by logic gate delays and routing imbalances compromised the claimed security.

Transitional Leakages

Transitions, or memory recombination, refer to the switching of the two stored values in consecutive clock cycles for memory elements in digital circuits. This phenomenon occurs over multiple clock cycles, which means the digital circuit maintains its state. Typical stateful circuits are the register files of a microcontroller or the value registers within a finite state machine. The former corresponds to software implementations of masked cryptographic systems, while the latter refers to parallelized hardware implementations of masked ciphers. Transitional leakage was first introduced in the software side-channel analysis [7, 37], where the Hamming distance between the two values can leak secret information stored in the memory elements.

Coupling Effects

The next step in the electronic design automation flow, after logic synthesis, is the place-and-route process (P&R) for the wires and logic elements in the circuit.

However, the wires routed close to each other can potentially have capacitive coupling effects. Meanwhile, the power distribution network connecting all logic elements in the circuits can also correlate the aggregated dynamic power consumption with the values on each wire. The coupling effects can cause the power consumption of one wire to be dependent on the value of another wire, which also breaks the independent leakage assumption. De Cnudde et al. [33] first reported such coupling effects in the context of the masked hardware implementation on FPGA.

2.4.4 Robust Probing Model

To address these imperfections, Reparaz et al. [94] proposed the first glitch-extended probing model. Based on this model, Faust et al. developed the robust probing model [45], which is also called (g, t, c)-extended probing model for the glitch, transition, and coupling effects respectively.

- Glitch-extended probes allow the adversary to observe all relevant primary inputs of the combinational logic block, which correspond to the outputs of the last register stage. This gives all possible value recombination of the logic block, which is the worst-case scenario.
- Transition-extended probes record two consecutive values of the memory element, which effectively double the number of adversarial probes and observe values before and after the memory transition.
- Coupling-extended probes allow observing leaked secret value, which depends on multiple wires, assuming there is a coupling channel between these wires.

Note that the coupling effect cannot be considered in the logic design and synthesis phases because it is either related to the P&R step of the design flow or related to the power distribution network-on-chip. The logic-level design and countermeasures cannot mitigate the physical-level coupling effect. Previous studies that tried to find evidence of leakage due to the coupling effect in a masked hardware cannon converge on this factor alone. Physical-level simulations and case studies of coupling effects on an FPGA demonstrate that the simulation step size or routing elements on devices can lead to false positives [77, 78, 34]. Meanwhile, there are remote attacks [101], which implement a sensor on the same FPGA running an unprotected AES, that successfully recover keys by performing CPA. Transitional leakage primarily occurs in software masking because overwriting the register with shares of a secret value in hardware secure masking is easy to avoid. The transitional leakage in hardware remains relevant

when the masked implementation is with a non-pipelined iterative structure [31]. Therefore, when the masking schemes are applied to hardware implementations, the robust probing model is commonly reduced to the (g)-extended probing model or the (g,t)-extended probing model. The work in this thesis focuses on the (g)-extended probing model.

2.5 Hardware Secure Masking

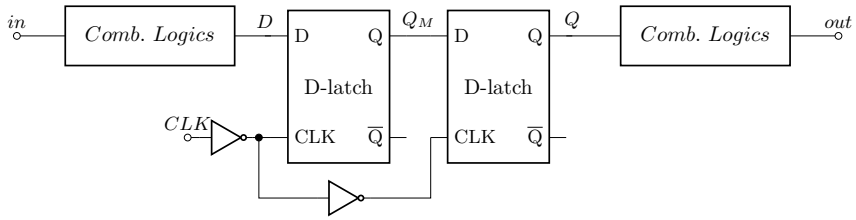
This thesis uses the term *hardware secure masking* to represent masking schemes considering the glitch effect in implementations of ciphers on FPGA or ASIC. Thus, it is the countermeasure against SCA in the (g)-extended probing model.

2.5.1 From Boolean Functions to Digital Circuits

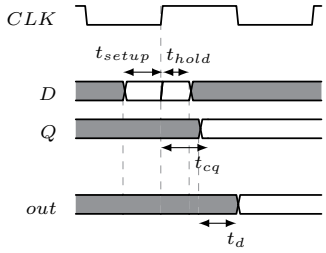
From the perspective of a hardware engineer, the implementation of a symmetric cipher in hardware like FPGA and ASIC considering countermeasures against SCA requires a systematic approach that balances performance, resource utilization, and security. The design process starts with choosing an architecture, such as a pipelined or iterative architecture, that optimizes the trade-off between throughput and area cost. At the logic level, the cipher operations, such as the S-box, permutations, and linear operations, are designed with mathematical cryptanalytic properties, which are represented in Boolean functions. The Boolean functions can be implemented using lookup tables or through combinational logic. The entire implementation then must be synthesized and optimized using FPGA or ASIC design tools into RTL netlists. Applying masking for the countermeasure against SCA will involve all the design steps up to logic synthesis and P&R. Furthermore, the security under the (g)-extended probing model adds design constraints related to latency and area cost from masked Boolean functions. Thus, the main overheads of hardware secure masking are registers for stopping glitch propagation and shares to mask the secret value, both contributing to additional area and latency cost.

Latency in Time and Clock Cycles

In digital circuit design, latency refers to the delay between when an input is applied to a circuit and when the corresponding output becomes stable. It can be expressed in terms of clock cycles or time. Latency-in-cycle indicates the number of clock cycles required for data to propagate through the circuit, which is determined by the pipeline depth or the number of sequential logic stages.



(a) A primary-secondary D-flip-flop (register) in a pipeline of sequential logic



(b) Definitions of set-up time, hold time, and propagation delay of a D-flip-flop [119, 91]

Figure 2.1: Registers in hardware secure masking

For example, a module with a latency of n cycles produces an output through n register stages after receiving the input. Latency-in-time, on the other hand, represents the actual duration of this propagation delay in time units such as nanoseconds, which depends on the clock frequency. It is calculated as the number of cycles multiplied by the clock period. For instance, if a circuit has a latency of n cycles and the clock frequency is 10 nanoseconds, i.e., 100 MHz clock frequency, the time latency is $10n$ nanoseconds.

Registers and Sequential Logic

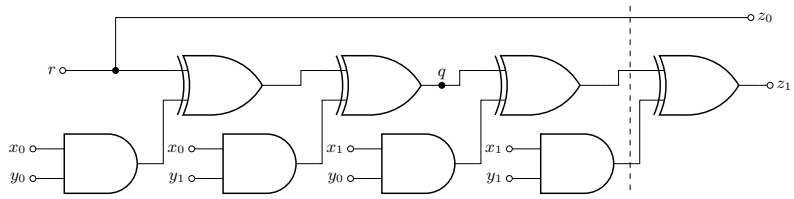
The design of sequential logic circuits, in addition to purely combinational logic, also incorporates memory elements, i.e., registers. These registers, also known as flip-flops, play a crucial role in synchronizing operations and ensuring the proper flow of data. They are logic circuits synchronized by a global clock and produce a stable output signal when the clock-edge trigger arrives. Thus, unlike combinational circuits, where outputs are determined solely by current inputs as defined by Boolean functions, sequential circuits rely on both current and past inputs, enabling stateful functionalities such as state machines and pipelined architectures.

A digital signal, either at a high or low level, represents a logic value 1 or 0. The rising edge, also referred to as the positive edge, represents the transition from a low to a high level in the clock signal and serves as the trigger for sequential elements, i.e., registers. The clock signal synchronizes all registers within a circuit, which results in a synchronous digital circuit. Figure 2.1a shows a structure of registers (D-flip-flop) consisting of latches. A latch functions as a level-sensitive element, meaning it has two states, transparent or hold, depending on its clock signal at a high (1) or low (0) level. In contrast, a flip-flop operates as an edge-sensitive element, meaning it updates its state only on a specific clock edge. A standard flip-flop is typically constructed using a primary-secondary configuration, also known as a master-slave configuration, where two cascaded latches ensure stable and controlled data propagation. This structure, as illustrated in Figure 2.1a, serves as pipeline registers with combinational logic connecting to its input and output.

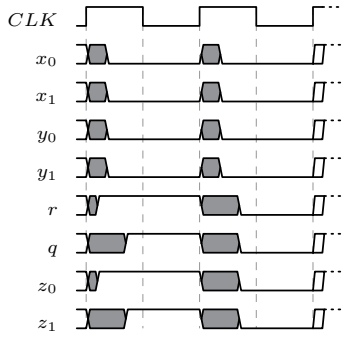
The timing characteristics of a flip-flop are essential to ensuring stable data registration and preventing errors, as illustrated in Figure 2.1b. The setup time refers to the minimum interval during which the input data D must remain stable before the rising edge of the clock signal. Similarly, the hold time specifies the duration after the clock edge during which D must remain unchanged. Another key timing parameter is the propagation delay t_{cq}, which measures the time taken for an input transition to be reflected at the output Q after the clock edge. The combinational logic connected to Q changes its value based on a stable Q output. The propagation delay t_d indicates the time needed for the values on Boolean gates to stabilize, during which glitches may occur. When a register is used, there are no glitches from the input D; thus, only the glitches from Q to the *out* port are considered. Thus, registers are critical in hardware secure masking by stopping glitch propagation.

Glitches in Digital Circuits

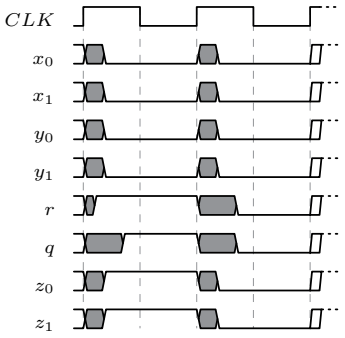
An example of how glitches contribute to leaking secret values from masking with the Trichina gate is shown in Figure 2.2. The vertical dashed line in Figure 2.2a represents an optional register stage inserted into the Trichina gate. This inserted register can prevent the glitches from propagating from the input directly to the output. Figures 2.2b and 2.2c show an illustrative example that indicates the difference caused by using a register stage to stop the glitch propagation through the combinational gates. On each signal, a time interval indicated in grey means metastability, when glitches occur. Based on whether the register stage is inserted, the output and intermediate signals, which values are described in Equation (2.3), have different transient behavior because of



(a) The Trichina gate and glitch propagation



(b) The timing diagram of the Trichina gate and glitches effect



(c) The timing diagram of the Trichina gate with registers

Figure 2.2: Glitches and leakages in masking with the Trichina gate

the glitch effect.

$$\begin{aligned} z_0 &= r \\ z_1 &= r \oplus x_0y_0 \oplus x_0y_1 \oplus x_1y_0 \oplus x_1y_1 \\ q &= r \oplus x_0y_0 \oplus x_0y_1 \end{aligned} \tag{2.3}$$

For simplicity, the input shares $\{x_0, x_1\}$ and $\{y_0, y_1\}$ are stable at zero and r is stable as one except the signal is on metastability. If there are no registers in between, as shown in Figure 2.2b, and the random bit r glitches even after the input shares are stable, the output value z_1 will change according to the value of r. When r becomes stable, intermediate signal q and output signal z_1 are still in metastability, so the dynamic power consumption related to the secret $x_0y_0 \oplus x_0y_1 \oplus x_1y_0 \oplus x_1y_1$ will be leaked. With registers, as depicted in Figure 2.2c, the circuit output does not depend on glitches on r, because the propagation of glitches is interrupted by the pipeline stage. However, the glitch correlated leakage is not entirely removed. The intermediate value q before the register stage can still glitch, causing the result of the `AND` gates x_0y_0 and x_0y_1 to be combined and leak the secret $y = y_0 + y_1$.

Performance Metrics and Clock Cycles

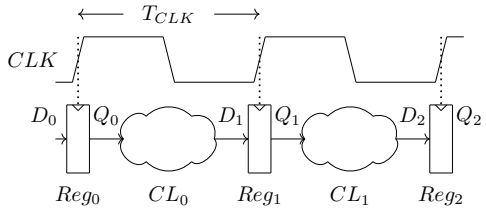


Figure 2.3: Synchronous digital circuits and their performance metrics

The critical path and the maximum frequency: The clock signal in a synchronous digital circuit controls the registers to update their states globally. The clock ensures that all registers update their output values to their past input simultaneously at regular intervals, i.e., when each positive clock edge arrives. Between these registers, combinational logic, including Boolean gates and interconnecting wires, performs the necessary computations of Boolean functions. Since these logic gates introduce propagation delays, the longest combinational path determines the maximum clock frequency $f_{max} = 1/T_{CLK}$, also known as the critical path.

Pipelines and throughput: To improve circuit performance, computation can be divided into sequential stages, each separated by registers, which results in a pipelined design. For instance, in a simple two-stage pipeline, as shown in

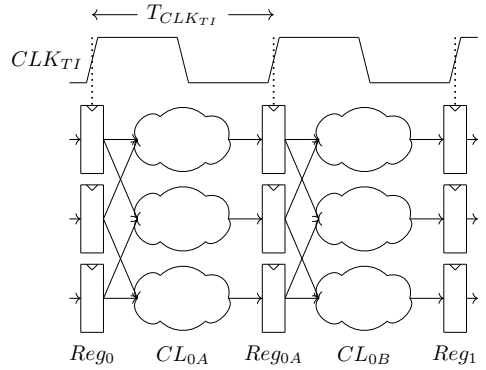


Figure 2.4: Hardware secure masking with registers and its timing and area overhead

Figure 2.3, an input value is stored in the first register Reg_0 during a single clock cycle, then processed through combinational logic, and stored in a second register Reg_1 in the next cycle. The final output becomes available in the last register Reg_2 after two cycles, meaning the latency for each input is two clock cycles. This technique enhances the throughput, i.e., the number of completed operations per cycle, by overlapping the execution of multiple operations. In this case, the throughput is one operation per cycle, ensuring continuous data processing without waiting for prior computations to finish. Thus, in digital hardware design that does not consider side-channel resistance, optimizing the trade-off between latency and throughput requires careful design of pipeline stage depth and critical path.

Performance metrics for masking: Hardware secure masking requires the (g)-extended probing security. This means that in the dth-order secure masked implementation, d probes can be placed in the circuits across different clock cycles and get values on the probed wires which connect the logic gates and registers. The placement of registers and the specification of Boolean functions between pipeline stages are algorithmically constrained by the dth-order secure (g)-extended probing security. Thus, the total number of clock cycles to complete one operation, such as an S-box evaluation, which corresponds to the *number-of-stages* in the circuit, is increased. As shown in Figure 2.4, the time delay of a glitch-immune masking implemented with registers is $T_{GSC_R} = \textit{number-of-stages} \times T_{CLK_{GSC_R}}$, where $T_{CLK_{GSC_R}}$ is the critical path of the masked implementation. Remember that the registers can separate the glitch propagation between stages, so the pipeline design will define the masked Boolean function that meets the security requirement. This makes the trade-off between area and latency a crucial consideration in the design of hardware secure masking.

2.5.2 Implementations of Hardware Secure Masking

Implementations for Physical Experiments

Hardware secure masking is often implemented on FPGAs for physical evaluations in laboratory environments. Commonly, the Sakura-X [100] and the Sakura-G [99] evaluation boards with Xilinx FPGA are used. However, extra caution should be taken when implementing the masking schemes on an FPGA as summarized by Gigerl et al. [49]. The design constraints `KEEP_HIERARCHY` or `DONT_TOUCH` are enabled, and `LUT_COMBINE` is disabled in Xilinx tools to prevent optimization across modules in the synthesis step, where multiple Boolean gates are combined into one LUT unit.

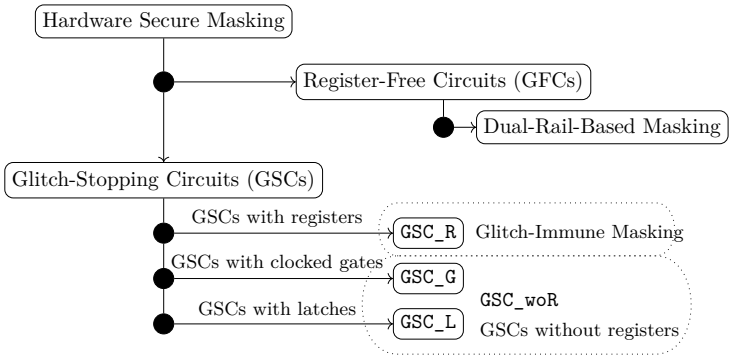


Figure 2.5: Categories of hardware secure masking

Performance Metric Evaluations

The area of the masked hardware is measured in gate equivalents (GE), i.e., the synthesized results using the Synopsys Design Compiler or YOSYS are normalized to the area of a 2-input `NAND` gate in a given cell library. This metric compares the area cost of different hardware implementations in terms of Boolean logic gates, while minimizing the impact of the cell library used. The critical path delay, or the maximum frequency, which depends on the timing metrics of the cell library used, is calculated using the Synopsys Design Compiler. Note that FPGA and ASIC have different characteristics in terms of the area and delay. However, the logic design step, where the masking schemes are implemented, is at the logic level and is shared between the hardware implementations on both platforms.

Categorizing Hardware Secure Masking

Mangard et al. [75] showed that glitches in hardware can cause leakages in power side channels, which the ISW masking overlooked. They also offered two strategies, which later Lammers et al. [82] named glitch-free and glitch-immune circuits. In this section, we will discuss the main strategies, which can be categorized into groups shown in Figure 2.5.

2.5.3 Glitch-Immune Masking

Given that predicting the glitching behaviour of a circuit is infeasible, algorithmic approaches to construct masking schemes that are (g)-extended probing secure

in the presence of glitches are developed. Each secret variable is represented by multiple shares, such that their reconstruction, e.g., bitwise `XOR`, equals the original variable. To ensure the dth-order (g)-extended probing security, any subset of at most d shares and their (g)-extended probed values provides no information about the secret variable. This security requirement constrains the minimum number of shares, so the corresponding glitch-immune masking is referred to as $d+1$ and $td+1$ sharing. Alternatively, the security requirements are formalized as security notions like (S)NI and PINI, which define the masked Boolean gates, also called secure gadgets.

Threshold Implementations

The first such masking scheme, being 1st-order (g)-extended probing secure, was introduced as the Threshold Implementations (TI) and later extended to any security order d [86, 18]. Theoretically, TI defines fundamental properties which make it provably secure under the (g)-extended probing model. Empirically, hardware implementations using TI are tested on physical platforms, which provide guidelines for creating glitch-immune masking.

TI is a masking scheme, which can be defined as shared Boolean functions, and satisfies properties of *correctness*, *non-completeness*, and *uniformity*. Masking schemes should produce the same calculation result as the unprotected circuit, thus ensuring correctness. Non-completeness ensures that even if glitches occur in the combinational circuit block, they will not leak all shares, preventing the adversary from reconstructing secret data. This property is fundamental in the design of glitch-immune masking schemes. The uniformity ensures that each share is independent of the secret. Meanwhile, TI keeps the uniformity from input sharing to output sharing so that the output shares can be securely fed into the next logic block.

Consider the target function $f : \mathbb{F}_2^n \to \mathbb{F}_2^m$ with n-bit inputs and m-bit outputs, where the input value $x \in \mathbb{F}_2^n$ and the output value $y \in \mathbb{F}_2^m$. A sharing (or masking) of the unshared value x, where the number of shares is s_x, is a tuple $\{x_1, \ldots, x_{s_x}\}$ such that $x = \sum_{i=1}^{s_x} x_i$, where each $x_i \in \mathbb{F}_2^n$. For every value x, there exist $2^{n(s_x-1)}$ possible sharings. Similarly, the output value y is shared into s_y shares.

The formal definition of TI and the properties are as follows, with detailed proofs summarized in the dissertation by Dr. Bilgin [17]. The masked Boolean function, i.e., TI, is defined as $\mathcal{F} : (\mathbb{F}_2^n)^{s_x} \to (\mathbb{F}_2^m)^{s_y}$, which consist of an s_y-tuple of coordinate functions $\mathcal{F}_j : (\mathbb{F}_2^n)^{s_x} \to \mathbb{F}_2^m$ for each $j \in \{1, \ldots, s_y\}$. The correctness of TI, with respect to the unprotected function $f : \mathbb{F}_2^n \to \mathbb{F}_2^m$, means that for every unshared value $x \in \mathbb{F}_2^n$ the sum of all the coordinate functions

$\mathcal{F}_j$ applied to a sharing of x results the same as the unprotected function f on the original value x, i.e.:

$$\sum_{j=1}^{s_y} \mathcal{F}_j(x1, \ldots, x_{s_x}) = f(x). \tag{2.4}$$

The dth-order non-completeness of TI $\mathcal{F}$ means there is at least one input share $x_i \in \{x_1, \ldots, x_{s_x}\}$ such that none of the combinations of up to d component functions $\mathcal{F}_j$ depends on x_i. The uniformity property is based on the definition of uniform sharing, which means each possible $\{x_1, \ldots, x_{s_x}\}$ sharing among the $2^{n(s_x-1)}$ number of valid sharings occurs with equal probability. If the sharing of input x is uniform, then the TI must preserve uniformity, which maps a uniform sharing to a uniform sharing $\{y_1, \ldots, y_{s_y}\}$ of output value y. To satisfy the above properties, a dth-order TI of an algebraic degree t unprotected vectorial Boolean function has minimum bounds for the number of input and output shares:

$$s_i \geq td + 1, \; s_o \geq \binom{td+1}{t}. \tag{2.5}$$

TI transforms the unprotected Boolean function into the shared Boolean functions, like ISW multiplication, but in a way that mitigates the effect of glitches on protection against SCA. The shared Boolean functions are implemented in digital combinational circuits that are connected with registers in between to prevent glitch propagation between sub-circuits. TI are also referred to as $td+1$ masking for the reason that the dth-order non-completeness property for a shared Boolean function of algebraic degree t is achieved with $td + 1$ shares.

Non-Completeness and Glitch Immunity

The properties of TI are inherited by the glitch-immune masking schemes and gadgets, e.g., the Consolidated Masking Scheme (CMS) and the Domain-Oriented Masking (DOM) [94, 57]. These schemes divide the masked circuits into expansion and compression layers, which are separated by registers. After the expansion layer, randomness is applied to ensure the uniformity property. Meanwhile, because the register layer stops the glitches from propagating between the expansion and compression layer, the dth-order non-completeness property can be achieved with only $d + 1$ shares.

For example, consider a 2-input `AND` gate, $z = xy$, where each input has two shares x_1, x_2 and y_1, y_2. The first-order $d + 1$ masking can be expressed by the following equations, which include CMS as Equations (2.6) and (2.7) and

DOM as Equation (2.8). The vertical dashed line represents register stages. In Equation (2.8), the horizontal line represents the separation of share domains.

$$\begin{array}{ll|l} x_1y_1 \oplus r_1 & \rightarrow z'_1 & \\ x_1y_2 \oplus r_2 & \rightarrow z'_2 & z'_1 \oplus z'_2 = z_1 \\ x_2y_1 \oplus r_3 & \rightarrow z'_3 & z'_3 \oplus z'_4 = z_2 \\ x_2y_2 \oplus r_1 \oplus r_2 \oplus r_3 & \rightarrow z'_4 & \end{array} \tag{2.6}$$

$$\begin{array}{ll|l} x_1y_1 \oplus r_1 \oplus r_2 & \rightarrow z'_1 & \\ x_1y_2 \oplus r_2 \oplus r_3 & \rightarrow z'_2 & z'_1 \oplus z'_2 = z_1 \\ x_2y_1 \oplus r_3 \oplus r_4 & \rightarrow z'_3 & z'_3 \oplus z'_4 = z_2 \\ x_2y_2 \oplus r_4 \oplus r_1 & \rightarrow z'_4 & \end{array} \tag{2.7}$$

$$\begin{array}{ll|l} x_1y_1 \oplus r & \rightarrow z'_1 & \\ x_1y_2 & \rightarrow z'_2 & z'_1 \oplus z'_2 = z_1 \\ \hline x_2y_1 & \rightarrow z'_3 & z'_3 \oplus z'_4 = z_2 \\ x_2y_2 \oplus r & \rightarrow z'_4 & \end{array} \tag{2.8}$$

Randomness and Uniformity

Randomness usage in CMS and DOM is essential to ensure uniformity of intermediate values and prevent leakage on the output shares. TI can be vectorial Boolean functions that are built with the uniformity property, ensuring the sharing has no statistical dependency on the secret value from the input to the output. However, constructing the uniformly shared Boolean functions is not straightforward. Common strategies are either increasing the number of input shares or inserting random bits into coordinate functions as shown in Equation (2.6) or Equation (2.7). The former is still not yet systematic, while the latter costs additional randomness.

Daemen [38] introduced a simple and efficient technique for constructing uniform TI that is 1st-order (g)-extended probing secure of any invertible S-box. Figure 2.6 shows an example of the changing-of-the-guards of a shared S-box. $\bar{S}$ is a masking of the S-box S with three shares, which is correct and non-complete, i.e., $\bar{S} = \{S_1(a_2, a_3), S_2(a_1, a_3), S_3(a_1, a_2)\}$ and $S(a_1 + a_2 + a_3) = S_1(a_2, a_3) + S_2(a_1, a_3) + S_3(a_1, a_2)$, and $\{a_1, a_2, a_3\}$ is a uniform sharing of the input secret. Two of the three shares are the guards for other functions $b'_1 = a_1$ and $b'_2 = a_2$, so the guards for $\bar{S}$ are uniform shares reused from other shared computations. These shares, which are used equivalently as random bits, make the output shares $\{a'_1, a'_2, a'_3\}$ uniform as

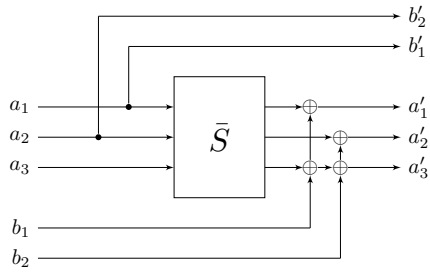


Figure 2.6: The changing-of-the-guards of the shared S-box $\bar{S}$ with three shares

shown in Equation (2.9).

$$\begin{aligned} a'_1 &= S_1(a_2, a_3) \oplus b_1 \\ a'_2 &= S_2(a_1, a_3) \oplus b_2 \\ a'_3 &= S_3(a_1, a_2) \oplus b_1 \oplus b_2 \end{aligned} \tag{2.9}$$

2.5.4 Glitch-Free Circuits

In secure hardware design, glitch-free circuits provide another effective countermeasure against side-channel attacks by eliminating glitches that would otherwise cause power leakage, so their power consumption remains nearly constant regardless of the input. Unlike glitch-immune masking schemes, which algorithmically ensure provable security under the (g)-extended probing model, glitch-free circuits eliminate glitches at the hardware level. By eliminating the need for additional registers, they reduce both latency and implementation overhead. This fundamental difference reduces the latency overhead associated with glitch-immune masking requiring registers while also achieving power equalization, meaning that the circuit's power consumption remains independent of the processed data.

Dual-Rail Pre-Charge Logic

One well-established method for designing glitch-free circuits is Dual-Rail Pre-Charge (DRP) logic, which ensures a uniform power consumption pattern and always consumes the maximum amount in each cycle. In DRP logic, each signal is transmitted using two complementary wires representing a single bit, which are firstly pre-charged and secondly switched to the intended value during

computation. This is also called differential encoding, in which the values of the two wires are either (0,1) representing a logic `1` or (1,0) representing a logic `0`. Thus, the original wire carries the actual data, and the complementary wire carries the inverse of the data. If two wires carry the same value, the actual data is invalid. In addition, the pre-charge phase sets the wires to the same value initially representing the *null* state, i.e., both the actual data and the complementary wire are (0,0). The pre-charge phase is followed by the evaluation phase, where the wires are switched to the values of the logic gates from (0,0) to either (0,1) or (1,0), which ensures the power consumption is constant in either logic value. Since power consumption is independent of the actual data, an attacker cannot infer secret values based on power consumption variations. Thus, the four possible wire states define the representation of a single-bit value, as shown in Table 2.1.

Original Value	Dual-rail encoding	Computing Phase
null	$(0, 0)$	Pre-charge the circuit to zero
1	$(1, 0)$	Compute the logic value
0	$(0, 1)$	Compute the logic value
invalid	$(1, 1)$	Invalid state

Table 2.1: Dual-rail encoding of one-bit value in DRP logics

Wave Dynamic Differential Logic

The Wave Dynamic Differential Logic (WDDL) [110] is an implementation of DRP logic cells that serves as a countermeasure against power analysis attacks, as shown in Figure 2.7. A pre-charge signal regulates circuit operation by enforcing a pre-charge phase, during which all signals are set to (0,0), followed by an evaluation phase, where signals transition to either (0,1) or (1,0). The pre-charge operation is executed at the first step of the combinational circuit, and the permissible logic operations are restricted to `AND`, `OR`, and `NOT` (implemented through wiring) in WDDL. Furthermore, WDDL gates are built from standard Boolean logic gates, like two-input `AND` and `OR` gates, which do not need additional customization at the transistor level. WDDL gates produce complementary outputs, denoted as $(q, \bar{q})$, ensuring balanced signal transitions and cascaded to the next gates also in DRP encoding, that can be composed to a larger logic circuit. WDDL can be integrated into the standard silicon design process and provide SCA resistance. WDDL ensures that the number of signal transitions occurring within a circuit during each operation cycle remains constant, not depending on the input values. Since dynamic power consumption in CMOS circuits is determined mainly by the number of gate transitions, this

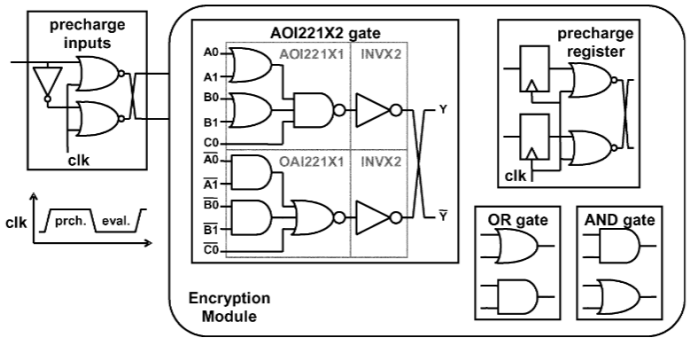


Figure 2.7: Wave Dynamic Differential Logic [60]

feature effectively mitigates data-dependent power variations. Consequently, WDDL was proposed to be an effective countermeasure against SCA by keeping the circuit's power consumption constant, thereby preventing attackers from extracting sensitive information based on side-channel leakages.

Factors of Leakages in DRP Logics

The P&R of the logic circuits plays a critical role in the power consumption of the DRP logic. For instance, the power leakage in WDDL can be caused by imbalanced loading capacitance and delays on wire [108, 109]. In an ideal DRP, a data value of logic `0` and logic `1` would produce identical timing on both the complementary wires. However, under imbalanced delays, the arrival time of a signal at the next gate depends on the data value. In other words, the pair of DRP is no longer perfectly synchronized, which breaks its fundamental assumption that both rails transition at the same time.

Loading Capacitance: The difference in loading capacitance values of the two complementary output wires can cause the power consumption during the CMOS gates transition to vary. Thus, although the total number of transitions from the pre-charged state (0,0) to either (0,1) or (1,0) on the two complementary wires is the same for any input value, the power consumption result can be dependent on the secret value. This source of leakage can be unintentional and layout-dependent because the P&R step is unpredictable. To mitigate the imbalance loading capacitance, manual or semi-automatic P&R techniques [111, 58] were developed as DPA countermeasures, but this also means the automated process was omitted.

Imbalanced Delays: The timing of the differential signals can cause data-dependent power consumption. In practice, maintaining perfect symmetry between the complementary rails of each signal is challenging. Imbalanced delays refer to differences in propagation delay between a signal's two rails after P&R, i.e., when one wire of a dual-rail pair reaches a gate later or earlier than its complement. This timing mismatch can arise from unequal routing paths between wires and various gate delays on a complicated combinational differential logic network.

Early Propagation: Early propagation refers to a phenomenon where a subset of DRP inputs can determine the output before all input signals arrive, which happens under the imbalanced delay of routed rails. This timing difference of each complementary gate does not change the computation result. However, due to the input values or delay time in a cycle, the switching time of the output can be dependent on whether the input comes early. Thus, early propagation reveals information because the timing or sequence of the output transition encodes part of the secret data, making the power consumption related to the signal values.

Countermeasures: Countermeasures have also been attempted by applying masking with DRP logic, such as the Masked Dual-Rail Pre-Charge Logic (MDPL) [89], or by modifying DRP logic gates [79, 16]. However, MDPL only used a one-bit mask and its complement for all input signals and constructed the DRP logic with majority gates. Suzuki and Saeki [108] proved that this masking method was not able to remove the inevitable timing difference resulting in SCA leakage. The modifications to DRP gates eliminated the early propagation effect in the pre-charge and evaluation phase, but still assumed no imbalanced delay introduced by P&R.

DRP Logics with Glitch-Immune Masking

Since DRP-like solutions still exhibited data-dependent early signal propagation leading to potential leakage, research shifted towards enhancing these solutions by incorporating glitch-immune masking. Motivated by the early propagation issue and the need for low-latency hardware secure masking, Moradi and Schneider [80] proposed integrating asynchronous WDDL with first-order masking. In recent years, solutions such as LUT-based Masked Dual-Rail with Pre-charge Logic (LMDPL) [97], Self-Synchronized Masking (SESYM) [84], and self-timed masking [107] have been developed. Section 3.4.1 provides a detailed discussion of these low-latency glitch-free circuit designs.

Incompatible with the Probing Model

Müller et al. [82] analyzed the security of these solutions in the presence of imbalanced delays on the dual-rail wires. Such imbalances can cause data-dependent time differences during the evaluation of differential logic, which can be exploited by TVLA. Among the solutions to this problem, the LMDPL considers the balanced routing issue and is shown to be secure by the authors' security analysis under the glitch-extended probing model.

In the security analysis of the DRP logics, the main issue is that the circuit model adopted by the (robust) probing model is a directed acyclic graph, which does not take the pre-charge and evaluation phases into account. In addition to the clock signal, DRP logics also include the global pre-charge signal, which brings the early propagation effect and imbalanced routing effect leading to SCA leakages. Müller et al. [82] pointed out that these effects are either ignored or analysed informally without rigorous justification.

2.6 Statistical Leakage Assessments

The leakage assessment methods are motivated by the fact that DPA and similar power analysis attacks exploit correlations between power consumption and intermediate values. Rather than testing against all potential power analysis attacks, leakage assessment methods focus on identifying leakage points in power traces, regardless of whether the leakages can be immediately exploited. In these methods, the power traces are either collected from physical experiments or generated via simulations.

Physical Tests

Test Vector Leakage Assessment (TVLA) originally proposed by Goodwill et al. [52] is a widely used methodology for evaluating cryptographic implementations against SCA. Instead of executing all possible attacks on the device under test, TVLA focuses on detecting any statistically significant leakage dependent on sensitive information in side-channel measurements, e.g., power traces, even if those leakages are not immediately exploitable. In other words, TVLA serves as a leakage assessment, which checks for the presence of secret-dependent information in the physical leakage, rather than attempting to extract the key directly. This approach has gained popularity due to its simplicity and broad applicability across different cryptographic designs.

TVLA testing is typically performed by applying Welch's t-test to compare two sets of collected power traces. In the non-specific TVLA, also known as the fixed-vs-random test, one set of traces $\mathcal{S}_0$ is measured using fixed plaintexts given a fixed secret key, and the other set $\mathcal{S}_1$ is measured using random plaintexts with the same key. This test determines whether any data-dependent variation exists without targeting any specific intermediate value. The null hypothesis is that the two trace sets have equal means, i.e., no leakage, and a high t-statistic indicates that this hypothesis should be rejected. The two sets of measurements are compared using the t-test statistic:

$$t = \frac{\mu(\mathcal{S}_0) - \mu(\mathcal{S}_1)}{\sqrt{\frac{\sigma^2(\mathcal{S}_0)}{|\mathcal{S}_0|} + \frac{\sigma^2(\mathcal{S}_1)}{|\mathcal{S}_1|}}} \tag{2.10}$$

where μ and σ^2 denote the sample mean and variance, respectively, and $|\mathcal{S}_0|$, $|\mathcal{S}_1|$ are the number of traces in each set. In practice, a threshold for the t-statistic, commonly $|t| > 4.5$, is used to decide whether the hypothesis is rejected. If the t-statistic exceeds 4.5 at any sample, it implies a significant difference in means and thus evidence of first-order leakage. If $|t|$ remains below the threshold for all samples, the implementation is considered to have passed, and no first-order leak is detected at the given confidence level. Any t value outside the ± 4.5 bound causes the null hypothesis to be rejected, flagging a potential leakage point. These detected points of leakage often correspond to time samples where a first-order DPA could be focused and exploit sensitive intermediate variables. To cover higher-order DPA attacks, the TVLA can be extended to higher-order cases, which involve pre-processing steps that combine more points in a power trace and perform a t-test on higher-order statistical moments, which is called d^{th}-order multivariate TVLA. Since the processing of the multivariate test can be time-consuming and often requires down-sampling of data points, on-the-fly computing methods and open source software have been developed by the research community [102, 27].

Pre-Silicon Statistical Verification

The pre-silicon phase refers to the stage at which the design is verified prior to chip fabrication. At this stage, the design is tested in a virtual environment for its side-channel security, following logic synthesis at the logic level or P&R at the physical level. Verification can be performed using either simulation or formal methods. In the context of side-channel analysis, PROLEAD by Müller et al. [83] is a simulation-based tool designed to verify a circuit's compliance with the (g, t)-robust probing model. Rather than exhaustively verifying every possible input, PROLEAD simulates the circuit with various shared inputs and examines the dependencies between the probe observations. To determine these

dependencies, the tool utilizes the G-test [59], which is considered as effective as TVLA in identifying potential leakage points. Formal verification tools such as SILVER [67] and MaskVerif [9] can be used to verify masked Boolean circuits or gadgets. Nevertheless, they are not scalable to large circuit designs.

Chapter 3

Low-Latency Masking in Hardware

This chapter discusses the timing overhead of hardware secure masking schemes and methods to reduce this latency. First, the concepts of latency-in-cycle and latency-in-time are explained in the context of masking, which represent two measurements of delays in masked implementations. Second, Low-Latency Threshold Implementations (LLTI), an extension of traditional TI, is discussed to reduce the latency-in-cycle. Thirdly, a circuit-level technique, Glitch-Stopping Circuits (GSC), is proposed to reduce the latency-in-cycle by replacing the register stages with clocked logic gates or latches, as illustrated in Figure 3.9.

3.1 Latency in Hardware Secure Masking

Hardware secure masking schemes require registers to stop glitch propagation and achieve sufficient protection against SCA, thus introducing latency in the hardware implementations of these schemes. As discussed in Section 2.5.1, latency is a critical timing overhead that can be defined in two ways: in terms of clock cycles or total propagation delay time. The choice of measuring latency depends on the specific use cases and constraints of the masked implementations. For instance, in high-performance computing systems, clock frequencies can be increased to the gigahertz range due to the higher thermal budget available in dedicated computing centers, which are equipped with advanced cooling solutions to dissipate the excess heat. This allows such systems to maximize their power consumption for compute-intensive applications. Thus, latency-in-

time should be considered so that hardware secure masking does not introduce a bottleneck on the critical path. In contrast, embedded systems prioritize energy efficiency and keep the clock frequency low while ensuring that performance requirements are met. They often operate in environments with limited cooling capabilities and strict power consumption constraints. Maintaining a lower clock speed reduces heat dissipation and power consumption to extend battery life and improve system reliability, which is crucial for applications in IoT applications.

If the clock frequency is constrained, the implementation block cannot operate faster than the rest of the integrated circuit. In that case, the number of clock cycles, i.e., cycle-wise latency, is a more relevant metric for the timing overhead. On the other hand, time-wise latency, representing the delay from the circuit's input to its output, is preferable in scenarios without clock frequency limitations, where both the maximum clock frequency and the number of clock cycles are considered. With these definitions of latency, this chapter discusses cycle and time-wise latency measurements to provide a comprehensive view of the circuit's performance. As is shown in Figure 2.4, the cycle-wise latency is the number of clock cycles required to complete the masked implementation, therefore, the total latency-in-time is given by $T_{\texttt{GSC_R}} = \text{number-of-stages} \times T_{CLK_{\texttt{GSC_R}}}$, with $T_{CLK_{\texttt{GSC_R}}}$ being the clock period, determined by the critical path of the glitch-immune masking implemented with registers.

3.1.1 Timing Overhead of Glitch-Free Circuits

As discussed by Section 2.5.4, glitch-free circuits exhibit no internal glitches during their operation by utilizing dual-rail logic with a pre-charging phase. DRP and its derivatives operate on the precharge and clock signal. Because there is inherently no glitch that happens in DRP logic, internal register stages are not required, which offers a significant advantage regarding reduced latency-in-cycles, although latency-in-time remains a constraint. Furthermore, when comparing the timing overhead of the GFC with hardware secure masking implemented in synchronous digital circuits, the pre-charging phase of the DRP like circuits should be considered. In the pre-charging phase and the evaluation phase, the GFC output should be stable given a pre-charge signal arrives, denoted as T_{GFC}. The DRP logic also interfaces with a synchronous digital logic, where the registers or memory elements controlled by a global clock signal store the state or key values of the cipher. Thus, as illustrated in Figure 3.1, if the latency is evaluated in clock cycles, this global clock is considered. The latency-in-time includes the total operation delays of the pre-charging and evaluation phase, which can be $2 \times T_{GFC}$.

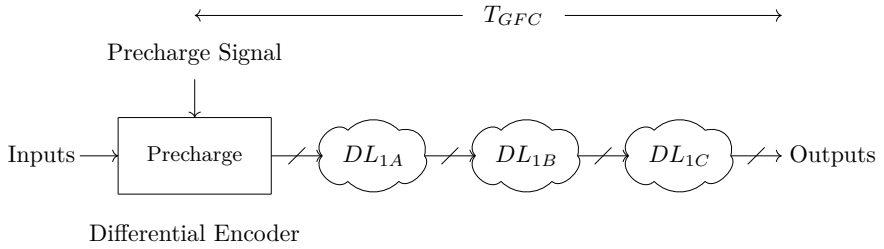


Figure 3.1: DRP logic and its timing metric

3.1.2 Timing Overhead of Glitch-Immune Masking

In glitch-immune masking with registers, the latency measured in clock cycles corresponds to the number of circuit stages requiring separation by registers. In contrast, time-wise latency is introduced by the computation of shared Boolean functions, which in hardware are combinational logic circuit blocks between register stages. Depending on the masking scheme used and the number of decomposition levels, the time-wise latency, which in digital logic is the propagation delay of the circuit, can be significant. Recall from Figure 2.5 that the Glitch-Immune Masking category includes the TI and DOM schemes, which are also known as $td+1$ and $d+1$ masking according to their number of input shares, where t is the algebraic degree of the masked Boolean function and d is the security order under the (g)-extended probing model.

3.2 Glitch-Immune Masking of Boolean Functions

This section discusses the construction of glitch-immune masking that achieves d^{th}-order (g)-extended probing security for Boolean functions. There are two distinct approaches. The first follows a top-down methodology, where high-degree functions are decomposed into lower-degree ones, and secure masked implementations are built for these subcircuits before composition. The second, a bottom-up approach, replaces logic gates from unprotected implementations with composable secure gadgets. The top-down approach leverages glitch-immune properties such as non-completeness, uniformity, and the use of registers to prevent glitch propagation. In contrast, the gadget-based approach relies on simulability-based proofs and direct composition of secure building blocks.

3.2.1 Complexity of Boolean Circuits

For an unprotected Boolean circuit in CMOS technology, a NAND gate is the fundamental element constructing large Boolean circuits. A NAND gate consists of four transistors, and the size of the Boolean circuit can be calculated as the number of NAND gates, also referred to as gate equivalents (GE). According to the Nangate Open Cell Library [85], the area costs of the two-input AND and XOR gates are 1.333 GE and 2 GE, corresponding to the non-linear and linear operations, respectively.

However, when implementing hardware secure masking using Boolean Masking, non-linear operations dominate the area cost. Boolean masking of linear layers is trivial because the shares are split with XOR operations, and there are no cross products of different shares. Therefore, a masked XOR requires $d+1$ or $td+1$ XOR gates. Consider an AND gate $y = a \cdot b$ and each variable is masked with s number of shares, $s = d+1$ or $s = td+1$. The masked implementation of an AND gate requires s^2 AND gates. The number of XOR gates depends on the Boolean masking scheme designed. Thus, the cost of masking AND gates dominates in hardware secure masking.

3.2.2 Multiplicative Depth

Several AND gates can be cascaded from an input to an output of a Boolean circuit. The maximum number of two-input AND gates on any such path can be described as multiplicative depth. The multiplicative depth indicates the latency cost of the masked implementation. From the perspective of Boolean functions, the property of multiplicative depth is defined as the algebraic degree of the functions. The maximal algebraic degree of an n-bit permutation, which is often used to design S-boxes in symmetric ciphers, is $n-1$.

Similar to the masked two-input AND gates, a monomial of high algebraic degree t is decomposed into s^t of shared monomial products, where s is the number of shares of the masking scheme. Thus, when the degree t is high, masking the Boolean function becomes infeasible. To mask a 4-bit S-box, which is a cubic function, it can either be shared directly or decomposed into two layers of quadratic functions. The AES 128-bit state is represented as a 4×4 matrix of elements in $\mathbb{F}_{2^8}$. The AES S-box is equivalent to an inversion in the field $\mathbb{F}_{2^8}$ followed by an invertible affine transformation, thus having an algebraic degree $t = 7$. Applying direct masking to such a high-degree function is infeasible.

3.2.3 Function Decompositions

High algebraic degree functions $f : \{0,1\}^n \to \{0,1\}$ can be decomposed into lower degree Boolean Functions. Given a Boolean function $f(x_1, x_2, \ldots, x_n)$, the decomposition seeks intermediate functions $g : \{0,1\}^m \to \{0,1\}^k$ and $h : \{0,1\}^k \to \{0,1\}$, and can be represented as:

$$f(x_1, x_2, \ldots, x_n) = h(g(x_1, x_2, \ldots, x_m), x_{m+1}, \ldots, x_n) \tag{3.1}$$

The Boolean function decomposition can be expressed using the composition operator $\circ$ $f = h \circ g$ Here, function g processes a subset of the inputs, and h maps the output of g along with any remaining variables to produce the final result. Boolean function decomposition is fundamental in circuit optimization, as it reduces circuit complexity and improves hardware efficiency.

Tower Field Decomposition: A tower field decomposition represents $\mathbb{F}_{2^8}$ as a composition of smaller fields, so the high-cost 8-bit inversion can be implemented in low-degree field operations with fewer gate counts in hardware. For example, Satoh et al. [98] and Canright [25] reported implementations of AES using this structure. Equation (3.2) and Figure 3.2 show the details of the tower field decomposition by Canright. Let the 8-bit of a variable x be $(x_8, ..., x_1)$, where x_8 denotes the MSB and x_1 the LSB.

- Input and output isomorphisms: The first operation is a change of basis through a linear map. The last inverse linear map includes the AES affine transformation, followed by the addition of the constant `0x63`.
- Finite field multiplications: The tower field decomposition uses multipliers over finite fields $\mathbb{F}_{2^2}$ and $\mathbb{F}_{2^4}$.
- Scaling functions: A "square scale function" and a "scale" function are linear mappings over 4-bits and 2-bits, respectively.
- Two-bit inversion: The inversion in $\mathbb{F}_{2^2}$ with a normal basis is linear, which corresponds to swapping bits in the binary representations of the field elements $(y_1, y_2) = (x_2, x_1)$. The inversion does not include logic gates.

$$
\begin{aligned}
&\textit{Linear map}\\
&y_8 = a_8 \oplus a_7 \oplus a_6 \oplus a_3 \oplus a_2 \oplus a_1 \qquad y_4 = a_8 \oplus a_5 \oplus a_4 \oplus a_2 \oplus a_1\\
&y_7 = a_7 \oplus a_6 \oplus a_5 \oplus a_1 \qquad y_3 = a_1\\
&y_6 = a_7 \oplus a_6 \oplus a_2 \oplus a_1 \qquad y_2 = a_7 \oplus a_6 \oplus a_1\\
&y_5 = a_8 \oplus a_7 \oplus a_6 \oplus a_1 \qquad y_1 = a_7 \oplus a_4 \oplus a_3 \oplus a_2 \oplus a_1\\
&\textit{Inverse linear map}\\
&y_8 = u_6 \oplus u_4 \qquad y_4 = u_8 \oplus u_7 \oplus u_6 \oplus u_5 \oplus u_4\\
&y_7 = u_8 \oplus u_4 \qquad y_3 = u_7 \oplus u_6 \oplus u_4 \oplus u_3 \oplus u_1\\
&y_6 = u_7 \oplus u_1 \qquad y_2 = u_6 \oplus u_5 \oplus u_2\\
&y_5 = u_8 \oplus u_6 \oplus u_4 \qquad y_1 = u_7 \oplus u_5 \oplus u_2\\
&\textit{Multiplier over } \mathbb{F}_{2^2}\\
&c_2 = (a_2 \oplus a_1)(b_2 \oplus b_1) \oplus a_2 b_2\\
&c_1 = (a_2 \oplus a_1)(b_2 \oplus b_1) \oplus a_1 b_1\\
&\textit{Multiplier over } \mathbb{F}_{2^4}\\
&\quad c_4 = \; a_4b_4 \oplus a_2b_4 \oplus a_1b_4 \oplus a_3b_3 \oplus a_2b_3 \oplus a_4b_2 \oplus a_3b_2 \oplus a_2b_2 \oplus a_1b_2\\
&\qquad\qquad \oplus a_4b_1 \oplus a_2b_1\\
&\quad c_3 = \; a_3b_4 \oplus a_2b_4 \oplus a_4b_3 \oplus a_3b_3 \oplus a_1b_3 \oplus a_4b_2 \oplus a_2b_2 \oplus a_3b_1 \oplus a_1b_1\\
&\quad c_2 = \; a_4b_4 \oplus a_3b_4 \oplus a_2b_4 \oplus a_1b_4 \oplus a_4b_3 \oplus a_2b_3 \oplus a_4b_2 \oplus a_3b_2 \oplus a_2b_2\\
&\qquad\qquad \oplus a_4b_1 \oplus a_1b_1\\
&\quad c_1 = \; a_4b_4 \oplus a_2b_4 \oplus a_3b_3 \oplus a_1b_3 \oplus a_4b_2 \oplus a_1b_2 \oplus a_3b_1 \oplus a_2b_1 \oplus a_1b_1\\
&\textit{Square scaling}\\
&(y_4, y_3, y_2, y_1) = (x_1 \oplus x_3, x_2 \oplus x_4, x_1 \oplus x_2, x_1)\\
&\textit{Scaling operation}\\
&(y_2, y_1) = (x_2, x_1 \oplus x_2)
\end{aligned}
\tag{3.2}
$$

Cubic Decomposition: There are also decompositions of the AES S-box into cubic functions [116, 115]. Equation (3.3) and Figure 3.3 show the details of a cubic decomposition. Define the function of the AES S-box as $F : \{0,1\}^8 \to \{0,1\}^8$, and let it be decomposed into two functions $G : \{0,1\}^8 \to \{0,1\}^8$ and $H : \{0,1\}^8 \to \{0,1\}^8$, where the input is an element of the finite field $x \in \mathbb{F}_{2^8}$.

$$
\begin{array}{ll}
\text{S-box} & F(x) = x^{-1} = x^{254} = (x^{26})^{49}\\
\text{Layer 1} & G(x) = x^{26}\\
\text{Layer 2} & H(x) = x^{49}
\end{array}
\tag{3.3}
$$

Composition: The composition of the sub-functions in hardware can be parallel or sequential. Let $\mathcal{C}_1$ and $\mathcal{C}_2$ be two circuits. The sequential composition of the two circuits can be denoted $\mathcal{C} = \mathcal{C}_2 \circ \mathcal{C}_1$. The inputs of $\mathcal{C}_2$ are directly

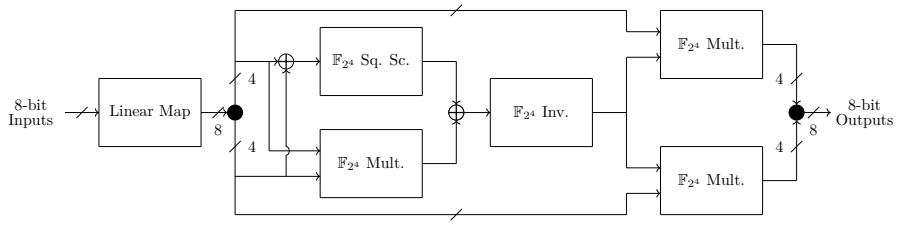


Figure 3.2: The tower field decomposition of the AES S-box

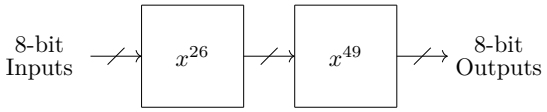


Figure 3.3: The cubic decomposition of the AES S-box

connected to the outputs of $\mathcal{C}_1$ by wire so the number of outputs of $\mathcal{C}_1$ is equal to the number of inputs of $\mathcal{C}_2$. Accordingly, the function implemented by $\mathcal{C}$ can be represented as the Boolean function composition, i.e., $f = f_2 \circ f_1$ as in Equation (3.4), which is the case of Figure 3.3.

$$f(x_1, x_2, \ldots, x_m) = f_2(f_1(x_1, x_2, \ldots, x_m)) \tag{3.4}$$

The parallel composition of the two circuits can be denoted as $\mathcal{C} = \mathcal{C}_1 \| \mathcal{C}_2$. The input or output wires of $\mathcal{C}$ are the concatenation of the inputs, or respectively outputs, of the circuits $\mathcal{C}_1$ and $\mathcal{C}_2$. The inputs of $\mathcal{C}_1$ and $\mathcal{C}_2$ may serve as shared inputs to both circuits. This happens in the tower field decomposition where an 8-bit wire is split into 4-bit nibbles for the $\mathbb{F}_{2^4}$ finite field multiplications and, before the inverse linear mapping, the 4-bit wires are concatenated back, as shown in Figure 3.2. Equation (3.1) is the mix of parallel and sequential composition, where the extra input wires of h are composed in parallel with the outputs of g while the remaining inputs of h are sequentially composed with the outputs of g.

3.2.4 Constructing Glitch-Immune Masking

After the high-degree function is decomposed, such as the tower field decomposition in Equation (3.2), each sub-function is constructed with its glitch-immune masking. Let each Boolean function be $f : \mathbb{F}_2^t \to \mathbb{F}_2$ representing a t-to-1 Boolean circuit. Each function is represented in its Algebraic Normal Form (ANF), as shown in Equation (2.1).

$d+1$ Masking

For each monomial in ANF, which is equivalent to the n input `AND` gate, the sharing in glitch-immune masking can be derived from its definition. For $d+1$ masking, the procedure for masking the monomials to achieve d^{th}-order security can be described as follows:

1. The $d+1$ input shares of t variables are expanded into $(d+1)^t$ cross products.
2. The refreshing layer is applied to all these cross products. In the case of CMS, randomness is added to the cross products following Equation (2.7) or Equation (2.6). In the case of DOM, randomness is applied to each domain. Across domains, the randomness cost can be saved, thus the total randomness cost is approximately divided by t compared to CMS.
3. After the register stage, compress the $(d+1)^t$ shares back into $d+1$ output shares.

For example, n-share $d+1$ masking of a two-input `AND` monomial is given in Equation (3.6) and Equation (3.5) for CMS and DOM, respectively.

$$\begin{array}{ccccc:c} x_1y_1 & x_1y_2 \oplus r_1 & x_1y_3 \oplus r_2 & \cdots & x_1y_n \oplus r_{n-1} & \oplus \rightarrow z_1 \\ x_2y_1 \oplus r_1 & x_2y_2 & x_2y_3 \oplus r_n & \cdots & x_2y_n \oplus r_{2n-2} & \oplus \rightarrow z_2 \\ x_3y_1 \oplus r_2 & x_3y_2 \oplus r_n & \cdots & \cdots & \cdots & \cdots \\ \cdots & \cdots & \cdots & \cdots & \cdots & \cdots \\ x_ny_1 \oplus r_{n-1} & x_ny_2 \oplus r_{2n-2} & \cdots & \cdots & x_ny_n & \oplus \rightarrow z_n \end{array} \tag{3.5}$$

$$\begin{array}{ccccc:c} x_1y_1 \oplus r_1 & x_1y_2 \oplus r_2 & x_1y_3 \oplus r_3 & \cdots & x_1y_n \oplus r_n & \oplus \rightarrow z_1 \\ x_2y_1 \oplus r_{n+1} & x_2y_2 \oplus r_{n+2} & x_2y_3 \oplus r_{n+3} & \cdots & x_2y_n \oplus r_{2n} & \oplus \rightarrow z_2 \\ x_3y_1 \oplus r_{2n+1} & \cdots & \cdots & \cdots & \cdots & \cdots \\ \cdots & \cdots & \cdots & \cdots & \cdots & \cdots \\ x_ny_1 \oplus r_{n(n-1)+1} & \cdots & \cdots & \cdots & x_ny_n \oplus r_{n^2} & \oplus \rightarrow z_n \end{array} \tag{3.6}$$

A Boolean function in ANF is expressed as a sum of monomials of degree up to t, e.g., the function has $s \geq t$ variables. For efficiency, randomness, i.e., the refreshing layer $\mathcal{R}$, can be applied after shared monomials $\mathcal{M}$ are combined, since two monomials in ANF do not have the same subset of variables. After the register indicated in the dashed line, the compression layer $\mathcal{C}$ returns the output shares. Thus, the resulting $d+1$ glitch-immune masking, i.e., DOM or CMS, is in the form of Equation (3.7).

$$\mathcal{M}_1 \oplus \mathcal{M}_2 \oplus \cdots \oplus \mathcal{R} \;\vdots\; \mathcal{C} \tag{3.7}$$

Because this ANF is obtained after the function decomposition process, its lower algebraic degree allows for the manual assignment of each shared monomial to shared coordinate functions. For large t, Bozilov et al. [23] and De Meyer et al. [115] proposed optimization and searching algorithms.

