KU LEUVEN

ARENBERG DOCTORAL SCHOOL
Faculty of Engineering Science

Efficient Hardware Masking Using Heuristic and Formal Methods

Dilip Kumar Shanmugasundaram Veeraraghavan

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

Supervisors:
Prof. dr. ir. Ingrid Verbauwhede
Dr. Benedikt Gierlichs

September 2025

EFFICIENT HARDWARE MASKING USING HEURISTIC AND FORMAL METHODS

Dilip Kumar SHANMUGASUNDARAM VEERARAGHAVAN

Supervisors:
Prof. dr. ir. Ingrid Verbauwhede
Dr. Benedikt Gierlichs

Members of the
Examination Committee:
Prof. dr. ir. Hendrik Van Brussel, chair
Prof. dr. ir. Wim Dehaene
Dr. Svetla Petkova-Nikova
Dr. Josep Balasch
(Rambus, The Netherlands)
Prof. dr. Amir Moradi
(TU Darmstadt, Germany)

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

September 2025

© 2025 Dilip Kumar Shanmugasundaram Veeraraghavan
Uitgegeven in eigen beheer, Dilip Kumar Shanmugasundaram Veeraraghavan, Kasteelpark Arenberg 10, bus 2452, B-3001 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

First, I would like to thank Prof. Ingrid Verbauwhede for giving me the opportunity to join COSIC as an intern in the summer of 2018 and later for allowing me to pursue my PhD in the group. I would also like to thank Benedikt and Josep for their guidance and for giving me the freedom to work on the problems of my choosing.

I would like to thank the members of my jury for their time and effort in reading and evaluating this thesis: Prof. Hendrik Van Brussel, Prof. Wim Dehaene, Dr. Svetla Petkova-Nikova, Dr. Josep Balasch, and Prof. Amir Moradi.

I am grateful to my colleagues at COSIC