Threshold Implementations

For TI, both the algebraic degree t and the security order d can increase the number of input shares and consequently the complexity of the masking scheme. Non-completeness and uniformity properties are challenging to achieve, and they even depend on the target function. For example, the first-order TIs of the `AND` gate $z = a \cdot b$ and the `AND-XOR` gate $z = a \cdot b \oplus c$, also called the Toffoli gate, are shown in Equation (3.9) and Equation (3.8). The function's algebraic degree is two, thus it requires three input and output shares for first-order security, i.e., $t + 1 = 3$. It is notable that both masking schemes of the `AND` and the Toffoli gate have similar cost and use the same number of shares, but differ in their approach to achieving the uniformity property of TI. The direct sharing on the `AND` gate $z = a \cdot b$ as Equation (3.9) is not uniform, thus a refreshing layer is introduced, where $(r_1, r_2) \leftarrow \$$ are sampled uniformly at random and $r_3 = r_1 \oplus r_2$. On the contrary, the sharing of the `AND-XOR` gate $z = a \cdot b \oplus c$ is uniform, because the independent variable c ensures uniformity through the `XOR` of its shares with the other terms.

$$\begin{aligned} z_1 &= c_1 \oplus a_1b_1 \oplus a_1b_2 \oplus a_2b_1 \\ z_2 &= c_2 \oplus a_2b_2 \oplus a_2b_3 \oplus a_3b_2 \\ z_3 &= c_3 \oplus a_3b_3 \oplus a_1b_3 \oplus a_3b_1 \end{aligned} \tag{3.8}$$

$$\begin{aligned} z_1 &= r_1 \oplus a_1b_1 \oplus a_1b_2 \oplus a_2b_1 \\ z_2 &= r_2 \oplus a_2b_2 \oplus a_2b_3 \oplus a_3b_2 \\ z_3 &= r_3 \oplus a_3b_3 \oplus a_1b_3 \oplus a_3b_1 \end{aligned} \tag{3.9}$$

Therefore, Threshold Implementations of degree t Boolean functions are constructed case-by-case. Bilgin et al. [20, 19] explore TI of generic Boolean functions, specifically n-bit permutations commonly used in cryptographic algorithms. Notably, when the number of shares required by Equation (2.5) is met, a first-order TI of a permutation can always be found. The main challenge lies in constructing a uniform sharing. The common ways to find a uniform TI are:

- Increase the number of shares, e.g., four instead of three shares for quadratic permutations with direct sharing.

- Search correction terms, or linear and lower-degree terms in a permutation. Equation (3.8) is an example. Using this method, 3- and 4-bit quadratic permutations are shown to be feasible to construct uniform TI with three shares in most cases.
- Use randomness. This adds cost but is an effective way to save the effort of searching for uniform sharing. In the case of high-order glitch-immune masking, this refreshing layer can guarantee the dth-order security given d^{th}-order non-completeness of the sharing.

The application of TI extends beyond permutations. For generic Boolean functions that are not permutations, uniformity can be achieved by applying a randomness-based refreshing layer before the output. The result TI is in the form of Equation (3.10), where the refreshing layer $\mathcal{R}$ can be optional depending on whether the direct sharing achieved uniformity.

$$\mathcal{M}_1 \oplus \mathcal{M}_2 \oplus \cdots \oplus \mathcal{R} \tag{3.10}$$

3.2.5 From Masked Gadgets to Masked Circuits

One bottom-up approach to compiling functional masked Boolean circuits is to replace each gate in the unprotected circuits with a masked circuit, which is called a gadget. The original definition of gadgets also covers arithmetic operations, but this thesis includes only Boolean masking and Boolean gates. The composition of the masked gadgets can also be parallel and sequential, similar to the function composition as shown in Equation (3.1), to construct the whole hardware secure masking. Crucially, both the individual gadgets and their method of composition must adhere to their respective security models. Otherwise, a composition of d-probing secure gadgets might not preserve d-probing security in the overall circuit. This bottom-up approach is motivated by the fact that formal verification on the large-scale masked circuit is computationally infeasible. In contrast, the use of verified building blocks (gadgets) enables scalable and efficient composition. Barth et al. [8, 10] started this modular construction approach by introducing the composable security notion and the simulation-based security proofs.

First, the simulation-based security proof is the foundation for the composable security notions. In this scenario, the adversary is defined as a distinguisher $\mathcal{D}$ with a probe set $\mathcal{P}$ and the gadget $\mathcal{G}$. The shared inputs of the target gadget can be denoted by $\mathcal{I}$. The adversary $\mathcal{D}$ interacts and observes two worlds: the

actual circuit $\mathcal{G}$ and the arbitrary one with a simulator $\mathcal{S}$. The simulator $\mathcal{S}$ only accesses a subset of the input shares of $\mathcal{G}$, $\mathcal{I}' \subset \mathcal{I}$, and lacks knowledge of the circuit's secrets from $\mathcal{I}'$, which is deliberately made secure. If $\mathcal{D}$ cannot distinguish between these two worlds, it implies that its observations are at most the shares provided to the simulator and independent of the secret, and therefore considered secure. A set of probes $\mathcal{P}$ is considered simulatable for certain input shares $\mathcal{I}'$, if a simulator, specific to the set of probes $\mathcal{P}$, can replicate the statistical behavior of the actual probes by simulated ones using only the value in $\mathcal{I}'$. This concept of simulatability ensures that the actual probed values do not depend on any input values not provided to the simulator. Consequently, probes can always be simulated if all input values are available, as the simulator can precisely emulate the gadget's behavior. If probes remain simulatable without access to any inputs, they are independent of the gadget's inputs. Most importantly, if a set of probes $\mathcal{P}$ within an s-shares gadget $\mathcal{G}$ can be simulated with $\mathcal{I}'$ that includes at most $s-1$ shares of each input, then $\mathcal{P}$ is considered secure.

Second, the simulatability of an individual gadget is extended to the parallel and sequential composition of gadgets, thus the composable notions are provided. Notions such as (S)NI and PINI [8, 10, 30] are proposed to achieve this goal. For example, a gadget $\mathcal{G}$ with s shares is d-Non-Interferent (d-NI) if any set of at most d probes can be simulated using at most d shares of each shared input. If $s = d + 1$, then this d-NI gadget can be considered d-probing secure. The simulability-based proof then requires listing all possible sets of probes and showing that the probes are safe under this construction. Instead of proving the whole circuit as d-probing secure, which would require enumerating all possible probes on a large Boolean circuit, the simulability-based proof is limited to a d-NI gadget. If all gadgets in a sequentially composed circuit are d-NI, then the simulability of a set of probes can be propagated to each gadget in the circuit. (S)NI and PINI cover other composition cases. For brevity, the technical details are omitted here.

Consequently, composable gadgets are constructed, e.g., (Generic) Hardware Private Circuits [68, 28]. There is also an automatic masked compiler that can transform the unprotected circuits to secure ones, e.g., AGEMA by Knichel et al. [66] However, the implementation overhead of this composable security strategy is high. Table 3.1 is a summary of the implementation cost of the proposed composable gadgets. HPC and its variations implemented the masked `AND` gate over $\mathbb{F}_2$. GHPC and its variations can mask the n-to-m Boolean circuits. Details regarding implementation and composability are omitted for brevity. Dr. Dhooghe's dissertation [42] states that any d^{th}-order (S)NI or PINI circuit necessarily requires randomness in hardware secure masking for practical symmetric key algorithms. For their composition, the latency is at least one

Gadget	d^{th}-order	Function	#-bits Randomness	#-stages Registers
HPC [28]	d	$\mathbb{F}_2$	$d(d+1)/2$	2
HPC3 [65]	d	$\mathbb{F}_2$	$d(d+1)$	1
GHPC [68]	1	$\mathbb{F}_2^n \to \mathbb{F}_2^m$	m	2
GHPC-LL [68]	1	$\mathbb{F}_2^n \to \mathbb{F}_2^m$	$m \cdot 2^n$	1

Table 3.1: Implementation costs of hardware composable gadgets

cycle due to registers.

3.2.6 Trade-Offs in Glitch-Immune Masking

The design of hardware secure masking involves various glitch-immune masking techniques, particularly the choice between $d+1$ and $td+1$ masking. Each approach entails trade-offs in terms of timing, area overhead, and randomness. The efficiency of a masked circuit depends on several factors, including the number of stages, the required registers, and the computational cost of operations like `AND` and `XOR` gates.

DOM introduces overhead due to the need for expansion and compression stages with registers in between. TI ensures security through constructing glitch-immune shared Boolean functions. For instance, a first-order TI of a cubic function can introduce latency-in-time equivalent to evaluating multiple levels of gates sequentially. On the contrary, DOM uses fewer shares, which means fewer wires and gates in the circuit, thus resulting in a lower area cost, a shorter critical path, but additional registers. Higher-order TI schemes, which require more shares and complex sharing functions, further increase the circuit size and processing delay. By carefully selecting a masking approach, designers must balance security requirements against practical constraints such as hardware area and performance.

3.3 Low-Latency Glitch-Immune Masking

This section discusses the development of TI to reduce latency and the development of low-latency glitch-immune masking. Low-Latency Threshold Implementations (LLTI) [4] is a methodology to mask circuits of any algebraic degree that builds upon the concepts of unrolled implementations [3]. LLTI formalizes and extends the concepts, which divide a non-linear operation and

conquer the masking separately, demonstrating its applicability to any security order and high algebraic degree.

3.3.1 Low-Latency Masking with $d+1$ Shares

The development of low-latency masking based on DOM starts from Generic Low-Latency Masking (GLM) [56] in hardware by Groß et al. Subsequent approaches using $d+1$ shares either integrate with glitch-free circuits, which will be described in Section 3.4.1, or are realized on the level of composable gadgets. The two representative low-latency gadgets are listed in Table 3.1, which are HPC3 [65] and GHPC-LL [68].

Compared to DOM as shown in Equation (3.7), GLM either omits the share compression layer $\mathcal{C}$, which increases the number of output shares, or introduces additional randomness to compress this sharing back. The inherited benefit from DOM is that GLM is scalable to any security order. HPC3 is a two-input `AND` hardware gadget trivially composable under the notion of PINI in the (g)-extended probing model. The cross-domain products, which are expanded in the same way as other $d+1$ masking, are independently blinded, i.e., one share is refreshed with randomness before registration. The multiplication happens after the register stage, and the correction term is also blinded with another random bit. This results in a doubling of the randomness cost compared to the trivial PINI gadget [28] while reducing the latency cost. Although each HPC3 gadget is limited to implementing quadratic functions as the two-input `AND`, it remains scalable to any desired security order.

GHPC, introduced alongside GHPC-LL, applies directly to vectorial Boolean functions and achieves first-order security. It operates on each coordinate function $f : \mathbb{F}_2^n \to \mathbb{F}_2$. Let $x = x_1, \ldots, x_n$, where $x \in \mathbb{F}_2^n$, denote the n-bit input vector, where each component $x_i \in \mathbb{F}_2$ for $1 \leq i \leq n$. Any Boolean function $f(x)$ can be expressed using Shannon decomposition:

$$f(x) = \overline{x_i} \cdot f|_{x_i=0} + x_i \cdot f|_{x_i=1} \tag{3.11}$$

$$f|_{x_i=0} = f(x_0, \ldots, x_{i-1}, 0, x_{i+1}, \ldots, x_n) \tag{3.12}$$

$$f|_{x_i=1} = f(x_0, \ldots, x_{i-1}, 1, x_{i+1}, \ldots, x_n) \tag{3.13}$$

Here, $f|_{x_i=0}$ and $f|_{x_i=1}$ are known as the Shannon cofactors, representing the function's behavior when the variable x_i is fixed to 0 and 1, respectively. For example, the GHPC of the two-input `AND` gate, $c = f(a, b) = ab$, has two shares $\{a_0, a_1\}$ and $\{b_0, b_1\}$. Then $c = f(a_0 + a_1, b_0 + b_1)$ can be expressed in Shannon

decomposition:

$$\begin{aligned} c = \;& \overline{a_0}\overline{b_0} \cdot f(0+a_1, 0+b_1) + \overline{a_0} b_0 \cdot f(0+a_1, 1+b_1) \\ & + a_0 \overline{b_0} \cdot f(1+a_1, 0+b_1) + a_0 b_0 \cdot f(1+a_1, 1+b_1) \\ = \;& \overline{a_0}\overline{b_0} \cdot f(a_1, b_1) + \overline{a_0} b_0 \cdot f(a_1, \overline{b_1}) \\ & + a_0 \overline{b_0} \cdot f(\overline{a_1}, b_1) + a_0 b_0 \cdot f(\overline{a_1}, \overline{b_1}). \end{aligned} \tag{3.14}$$

The resulting GHPC gadget utilizes a single random bit to mask one domain of shares and employs registers to prevent glitch propagation. As shown in Equation (3.15), the vertical dashed line is the register stage, and $r \leftarrow \$$ represents a random bit.

$$\begin{array}{c:c:cl} (f(a_1, b_1) \oplus r) & \cdot\, \overline{a_0}\overline{b_0} \rightarrow z_0 & & \\ (f(a_1, \overline{b_1}) \oplus r) & \cdot\, \overline{a_0} b_0 \rightarrow z_1 & c_0 = & z_0 \oplus z_1 \\ (f(\overline{a_1}, b_1) \oplus r) & \cdot\, a_0 \overline{b_0} \rightarrow z_2 & & \oplus z_2 \oplus z_3 \\ (f(\overline{a_1}, \overline{b_1}) \oplus r) & \cdot\, a_0 b_0 \rightarrow z_3 & = & c \oplus r \\ r & \rightarrow & c_1 = & r \end{array} \tag{3.15}$$

Its low-latency variation, as shown in Equation (3.16), uses randomness for each share and $r_4 = \overline{a_0}\overline{b_0} r_0 + \overline{a_0} b_0 r_1 + a_0 \overline{b_0} r_2 + a_0 b_0 r_3$ defining the second share. Therefore, the requirement for the second register layer is eliminated. Although this construction is limited to the first-order (g)-extended probing security, it scales with algebraic degree and results in a similar cost of GLM with a generic construction algorithm.

$$\begin{array}{c:c:cl} (f(a_1, b_1) \oplus r_0) & \cdot\, \overline{a_0}\overline{b_0} \rightarrow z_0 & & \\ (f(a_1, \overline{b_1}) \oplus r_1) & \cdot\, \overline{a_0} b_0 \rightarrow z_1 & c_0 = & z_0 \oplus z_1 \\ (f(\overline{a_1}, b_1) \oplus r_2) & \cdot\, a_0 \overline{b_0} \rightarrow z_2 & & \oplus z_2 \oplus z_3 \\ (f(\overline{a_1}, \overline{b_1}) \oplus r_3) & \cdot\, a_0 b_0 \rightarrow z_3 & = & c \oplus r_4 \\ r_4 & \rightarrow & c_1 = & r_4 \end{array} \tag{3.16}$$

3.3.2 Unrolled Threshold Implementations

Achieving low-latency in TI differs from $d+1$ masking. Low-latency designs require the masked circuit compiled with high algebraic degree Boolean functions, which impacts the number of input and output shares of TI as shown in Equation (2.5). Uniform sharing may require an increased number of shares. Thus, building an efficient low-latency design of TI presents different challenges.

A Boolean function with a high degree can be decomposed into lower-degree functions, i.e., $f = f_n \circ \ldots \circ f_1$ for n layers. On the contrary, low-degree functions can be merged or unrolled into a high-degree function such as $g' = g \circ g$. Arribas

Layer	1	$2 \cdots$	$\cdots\ n-1$	n
Algebratic degree	t_1	t_2	t_{n-1}	t_n
Non-completeness	$d_1 = t_2 d_2$	$d_2 = t_3 d_3$	$d_{n-1} = t_n d$	$d_n = d$
Compute d		$\Longleftarrow$		
Find sharing		$\Longrightarrow$		

Table 3.2: The function composition of a unrolled implementation of $f = f_n \circ \ldots \circ f_1$ in n layers

et al. [3] proposed the unrolled implementations to compute two rounds of Keccak in a single clock cycle. This means that non-linear operations of the unrolled rounds are computed without a register stage in between, so the non-completeness needs to be considered across stages. Thus, as a design constraint, the non-completeness property demands a divide-and-conquer approach rather than attempting to secure the entire complex function at once. The idea is that every layer of $f = f_n \circ \ldots \circ f_1$, where the algebraic degree of f_i is t_i, is secured with a different sharing scheme with a certain non-completeness order d_i to achieve an overall security order d for the function f, which is shown as follows:

$$d_i = \begin{cases} d & \text{if } i = n, \\ t_{i+1} * d_{i+1} & \text{if } i < n. \end{cases} \tag{3.17}$$

By determining the non-completeness requirement of the output layer f_n, the designer can recursively compute the non-completeness order needed in previous layers, from f_n to the input layer f_1, to achieve the desired overall security order. The shares of every layer are constructed following the constraints from Equation (3.17). It is important to note that d_i for each layer is the order of the non-completeness property. The security order d is defined for the overall masked function because glitch-extended probes can propagate throughout the entire construction. The computation of d_i for each layer starts from the output layer d_n and proceeds backward to the input layer. The construction of shared functions should begin at the input layer and proceed step-by-step to the output layer. Table 3.2 illustrates the computation process of an unrolled implementation of f.

The area cost of the masking scheme can be estimated by counting the number of `AND` and `XOR`. If the sharing is based on a monomial of degree t, this results in $S = (td + 1)^t$ cross-product terms, and $S \cdot t$ wires that are connected with `AND` and `XOR` gates. If the area costs of `AND` and `XOR` gates are assumed to be equal, the total number of wires estimates the total area cost. For example, the monomial's degree is assumed to be $t = 4$, the function $f = f_2 \circ f_1$ is decomposed into two quadratic layers, i.e., $t_1 = t_2 = 2$. Let the target security

Function	f	f_1	f_2	f	f_1	f_2
Non-completeness d	1	2	1	2	4	2
Algebraic degree t	4	2	2	4	2	2
# shares s_i^{in}	5	5	10	9	9	36
Total wires $(s_i^{in})^t \cdot t$	2500	50	200	26244	162	2592

Table 3.3: Estimate the area cost of the unrolled implementations on quartic monomials

order for f be $d = 1$. The non-completeness order of each layer is constrained by Equation (3.17). Accordingly, the non-completeness order of each layer is $d_2 = 1$ and $d_1 = 2$.

An additional constraint is the combined non-completeness of all layers, which makes finding a sharing technically more challenging. Due to the glitch-extended probing propagation, any probes on the last layer f_n should be propagated to the input f_1. In addition, since there is no register stage in between, the wires are directly connected, and no compression of shares is allowed. Thus, each output share of the previous layer s_i^{out} is an input share of the next layer $s_{i+1}^{in} = s_i^{out}$. Table 3.3 shows the area cost estimation for the target security order for f being $d = 1$ and $d = 2$. Note that the construction of TI is not automatic and depends on the ANF of the unprotected Boolean functions, so the numbers are estimates. However, this demonstrates the potential of the divide-and-conquer approach to reduce the area cost of TI for high-degree functions, making it applicable to low-latency designs.

Implementation on Keccak-f[200]

Arribas et al. [3] constructed the unrolled implementation of TI on the Keccak-f[200] permutation. The algebraic degree of each χ layer is two, which results in the unrolled implementation of two non-linear layers being quartic. The original TI of the quadratic round function of the Keccak-f[200] requires three shares. The low-latency unrolled implementation combines two rounds into a single clock cycle.

First, the area cost of the low-latency implementation is lower than the three-share implementation, while the latency is reduced from 25 cycles to 9 cycles, and the latency-in-time is reduced by 47.26%. This is achieved by the two-level unrolled implementation of TI with the numbers of shares $\{s_1^{in} \to s_2^{in} \to s_2^{out}\} = \{5 \to 10 \to 5\}$. These numbers follow the minimum bounds of TI as shown in Equation (2.5).

Second, increasing the number of input shares s_1^{in} can further enhance the efficiency of the unrolled implementation. Increasing s_1^{in} reduces s_2^{in}, therefore the total number of wires in the second layer is reduced, which can be estimated as $(s_2^{in})^2 \cdot 2$. The final sharing scheme proposed by Arribas et al. [3] is $\{s_1^{in} \rightarrow s_2^{in} \rightarrow s_2^{out}\} = \{6 \rightarrow 6 \rightarrow 6\}$, which reduced both area and latency-in-time cost of the Keccak-f[200] by 29.41% and 9.49%.

Set-Cover Problem and Non-Completeness

TI of arbitrary algebraic degree t needs a significant number of shares, potentially making the sharing of high-degree functions impractical to find. To address this, Petrides [87] reformulated the non-completeness property of TI in terms of the set coverings problem.

The set covering problem is a fundamental combinatorial optimization problem [118, 54, 55, 93], where the goal is to select a minimum number of subsets from a given collection to cover all elements in a universe set. Formally, given a universe $\mathcal{U} = \{1, \ldots, n\}$, which is a set of n elements, and a collection $\mathcal{S}$ of m subsets of $\mathcal{U}$, i.e., $\mathcal{S} = \{S_1, S_2, \ldots\}$, whose union covers the universe $\mathcal{U}$, a set cover is a subset $\mathcal{C} \subseteq \mathcal{S}$ whose union is $\mathcal{U}$. The set-cover problem can be a decision problem and an optimization problem. Given $(\mathcal{U}, \mathcal{S})$ and integer k, the decision problem is whether there is a $\mathcal{C}$ of size $\left|\mathcal{C}\right| \leq k$. In the optimization problem, the goal is to find a set cover $\mathcal{C}$ of minimal size, that is, minimize $min(\left|\mathcal{C}\right|)$ given $(\mathcal{U}, \mathcal{S})$. The decision version of the set covering problem is NP-complete, and the optimization, or search, version is NP-hard.

Non-complete set coverings are set coverings $\mathcal{C}$ with the constraint that reformulates non-completeness in TI. It is important to note that this reformulation does not account for the uniformity property of TI or the necessity of remasking. Multiple non-complete set coverings may exist, serving as guidelines for constructing a non-complete TI for any algebraic degree t. This reformulated definition aims to enable a computational way for constructing TI given any parameters, i.e., algebraic degree t and security order d.

A non-complete set covering $\mathcal{C}^{\text{nc}}(s, t, d)$ is a searching problem given the universe $\mathcal{U}_s = \{1, \ldots, s\}$ where s is the number of input shares of the masked Boolean function with algebraic degree t and security order d. Thus, $\mathcal{C}^{\text{nc}}(s, t, d)$ is a set of t-subsets of shares $\mathcal{U}_s$. Each t-subset of $\mathcal{U}_s$ is a subset of at least one element of $\mathcal{C}^{\text{nc}}(s, t, d)$, thus $\mathcal{C}^{\text{nc}}(s, t, d)$ is a correct TI. Each element of $\mathcal{C}^{\text{nc}}(s, t, d)$ has size at least t, so there are no redundant elements. The size of the non-complete set covering $\left|\mathcal{C}^{\text{nc}}(s, t, d)\right| \geq d + 1$ that cover $\mathcal{U}_s$, thus the represented TI is d^{th}-order non-complete.

Non-complete set coverings	Threshold Implementations
$\mathcal{C}^{\mathrm{nc}}(s,t,d)$	TI shares
$\left\lvert\mathcal{C}^{\mathrm{nc}}(s,t,d)\right\rvert$	Number of output shares
$\mathcal{U}_s = \{1,\ldots,s\}$	All input shares with total number s
t-subset $\mathcal{U}_{s,t} \subset \mathcal{U}_s$	The set of all degree-t shared cross products
Each element of $\mathcal{C}^{\mathrm{nc}}(s,t,d)$	Each coordinate function of TI
$\mathcal{NC}(s,t,d)$	All possible sharing solutions of TI
$\widehat{\mathcal{NC}}(s,t,d)$	The compact set of possible solutions
$\widehat{\mathcal{C}}^{\mathrm{nc}}(s,t,d)$	TI shares with minimal s_{out}

Table 3.4: The reformulation of the non-complete set covering

Based on defining $\mathcal{C}^{\mathrm{nc}}(s,t,d)$, each subset $S \in \mathcal{C}^{\mathrm{nc}}(s,t,d)$ corresponds to a shared function, and each element within a subset $S = \{s_1, s_2, \ldots\}$ represents a share that the sub-function is permitted to depend on. Table 3.4 shows the reformulation between non-complete set covering to TI. This reformulation brings the cardinality of S into focus. For every $S \in \mathcal{C}^{\mathrm{nc}}(s,t,d)$, it holds that:

$$\left|S\right| \leq s - (d-1)\cdot t - 1. \tag{3.18}$$

There is an upper bound on the number of shares each coordinate function can have. Therefore, the lower bound of output shares can be derived, i.e., the number of coordinate functions of TI. Note that this is not a new property of TI introduced by the reformulation, but rather a bound that helps designers in constructing more efficient sharings when the number of output shares is a critical factor.

For example, let $\mathcal{C}^{\mathrm{nc}}(3,2,1) = \{\{1,2\},\{2,3\},\{1,3\}\}$, which is the set covering corresponding to the sharing from Equations (3.8) and (3.9) on $z = ab$ or $z = ab \oplus c$. The randomness in Equation (3.9) is not represented in $\mathcal{C}^{\mathrm{nc}}(3,2,1)$ nor the uniformity of Equation (3.9). The parameters of the non-complete set covering can be expressed in one term, i.e., $\langle \mathcal{C}^{\mathrm{nc}}(s,t,d), s_o \rangle$. Reformulation of TI leads to the same propositions as Equation (2.5), which is the non-complete set covering with minimal cardinality:

$$\langle \widehat{\mathcal{C}}^{\mathrm{nc}}(td+1,t,d), \binom{td+1}{t} \rangle. \tag{3.19}$$

Equation (2.5) is equivalent to Equation (3.19).

Arribas et al. [3] extend this approach to the non-trivial case by increasing the number of input shares beyond the minimal bound s, and adjusting the number of output shares s_{out} to maintain both non-completeness and uniformity. Theorem 2.15 in [87] demonstrates that the minimum set cover size satisfies

i	S_i	$\overline{S_i}$
1	{1,2,3}	{4,5,6}
2	{1,4,5}	{2,3,6}
3	{1,4,6}	{2,3,5}
4	{2,5,6}	{1,3,4}
5	{3,5,6}	{1,2,4}
6	{2,3,4}	{1,5,6}

Table 3.5: The elements $S_i \in \widehat{\mathcal{C}}^{\mathrm{nc}}(6,2,2)$ and their complements

$\left|\widehat{\mathcal{C}}^{\mathrm{nc}}(6,2,2)\right| = 6$, as shown in Equation (3.18), and the corresponding sharing is reproduced in Table 3.5. The discovered non-trivial sharing illustrates the above-mentioned unrolled implementation, which is improved by using input shares larger than the minimum bound, i.e., $\{s_1^{in} \to s_2^{in} \to s_2^{out}\} = \{6 \to 6 \to 6\}$. However, a systematic method for finding non-trivial TI sharing remains undiscovered.

3.3.3 Low-Latency Threshold Implementations

The methodology for constructing Low-Latency Threshold Implementations (LLTI), by the author of this thesis together with Arribas and Nikova [4], is extended from unrolled implementations to achieve optimal non-complete sharing for functions of any algebraic degree. LLTI first decompose the non-linear operations into layers of lower algebraic degree. Second, the order of non-completeness of each decomposed layer can be calculated by following the same recursive backward process as shown in Table 3.2 and Equation (3.17). Third, the non-complete sharing of each layer, starting from the input layer and proceeding to the output, is constructed to achieve the desired security order using the non-complete set covering. Finally, the layers are directly wired to form the final implementation, so there is no register in between. The goal of this extension, while following the same procedure as the unroll implementation, is to achieve a universal function decomposition and masking solution for non-completeness.

As shown in Table 3.2 and Equation (3.4), the function composition of unrolled implementations in the form $f = f_n \circ \ldots \circ f_1$ does not cover all positive integer degrees $t \in \mathbb{Z}^+$. For example, when $t = 3$, the above sequential decomposition results in two quadratic operations, which is the same as $t = 4$. According to Equation (3.17), the decomposition yields $(d_1, d_2) = (2, 1)$ and results in the number of shares in each layer, i.e., $\{s_1^{in} \to s_2^{in} \to s_2^{out}\} = \{5 \to 10 \to 5\}$. However, the direct sharing of a cubic function requires $s = td + 1 = 4$

shares. As shown in Table 3.3, which estimates the total number of wires, direct sharing of a cubic function requires $s^t \cdot t = 192$, and this decomposition requires approximately 250 wires. The resulting decomposition is less efficient than direct sharing. Thus, the compositions of the unrolled implementation should be generalized to include both parallel and sequential compositions of Boolean functions, as shown in Equation (3.1), in order to achieve efficient overall implementations for any degree $t \in \mathbb{Z}^+$. Depending on the algebraic degree t, the decomposition can result in two cases: composite-degree and prime-degree functions. Each case requires a different strategy to achieve the required non-completeness order.

Determine LLTI Non-Completeness Requirement

This process generalizes the unrolled implementation as described in Equation (3.17) and Table 3.2. Based on whether the algebraic degree t of the Boolean function f is prime or composite, LLTI decomposes each monomial in the ANF of f into layers by determining the non-completeness order and the number of shares of each gate with a backward process.

Composite Degree Functions: Let the degree t of the target function be composite. There exists a prime factor decomposition of the integer t, i.e., $t = \prod_{i=1}^{n} t_i$ and $\{t_1, \ldots, t_n\}$ are prime numbers. Thus, the non-completeness order of each layer can be determined with the prime algebraic degrees, t_i, in the same way as the unrolled implementation, which divides the high-degree function into multiple low-degree layers at the cost of requiring high non-completeness order for the layers near the input. The advantage is that if the decomposition of f results in a symmetric tree of operations, then the number of input shares required for the first layer f_1 equals the minimum number of input shares needed when applying TI directly to the function f.

Prime-Degree Functions: The function f has a prime degree t in this case. As in the example mentioned above, where $t = 3$, the decomposition of the function into layers cannot be fully sequential. For prime degrees $t > 3$, by extracting the last two-input `AND` gate from the function, decomposition of the function f' with degree $t' = t - 1$ will be possible because t' is now composite.

Layer i	1	2	$\cdots$	$n-1$	n	$n+1$
Comp. deg.	t_1	t_2	$\cdots$	t_{n-1}	t_n	
d_i^{th}-order	$t_2 d_2$	$t_3 d_3$	$\cdots$	$t_n d$	d	
Prime deg.	t_1	t_2	$\cdots$	t_{n-1}	t_n	2
d_i^{th}-order	$t_2 d_2$	$t_3 d_3$	$\cdots$	$t_n d$	d	d
Compute d				$\Longleftarrow$		
Find sharing				$\Longrightarrow$		

Table 3.6: The LLTI function decomposition on a function with degree t

If $t \in \mathbb{P}$, $\{t_1, t_2, \ldots, t_{n-1}\}$ are the prime factors of $t-1$

$$d_i = \begin{cases} d & \text{if } i = n, \\ d & \text{if } i = n-1, \\ t_{i+1} * d_{i+1} & \text{if } i < n-1. \end{cases} \tag{3.20}$$

Note that the last two-input `AND` gate is asymmetric, i.e., one of its ports is driven by outputs from the previous layer, but the other receives value directly from the input. The non-completeness order of f' before this `AND` layer remains unchanged. Thus, the decomposition for f' follows the same procedure as that for composite-degree functions. Equation (3.20) presents the formula used to derive the non-completeness order for each layer. Table 3.6 shows the non-completeness order for both scenarios. Finally, the non-complete sharing of each layer is constructed to achieve the desired security order.

Examples: Figure 3.4 shows the non-completeness order of a decomposed cubic function with two cascaded `AND` gates on one side and a single `AND` gate on the other. This is an example of imbalanced sharing where the last `AND` gate has two inputs with different numbers of shares. After the constraints of the two layers are defined, the non-complete sharing of each layer is derived, as shown in Equation (3.21). For three inputs a, b, and c, the number of input shares is four. The first layer, denoted as $\mathcal{C}_1$, calculates the shared multiplication of a and b and produces six output shares. These six shares, along with the four shares of c, are processed by the second layer $\mathcal{C}_2$. The resulting coordinate functions are non-complete, with missing share indices $\{2, 3, 4, 1\}$, respectively.

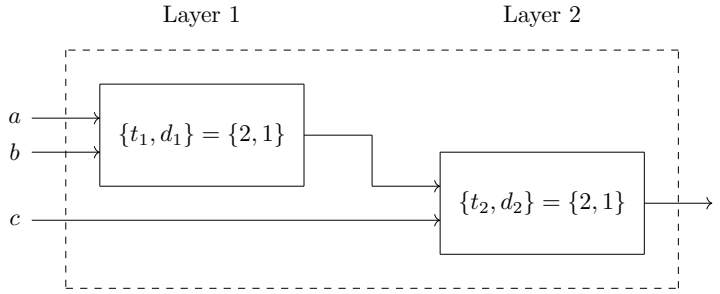


Figure 3.4: LLTI non-completeness requirements for a cubic function [4]

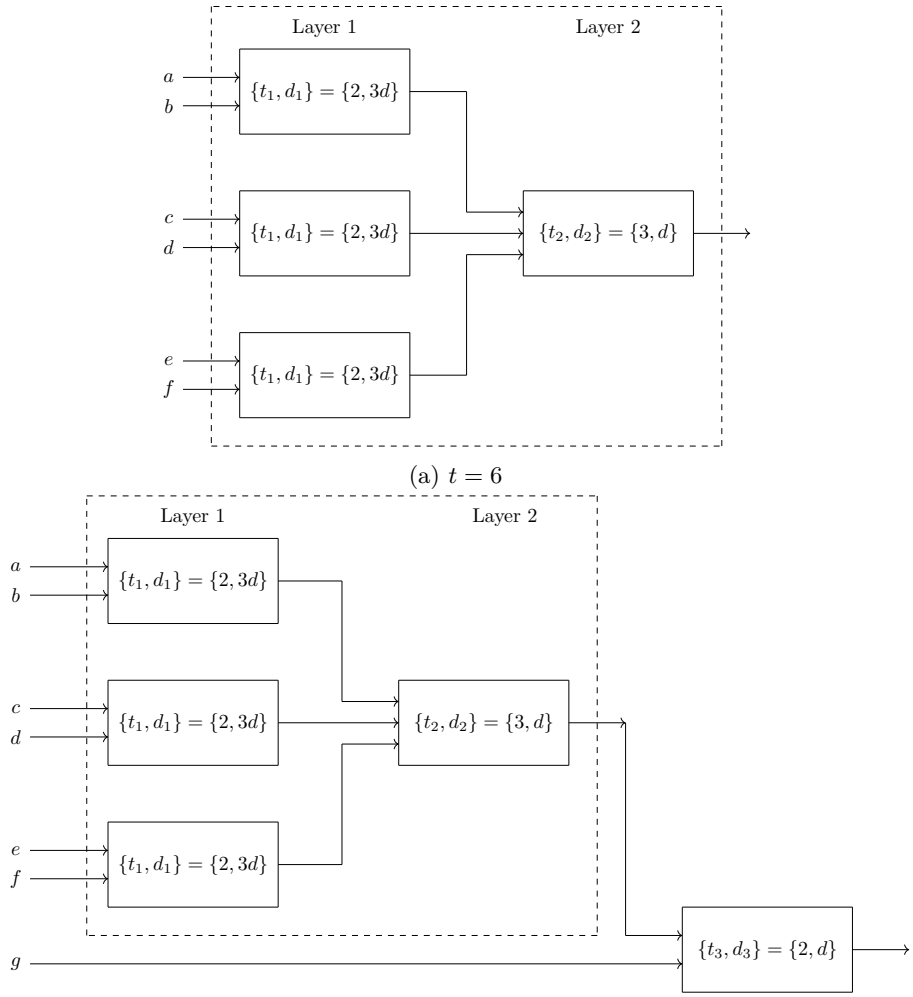
Input	Layer 1
$a = \{1, 2, 3, 4\}$ $b = \{1, 2, 3, 4\}$	$a \cdot b = \langle \mathcal{C}_1(4, 2, 1), 6 \rangle$ $= \{\{1, 2\}, \{2, 3\}, \{3, 4\},$ $\{1, 4\}, \{1, 3\}, \{2, 4\}\}$
Input	Layer 2
$ab = \{1, 2, 3, 4, 5, 6\}$ $c = \{1, 2, 3, 4\}$	$ab \cdot c = \langle \mathcal{C}_2([6, 4], 2, 1), 4 \rangle$ $= \{\{\{3, 1\}, \{3, 4\}, \{4, 3\}, \{5, 3\}, \{5, 4\}, \{4, 4\}\},$ $\{\{1, 1\}, \{1, 2\}, \{1, 4\}, \{4, 1\}, \{4, 2\}, \{6, 1\}\},$ $\{\{1, 3\}, \{2, 1\}, \{2, 3\}, \{5, 1\}, \{5, 2\}\},$ $\{\{2, 2\}, \{2, 4\}, \{3, 2\}, \{6, 2\}, \{6, 3\}, \{6, 4\}, \{3, 3\}\}\}$

(3.21)

Figure 3.5 illustrates the difference between the decompositions of degree $t = 6$ and $t = 7$. In this case, the composite degree function $t = 6$ is decomposed into one cubic function $t_2 = 3$ and three quadratic functions $t_1 = 2$. The non-completeness requirement for the first layer is derived as $d_1 = t_2 d_2 = 3d$. In the case of a prime degree $t = 7$, an imbalanced gate with $t_3 = 2$ is extracted, so the remaining constraint stays the same as $t = 6$.

Derive Non-Complete Set Covering for Each Layer

From the determined non-completeness order, the corresponding sharing can be derived. We proved that, for functions of composite degree, the lower bound on the number of LLTI input shares is the same as that of direct TI sharing. In such a case, there exists a compression sharing for the last layer n, such that it is a minimal d^{th}-order non-complete covering, where the number of output shares equals that of the direct TI sharing. The following formulas formalize



(a) $t = 6$

(b) $t = 7$

Figure 3.5: LLTI non-completeness requirements for composite ($t = 6$) and prime ($t = 7$) degree examples [4]

d^{th}-order Masking on Monomial Degree-t	Non-Complete Covering $\langle \mathcal{C}^{\text{nc}}(s,t,d), s_o \rangle$
$\{t,d\} = \{6,1\}$ TI	$\langle \widehat{\mathcal{C}}^{\text{nc}}_{F_6}(7,6,1), 7 \rangle$
$\{t_1,d_1\} = \{3,2\}$ LLTI Layer 1	$\langle \widehat{\mathcal{C}}^{\text{nc}}_{1}(7,3,2), 35 \rangle$
$\{t_2,d_2\} = \{2,1\}$ LLTI Layer 2	$\langle \widehat{\mathcal{C}}^{\text{nc}}_{2}(35,2,1), 7 \rangle$
$\{t,d\} = \{6,2\}$ TI	$\langle \widehat{\mathcal{C}}^{\text{nc}}_{F_6}(13,6,2), 1716 \rangle$
$\{t_1,d_1\} = \{3,4\}$ LLTI Layer 1	$\langle \widehat{\mathcal{C}}^{\text{nc}}_{1}(13,3,4), 286 \rangle$
$\{t_2,d_2\} = \{2,2\}$ LLTI Layer 2	$\langle \widehat{\mathcal{C}}^{\text{nc}}_{2}(286,2,2), 1716 \rangle$
$\{t,d\} = \{7,1\}$ TI	$\langle \widehat{\mathcal{C}}^{\text{nc}}_{F_7}(8,7,1), 8 \rangle$
$\{t_1,d_1\} = \{3,2\}$ LLTI Layer 1	$\langle \widehat{\mathcal{C}}^{\text{nc}}_{1}(8,3,2), 56 \rangle$
$\{t_2,d_2\} = \{2,1\}$ LLTI Layer 2	$\langle \widehat{\mathcal{C}}^{\text{nc}}_{2}(56,2,1), 1540 \rangle$
$\{t_3,d_3\} = \{2,1\}$ LLTI Layer 3	$\langle \widehat{\mathcal{C}}^{\text{nc}}_{3}([1540,8],2,1), 8 \rangle$
$\{t,d\} = \{7,2\}$ TI	$\langle \widehat{\mathcal{C}}^{\text{nc}}_{F_7}(15,7,2), 6435 \rangle$
$\{t_1,d_1\} = \{3,4\}$ LLTI Layer 1	$\langle \widehat{\mathcal{C}}^{\text{nc}}_{1}(15,3,4), 455 \rangle$
$\{t_2,d_2\} = \{2,2\}$ LLTI Layer 2	$\langle \widehat{\mathcal{C}}^{\text{nc}}_{2}(455,2,2), 103285 \rangle$
$\{t_3,d_3\} = \{2,2\}$ LLTI Layer 3	$\langle \widehat{\mathcal{C}}^{\text{nc}}_{3}([103285,15],2,2), 6435 \rangle$

Table 3.7: Non-complete set coverings present TI and LLTI in format $\langle \mathcal{C}^{\text{nc}}(s,t,d), s_o \rangle$ for algebraic degree t, non-completeness order d, number of shares at input s and output s_o

these conclusions.

$$\left|\mathcal{U}^t_s\right| = s_1 \geq td + 1 \tag{3.22}$$

$$\left|\widehat{\mathcal{C}}^{\text{nc}}_{f_n}(s_n,t_n,d_n)\right| = \left|\widehat{\mathcal{C}}^{\text{nc}}_{f}(td+1,t,d)\right| \Rightarrow \langle \widehat{\mathcal{C}}^{\text{nc}}_{f_n}(s_n,t_n,d_n), \binom{td+1}{t} \rangle. \tag{3.23}$$

For the case where the algebraic degree t is prime, the final layer consists of a `AND` gate with $t_n = 2$ and $d_n = d$. To preserve non-completeness when adding the final `AND` gate, additional d shares are required beyond those already present in the set $\mathcal{U}^{t-1}_s$. Consequently, the number of shares required for f to achieve the overall non-completeness set covering is:

$$\left|\mathcal{U}^t_s\right| = \left|\mathcal{U}^{t-1}_s\right| + d \tag{3.24}$$

Following Equation (3.24) and Equation (3.22), we can derive the set-covering for each layer knowing the degree t.

Generalizations

The decomposition when the algebraic degree t is prime can be expressed as Equation (3.1):

$$f(x_1, \ldots, x_t) = f_n(f'(x_1, \ldots, x_{t-1}), x_t) \tag{3.25}$$

where f_n denotes a two-input `AND` gate at the output layer. Equation (3.25) can be further generalized to an imbalanced tree of functions:

$$f(x_1, \ldots, x_t) = f_n(f^1_{n-1}(x_1, \ldots, x_a), f^2_{n-1}(x_{a+1}, \ldots, x_t)). \tag{3.26}$$

This generalization is beyond the scope of LLTI because the current construction of TI and LLTI on generic Boolean monomials that applied to ANF of a Boolean function, which is a two-level representation of `AND-XOR`. Future work may explore multilevel representations of Boolean functions that can apply this generalization.

The imbalanced decomposition of a cubic function is more efficient than its direct sharing. In addition, the decomposition of the composite degree t is commutative. Therefore, for functions $t < 9$, there exist solutions that are composed solely of shared two-input gates. Thus, the resulting decomposition can be seen as a binary multiplication tree. Future work may develop automatic searching algorithms based on this observation. This optimization is applied manually in LLTI on $t = 6$ and $t = 7$ in Table 3.8, which is described in the following text.

Optimizing the Trivial Construction of LLTI

Although LLTI constructions based on the above methods can achieve the desired security order and low latency-in-cycle, the number of gates in the resulting masked implementations can be further reduced. Comparing the examples of degree $t = 6$ with first- or second-order security, as depicted in Table 3.7, it is evident that increasing the number of shares at the middle layer substantially increases the hardware area cost. The same observation holds for the degree $t = 7$ examples in Table 3.7. The unrolled implementation demonstrates a considerable reduction in area cost when the two-level sharing changes from $\{s_1^{in} \to s_2^{in} \to s_2^{out}\} = \{5 \to 10 \to 5\}$ to a balanced structure like $\{6 \to 6 \to 6\}$. This observation suggests that a key optimization strategy is to avoid significant expansion in the number of intermediate shares.

We showed that for composite degrees t, it is possible to maintain constant set cardinality $\left|\mathcal{C}^{\mathrm{nc}}(s, t, d)\right|$ across layers by increasing the number of input shares s.

Specifically, for first-order masking, the set covering size $\left|\mathcal{C}^{\mathrm{nc}}(s = t+1, t, 1)\right| = \binom{t+1}{t} = s$ indicates that the same number of input and output shares can be maintained across all layers. This insight provides a guideline for optimizing decomposed sharing: rather than minimizing the individual set covering size per layer, designers should aim to avoid a significant increase in intermediate share. Designers must balance the total number of input shares against the risk of intermediate expansion. In the case of prime t, additional constraints are required on the imbalanced layer to control share growth. Furthermore, when translating the set coverings into actual masking schemes, non-completeness checks across layers remain essential. Consequently, optimizing the basic LLTI construction is still a manual process but offers designers valuable flexibility in reducing overall circuit complexity.

Applications

We present the application of LLTI on monomials of $t = \{3, 4, 5, 6, 7\}$ and various ciphers. Table 3.8 reproduces the optimized results on the shared monomials. The optimization steps are not fully automated or computationally optimal, but follow certain principles:

- Minimize the total number of shares $(s + s_o)$ while ensuring security constraints like non-completeness. The problem is closely related to the set-cover problem, a well-known NP-hard problem in combinatorial optimization, as explained in Section 3.3.2. In addition to the standard set covering problem, we need to add the non-completeness constraint. While existing set covering databases, e.g., [36, 53], provide useful guidance, they lack non-completeness. Thus, designers must verify or tailor these coverings accordingly. Future work should focus on developing dedicated bounds and optimized coverings that inherently satisfy both the covering and non-completeness constraints.

- Divide-and-conquer the layers and decompose cubic nodes with imbalanced sharing. For instance, $t = 6$, a quadratic function $t_2 = 2$ is shared with the target first-order security. The input layer is $t_1 = 3$, which can be applied with the imbalanced composition. Therefore, designers should find covering sets $t_{12} = t_{11} = 2$ and non-completeness order $d_{11} = d_{12} = 2$. Table 3.8 shows such results in the functions degree $t = 6$ and $t = 7$ at level R^{11} and R^{12}. When searching for these non-completeness set coverings on R^{11} and R^{12}, the set cardinality is constrained by later layers, so the candidates in the database are reduced.

Degree	Layer	Covering	Wires (#)	Reduction times (w.r.t) TI	Reduction times (w.r.t) Trivial LLTI
$t = 3$	R^1 R^2	$\langle \mathcal{C}_1(4,2,1),6\rangle$ $\langle \mathcal{C}_2([6,4],2,1),4\rangle$	70	$\times 2.68$	
$t = 6$	R^{11} R^{12} R^2	$\langle \mathcal{C}_{11}(9,2,2),32\rangle$ $\langle \mathcal{C}_{12}([32,9],2,2),13\rangle$ $\langle \mathcal{C}_2(13,2,1),9\rangle$	1 715	$\times 400$	$\times 2.5$
$t = 7$	R^{11} R^{12} R^2 R^3	$\langle \mathcal{C}_{11}(10,2,2),45\rangle$ $\langle \mathcal{C}_{12}([45,10],2,2),30\rangle$ $\langle \mathcal{C}_2(30,2,1),45\rangle$ $\langle \mathcal{C}_3([45,10],2,1),10\rangle$	4 695	$\times 3131$	$\times 6.9$

Table 3.8: Summary of final first-order LLTI coverings for $t = \{3, 6, 7\}$ applied to a single monomial [4]

A full AES implementation with four-share LLTI, which is a two-stage decomposition as in Figure 3.3, passed the TVLA test on physical experiments. Compared to the TI with the same number of shares, the LLTI version achieves 47.2% decrease in area and a 6.9 times increase in maximum frequency. Thus, LLTI improves the circuit complexity compared to TI.

3.3.4 Universal First-Order $t + 2$ TI for Permutations

Picionne et al. [88] proposed an optimal and universal first-order secure construction applicable to all permutations, given that they have at least $t + 2$ shares. We denote this method as *the* $t + 2$ *TI*. This construction satisfies all the TI properties, including uniformity, which means it does not require any randomness. In addition, this construction is universal for any permutation and applicable to all bijective Boolean functions, so it does not require any searching algorithm for the sharing scheme. Previous approaches, like traditional TI and LLTI, relied on techniques like correction terms and randomness to achieve uniformity, or searching algorithms to find uniform sharing for an S-box [20].

Let the $t + 2$ TI is $\mathcal{F} = (f_1, \ldots, f_{t+2})$ with $t + 2$ shares for every permutation F over $\mathbb{F}_2^n$ of algebraic degree at most t, for $t \geq 2$. For $t = 1$, the construction of TI of affine permutations is known. The dimension n of $\mathbb{F}_2^n$ does not affect this construction. Let each coordinate function $f_i : (\mathbb{F}_2^n)^{t+2} \to (\mathbb{F}_2)^{t+2}$, be defined

Design	Security d	$\mathcal{R}$ #-bits	Area $[kGE]$	Latency Cycles
HPC3, Knichel et al. [65]	1	68	1.85	4
HPC3, Knichel et al. [65]	2	204	4.86	4
GHPC, Knichel et al. [68]	1	8	77.15	2
GHPC-LL, Knichel et al. [68]	1	2048	64.11	1
GLM, Gross et al. [56]	1	2048	60.76	1
GLM, Gross et al. [56]	1	416	6.74	2
$t+2$ TI, Piccione et al. [88]	1	0	128.27	1
$t+2$ TI, Piccione et al. [88]	1	0	21.86	2
4-share LLTI, Arribas et al. [4]	1	0	25.78	2
4-share TI, Arribas et al. [4]	1	0	58.41	2

Table 3.9: AES S-box implementations using low-latency glitch-immune masking schemes

for every $\mathbf{x} = (x_1, \ldots, x_{t+2}) \in (\mathbb{F}_2^n)^{t+2}$ as

$$
\begin{aligned}
F_1(\mathbf{x}) &= x_1 \\
F_2(\mathbf{x}) &= \sum_{i=3}^{t+2} x_i + F\left(\sum_{i=2}^{t+2} x_i\right) \\
F_j(\mathbf{x}) &= x_j + \sum_{I \in \mathcal{I}_{j-2}} F\left(\sum_{i \in I} x_i + \sum_{i=j}^{t+2} x_i\right), \qquad j = 3, \ldots, t+1 \\
F_{t+2}(\mathbf{x}) &= x_{t+2} + x_1 + \sum_{I \in \mathcal{I}_t} F\left(\sum_{i \in I} x_i\right)
\end{aligned} \tag{3.27}
$$

where for any $k \geq 1$, $\mathcal{I}_k$ is the set of all subsets of $\{1, \ldots, k\}$ (including $\emptyset$).

This construction is applied to AES S-box with one cycle latency, which is the first direct sharing of a function with algebraic degree $t = 7$, to the best of our knowledge. Furthermore, the S-box with cubic decomposition is also implemented and achieves a lower area cost than that of LLTI. The reason is that the permutation F is directly used in the $t+2$ TI, thus the synthesizer can perform logic minimization on this permutation with no concern for side-channel security. The results are included in Table 3.9.

3.3.5 Conclusion

Table 3.9 compares low-latency AES S-box implementations with glitch-immune masking. Four implementations are based on TI, which do not consume any randomness and require $td + 1$ shares. The large number of input shares results in significant area overhead. Due to the difficulty of constructing TI for high-degree functions, a single-cycle S-box was not achieved until the use of the $t + 2$ sharing.

In contrast, $d + 1$ sharing requires a high randomness cost due to the need for compression layers. However, the use of randomness allows the shares to be compressed into $d + 1$ shares after the register stage. One key advantage of $d + 1$ sharing is its improved area efficiency. Additionally, the use of registers makes high-order secure implementations more feasible.

3.4 Low-Latency Masking without Registers

Registers are essential components in synchronous circuits, as they introduce latency in data paths and stop glitch propagation in circuits secured with glitch-immune masking. Section 3.3 covers the main technologies of low-latency glitch-immune masking schemes, which focus on removing the need for registers while keeping the glitch-immune security of the shared Boolean functions. The use of registers becomes a bottleneck in the design of low-latency masked implementations. Therefore, circuits without registers are studied in the context of low-latency hardware secure masking. As shown in Figure 2.5, there are two main categories of implementation without registers. One category is the glitch-free circuits, i.e., the DRP logic implementations. The other is the glitch-stopping circuits, which are based on the glitch-immune masking scheme, but use efficient circuit elements to stop glitch propagation.

3.4.1 Low-Latency Glitch-Free Circuits

Glitch-free circuits are circuits that do not exhibit internal glitches during operation. To reduce the clock cycles required for computing the entire masked function, glitch-free solutions have been proposed to ensure the masked function's non-linear operation remains secure. For masked block ciphers, the S-boxes are protected with glitch-free circuits, which are pre-charged and evaluated locally, while the surrounding circuits operate synchronously to interface with the remaining digital logic.

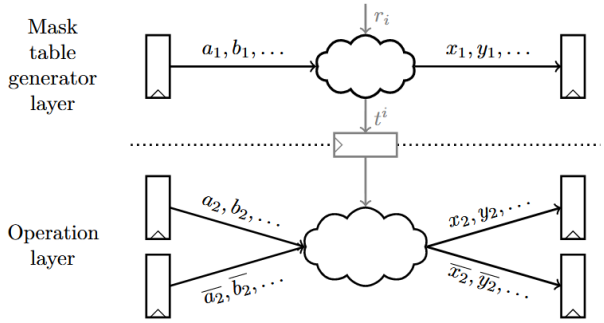


Figure 3.6: The LMDPL structure [97]

LMDPL

Sasdrich et al. [97] apply the LUT-based Masked Dual-Rail with Pre-charge Logic (LMDPL) technique, which was first introduced by Leiserson et al. [72]. The authors analyse its 1st-order (g)-extended probing security and also evaluate its simulability as a composable gadget. LMDPL merges LUT-based first-order masking on monotonic gates with DRP logic. For first-order security, LMDPL gadgets work on two shares, i.e., $x = x_1 \oplus x_2$. The gadget design can be conceptually divided into two horizontal levels or layers. The first layer is known as the mask-table generator that operates exclusively on the first share x_1. If the gadget is non-linear, this layer also generates an intermediate vector called the table output t, which is stored in a register stage that serves as an input to the second layer. The second layer, known as the operation layer, computes the second output share and its complement $\{x_2, \overline{x_2}\}$. The structure of LMDPL is shown in Figure 3.6.

Consider a two-input `AND` using LMDPL, denoted as $x = ab$. The mask LUT generator receives a random bit $r \leftarrow \$ \in \mathbb{F}_2$ which is assigned to $x_1 = r$. The table output vector is $t = \{t_0, t_1, \ldots, t_7\}$, as shown in Equation (3.28) where the vertical dashed line indicates the register stage.

$$\begin{array}{lll|ll}
\{0,0\} \rightarrow & t_0 = a_1 b_1 + \overline{r} & & \overline{x_2} = & \overline{a_2}\,\overline{b_2} t_0 + \\
\{0,1\} \rightarrow & t_1 = \overline{a_1} b_1 + \overline{r} & & & a_2 \overline{b_2} t_1 + \\
\{1,0\} \rightarrow & t_2 = a_1 \overline{b_1} + \overline{r} & & & \overline{a_2} b_2 t_2 + \\
\{1,1\} \rightarrow & t_3 = \overline{a_1}\,\overline{b_1} + \overline{r} & & & a_2 b_2 t_3 \\
\{0,0\} \rightarrow & t_4 = a_1 b_1 + r & & x_2 = & \overline{a_2}\,\overline{b_2} t_4 + \\
\{0,1\} \rightarrow & t_5 = \overline{a_1} b_1 + r & & & a_2 \overline{b_2} t_5 + \\
\{1,0\} \rightarrow & t_6 = a_1 \overline{b_1} + r & & & \overline{a_2} b_2 t_6 + \\
\{1,1\} \rightarrow & t_7 = \overline{a_1}\,\overline{b_1} + r & & & a_2 b_2 t_7
\end{array} \tag{3.28}$$

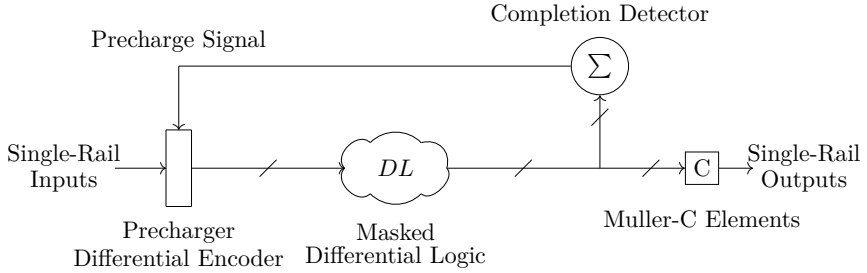


Figure 3.7: The SESYM structure [84]

If the target function is linear, such as an `INV` or `XOR` gate, the output table is empty. Therefore, no synchronization occurs, and the register stage is omitted.

The advantage of the LMDPL gadgets is one of their composability, i.e., the non-linear gadgets can be composed without requiring registers in between to realize a high algebraic degree function, while preserving the register of the table output t. The authors proved the gadgets themselves be 1st-order (g)-(S)NI. With such a composition, the mask-table generator layer processes on share x_1, and the operation layer processes on the other share $\{x_2, \overline{x_2}\}$.

A single-cycle AES implementation per round is developed with LMDPL. The AES S-box is implemented in dual-rail, which includes both the shared values and their inversion at the circuit's input and output. To optimize efficiency and maintain the glitch-extended security, the pre-charging unit is delegated to an external module that keeps the round computation also in dual-rail. In addition, an `XOR` gadget with an enabling signal and a multiplexer gadget with a select signal are introduced for the round computation of both AES encryption and decryption. The dual-rail data path includes two round functions, separated by pipeline registers to form an alternating flow of pre-charging and evaluation. When the first module pre-charges, the second evaluates the dual-rail computation.

SESYM

Nagpal et al. [84] proposed the self-synchronized masking (SESYM), which applies WDDL gates in DOM, that transforms the glitch-immune masking into a single-cycle glitch-extended secure design without requiring additional registers. Figure 3.7 shows the building blocks, which first convert single-rail inputs to dual-rail encoding, followed by a pre-charging module. The completion detector module is the key component for detecting the completion of the evaluation phase of the dual-rail circuit by calculating the product of the sum of all outputs.

This completion signal is used to drive the pre-charging of the circuits. An array of Muller C-elements ensures stable output signals, which also function as a dual-to-single-rail converter. SESYM adds the self-synchronizing mechanism and a stable single-port output using the Muller-C element. However, the inherited problem in dual-rail logic remains.

Problems of Imbalanced Delays

Müller et al. [82] conduct a thorough security analysis of the glitch-extended probing security on the DRP logic, such as SESYM and LMDPL. The analysis is based on the imbalanced delay on wires of the DRP logic. In DRP logic, the early propagation effect means the output of a DRP gate is stable before all the inputs arrive. This effect can break the assumption that the power consumption of DRP logic remains constant. In the timing window between the output and input becoming stable, power consumption changes instantaneously.

The analysis of imbalanced delay follows similar timing considerations. The delay concerns value transitions, also known as toggles, that happen on wires of the DRP logic circuit. During the pre-charge phase, the delay counts from the start of the phase, signaled by the pre-charge signal, to the transition of the wires from 1 to 0. This assumes that the DRP logic is operated in a valid state, i.e., with $\{0, 1\}$ or $\{1, 0\}$ on the dual-rail wires. During the evaluation phase, the delay measures from the start of the phase to the transition from the initial state to a valid state, i.e., from 0 to 1 on wires. The analysis is based on a WDDL gate by placing a probe at the inverted output of the two-input `WDDL-AND` gate, which is evaluated by an `OR` gate as shown in Figure 2.7. By adding an arbitrary delay on an input wire and observing the probed toggle pattern and its delay, the adversary can reveal the input value of the WDDL gate. This can result in a statistical information leakage of SESYM gate, which is based on WDDL gates.

This analysis on imbalanced delay is not captured by the robust probing model and demonstrates that information about the input of DRP gates can be revealed by observing their outputs. Additionally, the authors state that the early propagation effect and the imbalanced delay are not related to each other. The imbalanced delay can cause input value leakage even when the early propagation is avoided.

LMDPL separates two shares into two layers separated by registers, as shown in Figure 3.6. The probe is placed at the output of the operation layer, where non-linear calculations occur. The average delay observed at the probe is indistinguishable to the adversary, regardless of deliberate delay on input wires. This means that imbalanced delay does not affect the probing security of

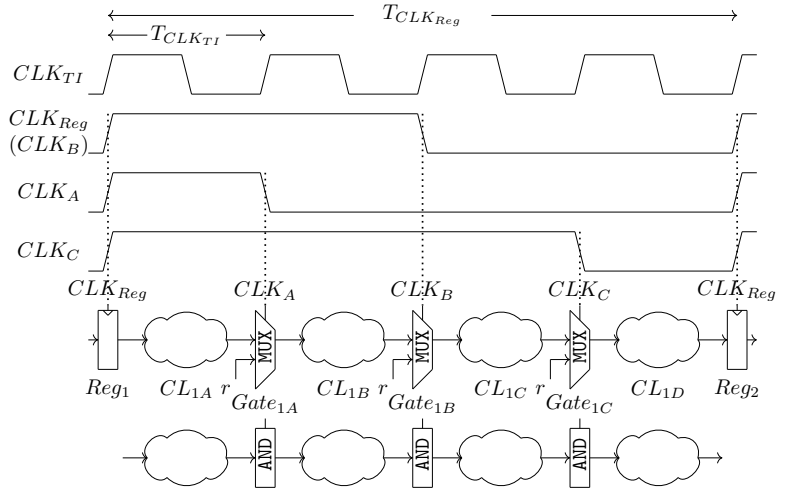


Figure 3.8: The glitch-stopping circuits with `AND` or `MUX` gates, and their clock configurations [120]

LMDPL. This analysis on LMDPL and SESYM is also validated by physical experiments on FPGA evaluation setup. They show that LMDPL remains secure, and, furthermore, illustrate how timing considerations in DRP logic, which is ignored by traditional circuit models of security analysis, can cause information leakage.

3.4.2 Glitch-Stopping Circuits

In contrast to the DRP-based logic, Zhang et al. [120], a work by the author of this thesis, explored the use of digital circuits other than registers to stop glitch propagation.

Glitch-stopping circuits maintain the synchronous nature of the masked implementation but use clocked logic gates or latches, apart from registers, to prevent glitch propagation. The synchronous nature means that the GSC method is still regulated by a clock signal, which synchronizes the logic output, but requires minor adjustments to the clock configuration. The propagation delay on wires of GSC is comparable to that of synchronous logic. Figure 3.9 and Figure 3.8 show the glitch-stopping gates and its clock configurations.

Design Procedure of Glitch-Stopping Circuits

The glitch-stopping circuits are constructed from a register-based hardware secure masking. The steps are as follows:

- Begin with a glitch-extended probing secure hardware implementation with registers.
- Determine the register layers and the critical path of the circuit by analyzing each combinational logic stage.
- Introduce the appropriate clock signals for the selected glitch-stopping method if the time-wise latency is a primary concern for further optimization.
- Replace registers with the chosen type of glitch-stopping latches or gates at the RTL or netlist level and connect the newly introduced clocks to their respective gates.
- Validate the behavioral correctness, and conduct leakage assessment tests.

Figure 3.8 shows the clock configurations and the circuit construction for a four-stage S-box at the logic level. The input and output of this masked circuit are registered, so the results are synchronized by the global clock CLK_{Reg} and used by the other hardware modules. The internal stages of this four-stage masked implementation are separated by `AND` or `MUX` gates, which can be instantiated as Figures 3.9c and 3.9d. The `AND` or `MUX` gates can be constructed using underlying logic gates as shown in Figures 3.9a and 3.9b, if the designer can access such logic cells in ASIC hardware design. In the case of FPGA, the instantiation of such circuits must follow the available primitives on chip, which is discussed in the following section.

The clock signals CLK_A, CLK_B and CLK_C are connected to drive the gates of stage CL_{1A}, CL_{1B} and CL_{1C}. Each clocked gate is triggered by the negative edge of the clock signals, so the three clock edges stop the glitches from propagating from the previous stage to the next. As shown in the figure, three clocks have duty cycles of 25%, 50%, and 75%, so CLK_B and CLK_{Reg} are equivalent. There is room for further optimization, as the duty cycles can be adjusted to accommodate the propagation delays of each stage. The implementation of such clock configurations differs between the clock configuration sources of FPGA and ASIC.

Physical Evaluation of Glitch-Stopping Circuits

Since glitch-stopping circuits are circuit-level constructions of SCA countermeasures, the physical leakage experiments are essential to evaluate the glitch-extended probing security, which is often performed using Xilinx FPGA side-channel evaluation platforms. Therefore, the `AND` and `MUX` instances of the GSC are mapped onto FPGA primitives. FPGAs consists of Configurable Logic Blocks with reconfigurable interconnects. There are primitives inside the configurable logic blocks, which can be used to perform desired logic operations. The primitives are typically Look-Up Tables (LUT), flip-flops, and other logic elements. On Xilinx FPGAs, the primitives that can serve as the `AND` and `MUX` instances of the GSC are `LUT`, `MUXF7`, and `AND2B1L`. They correspond to the Look-Up Table, the 2-to-1 multiplexer, and the latch configured as logic on a logic block. Furthermore, the clock configuration tools of Xilinx are used to create clock signals according to the clock configuration of Figure 3.8. The generated clocks are routed to GSC instantiation using FPGA logic primitives, so the option `CLOCK_DEDICATED_ROUTE` needs to be set to `FALSE` for these clock signals. Signal delays on interconnects, particularly those affecting derived clock signals, can degrade the quality of the clock that drives the primitives for glitch-stopping circuits. Nevertheless, these primitives effectively suppress glitch propagation. Therefore, it can be concluded that despite introducing minor degradation in the derived clock on non-clock-dedicated routing, glitch-stopping circuits effectively suppress glitch propagation and maintain practical usability. It is assumed that ASIC designers have flexibility and finer control over cell primitives and clock configuration resources, which facilitates the transfer of glitch-stopping circuits verified on FPGA to ASIC implementations.

Table 3.10 shows the experiments implementing an AES encryption using these candidates, which reveal whether leakage occurs. Additionally, we performed experiments on the cases of four-stage masked AES, high-frequency clock signal, and second-order multi-variant TVLA, which are omitted here for brevity. Due to the mismatch between FPGA platform and the ideal circuit model, some cases fail as indicated in the table. However, the multiplexer primitive `MUXF7` provides glitch-stopping behavior in all test cases, which illustrates the effectiveness of the glitch-stopping circuits constructed with gates.

Instance	FPGA	Implementation	First-Order TVLA
`AND` or `MUX` gates	`MUXF7`	2-stage serialized AES	*Pass*
`AND`-gate	`AND2B1L`	2-stage serialized AES	*Fail*
`AND`-gate	`LUT`	2-stage serialized AES	*Fail*

Table 3.10: Physical leakage tests of GSC using the FPGA primitives on Xilinx Sakura-G [120]

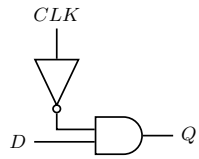
Discussion

By using glitch-stopping gates, the multi-stage masked implementation can be realized in one clock cycle, without changing its masking scheme and security order. In Table 1 of the paper, the authors compared GSC with the state-of-the-art masked implementations. For first-order security, LMDPL remains the implementation with the lowest area and critical path implementation. The four-stage `AND`-based GSC based on three-share TI is 123.9% of the area cost and 54.2% of the maximum clock frequency. However, given that the LMDPL AES S-box is two shares, if DOM is used to construct first-order secure GSC, the area cost can be potentially reduced. For future work, the register stage in LMDPL can also be potentially replaced and constructed as GSC, which will reduce its area and critical path cost. If the `AND`-based GSCs are compared to their register-based ones, the area cost is reduced to 89.6% and 88.8% for the four-stage and two-stage masked S-boxes, respectively.

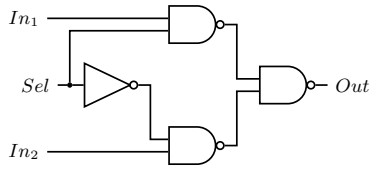
The advantage of GSC is larger for high-order masked implementations. In this work, a second-order implementation using `AND`-based GSCs is physically evaluated on FPGA to show its multi-variant security. For future work, high-order masking schemes constructed with GSC can be further explored to optimize area and latency-in-time. Furthermore, Figure 3.10 illustrates two optimizations that can be potentially applied. In particular, when latency-in-time matters, the timing constraint based on the propagation delay of each stage can be different. By adjusting these virtual cycles to align with the critical paths of each sub-circuit, the overall computation time can be reduced. Additionally, clocks with the same frequency can be used with a phase shift. This phase shift can be implemented using either a delay line or a phase-locked loop (PLL) circuit, which may offer a more straightforward solution compared to generating clocks with different duty cycles. These optimizations collectively contribute to reducing time-wise latency and improving overall performance.

3.4.3 Conclusion

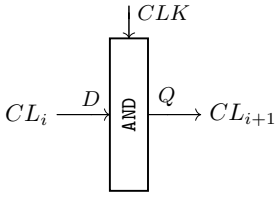
Registers are only used for stop glitch propagation, adding latency cost to hardware secure masking. To remove these registers, either by using glitch-stopping circuits or glitch-free circuits, is a method that keeps the security requirement of glitch immunity and avoids the area cost increase significantly. Table 3.11 shows the cost of AES S-box using these methods. They typically reuse the glitch-immune masking scheme, but implement the masked logic in the corresponding method with DRP logic. LMDPL is highly optimized for quadratic functions and first-order glitch-extended probing secure, so it results



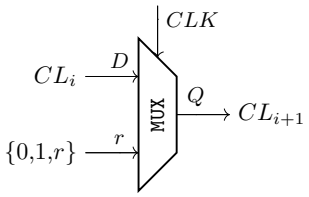
(a) A clocked AND-Gate



(b) A clocked MUX-Gate



(c) Integration of the AND-Gate into GSC



(d) Integration of the MUX-Gate into GSC

Figure 3.9: AND and MUX instances of the glitch-stopping circuits [120]

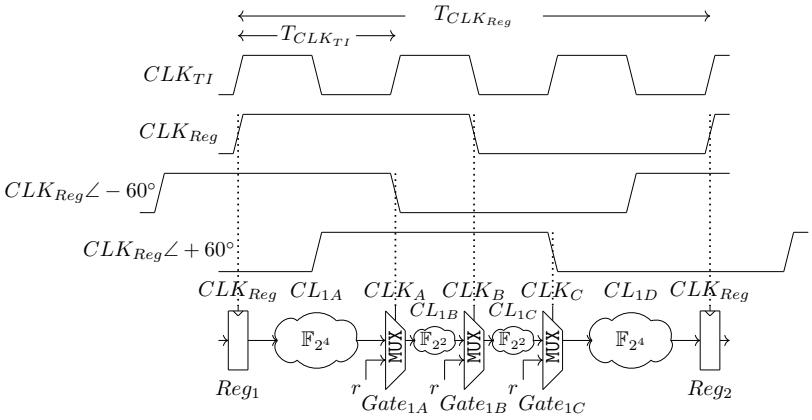


Figure 3.10: Clocked logic-gate-based glitch-stopping circuit with clocks with phase-shift

Design	Security d	$\mathcal{R}$ #-bits	Area $[kGE]$	Latency Cycles
LMDPL, Sasdrich et al. [97]	1	36	3.48	1
SESYM, Nagpal et al. [84]	1	18	7.59	1
SESYM, Nagpal et al. [84]	2	51	14.78	1
GSC-AND, Zhang et al. [120]	1	36	4.76	1

Table 3.11: AES S-box with tower field decomposition with low-latency masking without registers

in the most efficient area cost. Both GSC and SESYM can scale to high-order security, although SESYM is still vulnerable to the early propagation effect.

3.5 My Contributions

In this chapter, we have discussed the timing overhead of masking schemes and techniques for reducing the latency of masked implementations. The latency cost is introduced by the register stages, which are essential to prevent glitch propagation. This dissertation proposes two generic methods to reduce the latency of masked implementations: Low-Latency Threshold Implementations (LLTI) and Glitch-Stopping Circuits (GSC).

3.5.1 Extend Threshold Implementations to Low-Latency Schemes

The international journal paper from IEEE Transactions on Information Forensics and Security in 2021, titled "LLTI: Low-Latency Threshold Implementations" [4], presents a divide-and-conquer methodology to construct Threshold Implementations with cycle-wise latency down to one clock cycle. I am one of the primary authors of this paper.

Before this research, the unrolled implementation of TI was proposed to compute two rounds of Keccak in a single clock cycle, and the non-complete set covering was introduced to find the non-trivial TI sharing. This work extends these concepts to the masking of higher algebraic degree functions and any security order. The method, denoted as LLTI, is applied to construct high-degree monomials up to degree seven and results in a more efficient imbalanced sharing of cubic functions which can be applied to the two-stage decomposed AES S-box.

3.5.2 Glitch-Stopping Circuits

The international conference paper appeared at the ACM Conference on Computer and Communications Security (CCS) in 2024, titled "Glitch-Stopping Circuits: Hardware Secure Masking without Registers" [120], presents a method to secure masked implementations without registers. I am the primary author of this paper.

This work falls into the category of masked implementation without registers, which tries to reduce the latency by avoiding the register stages. Specifically, our work uses clocked logic gates or latches to stop glitch propagation, requiring only minimal adjustments to the clock configuration and maintaining the synchronous nature of the masked implementation.

Chapter 4

Reduce the Randomness Cost in Hardware Secure Masking

Hardware secure masking typically consumes a considerable amount of randomness per clock cycle. Such randomness ensures the uniformity of the output shares of the masked implementation. The masked implementations rely on high-entropy randomness sources to guarantee provable security against side-channel attacks. This chapter explores the randomness cost in hardware-secure masking and investigates methods to reduce the cost of generating and using random bits.

4.1 Randomness Requirement in Hardware Secure Masking

This section will discuss the randomness requirement in hardware secure masking from state-of-the-art studies and the practical randomness sources. In designing and implementing hardware secure masking, an ideal randomness source is typically assumed to operate concurrently with the masked implementation. The randomness cost of a specific design is usually measured in the number of random bits required per clock cycle or for the whole computation. The practical cost of implementing such randomness in hardware is often overlooked.

Two types of randomness are required in hardware secure masking. First, randomness consumed during computations of the masked implementation is

updated every cycle. As discussed in Section 2.5.3, this randomness is used in the refreshing layer or to achieve uniformity of the sharing in glitch-immune masking. Second, randomness is used to construct the masking of the input from the secret, which is generated once and used throughout the computation of the masked Boolean functions. Thus, the changing-of-the-guards [38] can be seen as using the randomness of masked inputs, which is called *guards*, as random bits in refreshing layers for achieving uniformity in TI. The primary contributor to the randomness cost is the refreshing layer, which is updated every cycle, because this requires a random number generator running in parallel with the masked implementation. Ensuring the quality and effectiveness of this randomness is the focus of this chapter.

4.1.1 Practical Randomness Sources

Masking relies on the uniformity and unpredictability of these random sources. Thus, a dedicated hardware module is required to generate randomness for practical implementation on FPGA or ASIC. For the implementations that consume randomness, both true random number generators (TRNG) and pseudo-random number generators (PRNG) are used. The difference between the two is that TRNG provides fresh entropy from its circuit operation noises, while PRNG produces deterministic outputs from the seeds it receives.

Entropy is a measure of the uncertainty of predicting the outcome of an experiment before it is known, which in this thesis refers to random bits. Thus, the high-entropy randomness means the output of each random bit is unpredictable. An ideal random source provides fresh bits with uniform distribution, thus each of the 2^n possible n-bit sequences has equal probability. Let $\mathbf{X}$ be a random variable and, for simplicity, we consider the case of single-bit random variables, so $\mathbf{X}$ has a finite set of two values $\{0, 1\}$. The entropy depends on the probabilistic distribution μ of $\mathbf{X}$, so let the binary discrete random variable $\mathbf{X}$ has outcomes from $\{0, 1\}$ with probabilities $\{p_0, p_1\}$ such that $p_0 + p_1 = 1$ and $p_0, p_1 \geq 0$. Specifically, two entropy measurements, i.e., Shannon entropy and min-entropy, are commonly used and NIST recommends using min-entropy as the metric of evaluation the randomness of TRNG [113]. Shannon entropy measures the expected, or average, gain of information from an observation, and min-entropy only looks at the most predictable outcome, which is a conservative measure.

$$\begin{array}{ll} \text{Shannon entropy} & H_1(\mathbf{X}) = -p_0 \log_2 p_0 - p_1 \log_2 p_1 \\ \text{Min-entropy} & H_{min}(\mathbf{X}) = -\log_2 \max\{p_0, p_1\} \end{array} \tag{4.1}$$

Consequently, if $\mathbf{X}$ is uniformly distributed, i.e., has equal probability $p_0 = p_1 = 0.5$, the two entropy measures are both equal to $H_{min}(\mathbf{X}) = H_1(\mathbf{X}) = -\log_2 0.5$

and reach their maximum values.

TRNGs introduce uncertainty by exploiting entropy sources in digital circuits, which rely on physically unpredictable processes, such as timing jitter, thermal noise, or metastability. After raw bits are extracted from entropy sources, a post-processing step is performed to improve the randomness quality by compressing the number of output bits. Due to the difficulty of implementing a TRNG, physically leakage assessment tests of hardware secure masking assume that the entropy is provided externally, either by another FPGA or a computer.

Cryptographically strong PRNGs produce output that is computationally indistinguishable from true randomness for any computationally bounded adversaries but do not add the information-theoretic entropy. They are widely used in cryptographic protocols for generating nonces, salts, and cryptographic keys. In hardware secure masking, the generated random bits ensure the uniformity of shares by `XOR` with each share. The (g)-extended probe does not recover the randomness values themselves, so the quality of randomness is assumed to be ideal and is overlooked both in terms of implementation cost and adversary model. However, uniform randomness is vital for overall security.

A common practice in hardware secure masking is to combine a TRNG and a PRNG and assume the TRNG prepares the random seed for the PRNG, which generates random bits every cycle. However, in practical TVLA experiments, only the PRNG is implemented alongside the target masked implementation on the same FPGA. After $\sim 10k$ computations, the PRNG's seeds are refreshed. Because the cost of PRNG was neglected in comparing hardware secure masking, there was a lack of best practice regarding the area and throughput of PRNG. For example, De Cnudde et al. [35] used a fully unrolled PRINCE implementation with OFB mode [22] with a fresh key for each masked AES execution. Wegner et al. [115] utilised a Linear Feedback Shift Register (LFSR) for each bit of randomness. The author of this thesis used the post-processing part of the Motorola TRNG, which consists of an LFSR and a Cellular Automata Shift Register (CASR), to replace the single LFSR mentioned before. Otherwise, a SHA-256 or SHA-3 hash function based PRNG is used to produce all needed random bits in advance and stored in registers, which is only possible for physical leakage assessment but not for practical applications. These methods produce low-quality randomness with light PRNG or use heavy implementations for better randomness. Therefore, a more systematic study is needed to explore efficient PRNG designs that balance high throughput, low area cost, and strong randomness quality.

4.1.2 Efficient Randomness Generation

Efficiently generating robust and high-entropy pseudo-randomness is a very important but frequently overlooked problem. Cassiers et al. [29] evaluate various efficient cryptographic building blocks based on their throughput-area ratio, exploring their potential for designing PRNGs optimized for efficient randomness generation. Their analysis indicates that the linearity of unrolled LFSR output can compromise the probing security of the masked implementation. Thus, using cryptographically more robust and secure primitives, i.e., stream ciphers, is necessary. Unrolled stream ciphers, such as Trivium and Bivium B, are found to be the most efficient building blocks for generating randomness in hardware-secure masking.

Trivium is a stream cipher developed by De Cannière [40] and was standardized as part of the lightweight stream cipher standard ISO/IEC 29192-3. Trivium operates on $\mathbb{F}_2$ and takes two input parameters: an 80-bit key, $\{K_1, \ldots, K_{80}\}$, and an 80-bit initialization vector, $\{IV_1, \ldots, IV_{80}\}$.

$$\begin{aligned}
&(s_1, s_2, \ldots, s_{93}) \leftarrow (K_1, \ldots, K_{80}, 0, \ldots, 0) \\
&(s_{94}, s_{95}, \ldots, s_{177}) \leftarrow (IV_1, \ldots, IV_{80}, 0, \ldots, 0) \\
&(s_{178}, s_{279}, \ldots, s_{288}) \leftarrow (0, \ldots, 0, 1, 1, 1)
\end{aligned} \tag{4.2}$$

$$\begin{aligned}
t_1 &\leftarrow s_{66} + s_{91} \cdot s_{92} + s_{93} + s_{171} \\
t_2 &\leftarrow s_{162} + s_{175} \cdot s_{176} + s_{177} + s_{264} \\
t_3 &\leftarrow s_{243} + s_{286} \cdot s_{287} + s_{288} + s_{69} \\
(s_1, s_2, \ldots, s_{93}) &\leftarrow (t_3, s_1, \ldots, s_{92}) \\
(s_{94}, s_{95}, \ldots, s_{177}) &\leftarrow (t_1, s_{94}, \ldots, s_{176}) \\
(s_{178}, s_{279}, \ldots, s_{288}) &\leftarrow (t_2, s_{178}, \ldots, s_{287})
\end{aligned} \tag{4.3}$$

First, the key and initial value bits initialize the 288 bits of the state register, as described by Equation (4.2). Second, in each cycle, the cipher updates its state registers by having three non-linear feedbacks, also called NLFSRs. The updated bits are the $\{t_1, t_2, t_3\}$ of Equation (4.3). After the 1152 initialization rounds, the output bits are streamed as $z = t_1 + t_2 + t_3$. Bivium B is a reduced-security stream cipher that uses two NLFSRs compared to the three of Trivium. Although practical attacks on this reduced security version exist, Bivium B is argued to remain applicable in the scenario of hardware secure masking, given that the complexity of the practical attacks is significant. Kreyvium, by Canteau et al. [26], extends Trivium with two additional registers to store the 128-bit key and 128-bit initialization vector and retains the original Trivium multiplicative depth. Compared to Trivium, the longer key and initialization vector lead to a higher security level and a greater maximal number of bits streamed from the cipher. The only difference is that the key and IV initialize two 128-bit shift registers in addition to the three NLFSRs. Given that the

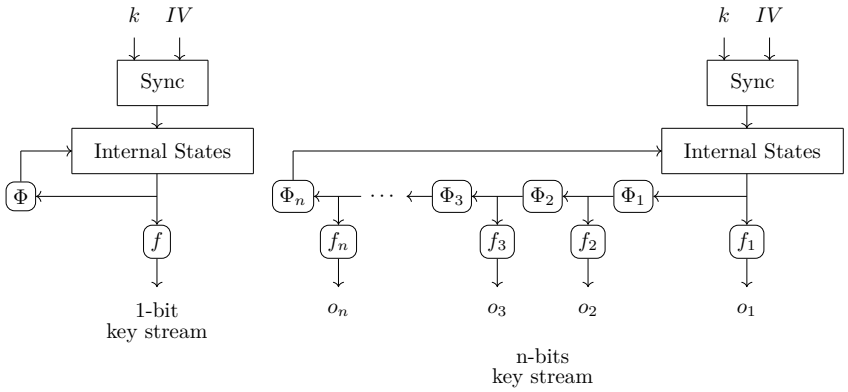


Figure 4.1: The structure of a key-stream generator and its unroll implementation [26]

number of internal state registers is increased, the throughput-area ratio of Kreyvium is lower than that of Trivium.

Similar to the iterated round functions in block ciphers or permutations, the state update and output functions of a stream cipher, which are typically executed sequentially, can be unrolled. This approach involves implementing multiple hardware instances without intermediate registers, allowing the combinatorial logic of multiple steps to be evaluated within a single cycle. Canteau et al. [26] divide the key-stream generation into a resynchronization function, a transition function, and a filter function.

- The resynchronization function *Sync* processes or expands through a pre-computation phase the IV and the key to produce an m-bit initial state.
- The transition function Φ computes the next state.
- The filtering function f derives a key-stream bit from the current internal state.

An unrolled implementation of such a stream cipher is thus n instances of transition and filtering functions to produce a key stream of n bits per cycle. Accordingly, the state updates of each cycle go through n transition functions, as shown in Figure 4.1.

Cassiers et al. [29] conclude that the unrolled Trivium gives the best throughput-area ratio for hardware implementations, among which a $\times 64$ unrolling is optimal.

This provides guidance for future PRNG implementation for hardware secure masking.

4.2 Reuse Randomness in Hardware Secure Masking

It is possible to reduce randomness cost without compromising security by reusing randomness to refresh intermediate shared products. The first such randomness-reuse technique is the changing-of-the-guards proposed by Daemen [38]. This technique can be applied to Threshold Implementations to achieve uniformity without consuming randomness. The changing-of-the-guards utilizes the masking shares from other uncorrelated inputs to correct the non-uniformity of the output shares. In this section, the first study generalizes the changing-of-the-guards applied to $d+1$ masking in addition to TI. The following studies explore the $d+1$ masking on Boolean functions that can achieve uniformity without the help of randomness, which is similar to uniform TI as shown in Equation (3.8). Section 3.3.1 summarizes the prior works of low-latency $d+1$ masking, which show the increased randomness cost associated with achieving low-latency gadgets. This section concludes with the state-of-the-art low-random and low-latency $d+1$ masking research overview.

4.2.1 Guarding the First-Order Masked AES Implementations

The changing-of-the-guards method was initially proposed for $td+1$ masking, but it can be extended to $d+1$ masking schemes, which inherently consume more randomness. Askeland et al. [6], including the author of this thesis, applied it to first-order masked implementations with three different sharing of the S-box. The generalized changing-of-the-guards technique, which incorporates multiple register stages and additional randomness, is applied to the serialized and round-based architecture of the AES implementations.

The Generalized Changing-of-the-Guards on First-Order Glitch-Immune Masking

As shown in Figure 2.6 and Equation (2.9), Daemen's original proposal was based on TI, i.e., the shared Boolean functions are non-complete, uniform, and executed within a single clock cycle. In the context of first-order glitch-immune hardware secure masking, multiple register stages and randomness are needed.

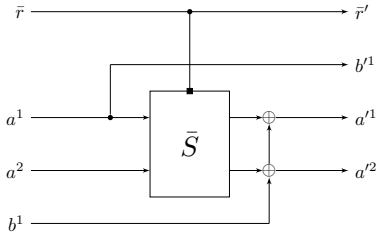


Figure 4.2: The changing-of-the-guards of the shared S-box $\bar{S}$ with two shares uses the randomness $\bar{r}$ [6]

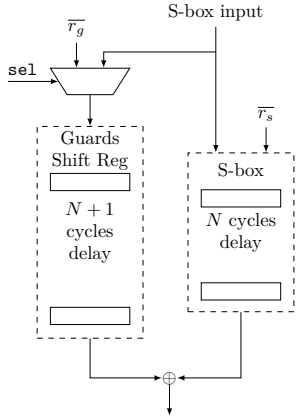


Figure 4.3: Application of the changing-of-the-guards in the serialized architecture [6]

The generalized changing-of-the-guards aims at preserving uniformity of the masked Boolean functions over these register stages while enabling the reuse of randomness. The two-share example is illustrated in Figure 4.2, where the shared S-box $\bar{S}$ incorporates glitch-immune masking through stages of function decomposition. The randomness $\bar{r}$ is applied to the refreshing layers of intermediate stages, and the reused random share b^1 collected from other S-boxes is applied at the output to ensure a uniform sharing of the masked S-box.

This method [6], as depicted in Equation (4.4) that is applied to a masked function $\bar{S}$, operates on input shares $\{a^1, ..., a^s\}$, guards $\{b^1, ..., b^{s-1}\}$, and randomness $\bar{r}$. The outputs consist of refreshed shares $\{a'^1, ..., a'^s\}$, guards to

the next $\{b'^1, ..., b'^{s-1}\}$, and the reused randomness.

$$\begin{array}{ll} \text{Reuse Randomness:} & \bar{r}' = \bar{r} \\ \text{Output Shares:} & a'^1 = S^1(a^1, \ldots, a^s, \bar{r}) \oplus b^1 \\ & a'^{s-1} = S^{s-1}(a^1, \ldots, a^s, \bar{r}) \oplus b^{s-1} \\ & a'^s = S^s(a^1, \ldots, a^s, \bar{r}) \oplus b^1 \oplus \ldots \oplus b^{s-1} \\ \text{Guards} & b'^1 = a^1, \ldots, b'^{s-1} = a^{s-1} \end{array} \tag{4.4}$$

This construction preserves correctness, first-order probing security, and uniformity. The correctness follows from the inherent correctness of $\bar{S}$, with each guard b^i being added to multiple output shares in the same way as the refreshing layer. First-order probing security is preserved under the assumption of a joint uniform input $\{a^1, ..., a^s\}$ and the first-order probing security of $\bar{S}$, since each guard b'^i depends solely on one share a^i, and the input guards $\{b^1, ..., b^{s-1}\}$ are non-complete. Finally, the method's uniformity is proved by demonstrating the construction $\{\bar{r}, a^1, ..., a^s, b^1, ..., b^{s-1}\}$ is invertible as a permutation.

Overall, the changing-of-the-guards method enables the transformation of any first-order masked $\bar{S}$ into a uniform masked function, which allows the efficient reuse of randomness within S-box computations.

AES S-box Structure

Askeland et al. [6] constructed three AES S-boxes using the generalized changing-of-the-guards, which are shared with tower field decomposition. Equation (3.2) gives the overarching components of the tower field decomposition. The three S-boxes demonstrate the trade-offs between the number of stages and the area cost. The detailed sharing of each S-box is skipped here and can be found in the original paper. Randomness in the designs can be reused across all S-box instances because the changing-of-the-guards makes the whole construction a permutation. Their design considerations for each design are summarized as follows:

- The first design employs two shares CMS, operates over five cycles, and utilizes 54 random bits. The masked quadratic multipliers follow Equation (2.6). This approach is the most efficient in terms of area among the three designs presented. However, this reduction in area comes at the cost of increased latency.
- The second employs three shares TI, operates over four cycles and utilizes 36 random bits. The three-share masked multipliers are computed and applied with refreshing randomness as Equation (3.9). Compared to the

two-share design, this S-box requires more area but benefits from one cycle less latency. The four-cycle latency is well-suited for the serialized architectures of AES, where the only S-box instance is used for computing both AES round functions and key scheduling.

- The third design utilizes three shares, operates over three cycles, and requires 40 random bits. This approach builds upon the three-shared multipliers of TI, which is also from Equation (3.9), but introduces a transition to four shares to enable non-complete masking of the inversion over $\mathbb{F}_{2^4}$, which is cubic vectorial Boolean functions. Each cubic term $x_a x_b x_c$ is computed with four-share direct sharing of TI. After each shared monomial is linearly added, a refreshing layer with random bits is applied as Equation (2.5). This approach reduces the latency to three cycles compared to the previous designs, achieving faster computation. However, this reduction in latency comes at the cost of a greater area cost. The round-based architecture of AES, which instantiates multiple S-boxes in parallel, prefers this design because of its lower latency structure.

Hardware Architecture with the Changing-of-the-Guard

In the round-based AES-128 architecture, 20 S-box instances are utilized, with 16 dedicated to the `SubByte` operation in the state function and 4 handling the `SubWord` operation in key expansion. The `MixColumns` and `ShiftRows` are linear operations on a bus width of #shares $\times$ 128 bits. After each round of AES encryption, the cipher state is stored in a register array. The changing-of-the-guards method is applied to the parallel S-boxes, following the definition by Daemen [38].

A serialized architecture uses a single S-box for key expansion and state processing. As a result, the bus width is #shares$\times 8$ bits, and its implementation is feasible for physical test on an FPGA evaluation board. Each masked byte, whether from `AddRoundKey` or `RotWord` in key expansion, is sequentially fed into the masked S-box. Consequently, the serialized architecture requires at least 20 cycles per round, best fitting the masked S-box with four-cycle latency. Figure 4.3 shows the implementations of the changing-of-the-guards in the serialized architecture. The guards $\overline{r_g}$ are initialized with random values, and subsequent guard values are derived from the S-box input. Guards shift registers store these shares, introducing an extra delay cycle relative to the S-box delay before they are applied to the output. Thus, $\overline{r_g}$ consists of $8 \times (\#\text{shares} - 1)$ random bits for initializing the guard shares, which remain static throughout the AES encryption process.

In conclusion, in the whole process of the cipher computation, the randomness is recycled using the generalized changing-of-the-guards to ensure first-order glitch-extended probing security, which replaces all randomness with so-called *guards* and can be flexibly applied to masked implementations with different levels of parallelism.

4.2.2 Amortize Randomness in First-Order Two-Share Masking

Rather than reusing randomness, the goal of reducing or amortizing randomness can be achieved by exploiting the algebraic properties of Boolean masking schemes. One initial example is the uniformity of TI, which can be achieved either by refreshing random bits or by constructing uniform shared Boolean functions. Recent works further explore the $d+1$ masking scenarios and develop techniques to achieve joint uniformity on the glitch-extended probes without using randomness.

Shahmirzadi and Moradi [105] achieve first-order secure hardware implementations with two-share $d+1$ masking that do not require fresh masks. The proposed method first looks beyond masking monomial AND gate $x = a \cdot b$, which is a quadratic function. Functions like $x = a \cdot b + c$ or $x = a \cdot b + b$ are considered and masked in DOM as illustrated in Equation (4.5). The masking shares of the second variable $\{b, c\}$ are used like the randomness in the refreshing layer of $d+1$ masking. This clearly does not break the non-completeness requirement, but uniformity needs to be evaluated for the overall probing security. The author examines the uniformity by placing a probe on each output share, which is $\{x_1, x_2\}$, and observes that its glitch-extended value $\{x'_1, x'_2\}$ or $\{x'_3, x'_4\}$ has an identical joint probability distribution. This means the probed values are uniformly distributed to the secret value, which leads to the overall probing security. This technique is systematically applied to two-share masking the quadratic and cubic functions for first-order glitch-extended probing security.

However, this gadget is not trivially composable and has not been evaluated under composable security notions. The authors gave a counter-example of parallel composing two masked functions, which results in an identical joint probability distribution requirement over the secret inputs of the two functions. Without the help of fresh random bits, this becomes a strong constraint that the proposed searching algorithms do not include.

$$\begin{array}{lll} a_1 b_1 \oplus \{c_1, b_1\} & \rightarrow x'_1 & \\ a_1 b_2 & \rightarrow x'_2 & x'_1 \oplus x'_2 = x_1 \\ \hline a_2 b_1 & \rightarrow x'_3 & x'_3 \oplus x'_4 = x_2 \\ a_2 b_2 \oplus \{c_2, b_2\} & \rightarrow x'_4 & \end{array} \tag{4.5}$$

Another interesting result is that a 2-input `AND` gate can be rewritten as $x = a \cdot b = \bar{a}b + b$, which fits into Equation (4.5). Thus, uniform $d + 1$ sharing exists for first-order security on quadratic monomials, which the authors also extend to higher-order scenarios.

4.2.3 Second-Order Three-Share Masking

For second-order glitch-extended probing security, randomness ensures the statistical independence of two probes across different cycles, which would otherwise cause values to correlate across multiple rounds. Therefore, to reduce the randomness cost, the aforementioned techniques must be extended accordingly.

Shahmirzadi and Moradi [106] extend the technique to two-share first-order masking without randomness on quadratic functions to the three-share second-order case. It is worth noting that the authors provide second-order masking solutions for certain quadratic functions such as $y = ab + c$ without requiring randomness, while other quadratic monomials and bijective functions still rely on fresh random bits to prevent multivariate leakages.

Shahmirzadi et al. [104] further investigate the first- and second-order masking of cubic functions. This method uses a similar searching process to find uniform sharing of the coordinate functions with the help of a pairing of two S-boxes. Figure 4.5 shows the pairing scheme of two masked S-boxes, which is an extension of the changing-of-the-guards technique. S-box inputs $\{\bar{x}, \bar{y}\}$ are applied to each other's coordinate functions so that the shares of the expansion layers are uniform. Random bits $\{r_1, r_2\}$ are applied in a ring refreshing manner, as Equation (2.7), which achieves second-order probing security over multiple rounds. When two probes are placed within the same pair of masked S-boxes, the randomness $\{r_1, r_2\}$ prevents them from revealing any secret information. When probing two different pairs, the inputs from the paired S-boxes serve as randomness. Thus, with the existence of such a pairing structure, these random numbers $\{r_1, r_2\}$ can be collected and reused, resulting in a saving of randomness.

Beyne et al. [15] introduced a security framework based on linear cryptanalysis, which achieves quantitative bounds on the adversary's advantage under the proposed bounded-query probing model. The model was later extended with noise on probes by Beyne et al. [14], so the bounds are improved, which leads to more efficient designs. The bounded-query probing model quantifies the security of a circuit $C(k)$ with input $k_b = \{k_0, k_1\}$ under the t-threshold-probing model with a left-or-right security game. The challenger $\mathcal{C}$ picks a random bit $b \leftarrow \$$ and provides a left-right oracle $\mathcal{O}^b$ to the adversary $\mathcal{A}$. The adversary is able

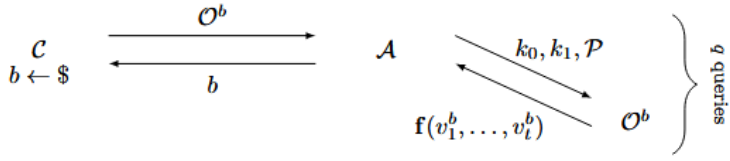


Figure 4.4: The bounded-query probing model consisting of a challenger $\mathcal{C}$, an adversary $\mathcal{A}$, a left-right oracle $\mathcal{O}^b$, two inputs k_0, k_1, a set of probes $\mathcal{P}$, and a set of probed wire values $(v_1^b, ..., v_t^b)$ of the circuit $C(k_b)$ [15]

to query the oracle, i.e., produce a set of probes $\mathcal{P}$ where $\left|\mathcal{P}\right| \leq t$. The probe set and two inputs $\{\mathcal{P}, k_0, k_1\}$ are sent to the oracle, which responds with the probed wire values of $C(k_b)$. Accordingly, a glitch-extended probe gets all wires leading to the probe until a register stage. The adversary $\mathcal{A}$ collects q queries in total and responds to the challenger with a guess for $b \in \{0, 1\}$. The advantage of the adversary is defined as:

$$\mathrm{Adv}_{t\text{-}\mathsf{thr}}(\mathcal{A}) = |\Pr[\mathcal{A}^{\mathcal{O}^0} = 1] - \Pr[\mathcal{A}^{\mathcal{O}^1} = 1]|\,. \tag{4.6}$$

Theorem 1 of Beyne et al. [15] gives an upper bound of this advantage. Under such a model, probed wires either closely reach uniform randomness or reveal no information about the secret input. For the former, the probed values can leak information with a substantial number of such probes, but remain indistinguishable from uniform random values unless an extremely significant number of probing queries is performed. In practice, these values correspond to shares of the state placed far apart, which corresponds to multiple rounds. Thus, this quantitative bound is constrained by the number of probes q and the conditional probability distribution on the probes that could leak information. This conditional probability distribution is deduced from linear cryptanalytic properties.

4.3 My Contributions

This chapter discusses the randomness requirement in hardware secure masking and methods to reduce the randomness cost. Randomness is essential to maintain the uniformity of output shares in $td + 1$ masking and to satisfy the non-completeness property in $d+1$ masking. The primary strategies for reducing the randomness cost involve extending the changing-of-the-guards technique and reusing randomness without compromising glitch-extended probing security.

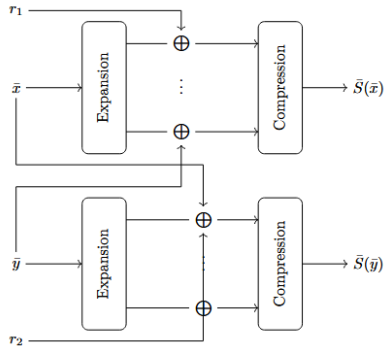


Figure 4.5: The pairing of two masked S-boxes [104]

By incorporating techniques such as linear cryptanalysis, the randomness requirements of higher-order masking can be further reduced.

4.3.1 Apply the Changing-of-the-Guards Technique to the First-Order Masked AES Implementation

The conference paper titled "Guarding the First Order: The Rise of AES Masking" [6] is presented at the Conference of Smart Card Research and Advanced Applications, CARDIS 2022. This work extends the changing-of-the-guards technique to achieve the first-order probing-secure masked AES implementation. I am one of the primary authors of this paper and presented the work at the conference.

This research generalizes the changing-of-the-guards technique to multiple register stages under $d + 1$ masking and applies it to the masked AES implementation with serialized and parallel architecture. In later research, this technique was further extended and combined with a randomness-reducing technique in $d+1$ masking. Saving randomness in higher-order masking remains a research direction.

Chapter 5

Conclusions

This chapter summarizes the achievements of this thesis with respect to the research problems posed in Section 1.1 and suggests potential future works in the field of hardware secure masking.

5.1 Summary

In this work, we have discussed two aspects of constructing hardware secure masking for symmetric key primitives. Chapter 2 explains the hardware secure masking as a countermeasure against passive physical side-channel attacks. It covers the threat model, adversary model, and the concepts of countermeasures in hardware. Chapter 3 focuses on reducing the latency cost of masking in glitchy digital circuits from the perspective of masking schemes and circuit implementations. Chapter 4 discusses the randomness cost of hardware secure masking, which should not be overlooked regarding the efficiency and security of generating randomness on the fly, meanwhile saving the total cost without compromising its glitch-immunity. This thesis advances the state-of-the-art with respect to low-latency and low-randomness hardware secure masking.

Low Latency Threshold Implementations: The emphasis of this work is on reducing the latency cost, which is intrinsically brought by register stages that stop glitch propagation. The work on LLTI explores removing registers between decomposed masked Boolean functions. To achieve the same latency in clock cycles, this work improves the overall area cost compared to direct TI sharing of

a cubic AND gate by a 60% reduction in area and a 137% increase in maximum frequency. To protect the AES encryption with two-cycle latency on an S-box, where the AES S-box is decomposed into two stages of cubic functions, the area cost is reduced by almost 50%. LLTI is a algorithms for $td + 1$ masking high-order Boolean functions. It leverages the non-completeness property by dividing the complex gates into low-degree layers and constructing sharing on each layer. This approach limits the randomness requirements while optimizing for reduced cycle latency. In comparison, low-latency $d + 1$ masking schemes, although advantageous in terms of area efficiency, increase register usage and require randomness. Overall, LLTI provides a practical and efficient approach based on TI to construct shared Boolean functions of higher algebraic degree.

Glitch-Stopping Circuits: The other line of low-latency masking is developed at the logic circuit level. Compared to the works based on glitch-free circuits, which are dual-rail with pre-charge circuits, our work focuses on using logic gates or latches to stop glitch propagation. This approach introduces minimal changes compared to conventional register-based masked implementations, avoiding the complexity of the pre-charging phase and synchronization mechanisms. It maintains the overall synchronous operations with a modified clock configuration and the glitch-immune masking scheme, making this method easy to implement on hardware. The area cost compared to register-based implementations is reduced by approximately 10%, and the computation of the dth-order masking can be completed in one clock cycle. Compared to glitch-free circuits like SESYM, our work does not have the risk of the early-propagation effects that exist in most DRP logic, except for LMDPL. In terms of the implementation cost, the four-stage AND-based GSC is 23.9% larger in the area cost and 45.8% less of the maximum clock frequency compared to LMDPL. However, LMDPL is restricted to first-order security, while our method preserves the glitch-immunity of high-order secure masking schemes. One additional observation related to the glitch propagation effect in digital circuits is that glitches happen before each wire has a stable value. As a result, the latency-in-time through the masked S-box stages corresponds to the overall propagation delay across all stages instead of the longest critical path in the register pipeline.

Randomness cost: Our proposal in this line of research aims to reduce the use of randomness in hardware. It has been demonstrated that the randomness cost can be reduced by generalizing the changing-of-the-guards technique to both $td + 1$ and $d + 1$ masking over multiple register stages. Thus, implementations that rely solely on reused randomness are possible in first-order masking, which can be subsequently extended to second-order cases. Reusing randomness eliminates the requirement for randomness generation updated every clock cycle,

which would otherwise require a secure and efficient implementation of unrolled stream ciphers.

Outcomes and Reflections: During the course of this doctoral research, defining these research problems and proposing solutions is a dynamic and evolving process. It requires flexibility as new findings emerge in our research community and a deep commitment to problem-solving. This flexibility is supported by the overview of the research area, which can be developed through the study of foundational literature and seminal publications. The two main categories of countermeasures against passive SCA, namely glitch-free circuits and glitch-immune masking, can still be categorized under the technique of hiding and masking explained in the book *Power analysis attacks: revealing the secrets of smart cards* [74] published in 2007. The attacks and the adversary models in concern are closely related to the original DPA proposed by Kocher et al. in CRYPTO 1999 [70]. Understanding these foundational and seminal works is essential to remain focused and committed in the developing field of hardware security.

Another challenge arises from physical experiments, particularly in cases of failure. Published research tends to favor successful countermeasures supported by well-demonstrated results of experiments. Implementations that leak, though they may provide a more profound understanding of the underlying research problems, are often neglected by researchers in academic discourse. Dr. De Cnudde, in his dissertation and with co-authors [34, 33, 41], documented the detected leakage, which results in the factor of coupling considered in the robust probing model. This thesis includes cases of failure in choosing FPGA primitives for constructing glitch-stopping circuits [120]. Additionally, the test setup, along with the electronic components and operation-dependent hardware implementations involved in the physical leakage test, must be well-configured. Such practical experience and analytical skills are essential, yet often underappreciated in research that requires real-world experiments.

5.2 Future Works

This section discusses future work in hardware-secure masking. Hardware security evolves in response to emerging computing paradigms and algorithms. This section highlights three directions: developing the adversary model for hardware-secure masking, designing tailored masking techniques for AI in hardware, and advancing masking for heterogeneous computing systems.

5.2.1 Succinct Security Models for Efficient Masking

The implementations in this thesis do not consider the transition-extended probing model, which is still under active discussion in hardware secure masking. The existing circuit model used in the glitch-extended model does not account for cyclic or stateful circuits, which means the transition-extended probing model does not match the implementations of hardware secure masking. Cassier et al. [31] extend the circuit model by unrolling the loop, defining the encoding, decoding, and execution of a circuit of each memory state on the level of a composable gadget. Müller et al. [81] proposed transitional probe extension algorithms used with statistical verification tools. Despite these advancements, the provable security of transition-extended probes is only based on the secure gadgets and does not fully account for the transition-extended probing security of the whole hardware implementation. Additionally, coupling-extended probes are often ignored in the design and implementation of hardware secure masking because this effect is at a different abstraction level. Attacks on FPGA using long-wires [48] and the power distribution network [51] show the need for countermeasures against such physical leakage channels. However, no matter how the model of such a leakage source and attack surface align with coupling, the countermeasures are not at the level of Boolean logic and masking. Circuit-level countermeasures, i.e., on power distribution network or P&R methods, can be developed against such attacks. Therefore, in practical evaluations of hardware secure masking, it is critical to document leakages not captured by current formal adversary models to guide future improvements in both modeling and countermeasure development.

5.2.2 Hardware Secure Masking for Trustworthy AI

Recent advancements in machine learning have driven the development of highly efficient neural network (NN) accelerators. Consequently, growing security concerns over data and model privacy due to SCA and fault attacks pose significant challenges to the development of trustworthy AI hardware. Maji et al. [73] proposed a TI-based NN accelerator that enhances resistance to power and electromagnetic SCA attacks by securing both model parameters and input data. DOM is used by Dubey et al. [43] to mask the machine learning operations, such as the activation function and the output layer classification. The LLTI and GSC proposed in this thesis can also be applied as masking schemes for neural network accelerations. However, the different characteristics of cryptographic and machine learning applications result in different performance and security requirements. Ashok et al. [5] integrated digital in-memory computing architectures to minimize data movement and improve energy

efficiency. Their proposed design addresses passive physical SCAs and memory bus-probing attacks by introducing lightweight encryption, secure compute techniques, and secret key generation using digital in-memory computing circuitry. Lee et al. [71] provided a systematic review of hardware security challenges in NN accelerators. They classified hardware attack surfaces into three categories: off-chip memory attacks, on-chip SCA threats, and hardware Trojans introduced during chip fabrication. This underscores the need for a unified security evaluation framework tailored for trustworthy AI hardware, where hardware secure masking plays a significant role. Current TI-based approaches introduce significant area and energy overhead. In addition, the security requirements of hardware AI systems are lighter compared to the symmetric key algorithms. Future work can explore efficient masking techniques, i.e., domain-specific approximations, to reduce the security-performance trade-off. Also, on emerging computing systems like in-memory computing, investigating its security mechanisms will be crucial for next-generation AI hardware.

5.2.3 Apply Masking to Hardware-Software Co-design

The work presented in this thesis primarily focuses on hardware secure masking, considering only the security of dedicated hardware executing a fixed algorithm. Masking in software [47] differs from hardware secure masking but shares the same adversary model, cf. Section 5.2.1. Combining the masking in hardware and software offers the flexibility to secure multiple algorithms while maintaining the efficiency of hardware-software co-design, which is essential for securing complex cryptographic algorithms. For instance, with the standardization of post-quantum cryptographic algorithms, dedicated hardware co-processors are expected to be deployed to accelerate such algorithms while protecting them against side-channel attacks. Recent research has explored the side-channel resistance of these ciphers through masking techniques such as secure arithmetic-to-Boolean masking conversions, masked comparisons [12, 39], and masked hardware accelerators using RISC-V instruction set extension [46]. Future work can focus on micro-architectural leakage behavior and secure interface implementations between hardware and software components. In this context, masking techniques that offer protection against SCA in both hardware and software represent an interesting research direction.

Bibliography

[1] Amazon Web Services. Lightweight Hypervisor - AWS Nitro System. https://aws.amazon.com/ec2/nitro/. [Accessed 24-01-2025].

[2] Arm. TrustZone for Cortex-M. https://www.arm.com/technologies/trustzone-for-cortex-m. [Accessed 09-12-2024].

[3] Arribas, V., Bilgin, B., Petrides, G., Nikova, S., and Rijmen, V. Rhythmic KECCAK: SCA Security and Low Latency in HW. *IACR Trans. Cryptogr. Hardw. Embed. Syst. 2018*, 1 (2018), 269–290.

[4] Arribas, V., Zhang, Z., and Nikova, S. LLTI: Low-Latency Threshold Implementations. *IEEE Trans. Inf. Forensics Secur. 16* (2021), 5108–5123.

[5] Ashok, M., Maji, S., Zhang, X., Cohn, J., and Chandrakasan, A. P. Digital In-memory Compute For Machine Learning Applications With Input And Model Security. *IEEE Journal of Solid-State Circuits* (2025), 1–13.

[6] Askeland, A., Dhooghe, S., Nikova, S., Rijmen, V., and Zhang, Z. Guarding The First Order: The Rise of AES Maskings. In *Smart Card Research And Advanced Applications - CARDIS 2022* (2022), I. Buhan and T. Schneider, Eds., vol. 13820 of *Lecture Notes in Computer Science*, Springer, pp. 103–122.

[7] Balasch, J., Gierlichs, B., Grosso, V., Reparaz, O., and Standaert, F. On The Cost of Lazy Engineering For Masked Software Implementations. In *Smart Card Research and Advanced Applications - 13th International Conference, CARDIS 2014, Paris, France, November 5-7, 2014. Revised Selected Papers* (2014), M. Joye and A. Moradi, Eds., vol. 8968 of *Lecture Notes in Computer Science*, Springer, pp. 64–81.

[8] Barthe, G., Belaïd, S., Dupressoir, F., Fouque, P., Grégoire, B., Strub, P., and Zucchini, R. Strong Non-Interference And Type-directed Higher-order Masking. In *Proceedings of The 2016 ACM SIGSAC*

Conference on Computer and Communications Security (2016), E. R. Weippl, S. Katzenbeisser, C. Kruegel, A. C. Myers, and S. Halevi, Eds., ACM, pp. 116–129.

[9] Barthe, G., Belaïd, S., Fouque, P., and Grégoire, B. maskVerif: A Formal Tool for Analyzing Software and Hardware Masked Implementations. *IACR Cryptol. ePrint Arch.* (2018), 562.

[10] Barthe, G., Dupressoir, F., Faust, S., Grégoire, B., Standaert, F., and Strub, P. Parallel implementations of masking schemes and the bounded moment leakage model. In *Advances In Cryptology - EUROCRYPT 2017 - 36th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Paris, France, April 30 - May 4, 2017, Proceedings, Part I* (2017), J. Coron and J. B. Nielsen, Eds., vol. 10210 of *Lecture Notes in Computer Science*, pp. 535–566.

[11] Batina, L., Gierlichs, B., Prouff, E., Rivain, M., Standaert, F., and Veyrat-Charvillon, N. Mutual Information Analysis: A Comprehensive Study. *J. Cryptol. 24*, 2 (2011), 269–291.

[12] Beirendonck, M. V., D'Anvers, J., and Verbauwhede, I. Analysis And Comparison of Table-based Arithmetic To Boolean Masking. *IACR Trans. Cryptogr. Hardw. Embed. Syst. 2021*, 3 (2021), 275–297.

[13] Belaïd, S., Coron, J., Prouff, E., Rivain, M., and Taleb, A. R. Random Probing Security: Verification, Composition, Expansion and New Constructions. In *Advances In Cryptology - CRYPTO 2020 - 40th Annual International Cryptology Conference, CRYPTO 2020, Santa Barbara, CA, USA, August 17-21, 2020, Proceedings, Part I* (2020), D. Micciancio and T. Ristenpart, Eds., vol. 12170 of *Lecture Notes in Computer Science*, Springer, pp. 339–368.

[14] Beyne, T., Dhooghe, S., Moradi, A., and Shahmirzadi, A. R. Cryptanalysis of Efficient Masked Ciphers: Applications To Low Latency. *IACR Trans. Cryptogr. Hardw. Embed. Syst. 2022*, 1 (2022), 679–721.

[15] Beyne, T., Dhooghe, S., and Zhang, Z. Cryptanalysis of Masked Ciphers: A not so random idea. In *Advances In Cryptology - ASIACRYPT 2020 - 26th International Conference on the Theory and Application of Cryptology and Information Security, Daejeon, South Korea, December 7-11, 2020, Proceedings, Part I* (2020), S. Moriai and H. Wang, Eds., vol. 12491 of *Lecture Notes in Computer Science*, Springer, pp. 817–850.

[16] Bhasin, S., Guilley, S., Flament, F., Selmane, N., and Danger, J. Countering Early Evaluation: An Approach Towards Robust Dual-Rail

Precharge Logic. In *Proceedings of The 5th Workshop On Embedded Systems Security, WESS 2010, Scottsdale, AZ, USA, October 24, 2010* (2010), ACM, p. 6.

[17] Bilgin, B. *Threshold Implementations : As Countermeasure against Higher-Order Differential Power Analysis.* PhD thesis, University of Twente, Enschede, Netherlands, 2015.

[18] Bilgin, B., Gierlichs, B., Nikova, S., Nikov, V., and Rijmen, V. Higher-order Threshold Implementations. In *Advances In Cryptology - ASIACRYPT 2014* (2014), P. Sarkar and T. Iwata, Eds., vol. 8874 of *LNCS*, Springer, pp. 326–343.

[19] Bilgin, B., Nikova, S., Nikov, V., Rijmen, V., and Stütz, G. Threshold Implementations of All 3×3 and 4×4 S-Boxes. In *Cryptographic Hardware And Embedded Systems - CHES 2012 Proceedings* (2012).

[20] Bilgin, B., Nikova, S., Nikov, V., Rijmen, V., Tokareva, N. N., and Vitkup, V. Threshold Implementations of Small S-boxes. *Cryptogr. Commun. 7*, 1 (2015), 3–33.

[21] Blog, G. Pixel 6: Setting a new standard for mobile security. `https://security.googleblog.com/2021/10/pixel-6-setting-new-standard-for-mobile.html`. [Accessed 09-12-2024].

[22] Borghoff, J., Canteaut, A., Güneysu, T., Kavun, E. B., Knežević, M., Knudsen, L. R., Leander, G., Nikov, V., Paar, C., Rechberger, C., Rombouts, P., Thomsen, S. S., and Yalçın, T. PRINCE - A Low-latency Block Cipher for Pervasive Computing Applications. Cryptology ePrint Archive, Paper 2012/529, 2012.

[23] Bozilov, D., Knezevic, M., and Nikov, V. Optimized Threshold Implementations: Minimizing The Latency of Secure Cryptographic Accelerators. In *Smart Card Research And Advanced Applications - CARDIS 2020* (2020), S. Belaid and T. Guneysu, Eds., Lecture Notes in Computer Science, Springer, pp. 20–39.

[24] Brier, E., Clavier, C., and Olivier, F. Correlation Power Analysis With A Leakage Model. In *Cryptographic Hardware And Embedded Systems - CHES 2004: 6th International Workshop Cambridge, MA, USA, August 11-13, 2004. Proceedings* (2004), M. Joye and J. Quisquater, Eds., vol. 3156 of *Lecture Notes in Computer Science*, Springer, pp. 16–29.

[25] Canright, D. A Very Compact S-box for AES. In *Cryptographic Hardware And Embedded Systems - CHES 2005 Proceedings* (2005), pp. 441–455.

[26] Canteaut, A., Carpov, S., Fontaine, C., Lepoint, T., Naya-Plasencia, M., Paillier, P., and Sirdey, R. Stream Ciphers: A Practical Solution for Efficient Homomorphic-Ciphertext Compression. *J. Cryptol. 31*, 3 (2018), 885–916.

[27] Cassiers, G., and Bronchain, O. Scalib: A Side-Channel Analysis Library. *J. Open Source Softw. 8*, 86 (2023), 5196.

[28] Cassiers, G., Grégoire, B., Levi, I., and Standaert, F. Hardware Private Circuits: From Trivial Composition To Full Verification. *IEEE Trans. Computers 70*, 10 (2021), 1677–1690.

[29] Cassiers, G., Masure, L., Momin, C., Moos, T., Moradi, A., and Standaert, F. Randomness Generation for Secure Hardware Masking - Unrolled Trivium to the Rescue. *IACR Commun. Cryptol. 1*, 2 (2024), 4.

[30] Cassiers, G., and Standaert, F. Trivially and Efficiently Composing Masked Gadgets With Probe Isolating Non-Interference. *IEEE Trans. Inf. Forensics Secur. 15* (2020), 2542–2555.

[31] Cassiers, G., and Standaert, F. Provably Secure Hardware Masking in the Transition- and Glitch-Robust Probing Model: Better Safe than Sorry. *IACR Trans. Cryptogr. Hardw. Embed. Syst. 2021*, 2 (2021), 136–158.

[32] Chari, S., Rao, J. R., and Rohatgi, P. Template Attacks. In *Cryptographic Hardware And Embedded Systems - CHES 2002, 4th International Workshop, Redwood Shores, CA, USA, August 13-15, 2002, Revised Papers* (2002), B. S. K. Jr., Ç. K. Koç, and C. Paar, Eds., vol. 2523 of *Lecture Notes in Computer Science*, Springer, pp. 13–28.

[33] Cnudde, T. D., Bilgin, B., Gierlichs, B., Nikov, V., Nikova, S., and Rijmen, V. Does Coupling Affect The Security of Masked Implementations? In *Constructive Side-Channel Analysis and Secure Design - 8th International Workshop, COSADE 2017, Paris, France, April 13-14, 2017, Revised Selected Papers* (2017), S. Guilley, Ed., vol. 10348 of *Lecture Notes in Computer Science*, Springer, pp. 1–18.

[34] Cnudde, T. D., Ender, M., and Moradi, A. Hardware Masking, Revisited. *IACR Trans. Cryptogr. Hardw. Embed. Syst. 2018*, 2 (2018), 123–148.

[35] Cnudde, T. D., Reparaz, O., Bilgin, B., Nikova, S., Nikov, V., and Rijmen, V. Masking AES with $d + 1$ Shares in Hardware. In *Cryptographic Hardware And Embedded Systems - CHES 2016 - 18th International Conference, Santa Barbara, CA, USA, August 17-19, 2016,*

Proceedings (2016), B. Gierlichs and A. Y. Poschmann, Eds., vol. 9813 of *Lecture Notes in Computer Science*, Springer, pp. 194–212.

[36] Colbourn, C. J., and Dinitz, J. H. *Handbook of Combinatorial Designs, Second Edition (Discrete Mathematics And Its Applications)*. Chapman & Hall/CRC, 2006.

[37] Coron, J., Giraud, C., Prouff, E., Renner, S., Rivain, M., and Vadnala, P. K. Conversion of Security Proofs From One Leakage Model To Another: A new issue. In *Constructive Side-Channel Analysis and Secure Design - Third International Workshop, COSADE 2012, Darmstadt, Germany, May 3-4, 2012. Proceedings* (2012), W. Schindler and S. A. Huss, Eds., vol. 7275 of *Lecture Notes in Computer Science*, Springer, pp. 69–81.

[38] Daemen, J. Changing of The Guards: A Simple and Efficient Method for Achieving Uniformity in Threshold Sharing. In *Cryptographic Hardware And Embedded Systems - CHES 2017 Proceedings* (2017), pp. 137–153.

[39] D'Anvers, J., Heinz, D., Pessl, P., Beirendonck, M. V., and Verbauwhede, I. Higher-order Masked Ciphertext Comparison For Lattice-based Cryptography. *IACR Trans. Cryptogr. Hardw. Embed. Syst. 2022*, 2 (2022), 115–139.

[40] De Cannière, C. Trivium: A Stream Cipher Construction Inspired by Block Cipher Design Principles. In *Information Security* (Berlin, Heidelberg, 2006), S. K. Katsikas, J. López, M. Backes, S. Gritzalis, and B. Preneel, Eds., Springer Berlin Heidelberg, pp. 171–186.

[41] De Cnudde, T., Rijmen, V., Nikova, S., and of engineering science. Department of electrical engineering degree granting institution, K. L. F. Cryptography Secured Against Side-channel Attacks, 2018.

[42] Dhooghe, S. *New Security Models for Passive and Active Physical Attacks*. phdthesis, KU Leuven, 2021.

[43] Dubey, A., Ahmad, A., Pasha, M. A., Cammarota, R., and Aysu, A. ModuloNet: Neural Networks Meet Modular Arithmetic for Efficient Hardware Masking. *IACR Trans. Cryptogr. Hardw. Embed. Syst. 2022*, 1 (2022), 506–556.

[44] Duc, A., Dziembowski, S., and Faust, S. Unifying Leakage Models: From Probing Attacks To Noisy Leakage. *J. Cryptol. 32*, 1 (2019), 151–177.

[45] Faust, S., Grosso, V., Pozo, S. M. D., Paglialonga, C., and Standaert, F. Composable Masking Schemes In The Presence of Physical Defaults the robust probing model. *IACR Trans. Cryptogr. Hardw. Embed. Syst. 2018*, 3 (2018), 89–120.

[46] Fritzmann, T., Beirendonck, M. V., Roy, D. B., Karl, P., Schamberger, T., Verbauwhede, I., and Sigl, G. Masked Accelerators and Instruction Set Extensions for Post-Quantum Cryptography. *IACR Trans. Cryptogr. Hardw. Embed. Syst. 2022*, 1 (2022), 414–460.

[47] Gaspoz, J., and Dhooghe, S. Threshold Implementations in Software: Micro-Architectural Leakages in Algorithms. *IACR Trans. Cryptogr. Hardw. Embed. Syst. 2023*, 2 (2023), 155–179.

[48] Giechaskiel, I., Rasmussen, K. B., and Szefer, J. Measuring Long Wire Leakage With Ring Oscillators In Cloud Fpgas. In *29th International Conference on Field Programmable Logic and Applications, FPL 2019, Barcelona, Spain, September 8-12, 2019* (2019), I. Sourdis, C. Bouganis, C. Álvarez, L. A. T. Díaz, P. Valero-Lara, and X. Martorell, Eds., IEEE, pp. 45–50.

[49] Gigerl, B., Pretterhofer, K., and Mangard, S. Security Aspects of Masking On Fpgas. In *IEEE International Symposium on Hardware Oriented Security and Trust, HOST 2024, Tysons Corner, VA, USA, May 6-9, 2024* (2024), IEEE, pp. 199–210.

[50] GitHub. GitHub - lowRISC/opentitan: OpenTitan: Open source silicon root of trust. `https://github.com/lowRISC/opentitan`. [Accessed 09-04-2025].

[51] Gnad, D. *Remote Attacks On FPGA Hardware*. PhD thesis, Karlsruhe University, Germany, 2020.

[52] Goodwill, G., Jun, B., Jaffe, J., and Rohatgi, P. A Testing Methodology For Side-channel Resistance Validation, September 2011.

[53] Gordon, D. M. Covering Designs.

[54] Gordon, D. M., Patashnik, O., and Kuperberg, G. New Constructions For Covering Designs. *Journal of Combinatorial Designs*, 4 (1995(3)).

[55] Gordon, D. M., Patashnik, O., Kuperberg, G., and Spencer, J. Asymptotically Optimal Covering Designs. *J. Comb. Theory(A)*, 2 (1996(75)).

[56] Gross, H., Iusupov, R., and Bloem, R. Generic Low-latency Masking In Hardware. *IACR Trans. Cryptogr. Hardw. Embed. Syst. 2018*, 2 (2018), 1–21.

[57] Gross, H., Mangard, S., and Korak, T. Domain-Oriented Masking: Compact Masked Hardware Implementations with Arbitrary Protection Order. In *Proceedings of The ACM Workshop on Theory of Implementation Security, TIS@CCS Vienna, Austria* (2016).

[58] Guilley, S., Hoogvorst, P., Mathieu, Y., and Pacalet, R. The "Backend Duplication" Method. In *Cryptographic Hardware And Embedded Systems - CHES 2005, 7th International Workshop, Edinburgh, UK, August 29 - September 1, 2005, Proceedings* (2005), J. R. Rao and B. Sunar, Eds., vol. 3659 of *Lecture Notes in Computer Science*, Springer, pp. 383–397.

[59] Hoey, J. The Two-way Likelihood Ratio (g) Test And Comparison To Two-way Chi Squared Test, 2012.

[60] Hwang, D., Tiri, K., Hodjat, A., Lai, B.-C., Yang, S., Schaumont, P., and Verbauwhede, I. AES-based Security Coprocessor Ic In $0.18 - \mu m$ CMOS With Resistance To Differential Power Analysis Side-Channel Attacks. *IEEE Journal of Solid-State Circuits 41*, 4 (2006), 781–792.

[61] Inc., A. Apple Platform Security. `https://support.apple.com/guide/security/welcome/web`. [Accessed 09-04-2025].

[62] Intel Corporation. Intel Trust Domain Extensions. `https://www.intel.com/content/www/us/en/developer/tools/trust-domain-extensions/overview.html`. [Accessed 24-01-2025].

[63] Ishai, Y., Sahai, A., and Wagner, D. A. Private Circuits: Securing Hardware Against Probing Attacks. In *Advances In Cryptology - CRYPTO 2003* (2003), D. Boneh, Ed., vol. 2729 of *Lecture Notes in Computer Science*, Springer, pp. 463–481.

[64] Karaklajic, D., Schmidt, J., and Verbauwhede, I. Hardware Designer's Guide To Fault Attacks. *IEEE Trans. Very Large Scale Integr. Syst. 21*, 12 (2013), 2295–2306.

[65] Knichel, D., and Moradi, A. Low-Latency Hardware Private Circuits. In *Proceedings of The 2022 ACM SIGSAC Conference on Computer and Communications Security, CCS 2022, Los Angeles, CA, USA, November 7-11, 2022* (2022), H. Yin, A. Stavrou, C. Cremers, and E. Shi, Eds., ACM, pp. 1799–1812.

[66] Knichel, D., Moradi, A., Müller, N., and Sasdrich, P. Automated Generation of Masked Hardware. *IACR Trans. Cryptogr. Hardw. Embed. Syst. 2022*, 1 (2022), 589–629.

[67] Knichel, D., Sasdrich, P., and Moradi, A. SILVER - Statistical Independence and Leakage Verification. In *Advances In Cryptology - ASIACRYPT 2020 - 26th International Conference on the Theory and Application of Cryptology and Information Security, Daejeon, South Korea, December 7-11, 2020, Proceedings, Part I* (2020), S. Moriai and H. Wang, Eds., vol. 12491 of *Lecture Notes in Computer Science*, Springer, pp. 787–816.

[68] Knichel, D., Sasdrich, P., and Moradi, A. Masked Accelerators and Instruction Set Extensions for Post-Quantum Cryptography. *IACR Trans. Cryptogr. Hardw. Embed. Syst. 2022*, 1 (2022), 323–344.

[69] Kocher, P. C. Timing Attacks on Implementations of Diffie-Hellman, RSA, DSS, and Other Systems. In *Advances In Cryptology - CRYPTO '96, 16th Annual International Cryptology Conference, Santa Barbara, California, USA, August 18-22, 1996, Proceedings* (1996), N. Koblitz, Ed., vol. 1109 of *Lecture Notes in Computer Science*, Springer, pp. 104–113.

[70] Kocher, P. C., Jaffe, J., and Jun, B. Differential Power Analysis. In *Advances In Cryptology - CRYPTO 1999* (1999), M. J. Wiener, Ed., vol. 1666 of *Lecture Notes in Computer Science*, Springer, pp. 388–397.

[71] Lee, K., Ashok, M., Maji, S., Agrawal, R., Joshi, A., Yan, M., Emer, J. S., and Chandrakasan, A. P. Secure Machine Learning Hardware: Challenges And Progress. *IEEE Circuits and Systems Magazine 25*, 1 (Firstquarter 2025), 8–34.

[72] Leiserson, A. J., Marson, M. E., and Wachs, M. A. Gate-level Masking Under A Path-Based Leakage Metric. In *Cryptographic Hardware And Embedded Systems - CHES 2014 - 16th International Workshop, Busan, South Korea, September 23-26, 2014. Proceedings* (2014), L. Batina and M. Robshaw, Eds., vol. 8731 of *Lecture Notes in Computer Science*, Springer, pp. 580–597.

[73] Maji, S., Banerjee, U., Fuller, S. H., and Chandrakasan, A. P. A Threshold Implementation-Based Neural Network Accelerator With Power And Electromagnetic Side-Channel Countermeasures. *IEEE Journal of Solid-State Circuits 58*, 1 (Jan 2023), 141–154.

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

[75] Mangard, S., Pramstaller, N., and Oswald, E. Successfully Attacking Masked AES Hardware Implementations. In *Cryptographic Hardware And Embedded Systems - CHES 2005* (2005), J. R. Rao and B. Sunar, Eds., vol. 3659 of *Lecture Notes in Computer Science*, Springer, pp. 157–171.

[76] Mell, P. M., and Grance, T. The NIST definition of cloud computing. Tech. Rep. NIST SP 800-145, National Institute of Standards and Technology, Gaithersburg, MD, 2011. Edition: 0.

[77] Monta, K., Nagata, M., Balasch, J., and Verbauwhede, I. On The Unpredictability of SPICE simulations for side-channel leakage verification of masked cryptographic circuits. In *60th ACM/IEEE Design Automation Conference, DAC 2023, San Francisco, CA, USA, July 9-13, 2023* (2023), IEEE, pp. 1–6.

[78] Monta, K., Oki, T., Hasegawa, R., Wadatsumi, T., Miki, T., Nagata, M., Lin, L., and Chang, N. A Hybrid Simulation Approach for Accurate and Fast System-Level Side-Channel Leakage Evaluation. In *Asian Hardware Oriented Security and Trust Symposium, AsianHOST 2024, Kobe, Japan, December 16-18, 2024* (2024), IEEE, pp. 1–6.

[79] Moradi, A., Kirschbaum, M., Eisenbarth, T., and Paar, C. Masked Dual-Rail Precharge Logic Encounters State-of-the-Art Power Analysis Methods. *IEEE Trans. Very Large Scale Integr. Syst. 20*, 9 (2012), 1578–1589.

[80] Moradi, A., and Schneider, T. Side-Channel Analysis Protection and Low-Latency in Action - Case Study of PRINCE and Midori. In *Advances In Cryptology - ASIACRYPT 2016* (2016), J. H. Cheon and T. Takagi, Eds., vol. 10031 of *Lecture Notes in Computer Science*, pp. 517–547.

[81] Müller, N., Knichel, D., Sasdrich, P., and Moradi, A. Transitional Leakage in Theory and Practice - Unveiling Security Flaws in Masked Circuits. *IACR Cryptol. ePrint Arch.* (2022), 23.

[82] Müller, N., Lammers, D., and Moradi, A. A Deep Analysis of Two Glitch-free Hardware Masking Schemes SESYM and LMDPL. *IACR Trans. Cryptogr. Hardw. Embed. Syst. 2024*, 3 (2024), 76–98.

[83] Müller, N., and Moradi, A. PROLEAD A Probing-Based Hardware Leakage Detection Tool. *IACR Trans. Cryptogr. Hardw. Embed. Syst. 2022*, 4 (2022), 311–348.

[84] Nagpal, R., Gigerl, B., Primas, R., and Mangard, S. Riding The Waves Towards Generic Single-cycle Masking In Hardware. *IACR Trans. Cryptogr. Hardw. Embed. Syst. 2022*, 4 (2022), 693–717.

[85] NANGATE. The NanGate 45nm Open Cell Library. Version: PDKv1.3_v2010_12.Apache.CCL. Available at `https://github.com/The-OpenROAD-Project/OpenROAD-flow-scripts/tree/master/flow/platforms/nangate45`.

[86] Nikova, S., Rechberger, C., and Rijmen, V. Threshold Implementations Against Side-channel Attacks And Glitches. In *Information and Communications Security, ICICS 2006 Proceedings* (2006).

[87] Petrides, G. On Non-Completeness In Threshold Implementations. In *Proceedings of ACM Workshop on Theory of Implementation Security Workshop, TIS@CCS London, UK* (2019).

[88] Piccione, E., Andreoli, S., Budaghyan, L., Carlet, C., Dhooghe, S., Nikova, S., Petrides, G., and Rijmen, V. An Optimal Universal Construction for the Threshold Implementation of Bijective S-boxes. *IEEE Trans. Inf. Theory 69*, 10 (2023), 6700–6710.

[89] Popp, T., and Mangard, S. Masked Dual-Rail Pre-charge Logic: DPA-Resistance Without Routing Constraints. In *Cryptographic Hardware And Embedded Systems - CHES 2005, 7th International Workshop, Edinburgh, UK, August 29 - September 1, 2005, Proceedings* (2005), J. R. Rao and B. Sunar, Eds., vol. 3659 of *Lecture Notes in Computer Science*, Springer, pp. 172–186.

[90] Quisquater, J., and Samyde, D. Electromagnetic Analysis (EMA): measures and counter-measures for smart cards. In *Smart Card Programming and Security, International Conference on Research in Smart Cards, E-smart 2001, Cannes, France, September 19-21, 2001, Proceedings* (2001), I. Attali and T. P. Jensen, Eds., vol. 2140 of *Lecture Notes in Computer Science*, Springer, pp. 200–210.

[91] Rabaey, J. M., Chandrakasan, A. P., and Nikolić, B. *Digital Integrated Circuits: A Design Perspective*, 2nd ed ed. Prentice Hall electronics and VLSI series. Pearson Education, Upper Saddle River, N.J, 2003.

[92] Rambus Inc. SCA Resistant Cores with DPA and FIA. `https://www.rambus.com/security/dpa-countermeasures/sca-resistant-cores/`. [Accessed 08-04-2025].

[93] Rees, R. S., Stinson, D. R., Wei, R., and van Rees, G. H. J. An Application of Covering Designs: Determining the Maximum Consistent Set of Shares in a Threshold Scheme. *Ars Comb. 53* (1999).

[94] Reparaz, O., Bilgin, B., Nikova, S., Gierlichs, B., and Verbauwhede, I. Consolidating Masking Schemes. In *Advances In Cryptology - CRYPTO 2015 Proceedings, Part I* (2015), pp. 764–783.

[95] Rivain, M., and Prouff, E. Provably Secure Higher-Order Masking of AES. In *Cryptographic Hardware And Embedded Systems, CHES 2010, 12th International Workshop, Santa Barbara, CA, USA, August 17-20, 2010. Proceedings* (2010), S. Mangard and F. Standaert, Eds., vol. 6225 of *Lecture Notes in Computer Science*, Springer, pp. 413–427.

[96] Roche, T., Lomné, V., Mutschler, C., and Imbert, L. A Side Journey To Titan. In *30th USENIX Security Symposium, USENIX Security 2021, August 11-13, 2021* (2021), M. D. Bailey and R. Greenstadt, Eds., USENIX Association, pp. 231–248.

[97] Sasdrich, P., Bilgin, B., Hutter, M., and Marson, M. E. Low-latency Hardware Masking With Application To AES. *IACR Trans. Cryptogr. Hardw. Embed. Syst. 2020*, 2 (2020), 300–326.

[98] Satoh, A., Morioka, S., Takano, K., and Munetoh, S. A Compact Rijndael Hardware Architecture With S-box Optimization. In *Advances In Cryptology - ASIACRYPT 2001, 7th International Conference on the Theory and Application of Cryptology and Information Security, Gold Coast, Australia, December 9-13, 2001, Proceedings* (2001), C. Boyd, Ed., vol. 2248 of *Lecture Notes in Computer Science*, Springer, pp. 239–254.

[99] Satoh Lab./UEC. The SAKURA-G FPGA board, 2014.

[100] Satoh Lab./UEC. The SAKURA-X (Sasebo-GIII) FPGA board, 2014.

[101] Schellenberg, F., Gnad, D. R. E., Moradi, A., and Tahoori, M. B. An Inside Job: Remote Power Analysis Attacks On Fpgas. *IACR Cryptol. ePrint Arch.* (2018), 12.

[102] Schneider, T., and Moradi, A. Leakage Assessment Methodology - A clear Roadmap for Side-Channel Evaluations. In *Cryptographic Hardware And Embedded Systems - CHES 2015 - 17th International Workshop, Saint-Malo, France, September 13-16, 2015, Proceedings* (2015), T. Güneysu and H. Handschuh, Eds., vol. 9293 of *Lecture Notes in Computer Science*, Springer, pp. 495–513.

[103] Sen, S., and Ghosh, A. Circuit-level Techniques For Side-channel Attack Resilience: A Tutorial. *IEEE Solid-State Circuits Magazine 16*, 4 (2024), 96–108.

[104] Shahmirzadi, A. R., Dhooghe, S., and Moradi, A. Low-latency And Low-randomness Second-order Masked Cubic Functions. *IACR Trans. Cryptogr. Hardw. Embed. Syst. 2023*, 1 (2023), 113–152.

[105] Shahmirzadi, A. R., and Moradi, A. Re-consolidating First-order Masking Schemes Nullifying Fresh Randomness. *IACR Trans. Cryptogr. Hardw. Embed. Syst. 2021*, 1 (2021), 305–342.

[106] Shahmirzadi, A. R., and Moradi, A. Second-Order SCA Security with Almost no Fresh Randomness. *IACR Trans. Cryptogr. Hardw. Embed. Syst. 2021*, 3 (2021), 708–755.

[107] Simoes, M., Bossuet, L., Bruneau, N., Grosso, V., and Haddad, P. Self-Timed Masking: Implementing First-Order Masked S-boxes Without Registers. *IACR Cryptol. ePrint Arch.* (2022), 641.

[108] Suzuki, D., and Saeki, M. Security Evaluation of DPA countermeasures Using Dual-Rail Pre-charge Logic Style. In *Cryptographic Hardware And Embedded Systems - CHES 2006, 8th International Workshop, Yokohama, Japan, October 10-13, 2006, Proceedings* (2006), L. Goubin and M. Matsui, Eds., vol. 4249 of *Lecture Notes in Computer Science*, Springer, pp. 255–269.

[109] Suzuki, D., Saeki, M., and Ichikawa, T. DPA Leakage Models for CMOS Logic Circuits. In *Cryptographic Hardware And Embedded Systems - CHES 2005, 7th International Workshop, Edinburgh, UK, August 29 - September 1, 2005, Proceedings* (2005), J. R. Rao and B. Sunar, Eds., vol. 3659 of *Lecture Notes in Computer Science*, Springer, pp. 366–382.

[110] Tiri, K., and Verbauwhede, I. A Logic Level Design Methodology For A Secure DPA resistant ASIC or FPGA implementation. In *Design, Automation and Test in Europe Conference and Exposition (DATE 2004* (2004), IEEE Computer Society, pp. 246–251.

[111] Tiri, K., and Verbauwhede, I. Place And Route For Secure Standard Cell Design. In *Smart Card Research And Advanced Applications VI, IFIP 18th World Computer Congress, TC8/WG8.8 & TC11/WG11.2 Sixth International Conference on Smart Card Research and Advanced Applications (CARDIS), 22-27 August 2004, Toulouse, France* (2004), J. Quisquater, P. Paradinas, Y. Deswarte, and A. A. E. Kalam, Eds., vol. 153 of *IFIP*, Kluwer/Springer, pp. 143–158.

[112] Trichina, E. Combinational Logic Design For AES SubByte Transformation on Masked Data. *IACR Cryptol. ePrint Arch.* (2003), 236.

[113] Turan, M. S., Barker, E., Kelsey, J., McKay, K. A., Baish, M. L., and Boyle, M. Recommendation For The Entropy Sources Used For Random Bit Generation. Tech. Rep. NIST SP 800-90b, National Institute of Standards and Technology, Gaithersburg, MD, Jan. 2018.

[114] Vaishnav, A., Pham, K. D., and Koch, D. A Survey On FPGA virtualization. In *28th International Conference on Field Programmable Logic and Applications, FPL 2018, Dublin, Ireland, August 27-31, 2018* (2018), IEEE Computer Society, pp. 131–138.

[115] Wegener, F., De Meyer, L., and Moradi, A. Spin Me Right Round Rotational Symmetry For FPGA-Specific AES: Extended Version. *J. Cryptology 33*, 3 (2020), 1114–1155.

[116] Wegener, F., and Moradi, A. Yet Another Size Record For AES: A first-order SCA secure AES s-box based on GF(2^8) multiplication. In *Smart Card Research and Advanced Applications, 17th International Conference, CARDIS 2018, Montpellier, France, November 12-14, 2018, Revised Selected Papers* (2018), B. Bilgin and J. Fischer, Eds., vol. 11389 of *Lecture Notes in Computer Science*, Springer, pp. 111–124.

[117] Wikipedia. Apple Silicon - Wikipedia. `https://en.wikipedia.org/wiki/Apple_silicon#Apple_T1`. [Accessed 09-12-2024].

[118] Wikipedia. Set Cover Problem - Wikipedia. `https://en.wikipedia.org/wiki/Set_cover_problem`. [Accessed 11-03-2025].

[119] Wikipedia. Flip-flop (electronics) - Wikipedia. `http://en.wikipedia.org/w/index.php?title=Flip-flop%20(electronics)&oldid=1193270436`, 2024. [Online; accessed 08-January-2024].

[120] Zhang, Z., Petkova-Nikova, S., and Nikov, V. Glitch-Stopping Circuits: Hardware Secure Masking Without Registers. In *Proceedings of The 2024 On ACM SIGSAC Conference on Computer and Communications Security, CCS 2024, Salt Lake City, UT, USA, October 14-18, 2024* (2024), B. Luo, X. Liao, J. Xu, E. Kirda, and D. Lie, Eds., ACM, pp. 3406–3420.

Part II

Part II. Publications

Chapter 6

LLTI: Low Latency Threshold Implementations

Publication Report

- **Title:** *LLTI: Low-Latency Threshold Implementations*
- **Authors:** Victor Arribas, Zhenda Zhang, and Svetla Nikova
- **Journal:** IEEE Transactions on Information Forensics and Security, Volume 16, 2021
- **DOI:** https://doi.org/10.1109/TIFS.2021.3123527
- **ePrint:** ePrint/2022/418

Contributions Summary: *Main author with Victor Arribas. I jointly conceptualized the idea, implemented the hardware, and conducted the experiments.*

LLTI: Low-Latency Threshold Implementations

Victor Arribas[1], Zhenda Zhang[2] and Svetla Nikova[2]

[1] Rambus Inc.,
varribas@rambus.com
[2] COSIC, KU Leuven, Belgium,
firstname.lastname@esat.kuleuven.be

Abstract. With the enormous increase in portable cryptographic devices, physical attacks are becoming similarly popular. One of the most common physical attacks is Side-Channel Analysis (SCA), extremely dangerous due to its non-invasive nature. Threshold Implementations (TI) was proposed as the first countermeasure to provide provable security in masked hardware implementations. While most works on hardware masking are focused on optimizing the area requirements, with the newer and smaller technologies area is taking a backseat, and low-latency is gaining importance. In this work, we revisit the scheme proposed by Arribas *et al.* in TCHES 2018 to secure unrolled implementations. We formalize and expand this methodology, to devise a masking scheme, derived from TI, designed to secure hardware implementations optimized for latency named Low-Latency Threshold Implementations (LLTI). By applying the distributive property and leveraging a divide-and-conquer strategy, we split a non-linear operation in layers which are masked separately. The result is a more efficient scheme than the former TI for any operation of algebraic degree greater than two, achieving great optimizations both in terms of speed and area. We compare the performance of first-order LLTI with first-order TI in securing a cubic gate and a degree-7 AND gate without using any registers in between. We achieve a 137% increase in maximum frequency and a 60% reduction in area for the cubic gate, and 3131 times reduction in area in the case of a degree-7 AND gate compared to TI. To further illustrate the power of our scheme we take a low-latency PRINCE implementation from the literature and, by simply changing the secure S-box with the LLTI version, we achieve a 46% max. frequency improvement and a 38% area reduction. Moreover, we apply LLTI to a secure a low-latency AES implementation and compare it with the TI version, achieving a 6.9 times max. freq. increase and a 47.2% area reduction.

Keywords: Hardware Security, Masking, Low-Latency, Non-Completeness

1 Introduction

With the exponential increase of IoT devices, Side-Channel Analysis (SCA) arises as one of the most threatening physical attacks. It exploits the side channels produced by the electronic device, such as the power consumption, the electromagnetic radiations, or the computation time. The most common attack of this kind is the Differential Power Analysis (DPA) [1]. Several strategies or countermeasures exist to help preventing side-channel attacks. One countermeasure that has caught great attention for the past years in the literature is masking [2, 3, 4, 5, 6, 7, 8, 9]. It is a technique derived from secret-sharing and multi-party computation (MPC), in which secret variables are split into multiple pieces or "shares", to make the information leaked independent of sensitive data.

Cryptographic algorithms combine linear and non-linear operations to achieve the desired security. The only non-linear part in symmetric cryptographic primitives is usually

the S-box. S-boxes have been designed with implementation costs in mind, but not taking into account the costs of side-channel resistance. A recent manuscript, from Bilgin *et al.* [10] stresses the importance of taking into account SCA resistance at design time, where the number of multiplications and the logical depth are the deciding factors to achieve efficient protected implementations. The logic depth has a direct impact on the latency of secure hardware implementations, which are prone to increase the unexpected dependencies caused by glitches [3]. Additionally, [10] analyzes larger S-boxes, with six and seven bits, limiting their investigations to quadratic operations due to the latency penalty when securing operations with a higher algebraic degree.

The perfect example of a construction with a big AND depth is the AES S-box, which has algebraic degree 7. There are multiple strategies to implement a masked AES S-box, the most common being the tower filed decomposition [11, 12, 13, 14, 8]. An also popular approach is the power maps decomposition [15, 4, 16, 17]. All these methods need several register layers to cope with the high non-linearity. To the best of our knowledge, there is no previous work securing the AES S-box by applying straightforward masking to the Algebraic Normal Form (ANF). Most of the literature regarding the masking of AES focuses on area efficient implementations. Only a few recent works proposed low-latency AES implementations [9, 18]. As chip manufacturing technology becomes smaller, the area constraints grow secondary, and lower latency implementations gain importance.

1.1 Related Work

The most popular type of masking used in hardware implementations is Boolean masking, where the pieces of the secret, or shares, are combined using an XOR operation to retrieve the unmasked data. Within this work, we focus on this type of masking. One of the first masked implementations is the one from Trichina [19], often referred to as the Trichina AND gate. A more formal study from Ishai, Shahai, and Wagner (ISW) [2], proposes the first security proofs and provable secure scheme for SCA protection at arbitrary order. They introduce the probing model, a widely used model in the design of masking primitives. Nevertheless, these techniques were not secure in hardware, since they did not capture in their models the hardware non-ideal behavior, *i.e.*, glitches.

The first masking scheme to provide first-order provable security in the presence of glitches was Threshold Implementations (TI) [3], later expanded to Higher-Order Threshold implementation (HOTI) [6], to cope with higher orders of security. These schemes are often referred as $td + 1$ masking. In a subsequent manuscript by Reparaz *et al.* [7], it was shown that the properties defined by HOTI were not enough to protect against multivariate attacks. Consolidating Masking Schemes (CMS) [7] proposes a reduction of the number of shares to achieve the same degree of security, in line with the proposal from [2]. This reduction additionally requires independence among the input variables to preserve the security warranties. Following CMS, Domain Oriented Masking (DOM) [8], improves on the randomness needed in CMS. These schemes are often referred as $d + 1$ masking. Finally, the paper from Moos *et al.* [20] shows that both CMS and DOM present flaws from third-order onwards.

On low-latency masking, the manuscript from Arribas *et al.* [21] presents the first low-latency TI applied to secure an unrolled implementation of Keccak. Following closely, Generic Low-Latency Masking (GLM) from Gross *et al.* [9] uses $d + 1$ masking as the main building block. The latest work by Sasdrich *et al.* [18] uses an alternative technique based on gate-level masking, known as LUT-based Masked Dual-Rail with Pre-charge Logic (LMDPL).

1.2 Our Contribution

In the cases where the AND depth equals to two, it is common practice to secure each AND gate separately, placing one register in between, as done in [22]. Hence, the greater the AND depth the more registers are needed. Current masking methodologies do not provide the means to directly secure high-algebraic-degree operations, such as the one found in the AES S-box, harming the latency of the design [10]. In this manuscript, we close this gap, proposing a methodology to mask any-logic-depth circuit without a single register, while still achieving competitive area results.

We present *Low-Latency Threshold Implementations* (LLTI), a masking methodology suitable for securing low-latency implementations, derived from the well-known masking scheme TI. The scheme we propose builds upon the concepts on unrolled implementations presented in Sect. 4 from previous work by Arribas *et al.* [21]. The scheme presented by Arribas *et al.* was utilized for securing unrolled implementations, proposing the first steps to devise the masking strategy needed within each unrolled round or layer. Within this manuscript, we formalize and extend the concepts from Arribas *et al.* to optimize the masking of higher-degree operations, and prove its applicability to any order of security and any algebraic degree.

We demonstrate how, by using our methodology, we can achieve the same minimal bounds for input and output shares as proposed in TI, with more efficient implementation results. Moreover, we prove that the optimization strategy used in [21] can be used at any security order and for any algebraic degree. Our scheme provides the means to mask any-degree operation without registers in between. It achieves realistic and competitive circuit complexity, compared to the direct sharing from [3, 6]. With our scheme we can not only implement efficiently high-algebraic-degree operations, but we can also use it to optimize the area requirements and the maximum frequency. LLTI works at the algorithmic level, so no synthesis or place-and-route constraints are needed for its implementation. It is important, however, to ensure that no optimizations happen during synthesis that could harm the security of the design.

Throughout this manuscript, we use the concept of latency to refer both to the number of clock cycles (or number of register layers), and the time from input to output. The reason behind this is that different communities use this term in distinct ways. On the one hand, the masking community uses it as cycle count or registers needed within a gadget ([9, 18, 10]), while, on the other hand, the works proposing PRINCE (or other full-cipher) implementations use latency as time from input to output ([23, 24, 25]). With this new scheme we aim to optimize both. The main feature of LLTI is to reduce the number of register layers. However, its application results in significant improvements in maximum frequency with respect to TI, leading to an enhanced input-output computation time.

Traditionally, in digital design, there is always a trade-off between area and latency: improving the design latency incurs in area penalty. In this work, we show how the application of LLTI minimizes this area increase in comparison with traditional strategies (like TI) when used for low-latency applications. Moreover, we show how it additionally increases the overall speed by further reducing the maximum frequency. We illustrate the advantages of LLTI by taking primitives implemented with low-latency in mind and comparing them to the corresponding TI secured version. We can see that while keeping the (low-) latency constant, LLTI highly reduces the area and frequency compared to the TI version.

Applying the concepts of the proposed scheme, we show a straightforward example of a first-order secure cubic AND gate (regularly used in symmetric primitives). Compared to using the direct sharing of traditional TI, with LLTI we achieve a maximum frequency improvement of 137% and a reduction in area of more than 60%. Additionally, we provide a sharing of a degree-7 operation, which is, to the best of our knowledge, the first sharing suitable to mask the AES S-box without adding any registers in between. This sharing,

applied on a single monomial, is more than 3 100 times smaller compared to the direct sharing if we were to use TI. To show the performance of LLTI when applied to realistic implementations, we experiment with PRINCE and AES designs. By simply swapping the secure S-box of a state-of-the-art PRINCE implementation targeting low-latency with an LLTI version, we achieve a 46% max. frequency improvement and a 38% area reduction. Moreover, we propose an LLTI AES with a single register layer and zero online randomness and compare it with the corresponding TI version, resulting in almost 7 times higher max. freq. and a reduction of 47% in area when using LLTI. We practically evaluate the security of both versions for up to 100 million traces.

The next sections of the paper are organized as follows. In Sect. 2, we present more in detail related preliminary works. Then, in Sect. 3, we introduce all the core concepts and proofs of our scheme. In the following Sect. 4, we illustrate different applications of our scheme. Finally, in Sect. 5, we present several implementations with LLTI.

2 Preliminaries

2.1 Notation

In this paper, we work in $GF(2)$, meaning that sums and multiplications correspond to XOR and AND gates in hardware. We describe any-algebraic-degree non-linear operations as AND gates of different number of inputs. We use t to denote the algebraic degree, and d for the order of security.

We focus entirely on Boolean masking, and thus, we represent a shared variable as $\boldsymbol{x} = (x_1, \ldots, x_s)$ such that $x = x_1 \oplus \ldots \oplus x_s$, where s refers to the number of shares and the number of input shares of a function. When referring to the number of output shares of a function we use s_o. Thus, the sharing of any function is denoted with the tuple (s, t, d, s_o). High algebraic degrees are designated with T, and F_{T} represents a function of such algebraic degree. Then, s_{T} refers to the input shares to this function, and d_{T} to the desired overall security order of F_{T}.

In addition to the masking notation, an equivalent set covering notation is used in the proofs presented in the following sections. We represent the corresponding set covering for the sharing tuple (s, t, d, s_o) as $\langle \mathcal{C}(s, t, d), s_o \rangle$, such that $\left|\mathcal{C}(s, t, d)\right| = s_o$. We also provide the sharing of a variable or a function with a set notation, *e.g.*, the sharing of the variable $\boldsymbol{x} = (x_1, \ldots, x_s)$ would be represented as $\{1, \ldots, s\}$. This notation is typically used in subsets of a covering set, where each subset represents the shares used in a shared function. For instance, a shared function with two output shares and three input shares could be represented as $\{\{1, 2\}, \{2, 3\}\}$, where the first shared function depends on shares 1 and 2, and the second one depends on shares 2 and 3. For clarity, we simplify this representation with the following $\{12, 23\}$.

2.2 Adversarial Model

The adversarial model used throughput this manuscript is the glitch extended d-probing model, first introduced in [7], and refined in [26]. It is also known as $(1, 0, 0)-$robust d-probing model [27]. Additionally, we assume independent leakage among shares. It was shown in [28], that a circuit which is secure in the d-probing model where each calculation is treated separately, is also secure against d^{th}-order SCA.

2.3 Threshold Implementations

Threshold Implementations (TI) [3] require an implementation to fulfill the following three properties in order to be first-order SCA secure:

- **Correctness**: a shared function $\mathbf{f}$ such that $f_i(\boldsymbol{x}) = y_i$, where $i = 1, \ldots, s$, is correct if $\sum y_i = y = f(x)$.
- **Non-completeness**: a shared function $\mathbf{f}$ is non-complete if every component function f_i is independent of at least one input share.
- **Uniformity**: Given a uniform sharing at the input of $\mathbf{f}$, the resulting output sharing must conform a uniform distribution as well.

The paper by Bilgin *et al.* [6] extended the original version of TI to Higher-Order Threshold Implementations (HOTI), to provide resistance against higher-order attacks. The former definition of non-completeness is expanded to:

Definition 1 (d^{th}**-Non-completeness [6]).** A shared function $\mathbf{f}$ is d^{th}-order non-complete if any combination of up to d component functions f_i is indep. of at least one input share.

This work proposes minimum bounds for the number of input and output shares to comply with the definition above as:

$$s_i \geq td + 1, \quad s_o \geq \tbinom{td+1}{t}. \tag{1}$$

Example: We provide an example of a first-order TI of the AND/XOR gate $z = a \oplus bc$, with $(s, t, d, s_o) = (3, 2, 1, 3)$:

$$\begin{aligned} z_1 &= a_1 \oplus b_1c_1 \oplus b_1c_2 \oplus b_2c_1, \\ z_2 &= a_2 \oplus b_2c_2 \oplus b_2c_3 \oplus b_3c_2, \\ z_3 &= a_3 \oplus b_3c_3 \oplus b_1c_3 \oplus b_3c_1. \end{aligned} \tag{2}$$

2.4 Set Coverings and Non-completeness

The non-completeness property of TI is reformulated in terms of set coverings by Petrides [29]. He leverages this theory to prove new concepts, which we use in the subsequent sections to aid our proofs. Below, we summarize his contributions.

Definition 2 (non-complete set coverings [29])**.** A dth-order non-complete set covering $\mathcal{C}^{\text{nc}}(s, t, d)$ is a set of subsets from the universe of the inputs, $\mathcal{U}_s = \{1, \ldots, s\}$, such that:

1. each t-subset of $\mathcal{U}_s$ is a subset of at least one element of $\mathcal{C}^{\text{nc}}(s, t, d)$,
2. each element of $\mathcal{C}^{\text{nc}}(s, t, d)$ has size at least t, and
3. a minimum of $d + 1$ elements of $\mathcal{C}^{\text{nc}}(s, t, d)$ are needed to cover $\mathcal{U}_s$.

Item 1 ensures correctness, item 2 prevents redundant elements not contributing to correctness from appearing, and item 3 ensures non-completeness. Note that the set covering theory does not capture the uniformity property of TI, nor necessary remasking. There can be several such non-complete set coverings, which can be used as a guideline for obtaining a non-complete TI for any algebraic degree t. Each subset represents a shared function, and each element of the subset is a share on which the sub-function is allowed to depend.
Example: We define $\mathcal{C}^{\text{nc}}(3, 2, 1) = \{12, 23, 13\}$, which is the set covering corresponding to the sharing from Eq. (2). It is trivial to see that it satisfies all the conditions from Def. 2, and that can be used to generate a non-complete sharing of $z = a + bc$.

$\mathcal{C}^{\text{nc}}(s, t, d)$ is a particular non-complete covering. We denote the set of all such coverings as $\mathcal{NC}(s, t, d)$, its subset of coverings with the smallest cardinality possible as $\widehat{\mathcal{NC}}(s, t, d)$, and a non-complete set covering with minimal cardinality by $\widehat{\mathcal{C}}^{\text{nc}}(s, t, d)$. Finally, $\mathcal{U}_{s,t}$ is the set of all t-subsets of $\mathcal{U}_s$.

2.4.1 Basic Results [29]

The first proposition presents the minimum number of input shares to achieve non-completeness:

Proposition 2.1. *If $s \geq td+1$ then $\mathcal{U}_{s,t} \subseteq \mathcal{NC}(s,t,d)$.*

Subsequently, conclusions regarding the trivial case, where $s = td+1$, are presented. By using the trivial value of s, we obtain non-complete set coverings with minimal cardinality:

Lemma 2.1.1. $\widehat{\mathcal{NC}}(s,t,d) = \mathcal{U}_{td+1,t}$.

Thus, a single non-complete set covering of minimal cardinality is represented as follows:

Corollary 2.1.1. $\langle \widehat{\mathcal{C}}^{nc}(td+1,t,d), \binom{td+1}{t} \rangle$.

Finally, the subsequent proposition establishes an upper bound for the size of subsets within a set covering, which represents the maximum number of shares a sub-function of a shared function can depend on.

Proposition 2.2. *For every $\mathcal{S} \in \mathcal{C}^{nc}(s,t,d)$ it holds that*

$$\left|\mathcal{S}\right| \leq s - (d-1)t - 1. \tag{3}$$

2.4.2 Extensions and Expansions [29]

In here, Petrides looks at concepts corresponding with bounds beyond the trivial ones defined in Eq. (1). The following lemma shows how by increasing the number of input shares, the number of output shares remains constant.

Lemma 2.2.1. *If $s \geq td+1$ then for every $\mathcal{C}^{nc}(s,t,d)$ there exists $\mathcal{C}^{nc}(s+1,t,d)$ of equal cardinality.*

Example: The set covering from the example above $\mathcal{C}^{\mathrm{nc}}(3,2,1)$ can be extended to $\mathcal{C}^{\mathrm{nc}}(4,2,1) = \{124, 134, 23\}$. From Prop. 2.2 we see that in this new case $\left|\mathcal{S}\right| \leq 4 - (1-1)2 - 1 = 3$.

Furthermore, by increasing the number of input shares, we can also reduce the number of output shares or the cardinality of the resulting set covering:

Corollary 2.2.1. *For $s \geq td+1$ we have $\left|\widehat{\mathcal{C}}^{nc}(s,t,d)\right| \geq \left|\widehat{\mathcal{C}}^{nc}(s+1,t,d)\right|$.*

The previous example has already minimum number of outputs, so no further increase in s will reduce the size of the set covering beyond 3. Nevertheless, this is extremely useful for higher orders, where $\left|\mathcal{C}^{\mathrm{nc}}(s,t,d)\right| \geq s$.

In summary, Petrides defines the non-complete set covering, reaching the same basic results for the trivial case of TI as [6] in Eq. (1). This is then extended to the non-trivial case when the number of input shares is increased beyond the minimum of the trivial case, with the respective change in the number of output shares to accommodate for non-completeness and uniformity. Up to now, no systematic way of finding TI non-trivial sharings has been found [29, 30].

2.5 Unrolled Implementations

The method proposed by Arribas *et al.* [21] is used to secure an unrolled implementation of Keccak, where two rounds are computed within one clock cycle. To achieve this, they proposed to split the non-linear operations of the unrolled rounds into different layers. Those layers are secured separately, with some conditions precomputed to ensure that the subsequent union preserves the security requirements. This allows the designer to

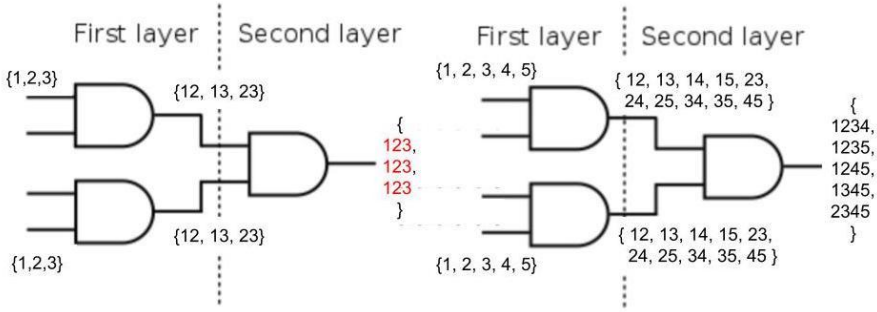


Figure 1: Layer-wise first-order sharing of a combinational quartic operation with an insecure sharing (left), and the secure counterpart (right) from applying the guidelines of Eq. (4) from [21], equivalent to a trivial LLTI with T = 4. The dashed lines represent the layers separation.

follow a divide-and-conquer procedure instead of solving the problem of securing a complex operation as a whole, which would be highly challenging or even impossible. Traditionally, these functions are secured placing a register in between and separately masking the smaller non-linear operations. In [21], the authors propose a methodology to secure such functions without placing registers in between. The designer is free to choose how to split the more complex non-linear operations into a total of N layers, each layer containing simpler non-linear operations. The n-th layer is denoted as R^n, its input shares s_{R^n} are expressed as s_n for better readability, the degree of security of this layer is referred as d_{R^n}, and its algebraic degree as t_{R^n}.

The idea is that every layer is secured with a different sharing scheme of a certain security order provided by the following recipe:

$$d_{R^n} = \begin{cases} d_F & \text{if } n = N, \\ t_{R^{n+1}} * d_{R^{n+1}} & \text{if } n < N \end{cases} \tag{4}$$

By defining the last layer's security requirements, the designer can recursively compute backwards the non-completeness degree needed in previous layers to achieve the desired security order at the output. The shares of every layer are constructed following the constraints from Eq. (4). Finally, the last layer conforms a *compression sharing*. This kind of sharing can be found for example in [31] and [13], and is defined in [21]:

Definition 3 (**Compression sharing [21]**)**.** A compression sharing is characterized by having $s > s_o$.

The authors of [21] use a quartic, or degree-four gate, as an example to illustrate the application of Eq. (4). This represents the algebraic degree resulting from removing the register layer usually employed to split two quadratic rounds. This is the case of Keccak, which is the algorithm analyzed in [21]. Fig. 1 presents this example, used to show how to leverage Eq. (4) to devise the masking strategy to secure such function. On the one hand, it shows an example in which the two layers are secured independently with three input shares (left), which makes it impossible to maintain the non-completeness property given the combinations from both layers. On the other hand, the secure version is presented using five input shares, derived from applying the concepts from Eq. (4). As we can see, in the end every output share depends on four input shares out of five, ensuring non-completeness.

Finally, the unrolled implementation can also be formulated as non-complete set coverings. The covering set for layer R^n is denoted as $\langle \mathcal{C}_n(s_n, t_n, d_n), s_{o_n} \rangle$. Sometimes, when the parameters of such covering are clear from the context, or not relevant at that moment, we use $\mathcal{C}_n$ for the sake of simplicity.

2.6 Unbalanced sharings

Unbalanced sharings have already been used in [26] App. A, but they were not properly introduced until the work of Wei *et al.* [32], used to devise a TI implementation of SM4.

Definition 4 (Unbalanced sharings [32]). In existing sharing schemes, every input variable has the same number of shares. However, it is perfectly possible that a sharing scheme in which different input variables are split into different numbers of shares still maintains the three essential properties for TIs. Such a scheme is named an unbalanced sharing scheme.

3 Low-Latency Threshold Implementations

In this section, we discuss our methodology. We present first an example where the methodology of Arribas *et al.* [21] does not achieve optimal results, and how to improve it. As a result, we propose a new sharing for a cubic operation with no registers in between, which at the same time represents the most basic element of our scheme. Then, we introduce *Low-Latency Threshold Implementations* (LLTI). We prove that our methodology achieves minimal sharings for any security order d, and any algebraic degree t, achieving the same bounds as with TI. Additionally, we demonstrate how by increasing the number of input shares we can achieve more optimal sharings. Finally, we address the issue of uniformity and randomness.

3.1 Sub-optimal cases of [21]

The scheme proposed by Arribas *et al.* can be applied to any non-linear operation, but it is not directly applicable to odd prime algebraic degrees to achieve the minimal sharing. We fix this and extend our methodology to obtain minimal sharings for any degree.

To illustrate the cases where a sub-optimal sharing is achieved when following the guidelines from Eq. (4), we present an example of a decomposed simple cubic gate for first-order security. With TI we can trivially share a cubic operation with $s = 3 \cdot 1 + 1 = 4$ shares for $d = 1$ (from Eq. (1)). However, when decomposing the cubic gate, the resulting two layers are composed by quadratic operations, so the result would be $s = 5$ shares for first-order security by applying Eq. (4), compared to the four input and output shares from the direct case.

In Fig. 2, we show that it is possible to achieve the minimum number of shares also from the layered decomposition, including the non-complete sharings for both layers. The two inputs to the second AND gate are split with a different number of shares: 6 and 4, respectively. The sharing in this layer is a compression sharing, and, additionally, an *unbalanced sharing*.

Unbalanced sharings are operated in the same way as regular sharings. However, note that the order of the elements within the subsets of the final covering matters, as opposed to the case when the sharings are symmetric. From the example in Fig. 2, since the bottom input is not driven by any other operation before, there are no further dependencies introduced in this line. Hence, it is enough to have six and four shares in respective inputs, instead of the 10 input shares as in the example from [21] shown in Fig. 1. As we see from the picture, the order in the indexes at the output of R^N is important, since one comes from a 6-shares sharing which at the same time has further dependencies, while the other

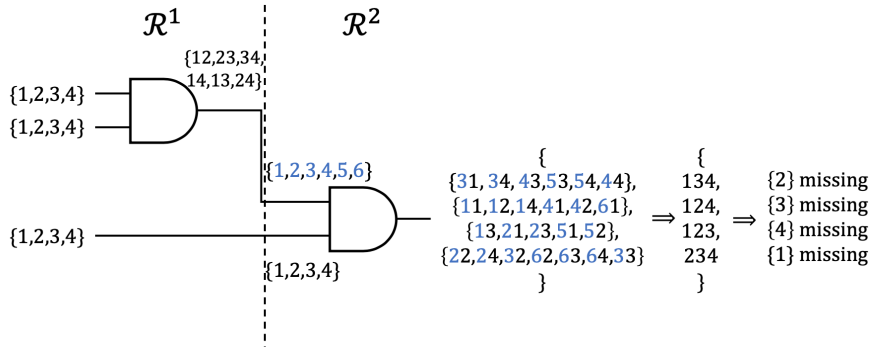


Figure 2: LLTI sharings for the two layers decomposition of a combinational cubic operation, for first-order security using unbalanced sharings with four and six shares.

comes from a 4-shares sharing equivalent to $\mathcal{U}_s$. Thus, the covering set for an unbalanced sharing must be expressed using all the cross-products from the two input sharings. With this example we show that the strategy proposed in [21] is not always optimal.

3.2 Low-Latency Threshold Implementations

In their work, De Meyer *et al.* [26] proved that non-completeness is a necessary condition for a masking scheme to provide provable security. Thus, we focus on finding efficient strategies to achieve the optimal non-complete sharing for a higher algebraic degree operation. We benefit from the set coverings theory from Sect. 2.4 to prove that we can achieve minimal non-complete coverings for any degree t function.

Our scheme builds upon the one in [21], providing a structured way of splitting the high-algebraic-degree non-linear operations into layers of lower ones to achieve more efficient sharings. As we have seen from the counterexample in Sect 3.1, we need to take into account cases when the tree decomposition is not symmetric. Fig. 2 corresponds to such a case, namely the decomposition of a cubic gate, of prime algebraic degree. On the contrary, Fig. 1 corresponds to the decomposition of a quartic operation with a composite degree, which results in a symmetric tree of smaller operations. Thus, we differentiate between two cases to cover all possible operations: first, we define the scheme for t composite, and subsequently, we extend the method for the cases where t is prime.

3.2.1 Composite algebraic degree

Given a non-linear function with a high algebraic degree T composite, denoted as F_{T}, we can split this operation into simpler non-linear operations following its prime factors decomposition (PFD), such that

$$\mathrm{T} = \prod_{i}^{N} t_i,$$

where every t_i prime factor designates a layer R^i, for a total of N layers. Given the degree of security d_{T} of $F_{\mathrm{T}} = R^N \circ \cdots \circ R^1$, the resulting sharing for each layer has a similar structure as Eq. (4). The decomposition and the corresponding sharing for each layer are

defined as follows:

$$\forall t_i \in \mathrm{PFD(T)} \text{ with } i \in [1, N],\ \exists R^i : \quad (5)$$
$$d_{R^i} = \begin{cases} d_{\mathrm{T}} & \text{if } i = N, \\ t_{R^{i+1}} * d_{R^{i+1}} & \text{if } i < N \end{cases}$$

Lemma 1. *Provided that the decomposition of F_{T} is a symmetric tree of operations, the number of input shares of the first layer R^1, obtained from applying Eq. (5), is equivalent to the minimal input shares resulting from applying Eq. (1) to the function F_{T}, i.e.,*

$$s_1 \geq t_1 \cdot d_1 + 1 = \mathrm{T} \cdot d_{\mathrm{T}} + 1.$$

In such a case, there exists always a compression sharing for the layer R^N, such that it is a minimal d_{T}^{th}-order non-complete covering $\widehat{\mathcal{C}}_N^{nc}(s_N, t_N, d_N)$ with:

$$\begin{aligned} \widehat{\mathcal{C}}_N^{nc}(s_N, t_N, d_N) \quad &= \quad \widehat{\mathcal{C}}_{F_{\mathrm{T}}}^{nc}(\mathrm{T}d_{\mathrm{T}} + 1, \mathrm{T}, d_{\mathrm{T}}) \Rightarrow \\ \langle \widehat{\mathcal{C}}_N^{nc}(s_N, t_N, d_N) \quad &, \quad \binom{\mathrm{T}d_{\mathrm{T}} + 1}{\mathrm{T}} \rangle. \end{aligned}$$

Proof. In the first layer R^1, the sharing scheme is a set covering in the form $\mathcal{C}_1(s_1, t_1, d_1)$, such that $s_1 \geq t_1 \cdot d_1 + 1$. From the definition in Eq. (5), we have that $d_1 = t_2 \cdot d_2$, resulting in $s_1 \geq t_1 \cdot (t_2 \cdot d_2) + 1$. If we keep going until recursively reaching the last layer, and from Eq. (1), we have that

$$s_1 \geq t_1 \cdot (t_2 \cdot \ldots (t_{N-1} \cdot (t_N \cdot d_N))) + 1 = (\prod_i^N t_i) * d_N + 1 = \mathrm{T}d_{\mathrm{T}} + 1$$

From this result, and together with Lemma 2.1.1, we conclude that the sharing of R^1 uses the minimal input shares corresponding to F_{T}.

Now we have to prove that the sharing for the last layer can be a non-complete compression sharing achieving the corresponding minimal number of output shares of F_{T}. From Def. 2 (3), we know that, for the last covering set to satisfy $d_{\mathrm{T}}^{\mathrm{th}}$-order non-completeness, we need $\left|\mathcal{C}_N\right| \geq d_{\mathrm{T}} + 1$. From Prop. 2.1, and taking the minimum number of elements at the input and output of R^N, it would result in $s_N = t_n \cdot (\left|\mathcal{C}_N\right| - 1) + 1$. Since $s_N = \left|\mathcal{C}_{N-1}\right|$, and $s_{N-1} = t_{N-1} \cdot (\left|\mathcal{C}_{N-1}\right| - 1) + 1$, we can write that $s_N = t_N \cdot t_{N-1} \cdot (\left|\mathcal{C}_N\right| - 1) + 1$. By recursively reaching the first layer, we will have that

$$s_1 = (\prod_1^N t_i) \cdot \underbrace{(\left|\mathcal{C}_N\right| - 1)}_{\left|\mathcal{C}_N\right| = d_{\mathrm{T}} + 1} + 1 \Rightarrow s_1 = (\prod_1^N t_i) * d_{\mathrm{T}} + 1 = \mathrm{T}d_{\mathrm{T}} + 1. \quad (6)$$

Thus, together with the first part of the proof, we can affirm that with the resulting number of input shares in R^1, $\mathcal{C}_N(s_N, t_N, d_N) \subseteq \mathcal{NC}(s_N, t_N, d_N = d_{\mathrm{T}})$, *i.e.*, is a $(d_N = d_{\mathrm{T}})^{th}$-order non-complete covering. Finally, as the result from Eq. (6) corresponds with the minimum number of input shares, from Cor. 2.1.1 we conclude that

$$\begin{aligned} &\langle \widehat{\mathcal{C}}_N^{\mathrm{nc}}(s_N, t_N, d_N), \begin{pmatrix} \prod_i t_i * d_{\mathrm{T}} + 1 \\ \prod_i t_i \end{pmatrix} \rangle \Leftrightarrow \\ &\Leftrightarrow \langle \widehat{\mathcal{C}}_{F_{\mathrm{T}}}^{\mathrm{nc}}(\mathrm{T}d_{\mathrm{T}} + 1, \mathrm{T}, d_{\mathrm{T}}), \binom{\mathrm{T}d_{\mathrm{T}} + 1}{\mathrm{T}} \rangle. \end{aligned}$$

□

3.2.2 Prime algebraic degree

When T is prime, there is no further decomposition possible. In this case, to split F_{T} into layers of simpler operations, we follow the above procedure to decompose $F_{\mathrm{T}-1}$ instead, where $\mathrm{T}-1$ will always be composite. To the output of the symmetric tree derived from $F_{\mathrm{T}-1}$, we just add an extra 2-input AND gate with the last input to realize a decomposition tree for F_{T}. Fig. 3 illustrates this concept with an example for $\mathrm{T}=7$. Hence, we have that R^{N-1} is the last layer of $F_{\mathrm{T}-1}$'s decomposition, and R^N includes the last AND gate to complete F_{T}. It is important to note that the sharing in the last layer of $F_{\mathrm{T}-1}$ is not a compression sharing anymore. In this case, the sharing in R^N is both an unbalanced sharing and a compression sharing. The sharing of the first input to the last AND gate will have $\left|\widehat{\mathcal{C}}^{\mathrm{nc}}_{F_{\mathrm{T}-1}}(s_{\mathrm{T}-1},\mathrm{T}-1,d_{\mathrm{T}})\right|$ number of input shares, while the corresponding sharing for the second input will have s_1 input shares, conforming the unbalanced sharing. Additionally, the number of output shares will be $\binom{\mathrm{T}d_{\mathrm{T}}+1}{\mathrm{T}}$, resulting in a compression sharing.

We can apply Eq. (5) to $F_{\mathrm{T}-1}$'s decomposition to get the different sharings for the first R^{N-1} layers, with $d_{N-1}=d_N=d_{\mathrm{T}}$ to achieve a full non-complete sharing. By definition, with this split, we always have that $t_N=2$, and $d_N=d_{\mathrm{T}}$. To ensure non-completeness is preserved when adding the last AND gate to form F_{T}, it suffices to add d_{T} additional elements to $\mathcal{U}_s^{\mathrm{T}-1}$, which refers to the number of elements in the universe of $F_{\mathrm{T}-1}$. This way it is also possible to achieve the minimum number of output shares, as shown in Fig. 2. These concepts are formalized in the following lemma:

Lemma 2. *Given F_{T}, with* T *prime, the decomposition of layers R^1 to R^{N-1} corresponds to the decomposition of $T-1$, where the conditions from Eq. (5) for $F_{\mathrm{T}-1}$ apply, with $d_{N-1}=d_N=d_{\mathrm{T}}$. Finally, an extra layer R^N is appended to realize F_{T}, with $t_N=2$ and $d_N=d_{\mathrm{T}}$. The sharing of R^N is both a compression and an unbalanced sharing, and it suffices to have*

$$\left|\mathcal{U}_s^{\mathrm{T}}\right| = \left|\mathcal{U}_s^{\mathrm{T}-1}\right| + d_{\mathrm{T}}$$

to achieve $\widehat{\mathcal{C}}^{nc}_{F_{\mathrm{T}}}(s,\mathrm{T},d_{\mathrm{T}})$.

Proof. From Lemma 1 we know that the decomposition of $F_{\mathrm{T}-1}$ results in a minimal non-complete covering $\widehat{\mathcal{C}}^{\mathrm{nc}}_{F_{\mathrm{T}-1}}(s_{\mathrm{T}-1},\mathrm{T}-1,d_{\mathrm{T}})$. From Eq. (1), in the minimal case we have that

$$s_{\mathrm{T}-1} = (\mathrm{T}-1)\cdot d_{\mathrm{T}} + 1 = \underbrace{\mathrm{T}\cdot d_{\mathrm{T}} + 1}_{s_{\mathrm{T}}} - d_{\mathrm{T}} \Rightarrow s_{\mathrm{T}} = s_{\mathrm{T}-1} + d_{\mathrm{T}},$$

from which we confirm that $\left|\mathcal{U}_s^{\mathrm{T}}\right| = \left|\mathcal{U}_s^{\mathrm{T}-1}\right| + d_{\mathrm{T}}$.

Then, we have to prove that with this number of input shares and following the guidelines for $F_{\mathrm{T}-1}$ we can achieve a non-complete minimal covering. From Def. 2 (2) and Prop. 2.2, for the trivial case

$$\forall \mathcal{S} \in \widehat{\mathcal{C}}^{\mathrm{nc}}_{F_{\mathrm{T}-1}}(s_{\mathrm{T}-1},\mathrm{T}-1,d_{\mathrm{T}}): \tag{7}$$
$$\left|\mathcal{S}\right| = t_1\cdot t_2\cdot \ldots \cdot t_{R^{N-1}} = \prod_i^{R^{N-1}} t_i.$$

Every $\mathcal{S}$ contains at most $\mathrm{T}-1$ elements of $\mathcal{U}_s^{\mathrm{T}}$, and since they conform a non-complete covering, any d_{T} combinations of sets contain at most $d_{\mathrm{T}}\cdot\prod_i^{N-1} t_i = d_{\mathrm{T}}\cdot(\mathrm{T}-1)$ elements.

Subsequently, an extra layer R^N is added to complete F_{T}. Since the non-linear operation in R^N is a simple 2-input AND gate, and the second input set covering corresponds to $\mathcal{U}_s^{\mathrm{T}}$:

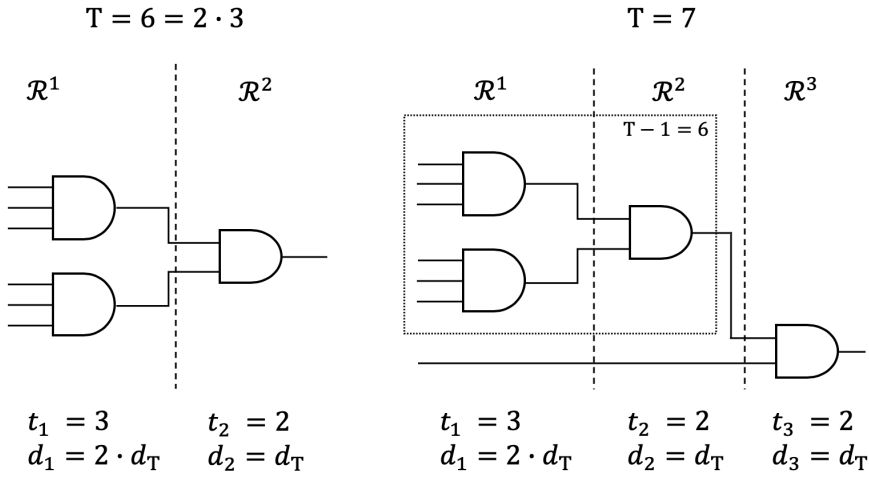
$$\forall \mathcal{S} \in \mathcal{C}_{F_{\mathrm{T}}}(s_{\mathrm{T}},\mathrm{T},d_{\mathrm{T}}): \left|\mathcal{S}\right| \leq t_1\cdot t_2\cdot \ldots \cdot t_{R^{N-1}} + 1 = \prod_i^{N-1} t_i + 1.$$

Every $\mathcal{S}$ contains at most $\mathrm{T}-1+1=\mathrm{T}$ elements. Similarly as before, any d_T combinations of sets will contain at most $d_\mathrm{T}\cdot(\prod_i^{N-1} t_i+1)=d_\mathrm{T}\cdot\mathrm{T}$ components. As we have that

$$s_\mathrm{T}=s_{\mathrm{T}-1}+d_\mathrm{T}=(\mathrm{T}-1)\cdot d_\mathrm{T}+1+d_\mathrm{T}= \\ \mathrm{T}\cdot d_\mathrm{T}-d_\mathrm{T}+1+d_\mathrm{T}=d_\mathrm{T}\cdot\mathrm{T}+1, \tag{8}$$

Def. 2 (3) is satisfied, and we conclude that $\mathcal{C}_{F_\mathrm{T}}(s_\mathrm{T},\mathrm{T},d_\mathrm{T})\subseteq\mathcal{NC}(s_\mathrm{T},\mathrm{T},d_\mathrm{T})$. Finally, from this reasoning and together with Cor. 2.1.1, we conclude that indeed $\widehat{\mathcal{C}}^{\mathrm{nc}}_{F_\mathrm{T}}(s_\mathrm{T},\mathrm{T},d_\mathrm{T})$ is a minimal non-complete set covering. □

Fig. 3 illustrates two examples, for $\mathrm{T}=6$ and $\mathrm{T}=7$, respectively. It includes the decomposition in several layers, together with the conditions that each layer must satisfy to achieve an overall minimal non-complete covering.



(a) $\mathrm{T}=6$ (b) $\mathrm{T}=7$

Figure 3: Trivial LLTI security requirements for combinational composite and prime Ts examples, following the application of Eq. (5) and Lemma 2.

In the case of $\mathrm{T}=6$ in Fig. 3a, the trivial TI sharing for first-order security given by Eq. (1) we have $s=7$ and $s_o=7$, or $\langle\widehat{\mathcal{C}}^{\mathrm{nc}}_{F_6}(7,6,1),7\rangle$. Then, if we build the set covering following the layered decomposition, the first thing to notice is that $s_1=7=s$, as stated in Lemma 1. The resulting set coverings for each layer are $\langle\widehat{\mathcal{C}}^{\mathrm{nc}}_{1}(7,3,2),35\rangle$ and $\langle\widehat{\mathcal{C}}^{\mathrm{nc}}_{2}(35,2,1),7\rangle$. For overall second-order security, with $d_\mathrm{T}=2$, if we calculate the trivial TI sharing for F_6, we have that:

$$s=6\cdot 2+1=13 \tag{9}$$
$$s_o=\binom{6\cdot 2+1}{6}=1716\Rightarrow\langle\widehat{\mathcal{C}}^{\mathrm{nc}}_{F_6}(13,6,2),1716\rangle$$

Similarly, if we calculate the coverings with LLTI we have that $\langle\widehat{\mathcal{C}}^{\mathrm{nc}}_{1}(13,3,4),286\rangle$ and $\langle\widehat{\mathcal{C}}^{\mathrm{nc}}_{2}(286,2,2),1716\rangle$.

Now, we do the same exercise for $\mathrm{T}=7$ from Fig. 3b. The trivial TI sharing for first-order would have $s=8$ and $s_o=8$, or $\langle\widehat{\mathcal{C}}^{\mathrm{nc}}_{F_7}(8,7,1),8\rangle$. Then, the set coverings for the

layered decomposition would be: for R^1, following Lemma 2,

$$\langle \widehat{\mathcal{C}}_1^{\text{nc}}(s_{\text{T}-1} + d_{\text{T}}, t_1, d_1) = \widehat{\mathcal{C}}_1^{\text{nc}}(7+1, 3, 2), \binom{8}{3} = 56 \rangle;$$

for R^2,

$$\langle \widehat{\mathcal{C}}_2^{\text{nc}}(56, 2, 1), \binom{56}{2} = 1540 \rangle;$$

and finally, for R^3, the unbalanced and compression sharing,

$$\langle \widehat{\mathcal{C}}_3^{\text{nc}}([1540, 8], 2, 1), 8 \rangle,$$

which has the minimal cardinality associated with F_7, as stated from Lemma 2. We can follow the same procedure for $d_{\text{T}} = 2$. On the one hand, the TI set covering for F_7 would be $\langle \widehat{\mathcal{C}}_{F_7}^{\text{nc}}(15, 7, 2), 6435 \rangle$. On the other hand, the set coverings corresponding to LLTI would result in $\langle \widehat{\mathcal{C}}_1^{\text{nc}}(13+2, 3, 4), 455 \rangle$, $\langle \widehat{\mathcal{C}}_2^{\text{nc}}(455, 2, 2), 103285 \rangle$, and $\langle \widehat{\mathcal{C}}_3^{\text{nc}}([103285, 15], 2, 2), 6435 \rangle$. With these examples, and the theory presented in this section we have shown that with LLTI we can achieve the same minimal bounds defined by TI in Eq. (1).

3.3 Optimizing LLTI

We should not forget that the numbers from the examples above refer to the number of shares. We can see that, already for second-order security, the cardinality of the sets grows enormously. However, these are only the examples corresponding to the trivial LLTI constructions, with the minimum number of input shares. In this section, we show how to keep the intermediate coverings cardinality from growing disproportionately by increasing the input shares. Throughout the rest of the manuscript we use trivial LLTI (presented above) to refer to LLTI constructed with sharings that strictly follow the guidelines of Eq. (1). The optimized LLTI or LLTI (presented below) refers to the construction of the scheme with non-trivial sharings, as the ones described in Sect. 2.4.

3.3.1 Preventing intermediate expansions

In [29], Petrides studies more in-depth the cases where the set coverings do not correspond to the trivial cases, which correspond to directly applying Eq. (1). As we see from Cor. 2.2.1, it is possible to reduce the cardinality of a set only by increasing the number of input shares. This technique was already used in [21] to convert the $5 \to 10 \to 5$ sharing to a $6 \to 6 \to 6$ one, avoiding the expansion of the shares in between. As explained in [6], the expansion does not entail a problem for uniformity, since then the sharing goes back to the original size within the same cycle. Nevertheless, it helps to optimize the area requirements. In the following, we prove that it is possible to prevent the expansion of shares in every layer independently, in a similar fashion as in [21], for any order of security and any algebraic degree without affecting the non-completeness.

Lemma 3. *Given $F_{\text{T}} = R^N \circ \cdots \circ R^1$ with* T *composite, we can achieve constant cardinality for every set covering across the different layers by increasing the number of input shares s_1 enough, such that they are still non-complete set coverings.*

Proof. Given the case where $s < s_o$, we can always achieve $\langle \mathcal{C}(s+S, t, d), s+S = s_o \rangle$ by applying Lemma 2.2.1. Applying this concept consecutively to every layer, we can achieve constant cardinality across the N layers. The non-completeness follows from applying Cor. 2.2.1 together with Lemma 1 to every layer as follows:

$$\begin{aligned} \left|\mathcal{C}^{\text{nc}}(s, t, d)\right| \geq \left|\mathcal{C}^{\text{nc}}(s+1, t, d)\right| \geq \\ \geq \left|\mathcal{C}^{\text{nc}}(s+2, t, d)\right| \geq \ldots \geq \left|\mathcal{C}^{\text{nc}}(s+S, t, d)\right|. \end{aligned}$$

Thus, we conclude that $\mathcal{C}(s+S, t, d) \subseteq \mathcal{NC}(s+S, t, d)$ and $\langle \mathcal{C}^{\text{nc}}(s+S, t, d), s+S \rangle$. □

For first-order security, $s = s_o$, so this technique is important to prevent intermediate expansions from growing out of control. For higher orders of security, it is always the case that $s_o > s$ for the trivial sharing, which means extra registers and fresh randomness may be needed to bring it back to $s = s_o$. Moreover, as we can see in the example from Fig. 3, the intermediate expansions of shares rapidly grows out of control. Thus, this property becomes of utmost importance in those cases to achieve sharings of reasonable sizes. For the non-trivial cases, which correspond to the cases where s does not correspond to the minimal one from Eq. (1), there is no bound for the minimal s_o anymore, and which is not straightforward to find.

3.3.2 Prime T

In the case where T is prime, Lemmas 2 and 3 are not enough to ensure non-completeness when seeking constant cardinality. For the non-trivial case, from Def. 2 (2) and Prop. 2.2, we have that $t_i \leq \left|\mathcal{S}_i\right| \leq s - (d_i - 1)t_i - 1 = k$ for a given R^i. Thus, Eq. (7) does not hold anymore, and the dependencies at the input of R^N are not precisely bound. The cause of this is the variability introduced by the changing number of input shares to seek constant cardinality. Layers R^1 to R^{N-1} adhere to the constraints given by Eq. (5), and hence, from Lemma 3, constant cardinality can be achieved without harming the non-completeness on those layers. Hence, only an additional constraint for layer R^N is needed. As we have seen from Lemma 2, this problem does not exist in the trivial case, since the universe of the inputs is strictly bound to $\left|\mathcal{U}_s^{\mathrm{T}}\right| = \left|\mathcal{U}_s^{\mathrm{T}-1}\right| + d_{\mathrm{T}}$. Despite not being sufficient, the guidelines from this lemma are still useful and can be used as a guide in the search for a more optimal sharing in the case of T prime. However, it does not guarantee any more that $\mathcal{C}_{F_{\mathrm{T}}}(s_{\mathrm{T}}, \mathrm{T}, d_{\mathrm{T}}) \subseteq \mathcal{NC}(s_{\mathrm{T}}, \mathrm{T}, d_{\mathrm{T}})$. It is necessary to include one additional constraint.

Proposition 1. *To achieve $\mathcal{C}^{nc}_{F_{\mathrm{T}}}(s_{\mathrm{T}}, \mathrm{T}, d_{\mathrm{T}})$ for T prime, the covering at layer R^{N-1} has to satisfy that*

$$\forall \mathcal{S} \subseteq \mathcal{C}^{nc}_{N-1}(s_{N-1}, N-1, d_N) : \left|\mathcal{S}\right| < \left\lceil \frac{s_1}{d_{\mathrm{T}}} \right\rceil - 1, \tag{10}$$

for $\mathcal{U}_s^{\mathrm{T}}$.

Proof. Suppose the maximum number of dependencies concerning $\mathcal{U}_s^{\mathrm{T}}$ at the output of layer R^{N-1} is denoted with $\left|\mathcal{S}_{N-1}\right|$. Since T is prime, $t_N = 2$ and the second input sharing is equivalent to $\mathcal{U}_s^{\mathrm{T}}$. Then, as we have seen in Lemma 2, the dependencies at the output of R^N are at most $\left|\mathcal{S}_{N-1}\right| + 1$, *i.e.*, $\left|\mathcal{S}_N\right| \leq \left|\mathcal{S}_{N-1}\right| + 1$. Given the security order d_{T}, we know that every output combination will have at most $(\left|\mathcal{S}_{N-1}\right| + 1) \cdot d_{\mathrm{T}}$. To ensure non-completeness, we need that $s_1 > (\left|\mathcal{S}_{N-1}\right| + 1) \cdot d_{\mathrm{T}}$, which brings us to

$$\left|\mathcal{S}_{N-1}\right| < \left\lceil \frac{s_1}{d_{\mathrm{T}}} \right\rceil - 1.$$

□

3.3.3 Achieving minimal circuit complexity

As concluded in Lemma 3, it is always possible to achieve constant covering cardinality across layers. However, this does not necessarily mean that this results in the most optimal implementation. To achieve this constant cardinality, one has to increase the number of input shares, which also impacts the linear layers (and other logic) surrounding the high-algebraic-degree non-linear operation under consideration. Thus, in some cases, it may be a better idea to moderately increase the number of inputs, such that there is still a constrained expansion of shares in the middle layers, to achieve the best compromise, taking into account the surrounding logic as well. It is then up to the designer to find a trade-off between the trivial case, with enormous expansion in the middle layers and the

least number of input shares (with the corresponding necessity of compressing s_o), and the constant cardinality case, where there is no expansion in the middle layers but where the input shares increase might drastically affect the overall architecture. It is noteworthy that increasing the number of shares is one of the strategies used in [30] to achieve uniformity. Therefore, the increase in the number of input shares can lead not only to improve the area performance but also to achieve uniformity, making it worthwhile even if more surrounding logic is needed.

3.3.4 Merging the N coverings

A final step is needed when using alternative sharings different from the trivial ones. The coverings at each layer must be compatible with each other, *i.e.*, that the composition of them results in a non-complete covering. For the trivial cases from Sect. 3.2 this is straightforward, since input and output shares are decided based on the worst-case expansions, where every set combination corresponds to a different element in the next layer. A more careful procedure is needed in the optimized case.

3.4 Uniformity and Resharing

With the concepts presented above, we ensure non-completeness is fulfilled, and hence, security within the same cycle. For first-order security, it suffices to additionally satisfy uniformity to achieve overall security. To achieve uniformity there are several strategies we can follow [30], including increasing the number of input shares, adding fresh randomness, or duplicating different cross-terms, such that correctness still holds. For LLTI, we can benefit from the optimization strategy of Sect. 3.3 to also find a uniform sharing.

For higher orders of security, ensuring uniformity is not enough to prevent multivariate attacks, as it was shown in [33]. Hence, we need to add extra randomness for remasking to keep the security guarantees. It was noted in [20] that the security claims of the ring refreshing from CMS [7] do not hold as of third-order. Similarly for the refreshing proposed in [8], which is broken from second-order on. In light of these results, for higher-orders, we propose to use the ISW refreshing after every non-linear operation masked with LLTI and before any registers layer (if any) that may be placed in between. In [20] it is shown as well that composing TI with an SNI refreshing does not result in an SNI gadget. However, in this work we do not seek to achieve SNI-ness, but glitch-probing security. Nevertheless, we stress that this is only a recommendation, and it is not in the scope of this paper to provide an optimal refreshing. As mentioned in [8], this is still an open question. Similarly, the optimization of the additional randomness requirements for securing higher-order implementations is not within the scope of this paper. In any case, LLTI could naturally benefit from any such proposals to reduce randomness or ensure composability.

4 Applications

As the name suggests, LLTI aims at achieving low-latency masked implementations. The main applications of LLTI are to secure higher algebraic degree functions such that no registers are needed in between, and to secure unrolled implementations. Nevertheless, it can be used to secure any composition of multiple functions. In this section, we show that we can achieve higher performance by applying LLTI already with the trivial cases (Sect. 3.2) than with the classic TI. Moreover, we show that the trivial cases can be improved greatly according to Sect. 3.3, to achieve yet more optimal implementations. Applying these concepts, we devise a first-order sharing for $\mathrm{T} = 7$, which is, to the best of our knowledge, the first sharing capable of directly securing the AES S-box's ANF.

4.1 High Algebraic Degree Functions

In this section, we show that the application of LLTI not only makes the masking of functions with $t > 2$ possible without incurring in further cycle penalties, but that the area reduction is also significant. First, we discuss the circuit complexity resulting from a masked implementation, giving an example for a simple cubic AND gate masked with TI and LLTI, and then show that the greater the security order and T, the greater is the complexity reduction when applying LLTI. Finally, we present highly optimized first-order coverings for different values of $\mathrm{T} = 5, 6, 7$.

4.1.1 Circuit complexity when masking

We can calculate the number of AND and XOR operations of a given sharing based on the number of input and output shares, and the algebraic degree. The number of AND gates and XOR gates needed to share a single monomial of degree t is given in Eq. (11):

$$\begin{aligned} \#AND &= (t-1) \cdot s^t \\ \#XOR &= s^t - s_o. \end{aligned} \tag{11}$$

The number of AND gates is given by all resulting cross-products s^t from the multiplication, all of them being degree t, which translates into $(t-1)$ multiplications per cross-product. These cross-products are XORed together and distributed among the different output shares. For the unbalanced sharing, since we have two different number of input shares, the cross-products are given by $[s_1 \cdot s_2]$, Eq. (11) resulting in:

$$\begin{aligned} \#AND &= [s_1 \cdot s_2] \\ \#XOR &= [s_1 \cdot s_2] - s_o. \end{aligned} \tag{12}$$

Finally, to calculate the gate numbers for a scheme applying LLTI, it suffices to apply these equations layer-wise and add the results together.

Example: To illustrate the power of LLTI, we show an example on a single cubic operation, *i.e.*, a 3-input AND gate in $GF(2)$, $F_3 = abc$. We evaluate the number of operations needed to mask it following the traditional TI (direct method), using Eqs. (1), versus the operations needed using our proposed method. Directly masking F_3, we get that

$$(s, t, d, s_o) = (4, 3, 1, 4) \Rightarrow \begin{cases} \#AND = 2 \cdot 4^3 = 128 \\ \#XOR = 4^3 - 4 = 60 \end{cases} \tag{13}$$

The number of gates needed in the example from Fig. 2 is

$$\begin{aligned} R^1: \quad (4, 2, 1, 6) \quad &\Rightarrow \begin{cases} \#AND = (2-1) \cdot 4^2 = 16 \\ \#XOR = 4^2 - 6 = 10 \end{cases} \\ R^2: \quad ([6, 4], 2, 1, 4) \quad &\Rightarrow \begin{cases} \#AND = (6 \cdot 4) = 24 \\ \#XOR = 6 \cdot 4 - 4 = 20 \end{cases} \\ F_3: \quad (4, 3, 1, 4) \quad &\Rightarrow \begin{cases} \#AND = 40 \\ \#XOR = 30 \end{cases} \end{aligned} \tag{14}$$

From Eq. (13), we obtain that sharing a cubic operation for first-order security with a direct sharing requires 188 gates, whereas with LLTI (Eq. (14)) it needs a total of 70 gates. This entails a 62.77% reduction in circuit complexity. As described in [10], a large number of symmetric primitives include non-linear cubic operations of AND depth two, including Prince [25], Present [22], or the inverse in $GF(16)$, often used in masking the AES S-box when implemented with tower field decomposition.

4.1.2 What is the reason for this significant reduction?

The reason lies in the basics of the distributive property. While the direct method expands every cross-product and then compresses them back, LLTI expands only the elements for the first multiplication, and compresses those back before continuing with the next multiplications. The equations below present these implications for the previous first-order example of $F_3 = abc$:

$$\begin{aligned}
&\text{TI}: \\
abc &= (a_1 \oplus a_2 \oplus a_3 \oplus a_4) \cdot (b_1 \oplus b_2 \oplus b_3 \oplus b_4) \cdot \\
&\quad (c_1 \oplus c_2 \oplus c_3 \oplus c_4) \\
&= a_1b_1c_1 \oplus a_1b_1c_2 \oplus \ldots \oplus a_4b_4c_4 = F_3 \\
&\text{LLTI}: \\
ab &= (a_1 \oplus a_2 \oplus a_3 \oplus a_4) \cdot (b_1 \oplus b_2 \oplus b_3 \oplus b_4) \\
&= a_1b_1 \oplus a_1b_2 \oplus \ldots \oplus a_4b_4 = w \\
wc &= (w_1 \oplus w_2 \oplus w_3 \oplus w_4 \oplus w_5 \oplus w_6) \cdot \\
&\quad (c_1 \oplus c_2 \oplus c_3 \oplus c_4) \\
&= w_1c_1 \oplus w_1c_2 \oplus \ldots \oplus w_6c_4 = z
\end{aligned}$$

We show an extremely simple example that perfectly illustrates the reason behind this great reduction in logic gates:

$$\begin{aligned}
(1+2+3)\cdot(1+2+3) = 6\cdot 6 = 36 \quad &\Rightarrow \quad \begin{cases} \#\times = 1, \\ \#+ = 4 \end{cases} \\
1\cdot 1 + 1\cdot 2 + 1\cdot 3 + 2\cdot 1 + 2\cdot 2 + & \\
+2\cdot 3 + 3\cdot 1 + 3\cdot 2 + 3\cdot 3 = 36 \quad &\Rightarrow \quad \begin{cases} \#\times = 9, \\ \#+ = 8 \end{cases}
\end{aligned} \tag{15}$$

From this example, it is clear that the expansion of the cross-products needs more "resources" to compute the same operation. With the direct sharing, all cross-products are distributed among the output shares. In LLTI, the sharings for the smaller degree operations are computed separately, avoiding the high amount of cross-products produced with high values of t.

With Eqs. (13) and (14) we have shown the area improvements for first-order and $\mathrm{T} = 3$. In the following, we calculate some additional circuit complexity numbers for other algebraic degrees to illustrate that the area improvements are more dramatic the greater the values of T and d_T are. In this comparison, presented in Tab. 1, we have only included the trivial cases (as described in Sect. 3.2) since for them we can define the sharings without searching for them.

As we can see from Tab. 1, for $\mathrm{T} = 7$ we can achieve 455 times reduction for 1st-order security, compared to the 2.68 from the example above with $\mathrm{T} = 3$. Although most of these numbers are impractical, this comparison shows the drastic reduction we can achieve with LLTI, making the design of these sharings one step closer to a practical implementation.

4.1.3 Further optimizing the area

According to Eq. (11), the circuit complexity increases notably with the number of input shares, with more severity the greater the algebraic degree. To reduce the area of the implementation of a given F_T operation we have to minimize the number of input shares on each layer. From Prop. 2.1, we know that the direct case uses the minimum number of shares to achieve a non-complete sharing, which means that the circuit complexity for

Table 1: Gate complexity comparison between TI and the trivial cases of LLTI for $T = 4, 5, 6, 7$ applied to a single monomial, for first- and second-order security, obtained by applying Eqs. (11) and (12).

	TI		trivial LLTI		Reduction
	Covering	Size (g)	Coverings	Size (g)	(g(TI)/g(LLTI))
			$d_T = 1$		
T = 4	$\langle \widehat{\mathcal{C}}^{nc}(5,4,1),5\rangle$	2495	$2 \times \langle \widehat{\mathcal{C}}^{nc}(5,2,2),10\rangle$ $\langle \widehat{\mathcal{C}}^{nc}(10,2,1),5\rangle$	275	×9
T = 5	$\langle \widehat{\mathcal{C}}^{nc}(6,5,1),6\rangle$	39K	$2 \times \langle \widehat{\mathcal{C}}^{nc}(6,2,2),15\rangle$ $\langle \widehat{\mathcal{C}}^{nc}(15,2,1),105\rangle$ $\langle \widehat{\mathcal{C}}^{nc}([105,6],2,1),6\rangle$	1.7K	×23
T = 6	$\langle \widehat{\mathcal{C}}^{nc}(7,6,1),7\rangle$	706K	$2 \times \langle \widehat{\mathcal{C}}^{nc}(7,3,2),35\rangle$ $\langle \widehat{\mathcal{C}}^{nc}(35,2,1),7\rangle$	4.4K	×160
T = 7	$\langle \widehat{\mathcal{C}}^{nc}(8,7,1),8\rangle$	14.7M	$2 \times \langle \widehat{\mathcal{C}}^{nc}(8,3,2),56\rangle$ $\langle \widehat{\mathcal{C}}^{nc}(56,2,1),1540\rangle$ $\langle \widehat{\mathcal{C}}^{nc}([1540,8],2,1),8\rangle$	32.3K	×455
			$d_T = 2$		
T = 4	$\langle \widehat{\mathcal{C}}^{nc}(9,4,2),126\rangle$	26K	$2 \times \langle \widehat{\mathcal{C}}^{nc}(9,2,4),36\rangle$ $\langle \widehat{\mathcal{C}}^{nc}(36,2,2),126\rangle$	2.7K	×9.6
T = 5	$\langle \widehat{\mathcal{C}}^{nc}(11,5,2),462\rangle$	804K	$2 \times \langle \widehat{\mathcal{C}}^{nc}(11,2,4),55\rangle$ $\langle \widehat{\mathcal{C}}^{nc}(55,2,2),1485\rangle$ $\langle \widehat{\mathcal{C}}^{nc}([1485,11],2,2),462\rangle$	37K	×22
T = 6	$\langle \widehat{\mathcal{C}}^{nc}(13,6,2),1716\rangle$	29M	$2 \times \langle \widehat{\mathcal{C}}^{nc}(13,3,4),286\rangle$ $\langle \widehat{\mathcal{C}}^{nc}(286,2,2),1716\rangle$	174K	×167
T = 7	$\langle \widehat{\mathcal{C}}^{nc}(15,7,2),6435\rangle$	1.2G	$2 \times \langle \widehat{\mathcal{C}}^{nc}(15,3,4),455\rangle$ $\langle \widehat{\mathcal{C}}^{nc}(455,2,2),103285\rangle$ $\langle \widehat{\mathcal{C}}^{nc}([103285,15],2,2),6435\rangle$	3.4M	×353

this method can not be reduced further. For LLTI, for the same reason, we cannot reduce further the area requirements of R^1. However, we can reduce the area for all subsequent layers by applying the concepts from Sect. 3.3. By increasing the number of input shares of R^1, we can reduce the number of output shares, which correspond to the number of input shares for R^2. We can proceed similarly for subsequent layers, to reduce the overall costs for the implementation of the non-linear operation.

This offers a great search space for the designer, to find the best trade-offs for a particular case. The expansion of input shares in R^1 translates in the reduction of input shares for subsequent layers, and ultimately, on the reduction of the area of F_T. However, the increase of input shares also entails a consequent increase of shares in the surrounding logic. Hence, the designer should find a balance between the area requirements of the non-linear operation in isolation, and the whole circuit. An example of this optimization procedure can be found in [21], where the coverings $\langle \widehat{\mathcal{C}}^{nc}(5,2,2),10\rangle \rightarrow \langle \widehat{\mathcal{C}}^{nc}(10,2,1),5\rangle$ were optimized to $\langle \mathcal{C}^{nc}(6,2,2),6\rangle \rightarrow \langle \mathcal{C}^{nc}(6,2,1),6\rangle$, increasing by one the number of input and output shares.

Computing the numbers as in Tab. 1 for a monomial of the $6 \rightarrow 6 \rightarrow 6$ sharing, we get a total number of gates of 198, a 1.4 times reduction compared to the 275 from the $5 \rightarrow 10 \rightarrow 5$ sharing. Moreover, this entails a total of 12.6 times reduction w.r.t. TI sharing with one input share less (see the first row from Tab. 1). The final implementation of

[21], despite having an extra share, achieved a better overall area and maximum frequency performance compared to the trivial case.

4.1.4 The problem of finding $min(s + s_o)$

For the cases where s is not the minimal value, for first-order security, it is always the case that $s_o \geq td + 1$, no matter how much one increases the number of input shares with respect to the trivial case. For higher orders, there is no longer a clear relationship between s and s_o. The problem of finding the minimum $(s + s_o)$ for $s > td + 1$ was already introduced in [6]. To date, it is still an open question, and there are no formulas that establish a relationship between them. A small step forward is taken in [29], where the construction for second-order security of a quadratic operation with $s = s_o = 6$ is proven to be the minimal one, *i.e.*, $\langle \widehat{\mathcal{C}}^{\text{nc}}(6, 2, 2), 6 \rangle$. Additionally, for $t = 3$ and $s = 9$, the bound $s_o \leq 13$ is given.

This problem is equivalent to the set cover problem in combinatorics and complexity theory, considered an NP-problem, and which has been studied for a long time [34, 35, 36, 37, 38]. These works propose similar bounds for the number of elements and the minimal number of subsets needed to entirely cover them. This corresponds exactly to our problem, with the exception that we need to include the non-completeness constraint. In these works, a (v, k, t)-covering design is a collection of k-element subsets of $\{1, 2, \ldots, v\}$, such that any t-element subset is contained in at least one block. In our notation, (v, k, t) corresponds to $(s, \left|\mathcal{S}\right|, t)$. The upper bounds for the coverings are given individually for every particular case, depending on v, k, and t. These bounds are found using different strategies, like greedy algorithms, induced from other coverings, with the help of finite geometry, etc. In [39], we can find the upper bounds for $t = \{2, 3, 4, 5\}$, depending on the different values of v and k.

Similar strategies and results are needed to find optimal TI sharings, including non-completeness as a constraint, and for the parameters (s, t, d). The problem of finding the minimum bound for $s + s_o$ is beyond the scope of this manuscript and remains as an interesting subject for future works. However, our methodology can be used already to bound the search to $t \leq 3$. The layered decomposition allows a divide-and-conquer strategy, where, instead of finding the bounds for $t = 6$, for instance, with the corresponding exponential increase in cross-terms, we can just search the minimum coverings for the two layers with $t = 3$ and $t = 2$, respectively. Once the minimal bound is established, it suffices to find a sharing within those bounds such that the merge of both coverings conforms a non-complete covering. For the case when T is prime, the algorithm should also add the constraint given in Prop. 1.

4.1.5 Optimized sharings for $\mathbf{T = 5, 6, 7}$

Here, we present highly optimized first-order non-complete coverings for $\mathrm{T} = 5, 6,$ and, 7, following the guidelines from Sect. 3.3 to reduce the gate requirements further compared to the ones presented in Tab. 1. To the best of our knowledge, these are the first sharings given for such algebraic degrees. To allow checking the validity of our results, we provide the reader the algorithms we used to build these coverings, the detailed constructions, and corresponding non-completeness checks in [40].

For the intermediate sharings, we use sharings from previous papers or take the coverings from the set covering literature. The covering sets described in [39] can be found in [41]. Note that these coverings are not constructed with the non-completeness property in mind, so it is important to check whether they satisfy it. While tailored bounds to our problem are still missing, this database is a great source for guiding the designer to get the sharing needed. Nevertheless, smaller coverings may be achieved by specifically searching the targeted set coverings with non-completeness as an initial constraint.

T = 5 It suffices to increase the input shares by one (as from Lemma 2) with respect to the optimal case for T = 4 from [21] to find a non-complete covering. The covering for the first layer is taken from the coverings of [41]. We choose a covering set such that it is second-order non-complete, according to the constraints from Eq. (5). Furthermore, the choice of this covering has to comply with the restriction from Prop. 1, so that the dependencies created in the second layer are within the required bounds. Finally, the unbalanced and compression sharing outputs the minimum number of shares. The resulting set coverings for each layer are:

$$\begin{aligned} R^1: &\quad \langle \mathcal{C}_1(7,2,2), 7\rangle \\ R^2: &\quad \langle \mathcal{C}_2(7,2,1), 21\rangle \\ R^3: &\quad \langle \mathcal{C}_3([21,7],2,1), 7\rangle \end{aligned}$$

According to Eqs. (11) and (12), the total gates required to realize this sharing is 546, 71 times smaller than the TI sharing, and 3 times smaller than the trivial case.

T = 6 To design this covering we follow the layered decomposition for T = 6 from Fig. 3a. For the first layer we need a second-order non-complete covering for $t_1 = 3$. Petrides [29] precisely proposes in his work such a covering, so we start from this set. Since second-order non-completeness is ensured at the output of R^1, the compression sharing for the second layer is therefore constructed ensuring that there is always one element from the input share missing:

$$\begin{aligned} R^1: &\quad \langle \mathcal{C}_1(9,3,2), 13\rangle \\ R^2: &\quad \langle \mathcal{C}_2(13,2,1), 9\rangle \end{aligned} \tag{16}$$

The total number of AND and XOR gates to implement this sharing is 4 677 gates. This entails a total of 151 times reduction with respect to TI, given in Tab. 1. In this case, however, the number of gates needed for the LLTI trivial case is smaller.

We can go even further and apply LLTI again in the first two cubic AND gates (See Fig. 3), to reduce the area of the first layer. Now we have a case analogous to that of Fig. 2, where we decompose R^1 into R^{11} and R^{12}. Finding the set covering for this case is easier since we already have the output shares indexes of R^{12}, defined by the output shares of the former R^1. It remains to find a suitable covering for R^{11}, such that when added the dependency from applying R^{12}, the final combination sets can be distributed into the outputs of R^1. R^{11} is devised by manually reducing the trivial sharing (with $s_o = \binom{9}{2} = 36$) as much as possible to 32 output sets. Hence, the R^1 set covering $\langle \mathcal{C}_1(9,3,2), 13\rangle$ can be split as follows:

$$\begin{aligned} R^{11}: &\quad \langle \mathcal{C}_{11}(9,2,2), 32\rangle \\ R^{12}: &\quad \langle \mathcal{C}_{12}([32,9],2,2), 13\rangle \end{aligned} \tag{17}$$

The combined number of AND and XOR gates to implement the corresponding sharing counting R^2 from Eq. (16) is 1 715, a final reduction of more than 400 times with respect to TI, and more than 2.5 times reduction with respect to the LLTI trivial case and the previous LLTI optimized case.

T = 7 As shown in Fig. 3b, the constraints for the first two layers are the same as in T = 6. The trivial bound according to Eq. (1) is $s = 8$, which is equivalent to the outcome of applying Lemma 2. We may increase the number of inputs by one, and proceed to calculate the set coverings of every layer with $s = 9$. The sharing we used from Petrides for the previous case also satisfies these constraints. However, it does not comply with the requirement from Prop. 1. The dependencies after the first layer are $|\mathcal{S}| = 5$ number of

shares, which after the second non-linear operation turns into at most 10 dependencies with respect to the input shares. From Eq. (10), the covering from the second layer must have $\left|\mathcal{S}\right| < \frac{9}{1} - 1 = 8 \not> 10$, with respect to $\mathcal{U}_s$. To find an alternative covering we go back to the database from [41]. We could not find any set with $s = 9$ such that the previous condition is fulfilled, so we go up to $s = 10$, where we succeed to find a suitable non-complete covering. To construct the R^2 sharing, we can again search for a non-complete covering with 30 inputs from [41], or construct it ourselves. We decide to construct the covering set ourselves with non-completeness in mind. We know that the maximum number of dependencies at the output of R^2 is $\left|\mathcal{S}\right| = 8$, and that the last layer adds just one extra dependency. Thus, ensuring that all output shares of R^2 have at least two shares missing from $\mathcal{U}_s$ will allow us achieve overall non-completeness. We calculate the dependencies of each combination at R^2 and get the complementary set ($\mathcal{S}^c$) with respect to $\mathcal{U}_s$. Then, we look at all different combinations such that $\left|\mathcal{S}^c\right| = 2$, and define them as the output shares baskets of R^2. By definition, the last layer adds a dependency to each element, which means that $\left|\mathcal{S}_3\right| = 9$. Since $s = 10$ it suffices to identify a single missing share at every output share of R^3 and distribute the cross-products according to this missing index dependency. A more detailed step-by-step commented code to generate this sharing is included in [40]. The set covering corresponding to the three layers is:

$$\begin{aligned} R^1: &\quad \langle \mathcal{C}_1(10,3,2),30\rangle \\ R^2: &\quad \langle \mathcal{C}_2(30,2,1),45\rangle \\ R^3: &\quad \langle \mathcal{C}_3([45,10],2,1),10\rangle \end{aligned} \tag{18}$$

Calculating the gate complexity with Eqs. (11), (12), we need a total of 8 585 gates to construct the covering defined in Eq. (18). This entails a 3.76 times reduction with respect to LLTI's trivial case, and a total of 1718 times reduction compared to TI.

Similarly as with T = 6, we reduce this covering further by again applying LLTI on the first cubic gates. By applying Eq. (10) to R^{11}, we obtain that $\left|\mathcal{S}\right| = 3$ is allowed. However, with such a sharing, when the extra dependency added from R^{12} is taken into account, it is impossible to distribute all combinations according to R^1 output shares. Thus, R^{11} is just the trivial sharing, where $s_o = \binom{10}{2} = 45$, and $\left|\mathcal{S}\right| = 2$. Finally, $\langle \mathcal{C}_1(10,3,2),30\rangle$ can be split in:

$$\begin{aligned} R^{11}: &\quad \langle \mathcal{C}_{11}(10,2,2),45\rangle \\ R^{12}: &\quad \langle \mathcal{C}_{12}([45,10],2,2),30\rangle \end{aligned}$$

Achieving an overall gate count of 4 695, taking into account R^2 and R^3 from Eq. (18). This means a reduction of 1.8 time with respect to the previous case, a reduction of more than x6 times comparing with the trivial case, and a final reduction of 3131 times with respect to TI.

Within this section we have proposed highly optimized first-order sharings for T = $3, 5, 6, 7$ (T = 4 already proposed in [21]), summarized in Tab. 2. The results presented in this section can be used to mask higher algebraic degrees of complex constructions efficiently, even if the AND depth is greater than one. Moreover, the covering corresponding to T = 7 can be used to directly mask the AES S-box ANF without registers, which is, to the best of our knowledge, the first sharing to allow it.

4.2 Securing a full cipher with LLTI

Previous sections focus on how to apply LLTI to a non-linear function, which is the key element to secure in a cipher. Below, we provide a recipe on how to apply LLTI to an entire cipher to achieve a fully secured implementation:

Table 2: Summary of final first-order LLTI coverings for $T = 3, 5, 6, 7$ applied to a single monomial.

Degree	Layer	Covering	Size (gates)	Reduction times (w.r.t) TI	Reduction times (w.r.t) Trivial LLTI
$T = 3$	R^1	$\langle \mathcal{C}_1(4, 2, 1), 6\rangle$	70	$\times 2.68$	
	R^2	$\langle \mathcal{C}_2([6, 4], 2, 1), 4\rangle$			
$T = 5$	R^1	$\langle \mathcal{C}_1(7, 2, 2), 7\rangle$	546	$\times 71$	$\times 3$
	R^2	$\langle \mathcal{C}_2(7, 2, 1), 21\rangle$			
	R^3	$\langle \mathcal{C}_3([21, 7], 2, 1), 7\rangle$			
$T = 6$	R^{11}	$\langle \mathcal{C}_{11}(9, 2, 2), 32\rangle$	1 715	$\times 400$	$\times 2.5$
	R^{12}	$\langle \mathcal{C}_{12}([32, 9], 2, 2), 13\rangle$			
	R^2	$\langle \mathcal{C}_2(13, 2, 1), 9\rangle$			
$T = 7$	R^{11}	$\langle \mathcal{C}_{11}(10, 2, 2), 45\rangle$	4 695	$\times 3131$	$\times 6.9$
	R^{12}	$\langle \mathcal{C}_{12}([45, 10], 2, 2), 30\rangle$			
	R^2	$\langle \mathcal{C}_2(30, 2, 1), 45\rangle$			
	R^3	$\langle \mathcal{C}_3([45, 10], 2, 1), 10\rangle$			

- **Number of shares**: when implementing an SCA-secure design, we first need to determine the desired security order. The non-linear operation is the limiting factor of the implementation, *i.e.* the most difficult element to secure. Thus, the security order together with the algebraic degree of the non-linear function will define the number of shares to employ in the design. By plugging these parameters in Eq. (5) and following the instructions from Sect. 3, we can obtain the complete sharing of the non-linear function and the number of shares for the overall implementation. In this work, we have presented a first-order sharing for each basic multiplication of algebraic degree up to $t = 7$, which are included in [40]. These sharings can be use directly to secure any function of such algebraic degree.
- **Refreshing**: then, we should include a refreshing layer to the masked non-linear function, as described in Sect. 3.4.
- **Full implementation**: finally, since linear operations are performed share-wise, every linear block from the design should have the same number of copies as number of shares in the design. Similarly, MUXes, buses, and Primary Inputs and Outputs should be also replicated as many times as the number of resulting shares from the application of LLTI to the non-linear operation.

Overall, securing a cipher with LLTI follows the same major steps as with any masking scheme, with the key differences in how to secure the non-linear layer.

4.3 Unrolled Implementations

The second application possible for LLTI is to secure unrolled implementations, as it was done in the work of Arribas *et al.* [21], where they proposed a first-order secure unrolled implementation of Keccak.

The typical design flow to secure a (simple) S-box, is to take the Algebraic Normal Form (ANF) and substitute each variable with the shared representation, distributing the cross-terms among the output shares respecting the security requirements to realize the shared function. This is impossible to do with unrolled implementations, since we cannot

get a single equation to define two (or more) rounds at once. Hence, the only alternative to mask the unrolled expression is by masking each round independently and then merging them, as in LLTI.

The equivalent overall degree t for a single round of the unrolled implementation is calculated by looking at the corresponding non-linear operations of each layer. Then, the strategy from Sect. 3 is applied to look for the sharings. The linear operations can be trivially included in the covering of the corresponding layer, since they do not create dependencies among different shares. They may create additional dependencies among different variables of the inputs within one share, but even if they do so, that is still not a problem since TI does not require independent inputs.

5 Implementations and Case Studies

In this section we present the results of applying LLTI to different primitives. We begin with the multiplication, the most basic non-linear operation in symmetric key primitives. Then, we present results for widely used ciphers, including example for PRINCE and AES.

5.1 Platform and Evaluation

The synthesis results were obtained with the Synopsis Design Compiler v2013.12, using the NANGATE 45nm Open Cell Library [42]. The `exact_map` compile option is used to prevent optimization across modules. Our implementation is deployed on a Xilinx Spartan6 FPGA on a Sakura-G evaluation board. For the synthesis, the `KEEP_HIERACHY` option is enabled in Xilinx ISE to prevent optimization across modules and shares in particular. The power measurements is taken on a dedicated output on Sakura-G board.

We perform a non-specific leakage detection test (TVLA) from Goodwill *et al.* [43]. The power measurements are split in two sets: the first set $\mathcal{S}_0$ receives fixed plaintexts and the second set $\mathcal{S}_1$ random plaintexts. The two sets of measurements are compared using the t-test statistic:

$$t = \frac{\mu(\mathcal{S}_0) - \mu(\mathcal{S}_1)}{\sqrt{\frac{\sigma^2(\mathcal{S}_0)}{|\mathcal{S}_0|} + \frac{\sigma^2(\mathcal{S}_1)}{|\mathcal{S}_1|}}} \tag{19}$$

The t-test verifies whether two populations have the same *mean*, *i.e.* its null hypothesis H_0 states that "the sets $\mathcal{S}_0$ and $\mathcal{S}_1$ are drawn from populations with the same mean." Large absolute values of this t-statistic indicate that the null hypothesis can be rejected with a high degree of confidence.

In order to verify that our setup is sound, we first perform TVLA to an unprotected implementation, by turning off the PRNG. With 100 000 power traces, we see clear evidence of leakage (see Fig. 5, left). When we turn on the PRNG, the masked AES by LLTI is secure up 100 million traces. (see Fig. 5, right). Second- and third-order leakage are detected, as expected. A threshold level of 4.5 is often used to decide whether the hypothesis should be rejected. In practice, this threshold should not be considered a hard limit. Fig. 5 shows that the first-order t-statistic reaches the "threshold" at some time samples. However, Fig. 4 demonstrates clearly that the first-order t-statistic does not grow over 100 million traces, and thus that we can say with confidence that the implementation is first-order secure with this many traces.

5.2 Multiplication

To show the advantages of our methodology, we implement several masked AND gates following the guidelines proposed in the previous section. We have implemented a cubic

AND gate and a degree-7 AND gate, for both TI and LLTI for first-order security. Additionally, we have implemented the cubic AND gate with the method explained in [21] for comparison purposes. Tab. 3 presents these results. As in [9], we refer to a combinational circuit as a circuit needing 0 clock cycles to compute the result. Similarly, a circuit with a single register layer needs 1 clock cycle. For the security analysis, we evaluate non-completeness and uniformity with VerMI [44], and, additionally, include a mutual information (MI) analysis.

Table 3: Comparison between TI and LLTI for several protected AND gates implementations for first-order of security.

	Method	Area $[GE]$	Rand.* $[bpc]$	# Reg. layers	Max. Freq. $[MHz]$	NC	Unif.	MI
				T = 3				
AND3 4sh.	TI	258	3	0	690	✓	✓	0.1379
AND3 5sh.	Rhythm.	134	4	0	1538	✓	✓	0.1399
AND3 4sh.	LLTI	**97**	3	0	1639	✓	✓	0.1379
				T = 7				
AND7 4sh.	TI	709	9	1	1176	✓	?	?
AND7 4sh.	LLTI	**357**	9	1	**1613**	✓	?	?
AND7 10sh.	LLTI	**5466**	≤ 9	**0**	375	✓	?	?

* Cost of fresh random bits per cycle (bpc) during computation.

The first three rows of Tab. 3 depict the results for the masked cubic gate, for TI, [21], and LLTI, respectively. With LLTI, the area complexity is reduced a 62% and a 28% w.r.t respective other techniques, as expected from the calculations made in Sect. 4.1. Furthermore, with this method we also achieve a 137% increase in the maximum frequency with LLTI compared to TI. In the case of TI, numerous cross-products have to be compressed to the given number of output shares, creating a deep XOR gates tree. With LLTI the total number of cross-products to compress is highly reduced, which leads to the corresponding improvement in maximum frequency. To ensure uniformity we add some extra bits of randomness. Note that in the examples we are only masking a single AND gate, which is highly unbalanced by definition. For other functions, it may be possible to reduce this number with (or even without) increasing the number of shares. Finally, we have calculated the mutual information of these examples. We have calculated as well the MI of an insecure 2-bit AND gate with simply two shares and a secure version with three shares from [30] to serve as reference. The results are 0.393 and 0.311, respectively. We can see how the implementations from Tab. 3 considerably reduce the MI information compared to these cases.

The second part of Tab. 3 presents the different implementations for a 7-input AND gate. The last row implementation corresponds to the function illustrated in Fig. 3b. The previous two rows implement the same function but with a register layer between R^1 and R^2, which leave only cubic operations at both sides of the register. To mask these cubic gates we benefit from the results of masking the previous cubic gates, the first version using TI, and the second using LLTI, both with 4 shares. The area complexity is almost 50% smaller for the case with LLTI, and the maximum frequency is 37% larger.

The last implementation is a masked 7-input AND gate without any register in between. So far, there is no masked implementation of the AES S-box purely combinational. Here we only present the implementation of a single monomial of the S-box ANF, but this implementation can be trivially leveraged to implement the remaining monomials to realize a full masked S-box implementation evaluated within the same clock cycle. As we can see, the reduction in maximum frequency is quite significant. Not only the critical path is split

in two by the register, but also the number of shares is greatly reduced, leading to a great reduction of XOR gates needed to compress all the cross-products. It is important to note that this notorious decrease in the frequency would not happen in a full implementation, since this operation would no longer belong to the critical path. Hence, the designer should ponder what is the impact of that one cycle reduction, and how much is the maximum frequency affected.

For these implementations, it was not feasible to evaluate the uniformity nor the MI. Since the 7-bit AND gate can be decomposed in three cubic operations, as shown in Fig. 3, we propose a total of 9 random bits, although further analysis is required since this composition does not guarantee uniformity.

5.3 PRINCE

To further illustrate the power of LLTI, we take existing masked implementations of PRINCE using TI and exclusively modify the TI cubic sharing with the LLTI version proposed in Fig. 2. In particular, we use the one from [24] with the lowest latency. PRINCE has been designed with latency in mind, unfortunately threatened by the use of registers that typically appear in decomposed SBoxes or in $d+1$-like masking. Thus, PRINCE highly benefits from the use of LLTI as we will see in the following. Multiple previous works in the literature have presented side-channel protected implementations of PRINCE, like [23, 24, 25].

Table 4: SCA protected PRINCE implementations.

PRINCE 1st order SCA	Method	Area $[GE]$	Rand. $[bpc]$	Cycles	Max. Freq. $[MHz]$	Time (ns)
1. No S-box dec.	LLTI	25857	48	**12**	**488**	**24.6**
2. No S-box dec. [24]*	TI	41628	48	**12**	335	35.8
3. No S-box dec. [25]	$d+1$	11596	48	24	376	63.8
4. S-box dec. [45]	TI	14153	**0**	36	268	134
5. S-box dec.[23]	TI	9292	**0**	40	250	160
6. S-box dec. [45]	$d+1$	**8701**	24	72	260	277

* Original implementation source code recompiled with the NANGATE45 library.

As we can see from the first two rows in Tab. 4, while maintaining the same number of cycles, we increase the maximum frequency by 46% and reduce the computation time of the implementation by 31% with simply applying LLTI to the protected S-box. Furthermore, this also reduces the total area by 38% compared to the TI version. Since LLTI improves the performance of the core element of the implementation, any shared implementation of the multiple ciphers in the literature using a cubic function would also result in similar performance optimizations.

5.4 AES

Finally, we analyze the benefits of applying LLTI to AES. So far, we can only find two implementations of the AES S-box in the literature with a single layer of registers, both using different flavors of masking. The first one, from Gross *et al.* [9], employs the proposed Generic Low-Latency Masking (GLM), a masking scheme at the algorithmic level. The second one, from Sasdrich *et al.* [18], uses LMDPL, a gate level masking scheme. Hereunder, we present a new alternative using LLTI.

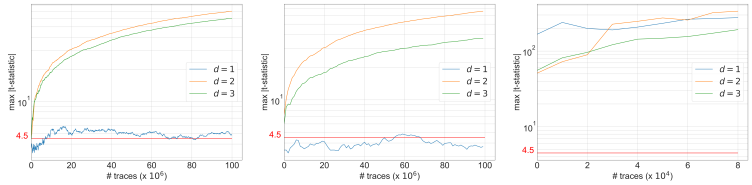


Figure 4: Max t-test value in logarithmic scale. From left to right: masks-on AES-TI, masks-on AES-LLTI, mask-off.

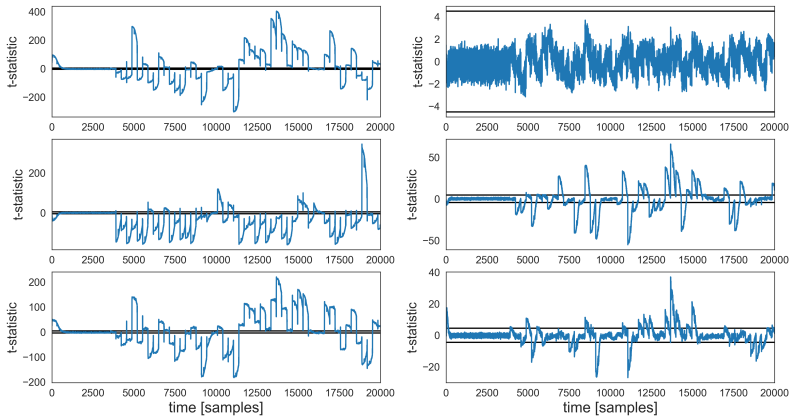


Figure 5: TVLA results for unmasked AES with 100k traces (left) and masked AES-LLTI with 100 million traces (right). Top to bottom: 1st-, 2nd- and 3rd-order t-statistic

The AES S-box consists of an inversion in $GF(2^8)$, and an affine layer. The inversion in $GF(2^8)$, which has algebraic degree seven, can be decomposed into two simpler cubic functions: $x^{-1} = x^{254} = (x^{26})^{49}$ [16, 46]. We leverage this decomposition to split our implementation into two stages with one register layer in between the two functions. Instead of splitting further these cubic functions, we directly mask them with a 1st-order non-complete $td + 1$ sharing.

Thanks to the non-completeness property, the affine transformation can be applied to each share independently, and compose with the non-linear part without using register. To illustrate the great performance optimization that LLTI can provide, we design our AES implementation using both TI and LLTI for comparison. Tab. 5 presents the results of both our implementations, together with previous works AES secure implementations for reference.

As we can see from Tab. 5, our implementations achieve minimal latency, with the S-box implemented with a single layer of registers. Moreover, no additional randomness besides the initial sharing is needed. We would like to stress that the purpose of these implementations is not to present a new or the best state-of-the-art AES implementation, but to illustrate the power of LLTI. As we can see from the first two rows, with LLTI we achieve 47.2% reduction in area and a 6.9 times increase in maximum frequency w.r.t. the TI implementation. The implementation from Gross *et al.* [9] using GLM is based on $d + 1$ masking, contrary to the $td + 1$ masking of LLTI. Since they are both maskings at the algorithmic level, it is easier to compare between them. We can see that with LLTI we manage to significantly improve both area and online randomness.

Table 5: 1st-order SCA secure AES Implementation Cost.

Design	Area* [kGE] S-box	Total	Rand. [bpc]	# Regs. S-box	Cycles Total	Max. Freq.* [MHz]	Time* (ns)
4 shares LLTI	25.78	**36.49**	**0**	**1**	216	277	779
4 shares TI	58.41	69.12	0	1	216	40	5 400
Bilgin *et al.* [13]	2.84	8.12	32	2	246	-	-
De Cnudde *et al.* [14]	1.98	**6.68**	54	5	276	-	-
Gross *et al.* [9]	60.76	-	2 048	**1**	-	356	-
Gross *et al.* [9]	6.74	-	416	2	-	**584**	-
Wegener-Moradi [47]	4.20	7.60	**0**	16	2 804	-	-
Sugawara [48]	3.50	17.10	**0**	4	266	-	-
Sasdrich *et al.* [18]	3.48	157.50	720	**1**	**10**	400	25

* Different designs used different standard cell libraries.

The latest design from Sasdrich *et al.* [18] proposes an efficient round-based implementation computed in 10 cycles. However, it is difficult to have a one-to-one comparison since Sasdrich *et al.* employ a gate level masking scheme. Although our implementation is slower due to the serialized design, it performs much better in terms of online randomness, using zero bits compare to the 720 bits per cycle from Sasdrich *et al.*'s design. This allows us to avoid any PRNG placed next to our design, which greatly affects the overall area and power performance. Moreover, the use of a PRNG creates additional vulnerabilities, easily exploitable by the attackers. Finally, compared to the implementations using zero online randomness, the latency in cycles of our S-box is reduced 16 and 4 times respectively, at the cost of larger area. Every implementation has different trade-offs, and we should choose the most fitting one tailored to the desired application. Our implementation is presented as a new alternative to the state-of-the-art.

We use TVLA to evaluate the security of both our implementations, deeming them secure for up to 100 million traces. Figs. 4 and 5 present these results. In the case of the TI implementation, we can see small leakage, but since it does not grow with the number of traces (as opposed to $d = 2, 3$) and stays close to the threshold we can say that this leakage would not be exploitable by an attacker. On the contrary, we can see that with LLTI this leakage no longer appears. From Fig. 4, we can see that LLTI not only does not harm the security of the implementation compared to TI, but it even reduces the amount of leakage. This can be attributed to LLTI's reduction in the glitching activity. Glitches introduce unexpected (and unwanted) combinations, which threaten the security of masking in hardware [49]. As we have seen from Eq. (15) LLTI considerably reduces the XOR trees that are needed to compress all cross-terms to the number of output shares. This leads to a considerable reduction in glitching activity, which potentially translates into better security warranties.

6 Conclusion

In this manuscript, we have presented Low-Latency Threshold Implementations (LLTI), a new masking methodology suitable for low-latency applications. It is derived and inherits the properties of TI, achieving a considerable improvement in the area, latency, and maximum frequency performances for any operation with an algebraic degree greater than two. For a first-order masked cubic AND gate, very often used in symmetric primitives, we achieved a 137% increase in maximum frequency and a 60% reduction in area compared to TI. Furthermore, we have presented different sharings for operations with $\mathrm{T} = 3, 5, 6$, and 7, achieving a remarkable 3131 times reduction in gate complexity for a first-order masking of a degree-seven monomial compared to TI. To show the practical impact of

LLTI, we applied it to both examples of PRINCE and AES. We took a previously proposed PRINCE implementation and simply substituting the S-box with the new LLTI version, we achieved substantial improvements in both area and latency. Similar gains could be achieved when securing any cipher featuring a cubic AND operation. Finally, we proposed a new AES design with a one-cycle S-box and its TI equivalent, obtaining, with LLTI, an implementation with half the area and seven times greater max. frequency.

7 Acknowledgements

This work was supported by CyberSecurity Research Flanders with reference number VR20192203.

References

[1] P. C. Kocher, J. Jaffe, and B. Jun, "Differential power analysis," in *CRYPTO*. Springer, 1999.

[2] Y. Ishai, A. Sahai, and D. A. Wagner, "Private circuits: Securing hardware against probing attacks," in *CRYPTO*. Springer, 2003.

[3] S. Nikova, C. Rechberger, and V. Rijmen, "Threshold implementations against side-channel attacks and glitches," in *ICICS*. Springer, 2006.

[4] E. Prouff and T. Roche, "Higher-order glitches free implementation of the AES using secure multi-party computation protocols," in *CHES*. Springer, 2011.

[5] S. Nikova, V. Rijmen, and M. Schläffer, "Secure hardware implementation of nonlinear functions in the presence of glitches," *J. Cryptology*, 2011.

[6] B. Bilgin, B. Gierlichs, S. Nikova, V. Nikov, and V. Rijmen, "Higher-order threshold implementations," in *ASIACRYPT*. Springer, 2014.

[7] O. Reparaz, B. Bilgin, S. Nikova, B. Gierlichs, and I. Verbauwhede, "Consolidating masking schemes," in *CRYPTO*. Springer, 2015.

[8] H. Groß, S. Mangard, and T. Korak, "An efficient side-channel protected AES implementation with arbitrary protection order," in *CT-RSA*. Springer, 2017.

[9] H. Groß, R. Iusupov, and R. Bloem, "Generic low-latency masking in hardware," *Trans. CHES*, vol. 2018(2).

[10] B. Bilgin, L. De Meyer, S. Duval, I. Levi, and F.-X. Standaert, "Low AND depth and efficient inverses: a guide on s-boxes for low-latency masking," in *IACR Transactions on Symmetric Cryptology*, vol. 2020,1.

[11] D. Canright, "A very compact s-box for AES," in *CHES*, 2005.

[12] A. Moradi, A. Poschmann, S. Ling, C. Paar, and H. Wang, "Pushing the limits: A very compact and a threshold implementation of AES," in *EUROCRYPT*. Springer, 2011.

[13] B. Bilgin, B. Gierlichs, S. Nikova, V. Nikov, and V. Rijmen, "Trade-offs for threshold implementations illustrated on AES," *IEEE Trans. on CAD of Integrated Circuits and Systems*, vol. 34, no. 7, 2015.

[14] T. D. Cnudde, O. Reparaz, B. Bilgin, S. Nikova, V. Nikov, and V. Rijmen, "Masking AES with d+1 shares in hardware," in *CHES*. Springer, 2016.

[15] M. Rivain and E. Prouff, "Provably secure higher-order masking of AES," in *CHES*. Springer, 2010.

[16] F. Wegener and A. Moradi, "Yet another size record for AES: A first-order SCA secure AES s-box based on $\(\mathrm {GF}(2^8)\)$ multiplication," in *CARDIS*. Springer, 2018.

[17] L. De Meyer, V. Arribas, S. Nikova, V. Nikov, and V. Rijmen, "M&m: Masks and macs against physical attacks," *Trans. CHES*, vol. 2019(1).

[18] P. Sasdrich, B. Bilgin, M. Hutter, and M. E. Marson, "Low-latency hardware masking with application to AES," *Trans. CHES*, vol. 2020(2).

[19] E. Trichina, "Combinational logic design for AES subbyte transformation on masked data," *IACR Cryptol. ePrint Arch.*, 2003.

[20] T. Moos, A. Moradi, T. Schneider, and F. Standaert, "Glitch-resistant masking revisited or why proofs in the robust probing model are needed," *Trans. CHES*, vol. 2019(2).

[21] V. Arribas, B. Bilgin, G. Petrides, S. Nikova, and V. Rijmen, "Rhythmic keccak: SCA security and low latency in HW," *Trans. CHES*, vol. 2018(1).

[22] A. Poschmann, A. Moradi, K. Khoo, C. Lim, H. Wang, and S. Ling, "Side-channel resistant crypto for less than 2, 300 GE," *J. Cryptology*, vol. 24, no. 2, 2011.

[23] A. Moradi and T. Schneider, "Side-channel analysis protection and low-latency in action - case study of PRINCE and Midori," in *ASIACRYPT*, 2016.

[24] D. Bozilov, V. Nikov, and V. Rijmen, "Design trade-offs in threshold implementations," in *ICECS*. IEEE, 2019, pp. 751–754.

[25] D. Bozilov, M. Knezevic, and V. Nikov, "Optimized threshold implementations: Minimizing the latency of secure cryptographic accelerators," in *CARDIS*. Springer, 2019.

[26] L. De Meyer, B. Bilgin, and O. Reparaz, "Consolidating security notions in hardware masking," *Trans. CHES*, vol. 2019(3).

[27] S. Faust, V. Grosso, S. M. D. Pozo, C. Paglialonga, and F. Standaert, "Composable masking schemes in the presence of physical defaults & the robust probing model," *Trans. CHES*, vol. 2018, no. 3, 2018.

[28] A. Duc, S. Faust, and F. Standaert, "Making masking security proofs concrete - or how to evaluate the security of any leaking device," in *EUROCRYPT*. Springer, 2015.

[29] G. Petrides, "On non-completeness in threshold implementations," in *TIS@CCS*. ACM, 2019, pp. 24–28.

[30] B. Bilgin, "Threshold implementations : as countermeasure against higher-order differential power analysis," Ph.D. dissertation, 2015.

[31] B. Bilgin, B. Gierlichs, S. Nikova, V. Nikov, and V. Rijmen, "A more efficient AES threshold implementation," in *AFRICACRYPT*, 2014.

[32] M. Wei, S. Sun, Z. Wei, and L. Hu, "Unbalanced sharing: a threshold implementation of SM4," in *Science China Information Sciences*, vol. 64. Springer, 2021.

[33] O. Reparaz, "A note on the security of higher-order threshold implementations," *IACR Cryptol. ePrint Arch.*, vol. 2015.

[34] J. Schönheim, "On coverings." *Pacific J. Math.*, vol. 14, no. 4, 1996.

[35] D. M. Gordon, O. Patashnik, and G. Kuperberg, "New constructions for covering designs," *Journal of Combinatorial Designs*, no. 4, 1995(3).

[36] D. M. Gordon, O. Patashnik, G. Kuperberg, and J. Spencer, "Asymptotically optimal covering designs," *J. Comb. Theory(A)*, no. 2, 1996(75).

[37] C. Dai, P. C. Li, and M. Toulouse, "A cooperative multilevel tabu search algorithm for the covering design problem," in *Artificial Evolution*. Springer, 2005.

[38] F. Montecalvo, "Some constructions of general covering designs," *Electr. J. Comb.*, vol. 19, no. 3, p. P28, 2012.

[39] C. J. Colbourn and J. H. Dinitz, *Handbook of Combinatorial Designs, Second Edition (Discrete Mathematics and Its Applications)*. Chapman & Hall/CRC, 2006.

[40] J. Notebook, "LLTI sharings: https://colab.research.google.com/drive/1BoJSX7zgiBcbYsC-1Jv1u83RM7ExbEY2?usp=sharing."

[41] D. M. Gordon, "Covering designs, https://www.dmgordon.org/cover/."

[42] NANGATE, "The NanGate 45nm Open Cell Library," available at https://github.com/The-OpenROAD-Project/OpenROAD-flow-scripts/tree/master/flow/platforms/nangate45.

[43] G. Goodwill, B. Jun, J. Jaffe, and P. Rohatgi, "A testing methodology for side-channel resistance validation," September 2011.

[44] V. Arribas, S. Nikova, and V. Rijmen, "Vermi: Verification tool for masked implementations," in *ICECS*. IEEE, 2018.

[45] D. Bozilov, M. Knezevic, and V. Nikov, "Optimized threshold implementations: Securing cryptographic accelerators for low-energy and low-latency applications," *IACR Cryptol. ePrint Arch.*, vol. 2018, p. 922, 2018.

[46] F. Wegener, L. De Meyer, and A. Moradi, "Spin me right round rotational symmetry for FPGA-specific AES: Extended version," *Journal of Cryptology*, 2020.

[47] F. Wegener and A. Moradi, "A first-order SCA resistant AES without fresh randomness," in *COSADE*, 2018.

[48] T. Sugawara, "3-Share Threshold Implementation of AES S-box without Fresh Randomness," *Trans. CHES*, vol. 2019(1).

[49] S. Mangard, N. Pramstaller, and E. Oswald, "Successfully attacking masked AES hardware implementations," in *CHES*, ser. Lecture Notes in Computer Science, vol. 3659. Springer, 2005, pp. 157–171.

Chapter 7

Glitch-Stopping Circuits: Hardware Secure Masking without Registers

Publication Report

- **Title:** *Glitch-Stopping Circuits: Hardware Secure Masking without Registers*
- **Authors:** Zhenda Zhang, Svetla Nikova, and Ventzislav Nikov
- **Conference:** Proceedings of the 2024 on ACM SIGSAC Conference on Computer and Communications Security (CCS) 2024, Salt Lake City, UT, USA, October 14-18, 2024
- **DOI:** https://doi.org/10.1145/3658644.3670335
- **ePrint:** ePrint/2024/891

Contributions Summary: *Principle author. I conceptualized the idea, designed the circuits, and conducted the experiments.*

Glitch-Stopping Circuits: Hardware Secure Masking without Registers

Zhenda Zhang[1], Svetla Nikova[1] and Ventzislav Nikov[2]

[1] COSIC, ESAT, KU Leuven, Belgium,
firstname.lastname@esat.kuleuven.be
[2] NXP Semiconductors, Belgium,
venci.nikov@gmail.com

Abstract. Masking is one of the most popular countermeasures to protect implementations against power and electromagnetic side channel attacks, because it offers provable security. Masking has been shown secure against d-threshold probing adversaries by Ishai et al. at CRYPTO'03, but this adversary's model doesn't consider any physical hardware defaults and thus such masking schemes were shown to be still vulnerable when implemented as hardware circuits. To addressed these limitations glitch-extended probing adversaries and correspondingly glitch-immune masking schemes have been introduced. This paper introduces glitch-stopping circuits which, when instantiated with registers, coincide with circuits protected via glitch-immune masking. Then we show that one can instantiate glitch-stopping circuits without registers by using clocked logic gates or latches. This is illustrated for both ASIC and FPGA, offering a promising alternative to conventional register-based masked implementations. Compared to the traditional register-based approach, these register-free solutions can reduce the latency to a single cycle and achieve a lower area cost. We prove and experimentally confirm that the proposed solution is as secure as the register-based one. In summary, this paper proposes a novel method to address the latency of register-based hardware masking without jeopardising their security. This method not only reduces the latency down to one clock cycle, but also improves the area cost of the implementations.

Keywords: AES · Hardware Secure Masking · Glitch-Stopping Circuits

1 Introduction

Every cryptographic implementation leaks certain information about the sensitive variables used for the computations through the so-called side-channels. These leakages can be of different nature: power consumption, electromagnetic, timing, micro-architectural, etc. In this paper we focus on leakages based on power and electromagnetic side-channels. The threat model is captured by the corresponding side-channel attacks (SCA) which are exploiting those power leakages by passively measuring the power consumption of millions of executions. Then using the collected power traces the attacker apples statistical methods, like differential power analysis [KJJ99], to recover the sensitive variables (e.g., the AES key) using divide and conquer revealing the key byte per byte. SCA can be very harmful if there is no proper countermeasure implemented. The known countermeasures fall into the following two categories, which however, can be jointly implemented:

- *Leakage hiding* tries to either remove the variations caused by the computations, or to create artificial noise in the circuit. The means to achieve hiding are shielding, power balancing, noise makers, dummy operations, dual-rail technology.

- *Noise amplification* leverages the existing noise in the given side-channels to make the measurements harder. Shuffling and masking are the two methods used to achieve noise amplification.

Through the years masking earned the reputation of being the most effective practical countermeasure. It is especially relevant because of its provable security. To prove the security of a cryptographic implementation, first the side-channel adversary's model has to be defined. The simplest model is the d-threshold probing model in which the adversary is only allowed to probe the values of d wires within the circuit [ISW03]. However, the d-probing model considers only ideal circuits and excludes any physical hardware defaults. To address this limitation, the glitch-extended probing model was first introduced by Reparaz et al. [RBN+15] at CRPYTO'15, followed by the formal definition of the robust probing model by Faust et al. [FGP+18] at CHES'18. The glitch-extended d-threshold probing model again allows the adversary to probe d wires within the circuit but in addition also all the input values of the probed wires are given to the adversary. In other words, the adversary can learn more than d values. We stress that although glitches are undesired (for security and power reasons) they are ubiquitous part of every CMOS technology. Masking with $d+1$ shares provides a successful defense against (glitch-extended) d-probing adversary if no information can be revealed for the sensitive variable for any choice of the d probes.

Early masked schemes didn't consider physical hardware defaults and Mangard et al. [MPO05] at CHES'05 showed that such schemes leak information due to glitches. It became clear that glitches in hardware designs can cause unintended re-combinations of masked values due to varying delays in the circuit, so the question was how to build resistant circuits. Mangard et al. also proposed two strategies which Lammers et al. [LMM23] named as *glitch-free circuits* and *glitch-immune circuits*.

Two main physical defaults have been identified: glitches and transitions. When a circuit stores a value in a memory element, the power consumption depends on the old (i.e., already stored value) and the new value, which overwrites the memory state. This overwriting effect is called a transition. The propagation delay across different data paths within the combinational logic is not uniform. A single gate may receive input signals that arrive at different times. As a result, the gate's output may undergo temporary changes. This process continues until the gate's input signals stabilize, after which the gate will produce a stable output. From a logical circuit perspective, these stable signals progressively propagate through the circuit from the input registers to the output registers throughout a cycle. Such short-lived effects of the signal are known as glitches.

A circuit is considered glitch-free if it experiences no internal glitches. One branch of research focused on investigating ways to eliminate glitches in circuits with the goal of achieving power equalization and thus leakage hiding. Early attempts to build glitch-free circuits have focused on dual-rail logic with pre-charging (DRP) and its extension WDDL [TV04]. All DRP circuits are usually asynchronous since, otherwise, they introduce time delay due to the pre-charging phase. Early-propagation in DRP logic, where a DRP logic's output stabilizes without all input signals being determined, causes side-channel leakages and such have been illustrated for both DRP and WDDL. Further extensions like DRPnoEE and AWDDL considered monotonic Boolean gates, but still standalone (without additional masking) those solutions were shown to be vulnerable.

In parallel to glitch-freedom, another branch of research focused on the glitch-immune circuits, namely masking (i.e., noise amplification) that maintains their security in the presence of glitches. The first glitch-immune circuit is proposed in 2006 [NRR06], where the Threshold Implementations (TI) methodology is introduced. TI outlines two principal guidelines for constructing glitch-immune masking:

- To render a circuit resistant to SCA in the presence of glitches, it is necessary to incorporate additional registers. The primary role of these additional registers is to

prevent the propagation of glitches between sub-circuits.

- Each masked logic block between two registers (starting and ending points) must adhere to the *non-completeness* requirement as defined by TI. This ensures that each such sub-circuit is glitch-immune.

Since then, many glitch-immune masking schemes have been proposed on the basis of, or expanding upon, the TI design principles, for example HOTI, CMS, DOM, HPC1, HPC2, and HPC3 [BGN+14, RBN+15, GMK16, CGLS21, KM22]. To reduce the latency overhead, GLM [GIB18] omitted share compression after each nonlinear operation but at the cost of increased intermediate share count for higher-degree S-boxes. More recently, LLTI, a low-latency masking technique based on TI, offers comparable area requirements to GLM while eliminating online randomness [AZN21]. Note that glitch-immunity refers to a circuit which achieves glitch-extended probing security. We are now ready to define glitch-stopping circuits as follows.

Definition 1. A *glitch-stopping circuit* (GSC) refers to a synchronous digital circuit that (1) incorporates glitch-immune masking schemes and (2) stops the propagation of glitches between consecutive blocks of combinational logic.

Meanwhile since DRP-like solutions standalone still exhibited data-dependent early signal propagation leading to potential leakage [MS16], research shifted to strengthen such solutions by applying glitch-immune masking in addition. The authors of [MS16] proposed the integration of WDDL and first-order TI masking. More recently LMDPL [SBHM20], self-synchronized masking (SESYM) [NGPM22] and self-timed masking [SBB+22] were proposed. Those solutions have been applied as asynchronous logic for part of a cipher implementation, while globally the cipher is a synchronous design. All these three recent solutions highlight the importance of data synchronization layers even in glitch-free designs, especially synchronizing in long combinational data paths inside such asynchronous circuits. Considering these works, which integrate DRP logic and masking, we redefine glitch-free circuits as follows.

Definition 2. A *glitch-free circuit* (GFC) refers to an asynchronous digital circuit that (1) incorporates glitch-immune masking schemes and (2) uses DRP logic and its extensions to eliminate any internal glitches.

A notable aspect of our investigation is the measurement of latency. Latency can be measured in two different ways: cycle-wise or time-wise. The choice between the ways to measure depends on the specific use cases and constraints. In scenarios where there are no limitations on clock frequency, time-wise measurement, which is the delay from the input of a circuit to its output, is preferable. However, in instances where the clock frequency is constrained, and the implementation block cannot be clocked faster than the rest of the integrated circuit (IC), cycle-wise measurement becomes more relevant. Recognizing the dual meaning of the term, we will provide latency measurements in both cycle-wise and time-wise formats, offering a comprehensive view of the circuit's performance under different conditions.

This work: Figure 1 illustrates the categories of hardware secure masking which includes GSCs as defined in Definition 1 and GFCs as defined in Definition 2. When glitch propagation is prevented by registers, we denote such circuit by `GSC_R`. Such circuits are equivalent to the conventional "glitch-immune circuits". We recall that during the last 10 years glitch-immune masking (like TI, DOM, HPC, etc.) became the only recognised way to do secure masking in hardware. Since additional registers are used for security purposes such masking schemes have the drawback of increased cost and latency. However, `GSC` can be instantiated not only with registers, and this paper focuses on building such instances

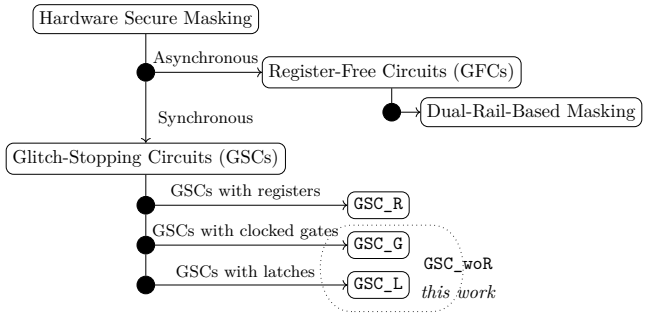


Figure 1: Categories of Hardware Secure Masking

without registers. Thus, our motivation is to keep the same level of security while we aim to reduce the cost in area and to improve the latency of secure HW implementations. In particular, we propose the usage of *clocked logic gates* or *latches* to build glitch-stopping circuits (`GSC_G` and `GSC_L`, respectively). We refer to those new solutions as `GSC_woR` and we show that they can stop glitch propagation between logic blocks as effectively as `GSC_R`, thus maintaining the security order of digital circuits against SCAs. The design and implementations for `GSC_woR` are feasible for digital logic circuits in both application-specific integrated circuits (ASIC) and field-programmable gate arrays (FPGA) environments. The key contributions of this paper are as follows:

- We introduce a novel approach to build glitch-stopping circuits that do not rely on registers `GSC_woR` while keeping the synchronous nature of the design. This method employs clocked logic gates or latches, which can stop the propagation of glitches as effectively as registers do.

- The novel `GSC_woR` require minimal changes in the clock configuration. Namely, additional clocks are added, that play a crucial role in ensuring the correct timing and synchronization of signals in the circuits. We provide complete procedure of how to make the required changes in the circuit, for both ASIC and FPGA, in a way that the security claims of the original circuit are preserved. This allows us to prove that `GSC_woR` maintain the same security as `GSC_R` under the glitch-extended probing model.

- A comprehensive comparison is provided between the proposed `GSC_woR`, the conventional `GSC_R` and state-of-the-art glitch-free circuits. This analysis covers area efficiency and latency in timing and in cycles. We show that `GSC_woR` can reduce the latency to a single cycle, reduce area and improve the overall time-wise latency.

- We performed a leakage evaluation of the proposed `GSC_woR` instantiated with four Xilinx FPGA primitives. This physical security evaluation provides empirical evidence that two out of the four primitives effectively implement glitch-stopping gates and thus achieve the first-order security.

2 Preliminaries

In this section we introduce all the notations and fundamental concepts of sequential logic circuits and their role in developing secure hardware implementations by either preventing glitches or their propagation.

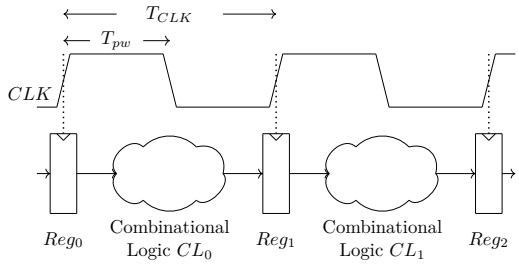


Figure 2: A synchronized circuit and its clock signal [WH10]



(a) A D-latch

(b) D-latch timing diagram

Figure 3: A D-latch that is level-sensitive to its clock signal [Wik24]

2.1 Sequential Logical Circuits

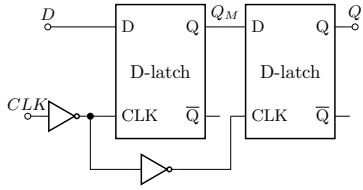
In digital design, *sequential logic* circuits provide outputs based on the current input values and the preceding input states. These preceding input states are stored in memory elements, usually referred to as *registers*. Conceptually, a register is characterized by a single input and output, governed by a clock signal. Combinational logic circuits connect the stages of registers. *Combinational logic* circuits consist of *(Boolean) gates* connected by wires carrying the signals. Most common circuits are *synchronized*, meaning a mix of combinational and sequential logic where the latter is driven by a global *clock signal*.

An example of sequential circuits is *pipelining*, as depicted in Figure 2. Pipelining is a technique aimed at optimizing resource utilization, enhancing functional *throughput*, and ultimately accelerating the operation of data paths in digital circuits. In this example, the pipeline has two stages of combinational logic, separated by registers. Every cycle, a new input is stored in Reg_0, after two cycles the output becomes ready at Reg_2. Thus, the latency for each input is two cycles. However, the circuit does not need to wait for the output to be ready before processing the next input. The throughput, quantifying the number of operations a circuit can process per cycle, is 1 bit/cycle in this example.

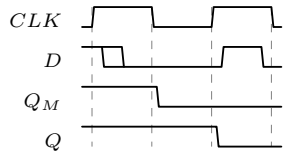
2.1.1 Registers

A digital signal has two states or voltage *levels*: the *low* state is denoted as 0, and the *high* state as 1. The clock signal is a special digital signal used to synchronize all gates in the circuit. The *rising edge*, or *positive edge*, of the clock signal is the transition from 0 to 1. Similarly, the *falling edge*, or *negative edge* of the clock signal is the transition from 1 to 0. Registers are commonly instantiated as flip-flops and sometimes as latches.

Latches: A *latch* operates as a level-sensitive memory element, while in contrast, a *flip-flop* operates as an edge-sensitive memory element. As depicted in Figure 3, when the clock signal controlling a D-latch is 0, the input is ignored, while the output is locked and maintained to the last input state. This state of the latch is referred to as the *hold*

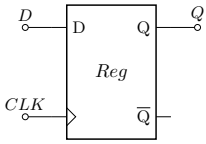


(a) A primary-secondary D-flip-flop (Register)

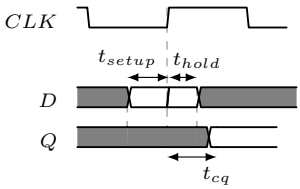


(b) D-flip-flop timing diagram with its internal value

Figure 4: A primary-secondary D-flip-flop that is triggered on the rising edge of the clock [Wik24, RCN03]



(a) A D-flip-flop (register)



(b) Timing characteristics

Figure 5: Definitions of set-up time, hold time, and propagation delay of a D-flip-flop [Wik24, RCN03]

mode since it effectively stores the data. When the clock signal becomes 1, the input propagates and passes continuously to the output. This is the *transparent* mode, which ensures real-time tracking of the input signal. The hold mode makes latches an essential building block for maintaining data in digital logic circuits, but any glitch on the input will propagate to the output when the clock signal is high, hence latches should be in transparent mode only when the input signal is stable.

Flip-Flops: D-flip-flops (FF) operate by sampling the input data typically at the rising edge of the clock. They can also be triggered on the falling edge, though such implementations are less common. A flip-flop is normally built based on the *primary-secondary*, also known as *master-slave*, configuration of two cascaded latches, as shown in Figure 4.

When CLK is 1 and before its falling edge, the primary D-latch is in hold mode and the secondary D-latch operates in transparent mode. The primary latch ignores any new input and holds its last input data constant at its output Q_M. The secondary latch, now transparent, receives the data Q_M and transfers it to its output Q, allowing the data to proceed to the combinational logic following the FF. Conversely, when CLK is 0 and before its rising edge, the primary latch accepts data from the D input and transfers it to its output Q_M. Meanwhile, the secondary latch is in hold mode, maintaining a stable output of its previously stored value. Consequently, the FF changes its output Q when the rising edge of the clock signal occurs. Unlike latches, either one of the FF's two latches always blocks any glitch inputs, preventing them from propagating to the FF's output.

2.1.2 Clock Signal Characteristics and Timing Constraints in Synchronous Circuits

The clock controls the updating of the sequential memory elements within an IC or a chip. As illustrated in Figure 2, a synchronous circuit is synchronized by a clock signal that is connected to all registers. The *clock period* (T_{CLK}) is the period between two rising clock edges. In other words, this is the sum of the rise time, high time, fall time, and low time

of the clock signal, as depicted on Figure 2. Within this clock period, or during one *cycle*, stable signal propagation through the circuits must be ensured, setting a minimum limit for T_{CLK}. The *frequency* of a synchronous circuit is defined as $1/T_{CLK}$, and consequently, it possesses a maximum limit. It is important to note that in complex system-on-chips (SoCs), different clock frequencies may coexist, being distributed to various parts of the integrated circuit.

Duty Cycle: *Pulse width* (T_{pw}) of a signal is the duration from its rising edge to its falling edge. The ratio of the positive pulse width T_{pw} and the total clock period T_{CLK} is defined as the *duty cycle* of the clock. Expressed as a percentage, the duty cycle is given by $D = T_{pw}/T_{CLK} \times 100\%$. Clock signals in modern circuit designs typically have a 50% duty cycle, although this is not always the case. Advanced electronic design automation tools (EDA) facilitate modifications of the duty cycle.

Timing Considerations of a flip-flop: The timing metrics of a flip-flop are critical for correctly registering an input to its output, as illustrated in Figure 5. The *setup time* is the interval during which the input data D must remain stable before the clock's rising edge. This period typically corresponds to the time required for the input signal to propagate through the primary latch. The *hold time* represents the duration that D cannot change for the data to transition to the secondary latch and the FF output. Propagation delay t_{cq} is the time the signal takes for the input change to be reflected at the output after the clock edge.

If the setup or hold time requirements are not met, the output of the FF is unpredictable and may even be unstable: the output may operate normally, take an invalid level, or oscillate. Therefore, setup and hold times define the uncertainty time windows in which the FF output is not deterministic. That is why in synchronous digital design clock periods are chosen such that no meta-stability of the FFs ever happens. In synchronous circuit design, the rising edge of the clock signal indicates that all signals have reached a valid and stable state. This is guaranteed by setting the clock period, during the design time, to the worst-case delay between two FFs. Usually, the timing requirements are controlled and ensured via the EDA tools.

The global clock signal has a tree structure from a single source and thus it is glitch-free. Note that if there are glitches in the clock signal then flip-flops might become meta-stable and hence the glitches on the data signals might not be stopped by a flip-flop. We will further consider the depicted in Figure 2 circuit to be a circuit without a side-channel attack resistance.

2.2 SCA Resistant Circuits

In this section, we consider the leakage model that captures the effect of glitches and we discuss the state-of-the-art for SCA resistant circuits.

Let's recall that a *glitch-extended probe* on a wire z models the impact of glitches by capturing all stable signals that contribute to the wire z. In other words, such a probe collects all flip-flop values leading to the wire z. Similarly, the *transition-extended probe* on a register w models the impact of transitions by recording two consecutive signals stored in w. In the rest of the paper, the main concept we will use is the glitch-extended probing model. In this model, it is assumed that an attacker can instantiate a glitch-extended probe from a wire z in the circuit and trace the wire values backwards to all registers which store signals contributing to z. However, tracing back beyond the register (into the preceding circuit) is not feasible as glitches are stopped by the register. Since both the glitch-free and the glitch-immune circuits include glitch-immune masking the glitch-extended probing model offers a framework to establish security proofs for both of them.

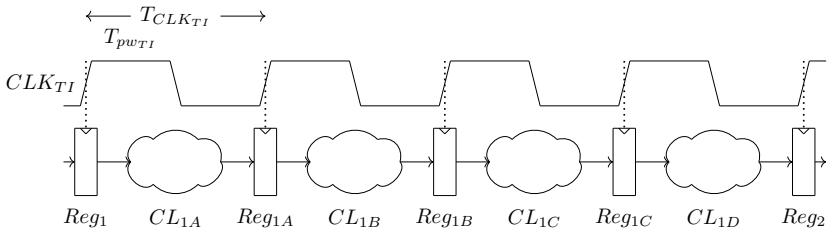


Figure 6: Glitch-stopping circuit with four-stages TI

2.2.1 Glitch-Immune Circuits

The authors of TI [NRR06] were the first to provide first-order protection against side-channel attacks in the presence of glitches. The TI methodology involves segmenting the combinational logic into smaller sub-circuits, as illustrated in Figure 6. Moreover, TI mandates non-completeness for each masked sub-circuit or gadget and requires that the masking must be both correct and uniform to guarantee first-order composability. Subsequently, the notion of *composability* was introduced [BBD+16]. Namely, a secure gadget is composable if and only if all possible combinations of such gadgets maintain probing security under the same security model. Various notions of composability, including NI, SNI, and PINI, have been introduced [BBD+16, CS20]. Secure masking schemes in hardware have also evolved to incorporate composability, resulting in the development of schemes such as HPC1, HPC2, and HPC3. We will not explore the specifics of these well-established and understood methods.

Figure 6 illustrates an example of a glitch-stopping circuit. For illustrative purposes, let's assume that the second combinatorial logic circuit CL_1 from Figure 2 is made side-channel protected by dividing it into four stages CL_{1A-1D} and applying first-order TI masking. The inserted additional registers expand the one-stage unprotected logic to four stages of glitch-stopping circuits, thus increasing from one cycle latency with T_{CLK} to four cycles with $T_{CLK_{TI}}$.

2.2.2 Glitch-Free Circuits

Dual-rail logic encodes each value {1} or {0} to a tuple {1,0} or {0,1} respectively, while {0,0} is known as empty value and {1,1} being not allowed. It is not allowed to have a transition from one valid encoding to another unless {0,0} is inserted in between. This return-to-zero is achieved by forcing a reset to the circuit i.e., pre-charging. The DRP logic e.g., WDDL [TV04], operates in two phases: a pre-charge phase setting all wires to an empty value, followed by an evaluation phase processing valid inputs to yield an output. WDDL is designed for power equalization and to reduce SCA leakages. However, WDDL's limitations in leaking secret information by unmatched wire delays and high routing uncertainty between rails led to its gradual replacement by more flexible glitch-immune masking techniques.

It has been noted that countermeasures relying solely on asynchronous logic are inadequate without masking, which led to the exploration of a hybrid approach. The research in [MS16] aimed to determine whether integrating a glitch-free WDDL-based circuit with TI, but without using registers, could achieve a design secure against SCA. The authors pointed out that "although placing registers between the shared non-linear functions was initially introduced to avoid the propagation of glitches, it also synchronizes the start of their evaluation to be independent of the timing of the previous stage." In their asynchronous circuit, the authors employed registers to isolate shared non-linear functions. However, as [MS16] reveals, this approach results in a design significantly larger than its

synchronous counterpart and faces latency issues due to the interleaved pre-charge and evaluation phases, ultimately compromising some advantages of asynchronous design due to the synchronization constraints.

The primary goal of the most recent asynchronous glitch-free designs is to achieve low latency, which means to reduce the number of clock cycles required for computing the entire masked function. However, these designs remain globally synchronous while being locally asynchronous so the efficiency improvement is focused on the masking of S-boxes in ciphers. One such approach is the LMDPL technique [SBHM20], which combines look-up table based first-order masking on monotonic gates with DRP logic. While this method has been shown to ensure first-order security, its performance and area overhead in comparison to similar designs without LMDPL are not clearly established due to the lack of comparative data in the same CMOS technology. Additionally, Nagpal et al. [NGPM22] proposed the SESYM technique. This method utilizes WDDL along with the Muller C-elements for synchronization, ensuring security across different critical components like S-boxes without the need for a fully dual-rail data path. Another approach is called self-timed masking [SBB+22], which modifies monotonic Boolean functions i.e., dual-rail OR and AND gates, and aims at eliminating the early-propagation. Data synchronization in [SBB+22] is done in similar way as in [NGPM22] using the Muller C-element. To our best understanding, the main difference between [SBB+22] and [NGPM22] is the use of AWDDL instead of WDDL logic.

We stress here that the recently proposed GFCs, that incorporate glitch-immune masking with DRPs, have demonstrated a notable resilience against SCAs, even at one order higher than the masking order. This enhanced security is primarily attributed to the dual-rail power-equalization effect.

3 Case Study: Threshold Implementations on AES

In this section, we present two different implementations of the AES S-box with serialized architecture of `AES-128` encryption. Let t be the algebraic degree of the corresponding Boolean function. The first implementation uses first-order $t+2$ TI sharing with five shares of the S-box decomposed in two stages. The second one uses first-order $t+1$ TI sharing (with three shares) of the S-box decomposed in four stages. We will apply those 2 implementations in Section 4 and Section 5.

3.1 Serialized AES Encryption

The `AES-128` encryption is implemented using a serialized architecture with a one-byte-width data path, incorporating an AES S-box instance as described in either Section 3.2 or Section 3.3. Due to the pipelining of the S-box, proper scheduling of operations is essential. Data is stored in s shares, where s equals either 5 or 3. The architecture comprises two state arrays of size $s \times 16$ bytes each, one for data and the other for the key. Data and key bytes in s shares move unidirectionally from one functional unit to the next, with their results being stored in the respective state registers after each clock cycle. The functional units consist of linear layers and the S-box, with registers in the latter capturing intermediate results due to pipelining.

In the AES round control flow, the ShiftRows operation is executed concurrently with the S-Box output processing. The output of the S-Box is written to the state arrays in which it would be stored after performing the ShiftRows operation. In other words, each column is written back diagonally with rotational wrapping around the state matrix. The MixColumns operation is performed as soon as all column bytes are ready. The result of MixColumns is ready in the following cycle. The key addition is performed in the next

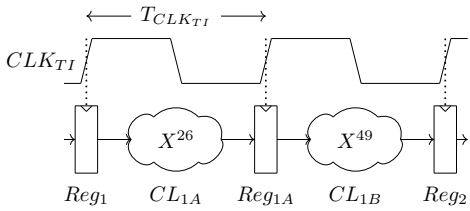


Figure 7: Representation of the two-stage masked AES S-box with five shares.

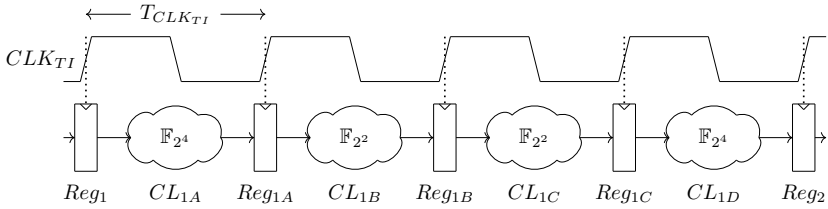


Figure 8: Representation of the four-stage masked AES S-box with three shares.

round iteration, while the final key addition is computed during the read-out operation. For the considered two pipelined designs an AES round fits into 20 cycles [BKN20].

3.2 Two-Stage AES S-box with Five Shares

The implementation of a two-stage AES S-box involves decomposing the inversion in $GF(2^8)$ of algebraic degree seven, into two cubic functions: namely $x^{-1} = x^{254} = (x^{26})^{49}$ [NNR19]. This decomposition facilitates a pipelined two-stage implementation with an interposed single glitch-stopping layer (register). First-order TI with $t+2$ shares [PAB+22] is applied to each of the cubic functions x^{26} and x^{49}. This strategy not only simplifies the design but also saves randomness cost, since the sharing is uniform no additional randomness is required. The architecture of the S-box is shown in Figure 7.

3.3 Four-Stage AES S-box with Three Shares

Canright [Can05] proposed an efficient tower field decomposition of the AES S-box in order to improve hardware costs. Since then this decomposition was widely used to create efficient hardware masking of AES. The tower-field decomposition of the S-box consists of the linear input/output isomorphism; and the nonlinear finite field multipliers, finite field scaling functions, and a multiplicative inversion. The tower-filed computes the inversion by starting from $GF(2^8)$ via $GF(2^4)$ to $GF(2^2)$ each time simplifying it until it becomes a linear operation. The implementation we will use is a modified version of the Design II by Askeland et al. [ADN+22] but with four stages instead. The shared linear stage from the input is merged into the first non-linear stage which results in one stage less than the Design II. The S-box uses a total of 36 random bits since we do not use the changing of the guards method. The architecture of the S-box is shown in Figure 8.

3.4 Timing Overhead of Glitch-Immune Masking Schemes

Revisiting the timing requirements, we compare the total delay of an unprotected circuit denoted as T_{CLK} to the time delay of a `GSC_R` protected circuit, represented by $T_{TI} =$ Number-of-stages $\times\, T_{CLK_{TI}}$. For the case of $t+2$ TI and a cubic function (Section 3.2) the timing overhead of `GSC_R` compared to unprotected cubic function is the time required

for the evaluation of $3 + 2 + 1 = 6$ XORs one after another. When applied to the shared AES S-box with two stages we obtain that the time delay compared to un-shared S-box is the time to pass through $2 \times 6 = 12$ XORs and 1 Register. Note, that when we talk about timing overhead in this section, we will refer to the number of gates but we will mean the time required for their consecutive evaluation.

Let us consider now masked Canright S-box and $t + 1$ TI on quadratic functions (Section 3.3). Taking into account that the overhead of finite field masked-multiplication is $1 + 1 = 2$ XORs, the overhead for the shared AES S-box (Canright with four stages) is $4 \times 2 = 8$ XORs and 3 Registers.

We also can consider masking with $d + 1$ shares (e.g., CMS, DOM) for quadratic functions. In this case the overhead of masked-multiplication is: $log_2(d + 1) \times (1 + 1)$ XORs and 1 Register. Hence, the overhead of a masked AES S-box (Canright with 5 stages) will become $log_2(d + 1) \times 4 \times 2$ XORs and 4 Registers. Similar calculations can be made for AES S-box implemented via Boyar-Peralta [BP11] representation which has the multiplicative depth of 4.

The general, the timing observation for `GSC_R` masked implementations is that the overhead comes from: a) the number of stages (and the required registers); b) the number of XORs used for re-sharing; c) the number of XORs used to compress the expanded number of sharings after multiplication.

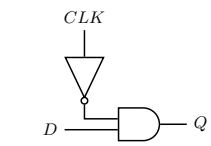
4 Glitch-Stopping Circuits in ASIC

In this section, we introduce glitch-stopping circuits without registers (`GSC_woR`) in contrast with the traditional approach with registers (`GSC_R`). We will show that glitch-stopping circuits can be built with different clocked combinational logic blocks, i.e. `MUX`-gate, `AND`-gate (`GSC_G`) and latch (`GSC_L`). This results in reduced latency and area. We will prove that the proposed `GSC_woR` provides security equivalent to `GSC_R` in the glitch-extended probing model. Different configurations of the clock signals can be used for `GSC_G` and `GSC_L`. We will use both high and low clock signal levels as well as rising and falling clock edges in our descriptions further. However, in cases when it is not advisable to use the falling clock edge then the described solutions can easily be adapted to only use the clock rising edge.

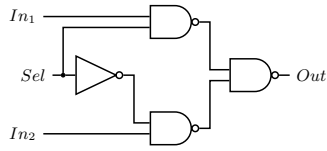
4.1 Clocked `AND`-Gate

Consider a device with `GSC_R` implementation as depicted in Figure 6. This implementation is either represented in hardware description language on register-transfer level (RTL) or gate-level description netlist. Now we preserve all the four sub-circuits $CL_{1\{A-D\}}$ and their critical paths. We replace each register inside the pipeline $Reg_{1\{A-C\}}$ with clocked logic `AND` gate, as depicted in Figure 9a. The `AND`-gate is placed between the data (in/out) signals and the inverted clock signal is used for control. Note that the term *clocked-gate* should not be confused with *clock gating*. Clock gating is a power-saving technique that disables clock signals in a synchronous logic module to reduce dynamic power dissipation.

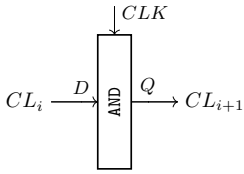
To ensure the same security as `GSC_R`, it is crucial to maintain the timing as shown in Figure 10, where $T_{CLK_{TI}}$ is included for reference (as originally given in Figure 6) to demonstrate the preserved critical path for $CL_{1\{A-D\}}$. Proper synchronization is achieved by introducing two additional clocks with the same period as the main clock $T_{CLK_{Reg}} = T_{CLK_B} = T_{CLK_A} = T_{CLK_C}$, yet with varying duty cycles. Specifically, D_{CLK_A} is set at 25%, D_{CLK_C} at 75%, and $D_{CLK_B} = D_{CLK_{Reg}}$ remains consistent with the global clock at 50%. To be able to refer to the sub-circuit evaluation in each of the four circuit stages we introduce the notion of "virtual cycle" as an analogue CLK_{TI}.



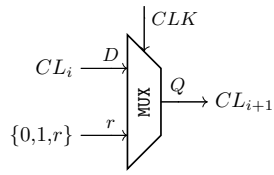
(a) A clocked `AND`-Gate



(b) A clocked `MUX`-Gate



(c) Integration of the `AND`-Gate into `GSC`



(d) Integration of the `MUX`-Gate into `GSC`

Figure 9: `AND` and `MUX` instances of the glitch-stopping circuits

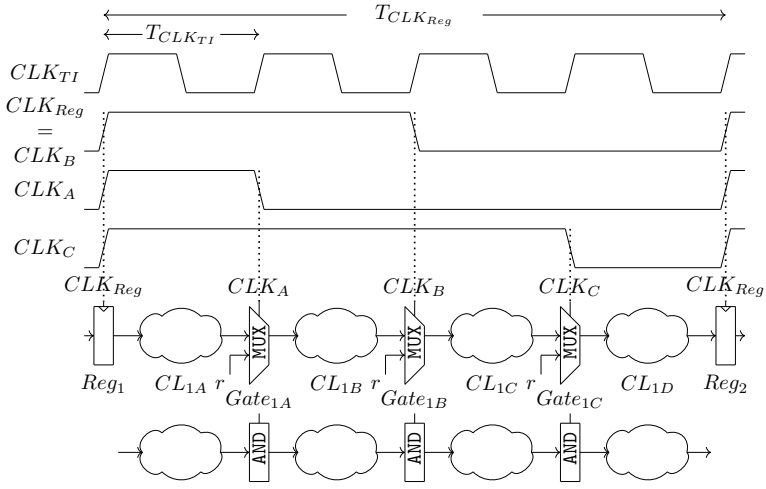


Figure 10: Clocked logic-gate-based glitch-stopping circuit

The three clock signals, CLK_A, CLK_B, CLK_C, manage signal flow through the stages $CL_{1\{A-D\}}$ in the `GSC_G`, ensuring only stable signals passing and stopping glitch propagation. In the first virtual cycle, all three clock signals are high, ending with CLK_A's falling edge. In other words, the positive pulse width $T_{pw_CLK_A}$ align with CLK_{TI}'s first clock period. Immediately the second virtual cycle starts with CLK_A low, while CLK_B and CLK_C remain high. It ends with CLK_B's falling edge, accordingly, $T_{pw_CLK_B} - T_{pw_CLK_A}$ matches with the second clock period of CLK_{TI}. The third virtual cycle has CLK_A and CLK_B low, CLK_C high, and finishes with CLK_C's falling edge. Thus $T_{pw_CLK_C} - T_{pw_CLK_B}$ aligns with CLK_{TI}'s third period. The final fourth cycle occurs with all clocks low, ending with their simultaneous rising edge, thus matching CLK_{TI}'s fourth clock period and corresponding to $T_{CLK_{Reg}} - T_{pw_CLK_C}$.

During the first virtual cycle in the clocked `AND`-gate `GSC_G` design, $Gate_A$ is closed, preventing signal propagation from CL_{1A} to CL_{1B}. $Gate_A$ output is stabilizing at zero. During this virtual cycle also $Gate_B$ and $Gate_C$ are closed, maintaining zero at their outputs. CL_{1A} is evaluated, while circuits CL_{1B-D} receive zero inputs. By design the virtual cycle finishes when CL_{1A}'s signals stabilize, and thus when $Gate_A$ opens at the beginning of the second virtual cycle the signals without glitches pass between CL_{1A} and CL_{1B}. In the second virtual cycle, $Gate_B$ and $Gate_C$ stay closed, sustaining their zero outputs hence $CL_{1\{C,D\}}$ remain unchanged. $Gate_A$ is open allowing CL_{1B}'s evaluation while CL_{1A} remains stable. Again by design, this virtual cycle ends when CL_{1B} stabilizes, leading to the opening of $Gate_B$ and the start of the third virtual cycle. In this third virtual cycle only $Gate_C$ remains closed, keeping its output zero. CL_{1C} undergoes evaluation while $CL_{1\{A,B\}}$ maintain their stable states. This virtual cycle ends with the stabilization of CL_{1C}'s signals, initiating the fourth and final virtual cycle in which CL_{1D} is evaluated. During this period, $CL_{1\{A-C\}}$ remain stable, and Reg_2 at the circuit's end halts further signal propagation. Thus, this strategy ensures stable output values at each stage circuit when the negative edge of the corresponding clock signal occurs.

4.2 Clocked Latch

As described in Section 2.1.1, registers, or flip-flops, consist of cascaded latches connected by the inverted clock signal. We show three ways to configure the clock signals for a glitch-stopping circuit with latches `GSC_L`. The first approach repeats the clock signals of the `GSC_G`. The second approach, depicted in Figure 11, differs from the first by introducing two inverted clocks CLK_P and CLK_N. The third, simpler method, requires only the CLK_N clock but both rising and falling edges of the clock are utilized to control the *Gates*. Notably, inverting the clock signal is not required when using the rising edge.

The `GSC_L`, while structurally similar to the `GCS_G`, diverges significantly in the way circuits $CL_{1\{A-D\}}$ are evaluated due to the inherent ability of the latch to retain its previous value. In the first virtual cycle, where CLK_A is high, the $Gate_A$ effectively halts any glitch progression between $CL_{1\{A,B\}}$. For the clocked `AND`-gate scenario, this results in $CL_{1\{B-D\}}$ having stable, zero-valued inputs. Conversely, in a latch-based setup, these circuits maintain stability by retaining their original data values.

4.3 Clocked `MUX`-Gate

The clocked `MUX`-gate solution is similar to the clocked `AND`-gate approach, with the primary distinction being the connection of the clock signal to the select input *sel* of the `MUX`, as shown in Figure 9b. This setup uses the same additional clocks as the `AND`-gate solution. The key difference lies in the use of a 2-to-1 `MUX` where, apart from the data input, the secondary input is variable. Setting this secondary input to a constant 0 repeats the `AND`-gate solution by zeroizing unprocessed circuits. Alternatively, a constant 1 input results in an identical operation with stable inputs 1. A more sophisticated option involves connecting

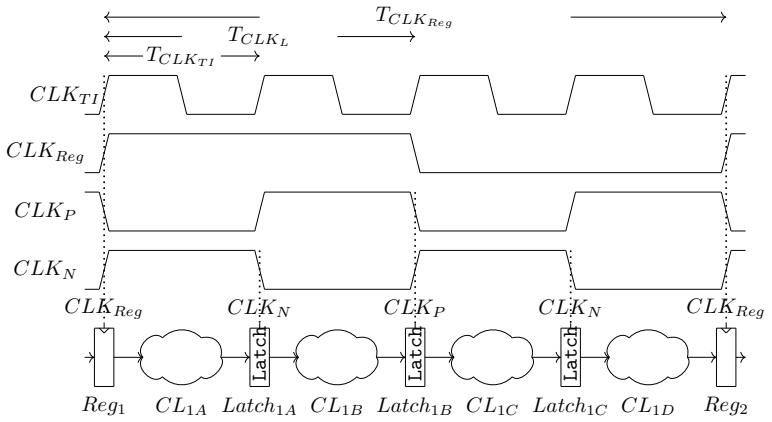


Figure 11: Clocked latch-based glitch-stopping circuit

the secondary input to a randomness source, thereby randomizing the un-evaluated circuits. This last choice adds noise and may result in a more robust solution.

4.4 Comparison of the Glitch-Stopping Circuits

In this section, we compare the four solutions: `GSC_R` given in Figure 6 and the 3 new `GSC_woR` given in Figures 10 and 11. We weigh up their glitch-stopping mechanism and their costs on timing and area.

For the new `GSC_woR` solutions we had to introduce additional clock signals. For example, in the clocked `AND`-gate construction of the four-stage masked circuit, two additional clock signals CLK_A and CLK_C are required. Instead of generating those clocks, a technique for detecting the clock glitches can also be used, as detailed in the recent overview by Askeland et al. [ANN23]. Namely, we will use in a different context the so-called parallel delay line circuit (PDL), as illustrated in Figure 12. The PDL derives from the main clock two lines of circuits, i.e. a fast and a delayed line, which are XOR-ed at the end of the path. We can use the output of the PDL (the XOR-ed value) as the trigger for the clocked-gate instead of the clock falling (or rising) edge as in Figures 10 and 11. The delay lines are using as many buffers (or back-to-back inverters) as needed to achieve the right timing until the moment comes when we need the trigger's edge. However, it might be not so straightforward to achieve the right timing, as indicated by [NT22]. When implemented in ASIC such delay lines have to be tunable in order to mitigate the dynamic timing variations caused by changes of the process, voltage, temperature and ageing. Without proper calibration in silicon, false positives may occur which in our case will result in glitches propagating via the clocked-gates. Such a glitch propagation violates the basic glitch-stopping assumption and might cause a leakage. Since from practical point of view such a calibration process requires additional efforts, we opted it out and favoured a solution with additional clocks.

Apart from the triggering/clocking mechanism the new solutions – `GSC_G` and `GSC_L` – differ from traditional `GSC_R` in their circuit evaluation process. In the pipelined register-based approach, all four circuits $CL_{1\{A-D\}}$ are evaluated simultaneously at each CLK_{TI} cycle. In contrast, in the new solutions, only one stage circuit (out of the four) will be evaluated at the virtual cycle corresponding to the CLK_{TI}. Circuits before the one currently being evaluated are preserved with their already computed values, while the subsequent circuits use predefined input values. For instance, in the clocked `AND`-gate solution, all circuits except for CL_{1A} are zeroized during the first virtual cycle and remain stable until their turn for evaluation arrives. In all four `GSC` solutions, glitches can occur

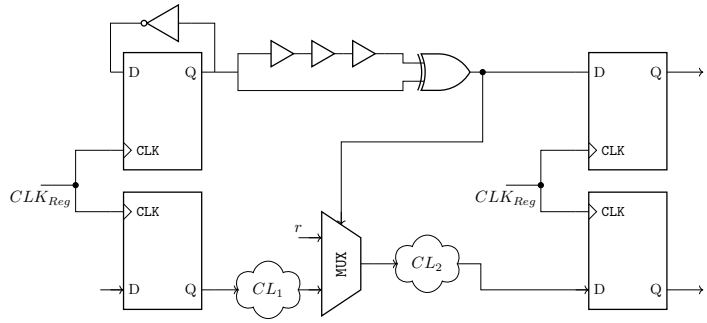


Figure 12: Use PDL-1 [ANN24] to generate trigger signal for the clocked-gate in two-stage masked circuit `GSC_woR`

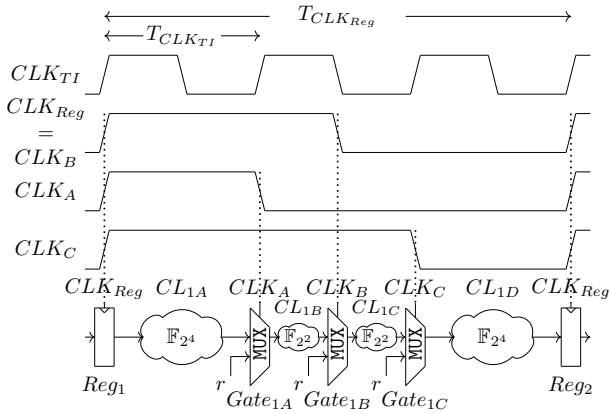


Figure 13: Clocked logic-gate-based glitch-stopping circuit with optimized virtual cycles.

during the evaluation of a sub-circuit within its designated (virtual) cycle. However, these sub-circuits are designed to be glitch-immune, adhering to the non-completeness property. This holds because `GSC_woR` preserved the original `GSC_R` masking scheme properties and in particular the non-completeness.

When comparing various glitch-stopping gate solutions as shown in Figures 3, 4, 9a and 9b, the `AND`-gate emerges as the simplest option with a cost of one AND and one NOT, with a delay on the data line of one AND. Next in simplicity is the `MUX`-gate with a cost of three NANDs and one NOT, and delay on the data line of two NANDs. The latch-gate, slightly more complex, requires four NANDs and two NOTs, also with a delay of two NANDs. Comparatively, the register (flip-flop) solution is more costly, doubling the latch cost while maintaining a similar signal delay.

All three new solutions offer better area efficiency than the register-based approach. In terms of cycle-wise latency, all these new solutions require only one single cycle for evaluation compared to four cycles in traditional register-based designs. However, it is important to note that the advantage in cycle reduction is mainly applicable to round-based architectures, as data pipelining is not feasible with the new solutions.

Last but not least, there are potential optimizations that reduce the complexity of the clock configuration and the time-wise latency. Firstly, time-wise latency can potentially be reduced in these new solutions by relaxing setup and hold times and thus slightly shortening the gating-related clocks. However, optimizing different critical paths per

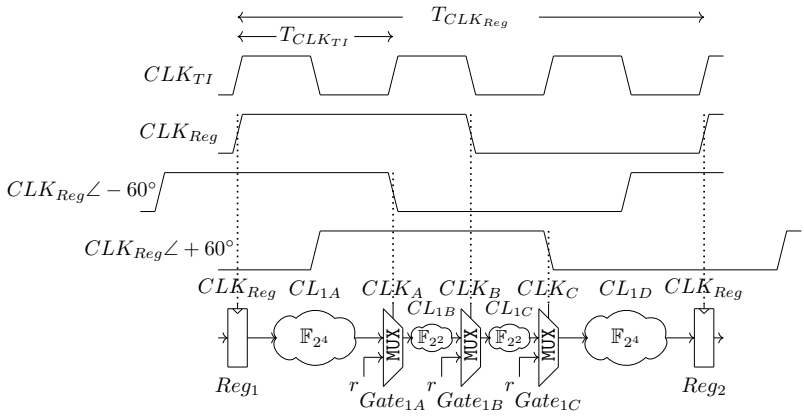


Figure 14: Clocked logic-gate-based glitch-stopping circuit with clocks with phase-shift.

stage instead of a single maximum critical path can further reduce time-wise latency as illustrated in Figure 13. In this example we assume that the sub-circuits in the first and fourth stages have 2 times longer critical paths than those sub-circuits in stages two and three. This is a plausible assumption given the fact that the first 2 sub-circuits contain multiplication over $GF(2^4)$ and affine operations compared to the later 2 circuits which contain multiplication over $GF(2^2)$. Therefore we can optimize the virtual cycles to fit the sub-circuits' critical paths and in that way the whole computation will be $3 * T_{CLK_{TI}}$ instead of $4 * T_{CLK_{TI}}$. Secondly, clocks with the same frequency can be used with phase shift, as shown in Figure 14. The phase shift can be achieved by either using a delay line or a phase-locked loop (PLL) circuit, which is less complex than clocks with different duty cycles.

4.5 Security in the Glitch-Extended Probing Model

In this section, Procedure 4.1 outlines the essential steps for designers to ensure the secure implementation of glitch-stopping circuits. Following this procedure, `GSC_woR` circuit gW is `derived` from `GSC_R` circuit gR. The steps described are critical to guarantee that glitches are effectively stopped across various stages. Subsequently, the security of the `GSC` hinges on the correct application of the masking techniques used. For example, Step 3 mandates the implementation of delays in the direct wires that pass through a combinational logic block from its input to the output. The aim is to mitigate (as an example) the risk that the time, required for the rising edge of CLK_A to reach $Gate_{1A}$ from Reg_1, is bigger than the time needed for the data propagating on a direct wire between them. Adding buffers (or back-to-back inverters) on such direct wire resolves this.

Our methodology for circuit `derivation` remains independent of the used security notion in the glitch extended probing model, such as TI, NI, SNI, or PINI. For simplicity, we collectively refer to these as *α-secure* `GSC`, where α represents any of these notions. Recall that a glitch-extended probe on a wire z models the impact of glitches by capturing all stable signals that contribute to the wire z. For the conventional `GSC_R` case those are flip-flop values, however for the `GSC_woR` cases such an extended probe captures the values of the clocked-gates instead. We are now ready to state our main result.

Theorem 1. *If a gadget gR is glitch-extended d-order α-secure `GSC_R`, then the `derived` gadget gW is also glitch-extended d-order α-secure `GSC_woR` (i.e., with clocked `AND`-gate, `MUX`-gate, and latch).*

Step 1. Start with a secure register-based implementation `GSC_R` named gR.

Optional. Perform leakage assessment tests on gR.

Step 2. Find the register layers and the critical path of gR.

Optional. Calculate the critical path for each combinational logic stages of gR, if the time-wise latency is to be optimised.

Step 3. Preserve the combinational logic of gR and insert delays on the direct wires.

Step 4. Add corresponding clocks for `GSC_L` or `GSC_G` as shown in Figures 10 and 11. Reduce the number of clocks whenever applicable.

Optional. Optimize the clocks if the time-wise latency is the goal.

Step 5. Replace the registers with the chosen type of glitch-stopping latches or gates for `GSC_L` or `GSC_G` respectively in the RTL or netlist level.

Step 6. Connect the gates with the correspondingly introduced new clocks and obtain `GSC_woR` named gW.

Step 7. Check the behavioural and timing correctness of gW.

Optional. Perform leakage assessment tests on gW.

Procedure 4.1: `GSC` design principles

If a composition of gadgets gR is glitch-extended d-order α-secure `GSC_R`, then the `derived` composition of gadgets gW is also glitch-extended d-order α-secure `GSC_woR`.

Proof. Consider a `GSC_woR` circuit gW which is built following the design principles given in Procedure 4.1 starting from a `GSC_R` circuit gR, in other words, the circuit gW is `derived` from the circuit gR. Such a masked circuit gW has clocked-gates instead of registers while it preserves the combinational sub-circuits and their timing conditions exactly as in gR. This guarantees that in gW no glitches propagate between two sequential sub-circuits via the clocked-gates exactly as this is the case for the gR circuit. At the beginning and the end of each virtual cycle all the sub-circuits in gW have stable signals exactly as for gR.

Let us now consider the extended probing model with glitches and compare the information received by the attacker observing both gW and gR. As explained above, gW has no pipelining and the sub-circuits are evaluated consecutively while in gR the pipelining is used and all sub-circuits are evaluated at the same cycle. The sub-circuits in gW that have been evaluated and those which are not yet in evaluation keep stable signals. A glitch-extended probe in a particular virtual cycle retrieves the same information for all sub-circuits which have already been evaluated and the one under evaluation in circuits gW and gR. The sub-circuits in gW which are not yet evaluated have no analogue in gR, however, those sub-circuits are evaluated with constant or random inputs which are independent of any data (sensitive or not) used in the gR circuit. Therefore, a glitch-extended probe over such (not yet evaluated) sub-circuit reveals no information to the attacker.

For both circuits gW and gR, the adversary has the same information and hence, the adversary advantage for both designs will be the same. So, we can conclude that if gR is a glitch-extended d-order α-secure gadget then gW is also a glitch-extended d-order α-secure gadget. Similarly, if gR is a composition of glitch-extended d-order α-secure gadgets then

gW is a composition of glitch-extended d-order α-secure gadgets. This concludes the proof. □

It has been noted that early signal propagation is the main security threat to asynchronous designs, particularly those based on DRP logic. In these glitch-free circuits, the timing information for the initial signal propagation is enough to reveal the secret. However, glitches are present in `GSC_R`, early signal propagations are not only related to the secret value and have not been shown to cause leakage. Thus `GSC_woR` are not vulnerable to early signal propagation similar to the `GSC_R`.

At the end of this section we stress that there are no layout-dependent constrains which a designer has to take into account. Procedure 4.1 (step. 2,3,4) demonstrates that the circuits are preserved and hence their time-wise behavior is preserved as well. The Place-and-Route procedure will make sure that the timing constrains are met and is not going to optimize any of the "gates" since this is prevented by the typical for masking "don't touch" instances.

5 Glitch-Stopping Circuits in FPGA

In this section, we describe how the three new solutions for ASIC presented in Section 4 can be implemented on FPGA taking into account the FPGA specifics. Unlike ASICs which are designed for a specific purpose and manufactured for that use case, FPGAs can be reprogrammed to perform a wide variety of tasks. However, this versatility comes with limitations and high complexity of placement and routing. Although the FPGA primitives can perform the same logical functions as the corresponding ASIC designs, their implementation is more complex. Indeed, the designer has to use the provided FPGA native primitives which might not entirely fit for the glitch-stopping purpose, for example, be more glitchy because they have more inputs. Compared to the register-based implementation of `GSC` which is on algorithmic RTL level, the `GSC` without registers requires deeper intrusion like preserving circuits time, taking care of the additional clocks, using less standard clocked-gates instead of registers. Some of these changes have to be performed on netlist level.

5.1 Primitives in FPGAs for Glitch-Stopping Circuit

FPGAs consist of Configurable Logic Blocks (CLBs) linked by a reconfigurable hierarchy of customizable interconnects. These CLBs can be customized to execute complex combinational functions typically utilizing Look-Up Table (LUT) circuits, flip-flops, and various logic components. This setup enables the implementation of both combinational and sequential digital circuits. Each CLB interfaces with a switch matrix, linking it to a broader routing matrix for flexible connectivity with other CLBs and I/O blocks within the FPGA.

Since we implement and evaluate our designs in the Xilinx Spartan-6 FPGAs on the Sakura-G board, we first introduce the instances of the required primitives in the Spartan-6 FPGAs. These primitives are also present in newer Xilinx FPGA models, accessible through identical instantiation methods. For Intel FPGAs, corresponding primitives adhering to the specifications of these FPGA models should be selected based on their glitch-stopping characteristics.

Look-up table and multiplexers in Spartan-6 Spartan-6 FPGAs utilize six-input Look-Up Table `LUT` circuits as function generators. Each `LUT` features six independent inputs, labelled `A1` to `A6`, and two independent outputs, `O5` and `O6`. These `LUT`s can execute any Boolean function with six inputs, including the `AND`-gate. Additionally, `SLICEL` and

SLICEM slices in Spartan-6 FPGAs incorporate three 2-to-1 multiplexers, named MUXF7, along with the basic LUTs. These multiplexers combine function generators, allowing for up to eight-input functions within a slice when integrated with LUTs. The MUXF7, a straightforward 2-to-1 multiplexer, receives one of its inputs from the LUT output O6 and can be driven by any internal net, making it suitable for use as a glitch-stopping MUX-gate

Latches and storage elements in Spartan-6 Each slice within the Spartan-6 FPGA structure incorporates also eight storage elements. Out of them, four storage elements can be configured either as edge-triggered FFs or as level-sensitive latches. Spartan-6 devices also feature an additional set of four storage elements that can exclusively be set as edge-triggered FFs.

In a normal synchronized circuit, the flip-flops are used to store the intermediate values of the circuit and are instantiated by using FDCE which represents a D-flip-flop with an asynchronous reset. This is done as part of the synthesis and mapping procedure of Xilinx design suits, so it is not exposed to the user. However, we can also explicitly instantiate the latches in the design by using LDCE which is a transparent level-sensitive data latch. We will use LDCE as a glitch-stopping latch.

Beyond the standard roles of such storage elements, they can also be used to implement logic gates. For example, a level-sensitive latch can emulate basic logic gates. Two primitives in Spartan-6 that help in leveraging the latch function as logic gates are the AND2B1L and OR2L. AND2B1L is a 2-input AND gate with one of the inputs inverted. This configuration uses the data input of the latch as the inverted AND-gate input (labelled as DI) and the latch's asynchronous clear input (labelled as SRI) as the other AND-gate input. OR2L is a 2-input OR gate that combines the data input of the latch with the latch's asynchronous preset.

The design purpose of these primitives is to broaden the logic capability by adding an additional external input. Typically, the latch data input is sourced from the output of a LUT located in the same slice. Similar to MUXF7, AND2B1L is a 2-input AND gate with one input inverted implemented on the latch function. We will use either AND2B1L or LUT as a glitch-stopping AND-gate. Thus on our Spartan-6, we have identified four ways to implement the three clocked-gates.

5.2 AES Implementations

In our implementations, we will use MUXF7 only with the second input set to 0, therefore implementing glitch-stopping AND-gate. Further, we will present several implementations for the two AES designs (S-boxes with two and four stages). Namely, for the design with two stages S-box we present four implementations AND2B1L, MUXF7, LUT and LDCE; while for the design with four stages S-box, we present two implementations MUXF7 and LDCE.

Implementations of AES with the Two-Stage S-Box In synchronous logical circuits, the clock signal is routed from its source, such as a Phase-Locked Loop, crystal oscillator, or clock divider, to all relevant components including registers and latches. This process, known as clock tree synthesis, is a crucial step in the circuit design workflow, taking place after the placement phase and before the routing phase.

We start from the GSC_R implementation, as described in Section 3.2, focusing solely on the 2-stage S-box illustrated in Figure 7. The remainder of the implementation is the serialized AES encryption architecture assuming the S-box is evaluated in a single cycle. We need to replace the register between the two stages with a glitch-stopping gate, which could be either a clocked latch or clocked AND. Note that in this simple configuration when only one register has to be replaced, we can maintain the same clock signal as in the

standard register configuration. However, the clock routing should connect this signal to the pin of the glitch-stopping gate as referred to in Figure 9a.

In our implementation, an external 6 MHz clock signal from a waveform generator is provided to the clock pin of the Spartan-6 FPGA. Internally, this 6 MHz clock signal is routed to all the registers, while we have to *manually* set the clock pins of the glitch-stopping gates to be `CLOCK_DEDICATED_ROUTE = FALSE`. Thus, the 5×8 bits of the registers are now replaced by the same quantity of clocked-gates.

Implementations of AES with the Four-Stage S-Box We start with the `GSC_R` implementation described in Section 3.3 and again focus only on the 4-stage S-box as depicted in Figure 8. The FPGA is supplied with a $24MHz$ clock. In this implementation, the Xilinx clock wizard is utilized to generate clock signals with different duty cycles, namely 25%, 50%, and 75% duty cycles. For a latch-based pipeline, the clock outputs are $6MHz$, $12MHz$, and $12MHz$ with 180° phase shift. The clocking wizard in the ISE or Vivado design tools for Xilinx has a GUI interface which allows implementing such clock signals directly. Again as it was the case for the S-box with 2 stages, we have to manually re-route to the corresponding registers clock pins of the the glitch-stopping gates replacing those pipeline internal registers. Similar to the 2-stage S-box scenario, it is necessary to manually set the clocks signal to the pins of the glitch-stopping gates `CLOCK_DEDICATED_ROUTE = FALSE`, which replace the three internal pipeline registers. Thus, the 114 bits of the registers are now replaced by the same quantity of clocked-gates.

5.3 Performance Comparison

In Table 1, we start by listing the AES unprotected S-box [MPL+11] based on Canright [Can05], which is a standard unprotected synchronous circuit with registers. We then compare this with the latest implementations of low-latency glitch-stopping circuits with registers (`GSC_R`) [GIB18] and state-of-the-art GFC, which are asynchronous [SBB+22, SBHM20, NGPM22]. A direct comparison is challenging due to variations in CMOS libraries across different studies. WDDL-based solutions typically eliminate the need for registers, thus reducing area costs, but they still have a substantial overhead in combinational logic due to DRP logic. Additionally, these circuits have a higher logic depth and require a pre-charging phase, which in synchronous designs translates to one cycle. However, GFCs are self-synchronizing and may get better latency in time. Clear advantage for the GFC versus `GSC_R` can be seen when latency is measured in cycles.

We have implemented two specific designs of `GSC_R` using $t + 1$ and $t + 2$ TI, which serve as a baseline for our comparisons. Both implementations are described in Section 3.2 and Section 3.3. All our implementations use `Nangate45` CMOS library. The syntheses are re-run with Synopsys Design Compiler 2017.09 using the command `compile -exact_map` and with the `set_dont_touch` flag applied to all nets. We made this choice because the `compile` command includes fewer default optimizations compared to `compile_ultra`. The command flag `-exact_map` prevents sequential mapping from encapsulating combinational logic into sequential cells. We note that the instantiating of glitching-stopping circuits in the synthesis libraries and on the FPGAs are different. Namely, in the HDL code the `AND`-gates, `MUX`-gates and latches are respectively instantiated with `AND2_X1`, `MUX2_X1` and `DLH_X1` elements for `Nangate45` while for Xiling FPGA (as already explained in Section 5.1) those are instantiated with `AND2B1L`, `LUT`, `MUXF7` and `LDCE`. In the last part of Table 1 we list our results for the implementations of `GSC_R`.

Firstly, the glitch-stopping circuits without registers `GSC_woR` are smaller than the corresponding circuits with registers `GSC_R`. The 2-Stage versus 4-Stage cases also show the trend that the more stages the initial solution has the larger is the area reduction. Secondly, our solutions match the cycle-wise latency of GFC. For latency measured in time, we estimate the maximum frequency based on the longest data arrival time from one

glitch-stopping (sequential) layer to another. In GSC_R, the propagation delay occurs on the data path between registers, triggered by the clock's positive edges. In circuits without registers GSC_woR, glitch-stopping layers operate under specific clock configurations, and their propagation delays span all stages of the combinational logic. The timing report is generated by Synopsys's static timing analysis (STA) tool, although no user-defined clock constraints are applied to the implementations. Note that the STA tool considers latches and registers as sequential elements, compared to AND and MUX gates as logic elements. Thus, the timing estimations of GSC_G and GSC_L are not directly comparable. As a rule of thumb the latency in time can be roughly calculated as the GSC_R masked latency multiplied with the number of stages.

We further elaborate on the impact of GSC_woR on the latency in time. As explained in Section 3.4, the latency overhead in traditional GSC_R is attributed to the XOR operations used for re-sharing and compression, along with the time consumed by registers inserted for separating the circuits. Typically, these implementations involve between 8 to 12 XORs and one to three registers. In contrast, GFC do not have the overhead of the registers. The GSC_woR also eliminates overhead of registers, but the synchronization of the circuits costs time. However, this synchronisation can be further optimized per stage and the overhead can be made similar to the asynchronous GFC case.

As it can be seen in Table 1 and Table 2, both GSC_R and Latch-based GSC_woR designs achieve the same time-wise latency for the S-box. Taking into account that the logic which in GSC_R was spread over 2 or 4 cycles, it is processed in GSC_woR in a single cycle we see that $609/4 = 152$ and $387/2 = 193$ holds. However, AND and MUX-based GSC_woR designs can achieve higher S-box time-wise latency by employing optimization of the virtual cycles as depicted in Figure 13. In the 2-stage AES S-box both stages have nearly equal critical paths hence only a small optimization (211 versus 193) can be achieved. However, the 4-stage AES S-box has 2 middle stages with critical path more than twice shorter than the other 2 stages, thus a frequency of 217 MHz instead of 152 MHz only.

In summary, implementing GSC_woR, as outlined in Procedure 4.1, involves the usage of additional clocks to maintain synchronization of the combinational logic in contrast to the conventional GSC_R. This approach not only improves latency in terms of time and cycles but also offers better area efficiency. However, the GSC_woR design method is more engineering-intensive in executing all the steps in Procedure 4.1. Whether the overall gain in cost compensates for the additional efforts and risks is case dependent. While direct comparison with glitch-free circuits is not straightforward, we anticipate achieving comparable latency in time and equal latency in cycles, but with better area efficiency. Moreover, this method is compatible with standard digital design tools and libraries, making it more accessible for practical implementation.

5.4 Leakage Evaluation

None of the existing formal verification tools like SILVER [KSM20], COCO [GHP+20], etc. can be used to evaluate the new glitch-stopping without registers (GSC_woR) designs. Thus similarly to [SBHM20, NGPM22, SBB+22] the only remaining way to demonstrate the security of our designs is to use Test Vector Leakage Assessment (TVLA) by Goodwill et al. [GJJR11] which is widely used to evaluate the security of cryptographic designs against SCA. Instead of testing against potential attacks, TVLA's goal is to identify, using the power traces, the leakage points of the secrets, irrespective of whether these leakages are immediately exploitable or not. The non-specific t-test is applied which verifies that our implementations do not show first-order leakage. For this, the measured power traces are divided into two sets, where the first set $\mathcal{S}_0$ uses fixed plaintexts and the second $\mathcal{S}_1$ receives random plaintexts. The t-test verifies whether the first-order statistical moments of these two sets are the same. The null hypothesis of the t-test states that "$\mathcal{S}_0$ and $\mathcal{S}_1$ are drawn from populations with the same mean."

Table 1: Comparison of first-order secure AES S-box implementations

Solutions	Area (kGE)	Latency ($cycles$)	Max clock (MHz)	CMOS Library
Unprotected S-box [MPL+11]	0.23	1	-	UMC L180
Glitch-Free Circuits (GFC)				
Self-Timed Masking [SBB+22]	6.07	1	4.8	STCMOSM40
LMDPL [SBHM20]	3.84	1	400	GF28
SESYM [NGPM22]	3.98	1	192	UMC65
Glitch-Stopping Circuits with Registers (GSC_R)				
GLM [GIB18]	60.73	1	356	UMC90
GLM [GIB18]	6.74	2	584	UMC90
5-share ($t+2$ TI) 2-Stage S-Box	21.86	2	387	Nangate45
3-share ($t+1$ TI) 4-Stage S-Box	5.31	4	609	Nangate45
Glitch-Stopping Circuits without Registers (GSC_woR)				
AND-based 2-Stage S-Box	19.41	1	211	Nangate45
MUX-based 2-Stage S-Box	19.45	1	211	Nangate45
Latch-based 2-Stage S-Box	19.63	1	193	Nangate45
AND-based 4-Stage S-Box	4.76	1	217	Nangate45
MUX-based 4-Stage S-Box	4.90	1	217	Nangate45
Latch-based 4-Stage S-Box	5.12	1	152	Nangate45

Table 2: Comparison of propagation delays (ns, Mhz) of the different stages in the masked GSC_R S-boxes synthesized with Synopsys

S-box Instance	Stage 1 (ns, Mhz)	Stage 2 (ns, Mhz)	Stage 3 (ns, Mhz)	Stage 4 (ns, Mhz)
3-share 4-stage AES S-box	1.58	0.73	0.64	1.64
	632	1369	1562	609
5-share 2-stage AES S-box	2.18	2.58	NA	NA
	458	387		

Large absolute values of this t-statistic provide evidence for rejecting the null hypothesis with a certain confidence. In side-channel research, a widely adopted threshold for the t-test is 4.5. If the t-test value computed from the power traces exceeds this threshold, it signifies a potential security vulnerability in the tested implementation. Note that we have used this threshold value for the t-tests, however one can apply the methodology by Ding et al. [DZD+17] to obtain a proper threshold when the number of samples is large, for example, $t = 6$ for 10K samples and $\alpha = 10^{-5}$ or $t = 7.4$ for 13K samples. Since our AES serialized implementations have 20K samples even higher t can be applied.

For the practical experiments, we mainly used a Xilinx Spartan-6 FPGA on a SAKURA-G evaluation board. We collect power traces using a digital oscilloscope at a sampling rate of $500MS/s$. The encryption is performed up to 100 million times receiving either fixed or random masked plaintexts. To do the t-test, we collect 4K sample points for the S-box implementations and 20K samples for the AES with serialized architecture implementations. As usual, we performed TVLA on our design with the PRNG both active and inactive to check that the tool-chain performs as expected. We perform the first-order t-test, the summary of the results is given in Table 3, while the figures for the t-tests are provided in Appendix A.

The FPGA primitives, although perform the same logical functions as the corresponding digital designs, can introduce glitches from internal hardware. For example, an LUT instance on FPGA when initiated to be an AND-gate is a glitchy hardware unit with five or six input signals. TVLA tests were conducted on both a standalone 2-stage 5-share S-box and the

Table 3: Summary of the first-order implementations using the FPGA primitives on Sakura-G

Synthesis	FPGA	Implementation	TVLA	Figure
AND or MUX gates	MUXF7	2-stage 5-share AES S-box	*Pass*	Fig. 15
AND-gate	AND2B1L	2-stage 5-share AES S-box	*Pass*	Fig. 16
AND-gate	LUT	2-stage 5-share AES S-box	*Fail*	Fig. 17
Latch	LDCE	2-stage 5-share serialized AES	*Pass*	Fig. 18
AND or MUX gates	MUXF7	2-stage 5-share serialized AES	*Pass*	Fig. 19
AND-gate	AND2B1L	2-stage 5-share serialized AES	*Fail*	Fig. 20
AND-gate	LUT	2-stage 5-share serialized AES	*Fail*	Fig. 21
Latch	LDCE	4-stage 3-share serialized AES	*Fail*	Fig. 22
AND or MUX gates	MUXF7	4-stage 3-share serialized AES	*Pass*	Fig. 23
MUX gates @30MHz	MUXF7	4-stage 3-share serialized AES	*Pass*	Fig. 24

Table 4: Summary of the serialized AES implementations using the FPGA primitives on Sakura-X

Synthesis on FPGA	Implementation	TVLA	Security Order	Figure
MUXF7	4-stage 3-shares	*Pass*	first-order uni-variate	Fig. 25
MUXF7	6-stage 3-shares	*Pass*	second-order bi-variate	Fig. 26

serialized AES-128 encryption incorporating such an S-box, evaluating the efficiency of AND-gate implementations within LUTs. As indicated in Figures 17 and 21, the implementation through LUT was ineffective in stopping glitches, evidenced by the failure of both t-tests.

This failure leads to the experiments using other primitives on FPGA, like MUXF7 and AND2B1L, that can build the same logic function of AND-gate. In the 2-stage 5-share S-box implementations, both MUXF7 and AND2B1L show no leakages as illustrated in Figures 15 and 16. However, when the tests are extended to the serialized AES-128 encryption Figure 19 shows that MUXF7 achieves the first-order security, while AND2B1L fails Figure 20. The key differences between the AND2B1L and MUXF7 primitives are that AND2B1L, is a larger unit shared with latches and registers and thus it requires more complex routing. Additionally, the full AES serialized encryption, combining synchronous state arrays and control logics, adds complexity and potential leakage sources. Due to negative results with AND2B1L and LUT, they were not tested on the 4-stage 3-share serialized AES. However, MUXF7 showed no leakage in the 4-stage 3-share implementation (Figure 23), suggesting its suitability as a glitch-stopping gate in FPGA environments.

The test using latches (LDCE) with the 2-stage 5-share S-box in AES serialized encryption showed no first-order leakage (Figure 18), where the clock configuration is the same as the AND-gate. However, the AES serialized encryption with a 4-stage 3-share S-box exhibited leakage (Figure 22), potentially due to complex clock configurations that inadequately prevented glitch propagation ac-ross imperfect clock edges.

For all evaluations on Sakura-G given in Table 3 we use $12MHz$ clock, except for the last one where we show that clocking faster at $30MHz$ (with 25%, 50% and 75% duty cycles) doesn't change the result for the AND or MUX gates.

In order to illustrate transfer-ability of the results to another platform we show in Table 4 that MUXF7 based 4-stage 3-share serialized AES implementation on Sakura-X is still first-order secure.

Finally, in order to demonstrate that the proposed approach is independent of the security order, we used the RTL of the 6-stage 3-share 2nd-order serialized AES implementation by De Cnudde et al. [CRB+16]. The modification from GSC_R to MUXF7 based GSC_woR implementation is second-order secure as shown in Table 4.

In summary, we have shown that MUX or AND glitch-stopping gate can be implemented

on FPGA via `MUXF7`. We leave as a further research topic to show that `LDCE` is also a way to implement latch-based glitch-stopping on FPGA.

6 Conclusions

In this paper, we introduced a method to build secure hardware masking without registers, offering improved performance efficiency and area reduction compared to conventional register-based methods. Our method employs clocked `AND`-gates, latches, and multiplexers to effectively stop glitch propagation. The `GSC_woR` approach is proven, in the glitch-extended probing model, to retain the security level of `GSC_R`. We experimentally confirmed, via TVLA evaluations, the first-order and second-order security of FPGA implementations using `MUXF7` and `LDCE`.

Acknowledgements. This work was supported by CyberSecurity Research Flanders with reference number VR20192203.

References

[ADN+22] Amund Askeland, Siemen Dhooghe, Svetla Nikova, Vincent Rijmen, and Zhenda Zhang. Guarding the first order: The rise of AES maskings. In Ileana Buhan and Tobias Schneider, editors, *Smart Card Research and Advanced Applications - CARDIS 2022*, volume 13820 of *Lecture Notes in Computer Science*, pages 103–122. Springer, 2022.

[ANN23] Amund Askeland, Svetla Nikova, and Ventzislav Nikov. Who watches the watchers: Attacking glitch detection circuits. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, 2024(1):157–179, Dec. 2023.

[ANN24] Amund Askeland, Svetla Nikova, and Ventzislav Nikov. Who watches the watchers: Attacking glitch detection circuits. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2024(1):157–179, 2024.

[AZN21] Victor Arribas, Zhenda Zhang, and Svetla Nikova. LLTI: low-latency threshold implementations. *IEEE Trans. Inf. Forensics Secur.*, 16:5108–5123, 2021.

[BBD+16] Gilles Barthe, Sonia Belaïd, François Dupressoir, Pierre-Alain Fouque, Benjamin Grégoire, Pierre-Yves Strub, and Rébecca Zucchini. Strong non-interference and type-directed higher-order masking. In Edgar R. Weippl, Stefan Katzenbeisser, Christopher Kruegel, Andrew C. Myers, and Shai Halevi, editors, *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, pages 116–129. ACM, 2016.

[BGN+14] Begül Bilgin, Benedikt Gierlichs, Svetla Nikova, Ventzislav Nikov, and Vincent Rijmen. Higher-order threshold implementations. In Palash Sarkar and Tetsu Iwata, editors, *Advances in Cryptology - ASIACRYPT 2014*, volume 8874 of *LNCS*, pages 326–343. Springer, 2014.

[BKN20] Dusan Bozilov, Miroslav Knezevic, and Ventzislav Nikov. Optimized threshold implementations: Minimizing the latency of secure cryptographic accelerators. In S. Belaid and T. Guneysu, editors, *Smart Card Research and Advanced Applications - CARDIS 2020*, Lecture Notes in Computer Science, pages 20–39. Springer, 2020.

[BP11] Joan Boyar and Rene Peralta. A depth-16 circuit for the aes s-box. Cryptology ePrint Archive, Paper 2011/332, 2011.

[Can05] David Canright. A very compact S-box for AES. In *Cryptographic Hardware and Embedded Systems - CHES 2005 Proceedings*, pages 441–455, 2005.

[CGLS21] Gaëtan Cassiers, Benjamin Grégoire, Itamar Levi, and François-Xavier Standaert. Hardware private circuits: From trivial composition to full verification. *IEEE Transactions on Computers*, 70(10):1677–1690, 2021.

[CRB+16] Thomas De Cnudde, Oscar Reparaz, Begül Bilgin, Svetla Nikova, Ventzislav Nikov, and Vincent Rijmen. Masking AES with d+1 shares in hardware. In Benedikt Gierlichs and Axel Y. Poschmann, editors, *Cryptographic Hardware and Embedded Systems - CHES 2016 - 18th International Conference, Santa Barbara, CA, USA, August 17-19, 2016, Proceedings*, volume 9813 of *Lecture Notes in Computer Science*, pages 194–212. Springer, 2016.

[CS20] Gaëtan Cassiers and François-Xavier Standaert. Trivially and efficiently composing masked gadgets with probe isolating non-interference. *IEEE Trans. Inf. Forensics Secur.*, 15:2542–2555, 2020.

[DZD+17] A. Adam Ding, Liwei Zhang, François Durvaux, François-Xavier Standaert, and Yunsi Fei. Towards sound and optimal leakage detection procedure. In Thomas Eisenbarth and Yannick Teglia, editors, *Smart Card Research and Advanced Applications - CARDIS 2017*, volume 10728 of *Lecture Notes in Computer Science*, pages 105–122. Springer, 2017.

[FGP+18] Sebastian Faust, Vincent Grosso, Santos Merino Del Pozo, Clara Paglialonga, and François-Xavier Standaert. Composable masking schemes in the presence of physical defaults & the robust probing model. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, (3):89–120, 2018.

[GHP+20] Barbara Gigerl, Vedad Hadzic, Robert Primas, Stefan Mangard, and Roderick Bloem. Coco: Co-design and co-verification of masked software implementations on cpus. Cryptology ePrint Archive, Paper 2020/1294, 2020.

[GIB18] Hannes Groß, Rinat Iusupov, and Roderick Bloem. Generic low-latency masking in hardware. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2018(2):1–21, 2018.

[GJJR11] Gilbert Goodwill, Benjamin Jun, Josh Jaffe, and Pankaj Rohatgi. A testing methodology for side-channel resistance validation, September 2011.

[GMK16] Hannes Groß, Stefan Mangard, and Thomas Korak. Domain-oriented masking: Compact masked hardware implementations with arbitrary protection order. *IACR Cryptol. ePrint Arch.*, 2016:486, 2016.

[ISW03] Yuval Ishai, Amit Sahai, and David A. Wagner. Private circuits: Securing hardware against probing attacks. In Dan Boneh, editor, *Advances in Cryptology - CRYPTO 2003*, volume 2729 of *Lecture Notes in Computer Science*, pages 463–481. Springer, 2003.

[KJJ99] Paul C. Kocher, Joshua Jaffe, and Benjamin Jun. Differential power analysis. In Michael J. Wiener, editor, *Advances in Cryptology - CRYPTO 1999*, volume 1666 of *Lecture Notes in Computer Science*, pages 388–397. Springer, 1999.

[KM22] David Knichel and Amir Moradi. Low-latency hardware private circuits. *Proceedings of the 2022 ACM CCS Conference*, 2022.

[KSM20] David Knichel, Pascal Sasdrich, and Amir Moradi. Silver - statistical independence and leakage verification. Cryptology ePrint Archive, Paper 2020/634, 2020.

[LMM23] Daniel Lammers, Nicolai Müller, and Amir Moradi. Glitch-free is not enough - revisiting glitch-extended probing model. Cryptology ePrint Archive, Paper 2023/035, 2023.

[MPL+11] Amir Moradi, Axel Poschmann, San Ling, Christof Paar, and Huaxiong Wang. Pushing the limits: A very compact and a threshold implementation of aes. In *EUROCRYPT*, volume 6632, pages 69–88, 2011.

[MPO05] Stefan Mangard, Norbert Pramstaller, and Elisabeth Oswald. Successfully attacking masked AES hardware implementations. In *Cryptographic Hardware and Embedded Systems - CHES 2005*, volume 3659 of *Lecture Notes in Computer Science*, pages 157–171. Springer, 2005.

[MS16] Amir Moradi and Tobias Schneider. Side-channel analysis protection and low-latency in action - - case study of PRINCE and midori -. In Jung Hee Cheon and Tsuyoshi Takagi, editors, *Advances in Cryptology - ASIACRYPT 2016*, volume 10031 of *Lecture Notes in Computer Science*, pages 517–547, 2016.

[NGPM22] Rishub Nagpal, Barbara Gigerl, Robert Primas, and Stefan Mangard. Riding the waves towards generic single-cycle masking in hardware. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2022(4):693–717, 2022.

[NNR19] Svetla Nikova, Ventzislav Nikov, and Vincent Rijmen. Decomposition of permutations in a finite field. In *Cryptography and Communications*, volume 11, pages 379—-384, 2019.

[NRR06] Svetla Nikova, Christian Rechberger, and Vincent Rijmen. Threshold implementations against side-channel attacks and glitches. In Peng Ning, Sihan Qing, and Ninghui Li, editors, *Inform. and Commun. Secur., ICICS 2006, Proceedings*, volume 4307 of *LNCS*, pages 529–545. Springer, 2006.

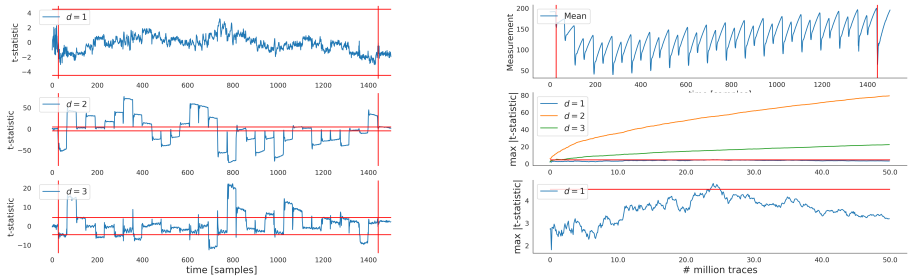
[NT22] Daniel Nemiroff and Carlos Tokunaga. Fault-Injection Countermeasures Deployed at Scale. Technical report, Intel Corporation, 2022.

[PAB+22] Enrico Piccione, Samuele Andreoli, Lilya Budaghyan, Claude Carlet, Siemen Dhooghe, Svetla Nikova, George Petrides, and Vincent Rijmen. An optimal universal construction for the threshold implementation of bijective s-boxes. Cryptology ePrint Archive, Paper 2022/1141, 2022.

[RBN+15] Oscar Reparaz, Begül Bilgin, Svetla Nikova, Benedikt Gierlichs, and Ingrid Verbauwhede. Consolidating masking schemes. In *Advances in Cryptology - CRYPTO 2015 Proceedings, Part I*, pages 764–783, 2015.

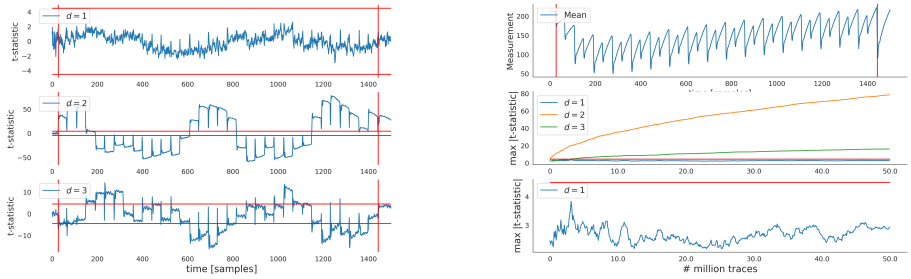
[RCN03] Jan M. Rabaey, Anantha P. Chandrakasan, and Borivoje Nikolić. *Digital integrated circuits: a design perspective*. Prentice Hall electronics and VLSI series. Pearson Education, Upper Saddle River, N.J, 2nd ed edition, 2003.

[SBB+22] Mateus Simoes, Lilian Bossuet, Nicolas Bruneau, Vincent Grosso, and Patrick Haddad. Self-timed masking: Implementing first-order masked s-boxes without registers. *IACR Cryptol. ePrint Arch.*, page 641, 2022.



(a) The final t-test result (b) The maximum t-test value evolving

Figure 15: The 2-stage 5-share AES S-box using `MUXF7`



(a) The final t-test result (b) The maximum t-test value evolving

Figure 16: The 2-stage 5-share AES S-box using `AND2B1L`

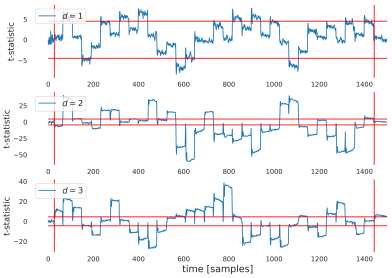
[SBHM20] Pascal Sasdrich, Begül Bilgin, Michael Hutter, and Mark E. Marson. Low-latency hardware masking with application to AES. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2020(2):300–326, 2020.

[TV04] Kris Tiri and Ingrid Verbauwhede. A logic level design methodology for a secure DPA resistant ASIC or FPGA implementation. In *Design, Automation and Test in Europe Conference and Exposition (DATE 2004*, pages 246–251. IEEE Computer Society, 2004.

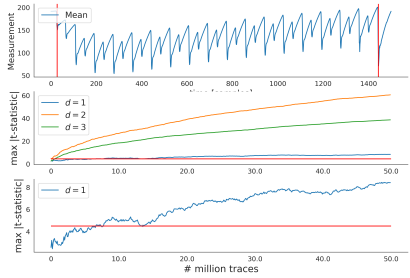
[WH10] Neil Weste and David Harris. *CMOS VLSI Design: A Circuits and Systems Perspective*. Addison-Wesley Publishing Company, USA, 4th edition, 2010.

[Wik24] Wikipedia. Flip-flop (electronics) — wikipedia, the free encyclopedia. http://en.wikipedia.org/w/index.php?title=Flip-flop%20(electronics)&oldid=1193270436, 2024. [Online; accessed 08-January-2024].

A TVLA Test Results

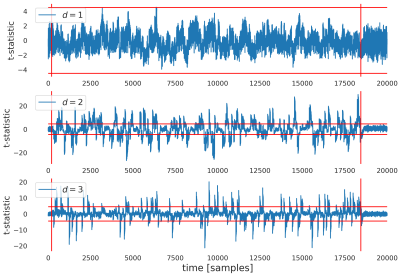


(a) The final t-test result

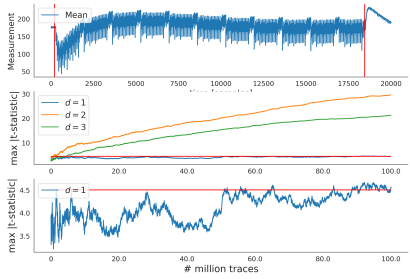


(b) The maximum t-test value evolving

Figure 17: The 2-stage 5-share AES S-box using `LUT`

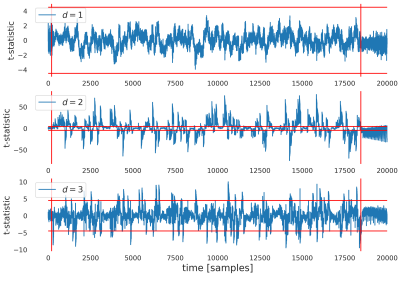


(a) The final t-test result

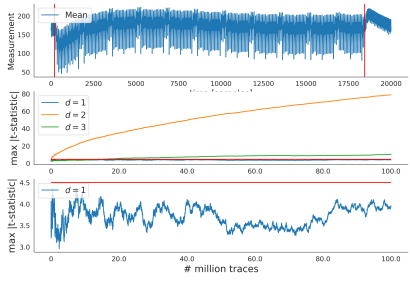


(b) The maximum t-test value evolving

Figure 18: The 5-share serialized AES using the 2-stage S-box in `LDCE`

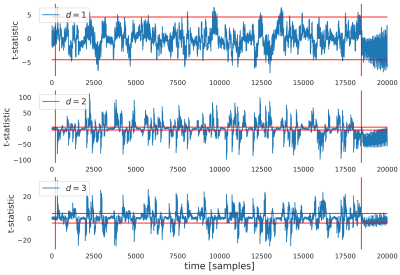


(a) The final t-test result

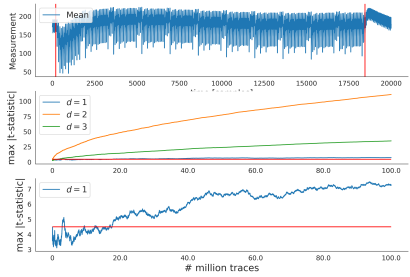


(b) The maximum t-test value evolving

Figure 19: The 5-share serialized AES using the 2-stage S-box in `MUXF7`

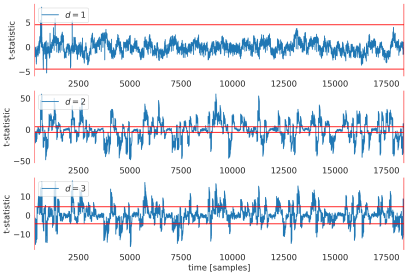


(a) The final t-test result

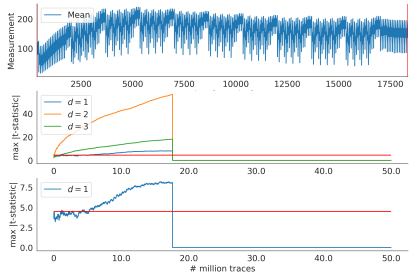


(b) The maximum t-test value evolving

Figure 20: The 5-share serialized AES using the 4-stage S-box in `AND2B1L`

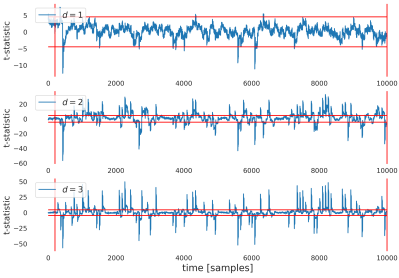


(a) The final t-test result

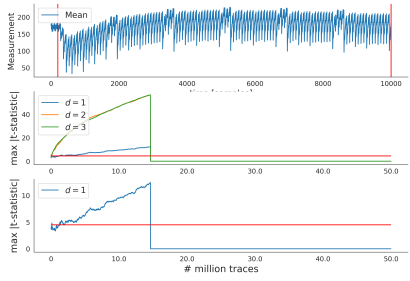


(b) The maximum t-test value evolving

Figure 21: The 5-share Serialized AES using `LUT`

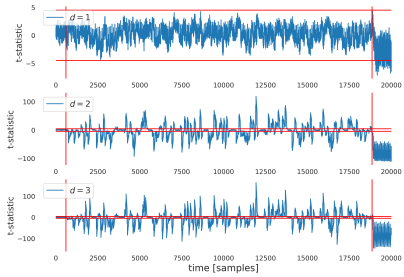


(a) The final t-test result

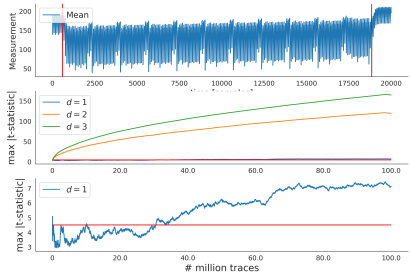


(b) The maximum t-test value evolving

Figure 22: The 3-share serialized AES using the 4-stage S-box in `LDCE`

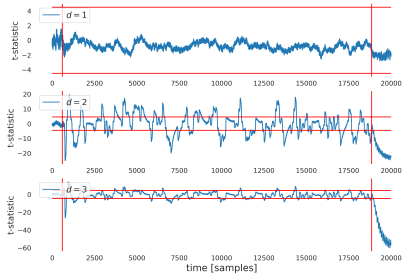


(a) The final t-test result

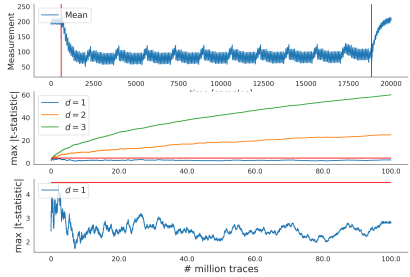


(b) The maximum t-test value evolving

Figure 23: The 3-share serialized AES using the 4-stage S-box in `MUXF7`

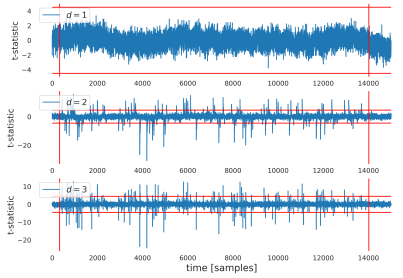


(a) The final t-test result

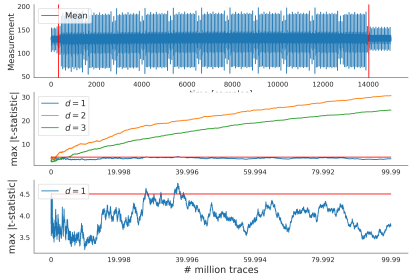


(b) The maximum t-test value evolving

Figure 24: The 3-share serialized AES using the 4-stage S-box in `MUXF7` on Sakura-G with 30Hz input clocks

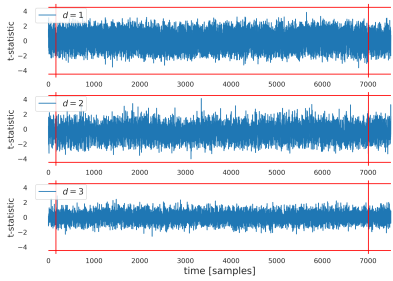


(a) The final t-test result

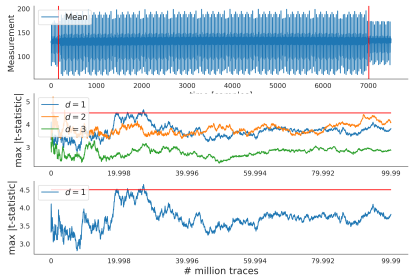


(b) The maximum t-test value evolving

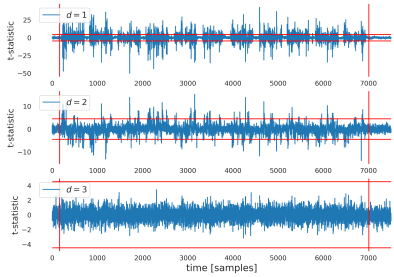
Figure 25: The 3-share first-order serialized AES using the 4-stage S-box in `MUXF7` on Sakura-X



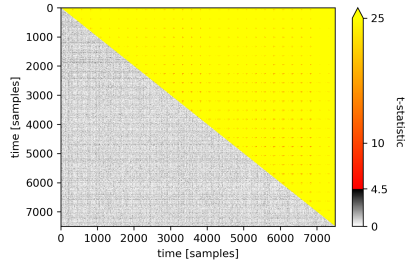
(a) The final uni-variate t-test result: PRNG on



(b) The maximum t-test value evolving



(c) The final uni-variate t-test result: PRNG off



(d) The final bi-variate t-test result: PRNG on (bottom left) and PRNG off (top right)

Figure 26: The 3-share second-order serialized AES using the 6-stage S-box in `MUXF7` on Sakura-X

Chapter 8

Guarding the First Order: The Rise of AES Maskings

Publication Report

- **Title:** *Guarding the First Order: The Rise of AES Maskings*
- **Authors:** Amund Askeland, Siemen Dhooghe, Svetla Nikova, Vincent Rijmen, and Zhenda Zhang
- **Conference:** Smart Card Research and Advanced Applications: 21st International Conference, CARDIS 2022, Birmingham, UK, November 7–9, 2022
- **DOI:** https://doi.org/10.1007/978-3-031-25319-5_6
- **ePrint:** ePrint/2021/734

Contributions Summary: *Main author with Amund Askeland and Siemen Dhooghe. I designed the hardware architecture and conducted the experiments.*

Guarding the First Order: The Rise of AES Maskings

Amund Askeland[2], Siemen Dhooghe[1], Svetla Nikova[1,2], Vincent Rijmen[1,2], Zhenda Zhang[1]

[1] imec-COSIC, ESAT, KU Leuven, Belgium
firstname.lastname@esat.kuleuven.be
[2] University of Bergen, Bergen, Norway
firstname.lastname@uib.no

Abstract. We provide three first-order hardware maskings of the AES, each allowing for a different trade-off between the number of shares and the number of register stages. All maskings use a generalization of the changing of the guards method enabling the re-use of randomness between masked S-boxes. As a result, the maskings do not require fresh randomness while still allowing for a minimal number of shares and providing provable security in the glitch-extended probing model. The low-area variant has five cycles of latency and a serialized area cost of 8.13 kGE. The low-latency variant reduces the latency to three cycles while increasing the serialized area by 67.89% compared to the low-area variant. The maskings of the AES encryption are implemented on FPGA and evaluated with Test Vector Leakage Assessment (TVLA).

Keywords: No keywords given.

1 Introduction

The Advanced Encryption Standard (AES) [8] is one of the most used cryptographic building blocks in practice. The cipher has secured many applications, including the world wide web. However, for some applications, like embedded devices, naive implementations of the AES are vulnerable to side-channel attacks such as Differential Power Analysis (DPA) due to Kocher *et al.* [18]. The current agreed-upon method to protect implementations against DPA is masking. In masking, each key-dependent variable is split into several random shares such that an adversary needs to view the power consumption of each share to gain information on the secret variable.

Several maskings of the AES appeared in the literature in the past twenty years. The efficiency and security of them have been significantly improved over time. Threshold implementations by Nikova *et al.* [21] allowed for maskings that protect against glitches in hardware. The uniformity aspect of threshold implementations allows for the reduction of randomness. The changing of the guards technique by Daemen [7] showed how to make a masking uniform without significantly increasing costs. Canright [4] proposed an efficient tower field decomposition of the AES S-box in order to improve hardware costs. This decomposition was then used by De Cnudde *et al.* [9] to create efficient threshold implementation maskings of the AES. However, the authors noted that the randomness cost of their designs is high, which makes it infeasible to generate the randomness in a cryptographic secure way.

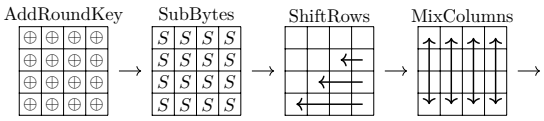


Figure 1: Representation of the AES.

Contributions. In this paper, we generalize the changing of the guards technique to include maskings that use randomness in Section 3. The method allows the re-use of randomness between all masked S-boxes and retains first-order probing security. As a result, the generalization tackles the open question by De Cnudde *et al.* as we significantly reduce the randomness cost of their masking, at least for first-order security. We then provide three variants of the masking in Section 4, which show the trade-off between the number of register stages and the number of shares.

We apply these maskings of S-boxes to both the serialized and round-based hardware architecture of the AES-128 encryption. These constructions are proven secure in the first-order glitch-extended robust probing model. The low-area variation of the serialized AES, described in Section 4.2, has an area cost of 8.13 kGE and 5 cycles of latency. The latency is reduced to 4 cycles for the S-box in Section 4.3 and 3 cycles for the S-box in Section 4.4 with the area cost of the serialized AES increased by 56.75% and 67.89%, respectively. The serialized AES implementations are tested on our side-channel leakage assessment setup to show the first-order security of the designs. Our implementations amortize the cost of online randomness, at the same time provide provable and physically tested first-order probing security, and achieve one of the most efficient area versus latency trade-offs in the literature.

2 Preliminaries

In this section, we go over the used notation, introduce the AES, the probing side-channel security model, and threshold implementations.

2.1 Notation

We denote bits by subscript and shares by superscript. We denote the most significant bit by a bigger subscript. For example, given $(a_1, ..., a_8)$, a_8 denotes the most significant bit (MSB) and a_1 the least significant bit (LSB). Finally, the concatenation of bits is denoted by $\|$.

2.2 Description of AES

We quickly introduce the standardized AES cipher by Daemen and Rijmen [8]. There are three levels of security 128, 192, and 256. AES consists of a 128-bit state and 128, 192, or 256-bit key, respectively, divided into bytes. The cipher is composed of 10, 12, or 14 rounds, respectively, each applying an addition of a subkey, a bricklayer of S-Boxes, a `ShiftRows` operation, and a `MixColumns` operation. The AES S-Box consists of an inversion in the field $\mathbb{F}_{2^8}$ and the application of an affine layer. This is visually represented in Figure 1.

The key schedule for AES-128, which operates on 4 columns of 32 bits each, is depicted in Figure 2. Each round of the AES state function has a parallel round of the key schedule. We provide a description for the AES-128 key schedule. Denote V_j, with $j \in \{1, ..., 4\}$, the jth word of the key state at round i and W_j the jth word of the key state at round $i + 1$.

Then a round of the key schedule is defined as

$$
\begin{aligned}
W_1 &= V_1 \oplus \texttt{RotWord}(\texttt{SubWord}(V_4)) + C_{i+1}\,, \\
W_2 &= V_2 \oplus W_1\,, \\
W_3 &= V_3 \oplus W_2\,, \\
W_4 &= V_4 \oplus W_3\,.
\end{aligned}
$$

With `RotWord`, the left circular shift, `SubWord`, the application of four AES S-boxes, and C_{i+1}, the round constants for round $i+1$.

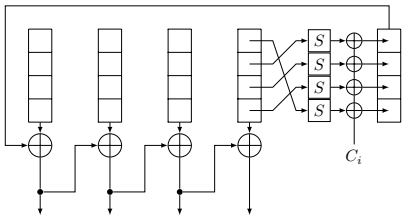


Figure 2: The AES-128 key schedule. $C_i \in \mathbb{F}_2^{32}$ denotes the ith round constants, and S denotes the AES S-box.

2.3 The Threshold Glitch-Extended Probing Model

This section introduces the threshold probing model.

Threshold probing. A dth-order threshold probing adversary $\mathcal{A}$, as first proposed by Ishai *et al.* [17], can view the values present on up to d gates or wires in a circuit implementing a cipher during a single execution (cipher evaluation). We note that by "probe" we do not mean a physical probe such as an EM probe. Instead, the word probe is used as an abstract concept through which an adversary can perfectly observe a part of the computation.

The adversary $\mathcal{A}$ is computationally unbounded, and must specify the location of the probes before querying the circuit. However, the adversary can change the location of the probes over multiple cipher queries. The adversary's interaction with the circuit is mediated through the encoder and decoder algorithms, neither of which can be probed.

The security model is a simulation model where the simulator needs to simulate the probed values from scratch, more specifically, the simulator is not given the input (including the key of a block cipher) of the circuit. The adversary needs to distinguish the probed values from the real circuit with the returned values from the simulator. A failure in doing so proves that the adversary can not learn anything from the circuit's input via the probes. In a security proof, this essentially comes down to proving that the probed values follow a distribution which is independent of the value of the circuit's input.

Glitches. The above model is extended to capture the effect of glitches on hardware. Whereas one of the adversary's probes normally results in the value of a single wire, a glitch-extended probe allows obtaining all the registered inputs leading to the gate/wire which is probed. This extension of the probing model has been discussed in the work of Reparaz *et al.* [22] and formalized by Faust *et al.* [12]. The formulation of the latter work is as follows: "For any ϵ-input circuit gadget G, combinatorial recombinations (aka glitches) can be modeled with specifically ϵ-extended probes so that probing any output of the function allows the adversary to observe all its ϵ inputs."

2.4 Boolean Masking and Threshold Implementations

Boolean masking was independently introduced by Goubin and Patarin [15] and Chari *et al.* [5]. It serves as a sound and widely-deployed countermeasure against side-channel attacks. The technique is based on splitting each secret variable $x \in \mathbb{F}_2$ in the circuit into shares $\bar{x} = (x^1, x^2, \ldots, x^{s_x})$ such that $x = \sum_{i=1}^{s_x} x^i$ over $\mathbb{F}_2$. A random Boolean masking of a fixed secret is uniform if all maskings of that secret are equally likely.

There are several approaches to protect a circuit by masking. In this work, we make use of threshold implementations, proposed by Nikova *et al.* [21]. In particular, we focus on "first-order threshold implementations" as those which protect against first-order side-channel attacks. The interested reader is referred to the works by Bilgin *et al.* [2] and Beyne *et al.* [1] for more information on how to use threshold implementations to secure against higher-order attacks. In the following, the main properties of threshold implementations as introduced by Nikova *et al.* are reviewed.

A threshold implementation consists of several layers of Boolean functions, as shown in Figure 3. As for any masked design, a black-box encoder function generates a uniform random masking of the input before it enters the masked circuit. At the end of each layer, synchronization is ensured by means of registers which stop the propagation of glitches.

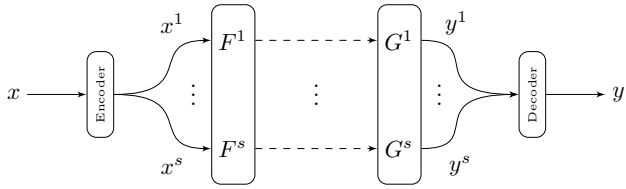


Figure 3: Schematic illustration of a threshold implementation assuming an equal number of input and output shares [11].

Let $\bar{F}$ be a layer in the threshold implementation corresponding to a part of the circuit $F : \mathbb{F}_2^n \to \mathbb{F}_2^m$. For example, F might be the linear layer of a block cipher. The function $\bar{F} : \mathbb{F}_2^{ns_x} \to \mathbb{F}_2^{ms_y}$, where we assume s_x shares per input bit and s_y shares per output bit, will be called a *masking* of F. The ith share of the function $\bar{F}$ is denoted by $F^i : \mathbb{F}_2^{ns_x} \to \mathbb{F}_2^m$, for $i \in \{1, .., s_y\}$. Maskings can have a number of properties that are relevant in the security argument for a threshold implementation; these properties are summarized in Definition 1.

Definition 1 (Properties of threshold implementations [21])**.** Let $F : \mathbb{F}_2^n \to \mathbb{F}_2^m$ be a function and $\bar{F} : \mathbb{F}_2^{ns_x} \to \mathbb{F}_2^{ms_y}$ be a masking of F. The masking $\bar{F}$ is said to be

correct
if $\sum_{i=1}^{s_y} F^i(x^1, \ldots, x^{s_x}) = F\left(\sum_{i=1}^{s_x} x^i\right)$ for all shares $x^1, \ldots, x^{s_x} \in \mathbb{F}_2^n$,

non-complete
if any function F^i, measured between register stages, depends on at most $s_x - 1$ input shares,

uniform
if $\bar{F}$ maps a uniform random masking of any $x \in \mathbb{F}_2^n$ to a uniform random masking of $F(x) \in \mathbb{F}_2^m$.

Considering that, in a threshold implementation, all input/outputs of the functions are stored in registers, placing a glitch-extended probe in a layer of a threshold implementation returns all inputs of the probed masked Boolean function. If all layers of a threshold implementation are non-complete and uniform, the resulting masked circuit can be proven secure in the first-order probing model with glitches [11].

3 Changing of the Guards with Randomness

The changing of the guards method proposed by Daemen [7] is a technique that transforms a non-complete masking into a uniform and non-complete masking. The technique works by embedding the masking into a Feistel-like structure. In this paper, we slightly generalize the method by considering a first-order probing secure masking. Such a masking potentially requires multiple register stages and extra randomness to guarantee its security. The adapted changing of the guards method still ensures uniformity while allowing the re-use of the randomness. An example of the method with two shares is shown in Figure 4.

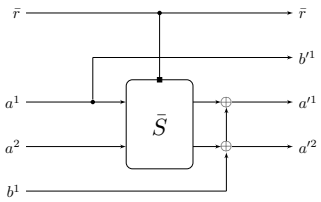


Figure 4: Changing of the guards method with two shares where the masked S-box $\bar{S}$ uses the randomness $\bar{r}$.

With the original changing of the guards method, the function $\bar{S}$ was non-complete. Meaning that $\bar{S}$ typically did not use fresh randomness and was computed in one cycle. Instead, with the generalized method, $\bar{S}$ can use fresh randomness (which can be recycled and used for different S-boxes) and be computed in multiple stages (in this work the whole tower-field decomposition of the AES S-box).

We give the changing of the guards method formally in Definition 2.

Definition 2. The generalized changing of the guards method applied to a masked map $\bar{S}$ given inputs $(a^1, ..., a^s)$, $(b^1, ..., b^{s-1})$, and randomness $\bar{r}$ is calculated as follows

$$\bar{r}' = \bar{r} \tag{1}$$

$$a'^1 = S^1(a^1, ..., a^s, \bar{r}) \oplus b^1, \quad \dots \quad , \quad a'^{s-1} = S^{s-1}(a^1, ..., a^s, \bar{r}) \oplus b^{s-1}, \tag{2}$$

$$a'^s = S^s(a^1, ..., a^s, \bar{r}) \oplus b^1 \oplus \ldots \oplus b^{s-1} \tag{3}$$

$$b'^1 = a^1, \quad \dots \quad , \quad b'^{s-1} = a^{s-1}\,. \tag{4}$$

In general, we refer to the $(b^1, ..., b^{s-1})$ as the *guards* of the masked S-box $\bar{S}$. We show that the changing of the guards construction with randomness retains the correctness and probing security properties from $\bar{S}$ and makes the masking uniform.

Theorem 1. *The method from Definition 2 is correct, first-order probing secure, and uniform.*

Proof. The correctness of the construction follows from the correctness of $\bar{S}$ and the fact that each share b^i is added to two different output shares in Eq. (2)-(3).

First-order probing security of the construction, assuming a joint uniform input, follows from the first-order probing security of $\bar{S}$ and the facts that the share b^s is not used in the construction and that each share b'^i is calculated using only one share a^i using Eq. (2)-(3).

For the proof of uniformity, we first take an arbitrary input secret a. We show that the above construction is invertible. In other words, given the secret a and the outputs $(a'^1, ..., a'^s)$, $(b'^1, ..., b'^{s-1})$, $\bar{r}'$, we show it is possible to construct the inputs $(a^1, ..., a^s)$, $(b^1, ..., b^{s-1})$, $\bar{r}$.

Since the input secret a is given, we can construct the inputs $(a^1, ..., a^s)$ from $(b'^1, ..., b'^{s-1})$ using Eq. (4). From $\bar{r}'$ we can evidently construct $\bar{r}$ since the two are equal (see Eq. (1)). By running $(a^1, ..., a^s)$ and $\bar{r}$ through $\bar{S}$ and XORing the output $(a'^1, ..., a'^s)$, we can also construct $(b^1, ..., b^s)$ (see Eq. (2)-(3)) which concludes the proof. □

Thus, the changing of the guards method allows for the transformation of any first-order probing secure masking into a uniform one which allows the re-use of the randomness used in the S-box.

4 Maskings of the S-Box

This section describes the three first-order glitch-extended probing secure S-box designs of the AES. We first go over the components which are masked between all designs and then provide the three designs themselves.

4.1 Overarching Components

We quickly review the functions used in the tower-field decomposition of the S-box.

Input/output isomorphism. The first operation occurring in the decomposed S-box performs a change of basis through a linear map. This mapping is implemented in combinational logic, and it maps the 8-bit input $(a_1^i, ..., a_8^i)$ to the 8-bit output $(y_1^i, ..., y_8^i)$ for each share $i \in \{1, 2\}$ as follows:

$$
\begin{aligned}
y_8^i &= a_8^i \oplus a_7^i \oplus a_6^i \oplus a_3^i \oplus a_2^i \oplus a_1^i & \qquad y_4^i &= a_8^i \oplus a_5^i \oplus a_4^i \oplus a_2^i \oplus a_1^i \\
y_7^i &= a_7^i \oplus a_6^i \oplus a_5^i \oplus a_1^i & \qquad y_3^i &= a_1^i \\
y_6^i &= a_7^i \oplus a_6^i \oplus a_2^i \oplus a_1^i & \qquad y_2^i &= a_7^i \oplus a_6^i \oplus a_1^i \\
y_5^i &= a_8^i \oplus a_7^i \oplus a_6^i \oplus a_1^i & \qquad y_1^i &= a_7^i \oplus a_4^i \oplus a_3^i \oplus a_2^i \oplus a_1^i
\end{aligned}
$$

The inverse linear map (including the AES affine transformation) maps the 8-bit input $(u_1^i, ..., u_8^i)$ to the 8-bit output $(y_1^i, ..., y_8^i)$ for each share i as follows:

$$
\begin{aligned}
y_8^i &= u_6^i \oplus u_4^i & \qquad y_4^i &= u_8^i \oplus u_7^i \oplus u_6^i \oplus u_5^i \oplus u_4^i \\
y_7^i &= u_8^i \oplus u_4^i & \qquad y_3^i &= u_7^i \oplus u_6^i \oplus u_4^i \oplus u_3^i \oplus u_1^i \\
y_6^i &= u_7^i \oplus u_1^i & \qquad y_2^i &= u_6^i \oplus u_5^i \oplus u_2^i \\
y_5^i &= u_8^i \oplus u_6^i \oplus u_4^i & \qquad y_1^i &= u_7^i \oplus u_5^i \oplus u_2^i
\end{aligned}
$$

The constant 0x63 is then added to the first share $(y_8^1, ..., y_1^1)$.

Finite field multipliers. In the computation of the masking of the S-box, we make use of multipliers over $\mathbb{F}_{2^2}$ and $\mathbb{F}_{2^4}$. We note that these equations only hold for the multipliers in the masked S-box and not for any other operation in the AES like the `MixColumns`. We recall the equations for the multiplication over $\mathbb{F}_{2^2}$. These map the two 2-bit inputs $(a_1, a_2), (b_1, b_2)$ to the 2-bit output (c_1, c_2) as follows:

$$c_1 = (a_2 \oplus a_1)(b_2 \oplus b_1) \oplus a_1 b_1 \qquad c_2 = (a_2 \oplus a_1)(b_2 \oplus b_1) \oplus a_2 b_2$$

We then recall the equations for the multiplication over $\mathbb{F}_{2^4}$. This maps the two 4-bit inputs $(a_1, a_2, a_3, a_4), (b_1, b_2, b_3, b_4)$ to the 4-bit output (c_1, c_2, c_3, c_4) as follows:

$$
\begin{aligned}
c_1 =& a_4b_4 \oplus a_2b_4 \oplus a_3b_3 \oplus a_1b_3 \oplus a_4b_2 \oplus a_1b_2 \oplus a_3b_1 \oplus a_2b_1 \oplus a_1b_1 \\
c_2 =& a_4b_4 \oplus a_3b_4 \oplus a_2b_4 \oplus a_1b_4 \oplus a_4b_3 \oplus a_2b_3 \oplus a_4b_2 \oplus a_3b_2 \oplus a_2b_2 \oplus a_4b_1 \oplus a_1b_1 \\
c_3 =& a_3b_4 \oplus a_2b_4 \oplus a_4b_3 \oplus a_3b_3 \oplus a_1b_3 \oplus a_4b_2 \oplus a_2b_2 \oplus a_3b_1 \oplus a_1b_1 \\
c_4 =& a_4b_4 \oplus a_2b_4 \oplus a_1b_4 \oplus a_3b_3 \oplus a_2b_3 \oplus a_4b_2 \oplus a_3b_2 \oplus a_2b_2 \oplus a_1b_2 \oplus a_4b_1 \oplus a_2b_1
\end{aligned}
$$

Scaling functions. In the decomposed S-box, there are two scaling functions, a "square scale function" and a "scale" function. The linear square scaling $SqSc$ maps the 4-bit input $(x_1, ..., x_4)$ to the 4-bit output $(y_1, ..., y_4) = (x_1, x_1 \oplus x_2, x_2 \oplus x_4, x_1 \oplus x_3)$. The scaling operation $Scale$ maps a 2-bit input (x_1, x_2) to the 2-bit output $(y_1, y_2) = (x_1 \oplus x_2, x_2)$.

Two-bit inverter. The inversion Inv in $\mathbb{F}_{2^2}$ is linear and is implemented by swapping the bits. The inversion operation maps a 2-bit input (x_1, x_2) to the 2-bit output $(y_1, y_2) = (x_2, x_1)$.

4.2 Design I: Two-Share S-Box

The first design uses two shares and is divided into five cycles. The S-box uses a total of 54 random bits which can be re-used over all the S-boxes. The design of the masking is given in Figure 5.

The masked multipliers are calculated by first computing all cross products of the input shares and then re-masking them with a random masking of zero before compressing the cross products back to the input number of shares. More specifically, the two-shared multiplier is given by

$$
\begin{array}{lcllcl}
a^0 & \rightarrow & a^0b^0 + r^0 & & & \\
a^1 & \rightarrow & a^0b^1 + r^1 & \rightarrow & x^0 = a^0b + r^0 + r^1 & \\
b^0 & \rightarrow & a^1b^0 + r^2 & \rightarrow & x^1 = a^1b + r^2 + r^3 & , \\
b^1 & \rightarrow & a^1b^1 + r^3 & & &
\end{array}
\tag{5}
$$

where $r^3 = r^0 + r^1 + r^2$. Note that the above multiplier is non-complete and uniform in both stages when fresh randomness r^i is used. More specifically, the first stage expands the two input shares to a four-sharing and the second stage compresses the uniform four shares back to two output shares.

This masking is the most efficient in terms of area considering the three designs in this paper. However, the decrease in area is traded for by an increase in latency.

4.3 Design II: Three-Share S-Box

The second design uses three shares and is divided into four cycles. The S-box uses a total of 36 random bits which can be re-used in every S-box. The masking is shown in Figure 6.

The masked multipliers are calculated by combining the cross-products in a non-complete way and re-masking them with a random masking of zero. More specifically, the three-shared multiplier is given by

$$
\begin{array}{lcl}
a^0, b^0 & \rightarrow & a^0b^0 + a^0b^1 + a^1b^0 + r^0 \\
a^1, b^1 & \rightarrow & a^1b^1 + a^1b^2 + a^2b^1 + r^1 \\
a^2, b^2 & \rightarrow & a^2b^2 + a^2b^0 + a^0b^2 + r^2
\end{array}
\quad , \tag{6}
$$

where $r^2 = r^0 + r^1$.

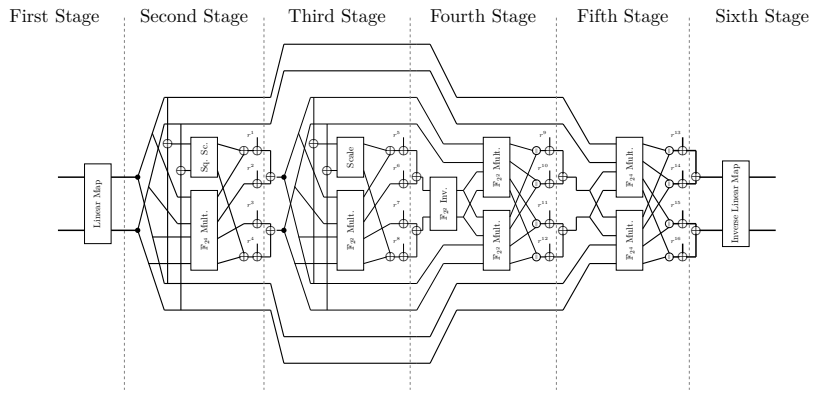


Figure 5: Representation of the S-box of design I. Register stages are denoted by dashed vertical lines.

Compared to the previous design in Section 4.2, this S-box is larger in terms of area but has a reduced latency of four cycles compared to five of the previous design. This design works well with both a serialized and a round-based architecture.

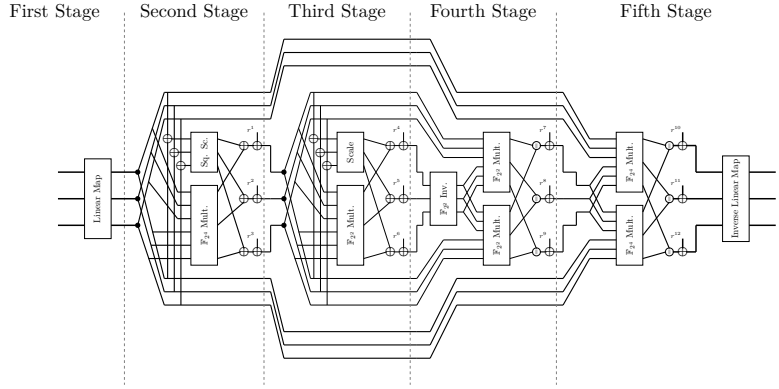


Figure 6: Representation of the S-box of design II. Register stages are denoted by dashed vertical lines.

4.4 Design III: Three-Share S-Box

The third design works over three shares and is divided into three cycles. The masking requires a total of 40 random bits which can be re-used over all S-boxes. The design is shown in Figure 7. The design uses the three-shared multipliers from design II, but the design switches to four shares in order to make a non-complete masking of the inversion over $\mathbb{F}_{2^4}$. For the inversion Inv in $\mathbb{F}_{2^4}$, the resulting equations are given by:

$$
\begin{aligned}
y_1 &= x_1x_3 \oplus x_1x_4 \oplus x_2x_3x_4 \oplus x_2x_4 \oplus x_3 \\
y_2 &= x_1x_3x_4 \oplus x_1x_4 \oplus x_2x_4 \oplus x_3 \oplus x_4 \\
y_3 &= x_1x_2x_4 \oplus x_1x_3 \oplus x_1 \oplus x_2x_3 \oplus x_2x_4 \\
y_4 &= x_1x_2x_3 \oplus x_1 \oplus x_2x_3 \oplus x_2x_4 \oplus x_2
\end{aligned}
$$

For each cubic term $x_a x_b x_c$, the ith share for $i \in \{1,2,3,4\}$ is calculated as follows, where the convention is used that the superscripts wrap around at four.

$$\begin{aligned} x_a x_b x_c^i = {} & x_a^i x_b^i x_c^i \oplus x_a^i x_b^{i+1} x_c^i \oplus x_a^i x_b^i x_c^{i+1} \oplus x_a^i x_b^{i+1} x_c^{i+1} \oplus x_a^i x_b^{i+1} x_c^{i+2} \\ & \oplus x_a^i x_b^{i+2} x_c^{i+1} \oplus x_a^i x_b^i x_c^{i+2} \oplus x_a^i x_b^{i+2} x_c^i \oplus x_a^i x_b^{i+2} x_c^{i+2} \oplus x_a^{i+2} x_b^{i+1} x_c^i \\ & \oplus x_a^{i+1} x_b^{i+2} x_c^i \oplus x_a^{i+2} x_b^i x_c^{i+1} \oplus x_a^{i+1} x_b^i x_c^{i+2} \oplus x_a^{i+2} x_b^{i+1} x_c^{i+1} \\ & \oplus x_a^{i+2} x_b^{i+2} x_c^{i+1} \oplus x_a^{i+2} x_b^{i+1} x_c^{i+2} \end{aligned}$$

The ith share for $i \in \{1,2,3,4\}$ of a quadratic term $x_a x_b$ is calculated as follows:

$$x_a x_b^i = x_a^i x_b^i \oplus x_a^i x_b^{i+1} \oplus x_a^i x_b^{i+2} \oplus x_a^{i+2} x_b^{i+1}$$

The linear terms are added share-wise. The output of the inversion is then refreshed with a zero-masking.

Compared to the previous designs, this one reduces the latency to three cycles, but it trades off a larger area cost. This design is better suited for a round-based architecture.

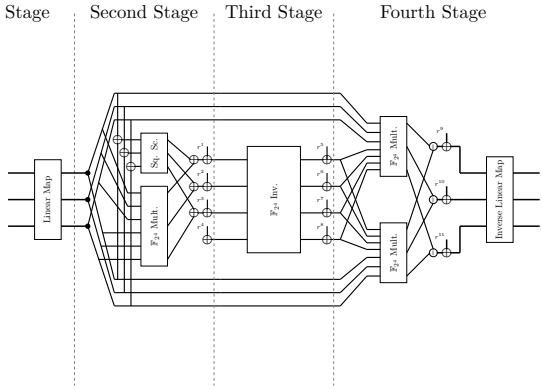


Figure 7: Representation of the S-box of design III. Register stages are denoted by dashed vertical lines.

5 Architecture

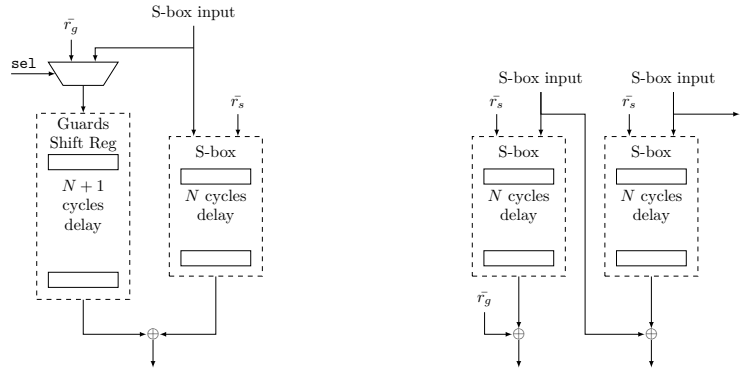
Our designs of the masked AES-128 encryption are implemented in both a serialized architecture and a round-based architecture.

In the round-based AES-128 architecture, 20 S-boxes are instantiated in which 16 S-boxes process `SubBytes` for the state function and 4 for `SubWord` in the key expansion. The bus width of `MixColumns` and `ShiftRows` is $n_shares \times 128$ bits. After each round of the AES encryption, the state of the cipher is stored in a register array. Thus, the whole encryption needs 10 rounds of $sbox_latency + 1$ cycles plus the latency by the control logic. The changing of the guards method is applied to the bricklayer of S-boxes following Definition 1 by Daemen [6].

The implementations that are evaluated in Section 7.2 use the serialized architecture where there is one S-box for both the key expansion and the state. The bus width is thus $n_shares \times 8$ bits. The state registers and key registers can be viewed as 4×4 arrays,

similar to the serialized encryption modules by Shahmirzadi and Moradi [24] and De Meyer *et al.* [10]. Each masked byte after `AddRoundKey`, or after `RotWord` in the key expansion, is fed into the masked S-box. Thus, the serialized architecture needs at least 20 cycles per round. If the latency of the S-box is 4 cycles, the stages of the S-box are pipelined without wasting a cycle. If the latency is larger than 4 cycles, waiting cycles are inserted at the end of each round. If the latency is less than 4 cycles, the round-based architecture is preferred for lower latency applications.

The guard shares are initialized with randomness and the subsequent guards are taken from the input of the S-box. Shift registers are used to store these shares in such a way that they are delayed by one more cycle than the S-box delay before they are applied to the output. This is depicted in Figure 8. A total of $8 \times (n_shares - 1)$ random bits are required to initialize the guard shares. These can be static throughout the AES encryption. The random bits in the masked S-boxes are rotated per applied S-box. This prevents transitional leakages from happening in the pipeline registers. Taking the two-shared multiplication of Eq. (5) as an example, by rotating (r^0, r^1, r^2, r^3), two consecutive executions (first with a, b and then with c, d) calculating x^0 gives Hamming distance leakage $HD(a^0b + r^0 + r^1, c^0d + r^1 + r^2)$ which is masked by r^0, r^2.



(a) Application of the guards in the serialized architecture.

(b) Application of the guards in the round-based architecture.

Figure 8: Application of changing of the guards in our implementations.

Before the AES starts for the serialized implementations, the masked key and plaintext are loaded to the register array in 16 cycles. The ciphertext is read out from the state register array in 16 cycles when the computation is finished. Table 1 depicts the latency in the number of cycles from when the load signal arrives to when the ciphertext is ready at the output.

6 First-Order Probing Security

In this section, we argue the first-order probing security of the three designs. We show this by arguing that the designs are threshold implementations, see Definition 1. We refer to Dhooghe *et al.* [11] for the proof that a threshold implementation is first-order robust probing secure.

The masked S-boxes are first-order probing secure due to the extra randomness added to each register stage. This refreshing works as follows. Given an input $(a^1, ..., a^s)$, an arbitrary masked map $\bar{F}$, and randomness $r^1, ..., r^{s-1}$, refreshing is done as follows, for

$i \in \{1, ..., s-1\}$

$$a'^i = F^i(a^1, ..., a^s) \oplus r^i, \qquad a'^s = F^s(a^1, ..., a^s) \oplus r^1 \oplus ... \oplus r^{s-1}, \tag{7}$$

where each F^i is non-complete.

Lemma 1. *The refreshing following Eq (7) gives a uniform output.*

Proof. We show that the function, taking $(a^1, ..., a^s)$ and $(r^1, ..., r^{s-1})$ as input and $(a^1, ..., a^s)$, $F^i(a^1, ..., a^s) \oplus r^i$ for $i \in \{1, ..., s-1\}$, and $F^s(a^1, ..., a^s) \oplus r^1 \oplus ... \oplus r^{s-1}$ as output, is invertible. Removing the $(a^1, ..., a^s)$ then gives a balanced (or uniform) output of the refreshing detailed in Eq (7).

The derivation is straightforward. Since $(a^1, ..., a^s)$ is given in the output, one can calculate $F^i(a^1, ..., a^s)$ for $i \in \{1, ..., s\}$. Subtracting this from the output $(a^1, ..., a^{s-1})$ then gives $(r^1, ..., r^{s-1})$ showing the map is invertible. □

Theorem 2. *Designs I, II, and III from Sections 4.2,4.3, and 4.4 are threshold implementations as given by Definition 1.*

Proof. First, each design uses a changing of the guards method. We refer to Theorem 1 for the proof that the masked input and output of each masked S-box is uniform.

Since firstly, the linear layers of the construction are evidently non-complete; secondly, they work share-wise and uniform; and thirdly, the unmasked linear functions are permutations, these comply with the properties from Definition 1.

We then show that the masked S-box from designs I, II, and III are first-order probing secure. Since a probe in the designs can only view one masked S-box, it suffices to show that each stage in the masked S-box complies with the threshold implementation properties.

Each stage in the masked S-box either maps a part of the input to the output (such as in the outer wires of Figure 5) or the output is masked using the randomness $\bar{r}$. Due to the changing of the guards structure this randomness $\bar{r}$ is joint uniform with the input of the masked S-box. From Lemma 1, we find that each stage of the masked S-box of design I, II, or III is uniform.

Finally, each stage in the masked S-box is also non-complete. As a result, the masked S-boxes from designs I, II, and III are itself threshold implementations and thus first-order robust probing secure. □

7 Implementations and Physical Evaluations

In this section, we explain the hardware implementations of the three AES designs and their side-channel analysis security.

7.1 Implementations and Comparison to Related Work

We implement our three AES designs for a Xilinx Kintex-7 FPGA mounted on a Sakura-X [19] evaluation board. The implementations are synthesized and programmed using Xilinx ISE. The `KEEP_HIERACHY` option is enabled in the Xilinx ISE to prevent optimization across modules in the synthesis step.

The area of the masked ciphers is measured in gate equivalences (GE), i.e., the cipher area normalized to the area of a 2-input `NAND` gate in a given standard cell library. The cell library we use is the `NANGATE` 45nm Open Cell Library [20], and the synthesis results are obtained with the Synopsis Design Compiler v2021.06. Comparing the area cost in gate equivalences can reduce the impact of different cell libraries. However, the delay on the critical path, or the maximum frequency, depends on the timing metrics of the used

Table 1: Implementation cost of the serialized AES. Latency for the S-box is given in the number of register layers and for the AES in cycles.

Design	Area (S-box) $[kGE]$	Area (AES) $[kGE]$	Latency (S-box) $[regs.]$	Latency (AES) $[cc]$	Random[1] $[bpc]$	f_{max} $[MHz]$[2]
Design I: Sec. 4.2	2.71	8.13	5	242	0	820
Design II: Sec. 4.3	4.33	12.75	4	232	0	515
Design III: Sec. 4.4	5.83	13.66	3	232	0	534
Bilgin *et al.* [3][4]	2.84	8.12	2	246	32	-
De Cnudde *et al.* [9]	1.98	6.68	5	276	54	-
Wegener-Moradi [26]	4.20	7.60	16	2 804	0	-
Sugawara [25]	3.50	17.10	4	266	0	-
Shahmirzadi-Moradi [24][3]	-	7.14	5	246	1	160
Shahmirzadi-Moradi [24][3]	-	7.71	5	246	0	160

1. Cost of online fresh random bits per cycle (*bpc*)
2. Depending on different standard cell libraries
3. With a non-uniform masked S-box
4. Requires an additional register stage in the S-box to store the state

cell library. The latency is measured in the number of cycles to get from the input to the output (*e.g.* from an S-box or a cipher).

Table 1 and Table 2 show the results of our implementation. Table 1 contains the serialized AES implementations and their comparison to other implementations in the literature. Design I gives a 2-share masked AES with the changing of the guards techniques and opts for a low area cost and a moderate latency. Compared to the 4-share AES by Wegener and Moradi [26] and the 3-share AES by Sugawara [25] which also uses the changing of the guards technique, this implementation costs 91.37% fewer cycles and 52.46% less area, respectively. Design II reduces 10 cycles of latency in the AES encryption with the trade-off on a 56.75% larger area and a 37.11% lower maximum frequency. Shahmirzadi and Moradi [24] present a randomness-free 2-share AES implementation with a 5.2% smaller area and a design using one bit per S-box with a 12.2% smaller area. Both designs require a lower maximum frequency compared to design I, although the UMC 180 standard cell library was used. However, we wish to emphasize that both designs are based on using a non-uniform masked S-box whereas our S-box is uniform allowing for a stronger security argument. Moreover, in the work of Shahmirzadi and Moradi, it is mentioned that "the application of changing of the guards on 2-share implementations to nullify the required fresh randomness does not seem trivial (or even possible)". As a result, our work indicates that this is possible. Table 2 covers the round-based implementations. The low-latency S-box reduces the latency of the whole encryption down to 42 cycles using design III. Although the same S-boxes are used in the round-based AES compared to the serialized implementations in Table 1, each estimated f_{max} is 3% to 20% lower than its serialized counterpart. The reason of this reduction is the wider bus in the round-based architecture making the fan-out in control logic cost an extra delay on the critical path. Design III costs 26.52% less area compared to the work by Sasdrich *et al.* [23]. Note that Sasdrich *et al.* used LUT-based Masked Dual-Rail with Pre-charge Logic (LMDPL) and included the mask-table generation circuit in the implementation.

7.2 Evaluation

In this section, we describe the practical evaluation of the three masked designs. We collect the amplified power traces from the measurement point on the Sakura-X board. The power measurements are amplified by a 30 dB SMA pre-amplifier [13]. The traces are captured by an oscilloscope at a sample rate of 500 MS/s while the FPGA is clocked at 6.144 MHz.

The non-specific leakage detection test from Goodwill *et al.* [14] is performed which

Table 2: Implementation cost of the round-based AES. Latency for the S-box is given in the number of register layers and for the AES in cycles.

Design	Area (S-box) [kGE]	Area (AES) [kGE]	Latency (S-box) [$regs.$]	Latency (AES) [cc]	Random[1] [bpc]	f_{max} [MHz][2]
Design I: Sec. 4.2	2.71	63.91	5	62	0	800
Design II: Sec. 4.3	4.33	102.44	4	52	0	505
Design III: Sec. 4.4	5.84	115.73	3	42	0	426
Gross *et al.* [16]	60.76	-	1	-	2 048	356
Gross *et al.* [16]	6.74	-	2	-	416	584
Sasdrich *et al.* [23]	3.48	157.50	1	10	720	400

1. Cost of online fresh random bits per cycle (*bpc*)
2. Depending on different standard cell libraries

verifies that our implementations do not show first-order leakage. The measured power traces are partitioned into two sets, where the first set $\mathcal{S}_0$ receives fixed plaintexts and the second set $\mathcal{S}_1$ contains random plaintexts. The two sets of measurements are compared using the t-test statistic:

$$t = \frac{\mu(\mathcal{S}_0) - \mu(\mathcal{S}_1)}{\sqrt{\frac{\sigma^2(\mathcal{S}_0)}{|\mathcal{S}_0|} + \frac{\sigma^2(\mathcal{S}_1)}{|\mathcal{S}_1|}}} \tag{8}$$

The t-test verifies whether the two sets have the same first-order moment. Its null hypothesis H_0 states that "the sets $\mathcal{S}_0$ and $\mathcal{S}_1$ are drawn from populations with the same mean." Large absolute values of this t-statistic indicate that the null hypothesis can be rejected with a high degree of confidence. The threshold value of the t-test commonly used by the side-channel research community is 4.5. If the t-test value of the measured power trace grows over 4.5, the implementation under test is considered as insecure.

Figure 9, 11, and 12 illustrate the TVLA result of the three serialized masked AES implementations. Figure 10 shows the maximum univariate t-test value changing over time which is denoted by the number of traces. The first-order probing security is verified physically for the implementations. The second-order leakages shows as anticipated. The exception is the third order leakage for design I which does not leak due to the noise amplification for third-order moments.

8 Conclusion

We proposed three first-order secure maskings of the AES with a different number of shares and register stages. These maskings allow the area versus latency trade-off in the hardware design of AES. The maskings use the generalized changing of the guards technique, which allows for the re-use of their randomness between the S-boxes. As a result, all designs proposed in this paper do not require fresh randomness in their calculation.

The three variations of the masked AES S-boxes show the area versus latency trade-off. Design I needs 5 cycles to compute the S-box and achieves a low-area serialized AES which costs only 8.13 kGE. Design II balances the area and latency cost. It reduces the S-box latency to 4 cycles and increases the area cost to 12.75 kGE. Design III is made for low latency applications, which requires 3 cycles for the S-box and costs 13.66 kGE for the serialized AES encryption. The designs do not require online randomness and are proven to be probing secure. The maskings are also implemented on FPGA and tested in practice. The TVLA results verify that the designs are first-order secure.

Acknowledgments. This work was supported by CyberSecurity Research Flanders with reference number VR20192203. Siemen Dhooghe is supported by a PhD Fellowship from

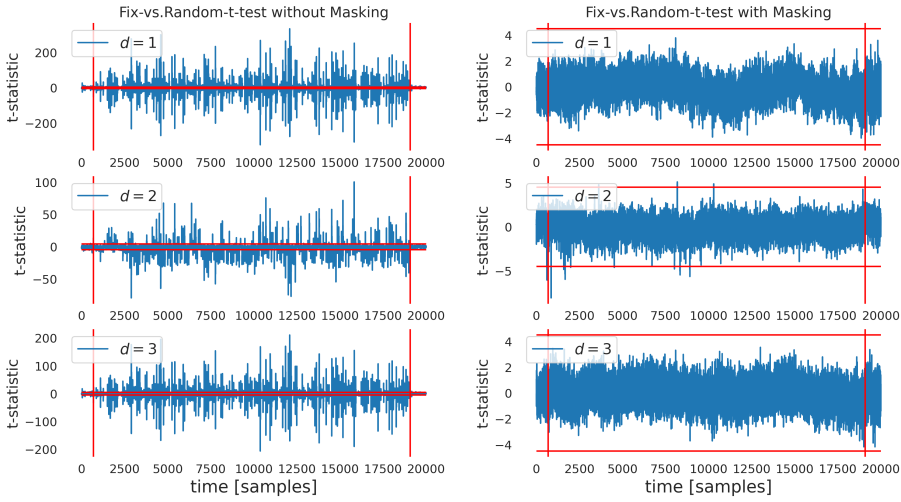
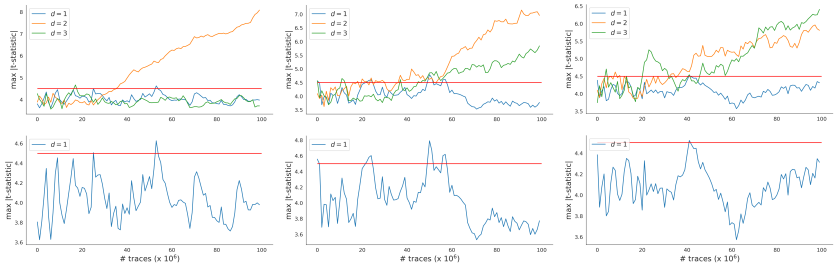


Figure 9: Univariate fixed-vs.-random t-test results for the serialized AES-128 encryption for design I. The right three plots show the results for the first to the third order statistical moments of the masked implementations at 10^8 traces. The results in the left column are for the unmasked implementations at 10^4 traces. The regions of interest are between vertical red lines, which indicate the start and end of the AES encryption.



(a) Design I. (b) Design II. (c) Design III.

Figure 10: The Y-axes of each subfigure is the maximum value of the univariate t-test results for the serialized AES-128 designs in this paper. The X-axes are the numbers of traces. The first two figures show the maximum t-test values of the first to the third statistical moments. The last one shows the first-order maximum t-test values only.

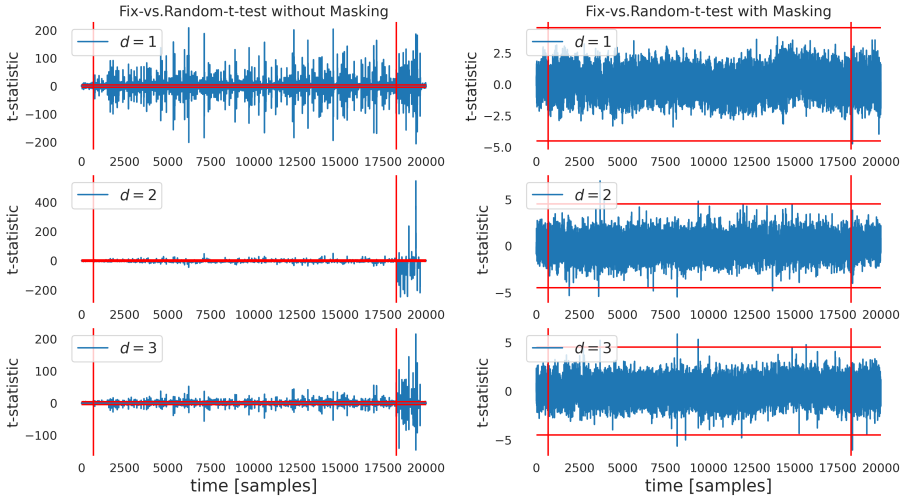


Figure 11: Univariate fixed-vs.-random t-test results for the serialized AES-128 encryption for design II. The layout is the same as in Figure 9.

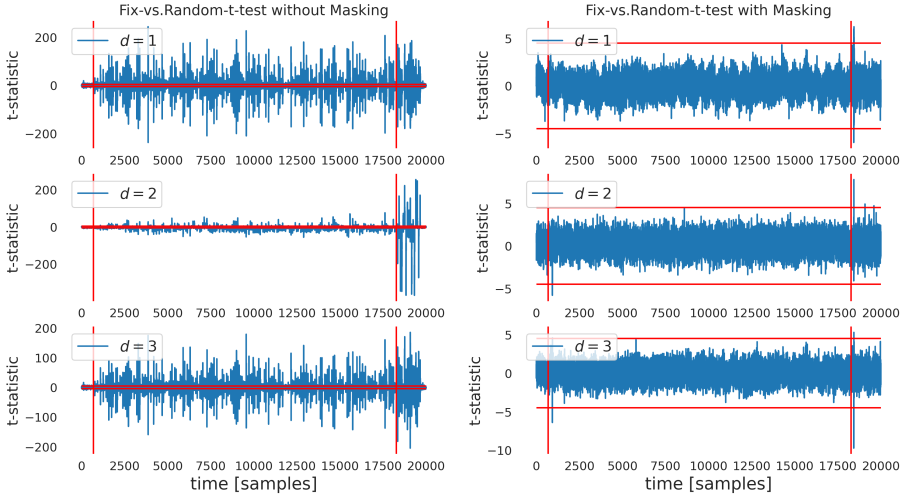


Figure 12: Univariate fixed-vs.-random t-test results for the serialized AES-128 encryption for design III. The layout is the same as in Figure 9.

the Research Foundation - Flanders (FWO). Zhenda Zhang is funded by a research grant from KU Leuven.

References

[1] Beyne, T., Dhooghe, S., Zhang, Z.: Cryptanalysis of masked ciphers: A not so random idea. In: Moriai, S., Wang, H. (eds.) ASIACRYPT 2020, Part I. LNCS, vol. 12491, pp. 817–850. Springer, Cham (Dec 2020). https://doi.org/10.1007/978-3-030-64837-4_27

[2] Bilgin, B., Gierlichs, B., Nikova, S., Nikov, V., Rijmen, V.: Higher-order threshold implementations. In: Sarkar, P., Iwata, T. (eds.) ASIACRYPT 2014, Part II. LNCS, vol. 8874, pp. 326–343. Springer, Berlin, Heidelberg (Dec 2014). https://doi.org/10.1007/978-3-662-45608-8_18

[3] Bilgin, B., Gierlichs, B., Nikova, S., Nikov, V., Rijmen, V.: Trade-offs for threshold implementations illustrated on AES. IEEE Trans. on CAD of ICs and Systems **34**(7), 1188–1200 (2015). https://doi.org/10.1109/TCAD.2015.2419623, https://doi.org/10.1109/TCAD.2015.2419623

[4] Canright, D.: A very compact S-box for AES. In: Rao, J.R., Sunar, B. (eds.) CHES 2005. LNCS, vol. 3659, pp. 441–455. Springer, Berlin, Heidelberg (Aug / Sep 2005). https://doi.org/10.1007/11545262_32

[5] Chari, S., Jutla, C.S., Rao, J.R., Rohatgi, P.: Towards sound approaches to counteract power-analysis attacks. In: Wiener, M.J. (ed.) CRYPTO'99. LNCS, vol. 1666, pp. 398–412. Springer, Berlin, Heidelberg (Aug 1999). https://doi.org/10.1007/3-540-48405-1_26

[6] Daemen, J.: Changing of the guards: a simple and efficient method for achieving uniformity in threshold sharing. Cryptology ePrint Archive, Report 2016/1061 (2016), https://eprint.iacr.org/2016/1061

[7] Daemen, J.: Changing of the guards: A simple and efficient method for achieving uniformity in threshold sharing. In: Fischer, W., Homma, N. (eds.) CHES 2017. LNCS, vol. 10529, pp. 137–153. Springer, Cham (Sep 2017). https://doi.org/10.1007/978-3-319-66787-4_7

[8] Daemen, J., Rijmen, V.: Advanced Encryption Standard (AES). National Institute of Standards and Technology (NIST), FIPS PUB 197, U.S. Department of Commerce (Nov 2001)

[9] De Cnudde, T., Reparaz, O., Bilgin, B., Nikova, S., Nikov, V., Rijmen, V.: Masking AES with d+1 shares in hardware. In: Gierlichs, B., Poschmann, A.Y. (eds.) CHES 2016. LNCS, vol. 9813, pp. 194–212. Springer, Berlin, Heidelberg (Aug 2016). https://doi.org/10.1007/978-3-662-53140-2_10

[10] De Meyer, L., Reparaz, O., Bilgin, B.: Multiplicative masking for AES in hardware. IACR TCHES **2018**(3), 431–468 (2018). https://doi.org/10.13154/tches.v2018.i3.431-468, https://tches.iacr.org/index.php/TCHES/article/view/7282

[11] Dhooghe, S., Nikova, S., Rijmen, V.: Threshold implementations in the robust probing model. In: Bilgin, B., Petkova-Nikova, S., Rijmen, V. (eds.) Proceedings of ACM Workshop on Theory of Implementation Security, TIS@CCS 2019, London, UK, November 11, 2019. pp. 30–37. ACM (2019). https://doi.org/10.1145/3338467.3358949, https://doi.org/10.1145/3338467.3358949

[12] Faust, S., Grosso, V., Merino Del Pozo, S., Paglialonga, C., Standaert, F.X.: Composable masking schemes in the presence of physical defaults & the robust probing model. IACR TCHES **2018**(3), 89–120 (2018). https://doi.org/10.13154/tches.v2018.i3.89-120, https://tches.iacr.org/index.php/TCHES/article/view/7270

[13] GmbH, L.E.T.: Langer emv - pa 303 sma, preamplifier 100 khz up to 3 ghz

[14] Goodwill, G., Jun, B., Jaffe, J., Rohatgi, P.: A testing methodology for side-channel resistance validation (September 2011)

[15] Goubin, L., Patarin, J.: DES and differential power analysis (the "duplication" method). In: Koç, Çetin Kaya., Paar, C. (eds.) CHES'99. LNCS, vol. 1717, pp. 158–172. Springer, Berlin, Heidelberg (Aug 1999). https://doi.org/10.1007/3-540-48059-5_15

[16] Gross, H., Iusupov, R., Bloem, R.: Generic low-latency masking in hardware. IACR TCHES **2018**(2), 1–21 (2018). https://doi.org/10.13154/tches.v2018.i2.1-21, https://tches.iacr.org/index.php/TCHES/article/view/871

[17] Ishai, Y., Sahai, A., Wagner, D.: Private circuits: Securing hardware against probing attacks. In: Boneh, D. (ed.) CRYPTO 2003. LNCS, vol. 2729, pp. 463–481. Springer, Berlin, Heidelberg (Aug 2003). https://doi.org/10.1007/978-3-540-45146-4_27

[18] Kocher, P.C., Jaffe, J., Jun, B.: Differential power analysis. In: Wiener, M.J. (ed.) CRYPTO'99. LNCS, vol. 1666, pp. 388–397. Springer, Berlin, Heidelberg (Aug 1999). https://doi.org/10.1007/3-540-48405-1_25

[19] Lab./UEC, S.: Sakura (sasebo-giii) (2014), https://satoh.cs.uec.ac.jp/SAKURA/hardware/SAKURA-X.html

[20] NANGATE: The NanGate 45nm Open Cell Library, version: PDKv1.3_v2010_12.Apache.CCL. Available at https://github.com/The-OpenROAD-Project/OpenROAD-flow-scripts/tree/master/flow/platforms/nangate45

[21] Nikova, S., Rechberger, C., Rijmen, V.: Threshold implementations against side-channel attacks and glitches. In: Ning, P., Qing, S., Li, N. (eds.) ICICS 06. LNCS, vol. 4307, pp. 529–545. Springer, Berlin, Heidelberg (Dec 2006). https://doi.org/10.1007/11935308_38

[22] Reparaz, O., Bilgin, B., Nikova, S., Gierlichs, B., Verbauwhede, I.: Consolidating masking schemes. In: Gennaro, R., Robshaw, M.J.B. (eds.) CRYPTO 2015, Part I. LNCS, vol. 9215, pp. 764–783. Springer, Berlin, Heidelberg (Aug 2015). https://doi.org/10.1007/978-3-662-47989-6_37

[23] Sasdrich, P., Bilgin, B., Hutter, M., Marson, M.E.: Low-latency hardware masking with application to AES. IACR TCHES **2020**(2), 300–326 (2020). https://doi.org/10.13154/tches.v2020.i2.300-326, https://tches.iacr.org/index.php/TCHES/article/view/8553

[24] Shahmirzadi, A.R., Moradi, A.: Second-order SCA security with almost no fresh randomness. IACR TCHES **2021**(3), 708–755 (2021). https://doi.org/10.46586/tches.v2021.i3.708-755, https://tches.iacr.org/index.php/TCHES/article/view/8990

[25] Sugawara, T.: 3-share threshold implementation of AES s-box without fresh randomness. IACR TCHES **2019**(1), 123–145 (2018). https://doi.org/10.13154/tches.v2019.i1.123-145, https://tches.iacr.org/index.php/TCHES/article/view/7336

[26] Wegener, F., Moradi, A.: A first-order SCA resistant AES without fresh randomness. In: Fan, J., Gierlichs, B. (eds.) COSADE 2018. LNCS, vol. 10815, pp. 245–262. Springer, Cham (Apr 2018). https://doi.org/10.1007/978-3-319-89641-0_14

Statement on the use of Generative AI

I did use the generative AI assistance tools, ChatGPT and Grammarly Pro. Grammarly Pro is a grammar checking tool with Gen AI backend, which was used for correcting mistakes in this writing. ChatGPT was used for grammar checks and as a search engine to learn about a particular topic, i.e., information about industrial standards and technology development.
The text, code, and images in this thesis are my own (unless otherwise specified). Generative AI has only been used in accordance with the KU Leuven guidelines and appropriate references have been added. I have reviewed and edited the content as needed and I take full responsibility for the content of the thesis.

Curriculum Vitae

ORCID: 0000-0002-6508-0350

Education

Ph.D. in Electrical Engineering, KU Leuven, Belgium
Dec 2020 – Aug 2025

M.Sc. in Electrical Engineering, KU Leuven, Belgium
Sept 2014 – Sept 2017

B.Sc. in Microelectronics, Fudan University, China
Sept 2010 – July 2014

Professional Experience

Research Assistant, COSIC, KU Leuven
Jan 2020 – Dec 2020

Secure Development Analyst, Worldline NV/SA, Belgium
Jan 2018 – Jan 2020

List of Publications

1. **Zhenda Zhang**, Svetla Nikova, and Ventzislav Nikov, "Glitch-Stopping Circuits: Hardware Secure Masking without Registers," *Proceedings of the 2024 ACM Conference on Computer and Communications Security (CCS)*, ACM, 2024. https://doi.org/10.1145/3658644.3670335 https://ia.cr/2024/891

2. Amund Askeland, Siemen Dhooghe, Svetla Nikova, Vincent Rijmen, and **Zhenda Zhang**, "Guarding the First Order: The Rise of AES Maskings," *Proceedings of the Smart Card Research and Advanced Application Conference (CARDIS)*, 2022. https://doi.org/10.1007/978-3-031-25319-5_6. https://ia.cr/2021/734

3. Victor Arribas, **Zhenda Zhang**, and Svetla Nikova, "LLTI: Low-Latency Threshold Implementations," *IEEE Transactions on Information Forensics and Security*, vol. 16, pp. 5108–5123, 2021. https://doi.org/10.1109/TIFS.2021.3123527. https://ia.cr/2022/418

4. Tim Beyne, Siemen Dhooghe, and **Zhenda Zhang**, "Cryptanalysis of Masked Ciphers: A Not So Random Idea," *Advances in Cryptology – ASIACRYPT 2020*, Lecture Notes in Computer Science, Springer, 2020. https://doi.org/10.1007/978-3-030-64837-4_27. https://ia.cr/2020/993.

5. Fereshteh Poormohammadi, Merijn Van Deyck, Martijn Deckers, Abdul Saboor, Bowen Wang, Pouya Mehrjouseresht, **Zhenda Zhang**, Arne Symons, Pieter Pas, Alexander Bodard, Hans Van Rooij, Marian Verhelst, Alexander Bertrand, Ruth Vazquez Sabariego, Panagiotis Patrinos, and Peter Coppens, "The 'Eagle' Approach to Train Electrical Engineers with Collaborative Problem-Solving Skills," *Proceedings of the 51st Annual Conference of the European Society for Engineering Education (SEFI)*, 2023. https://doi.org/10.21427/3NC0-AB72

FACULTY OF ENGINEERING SCIENCE
DEPARTMENT OF ELECTRICAL ENGINEERING
Kasteelpark Arenberg 10, bus 2452
B-3001 Leuven
zhenda.zhang@esat.kuleuven.be
https://www.esat.kuleuven.be/cosic/

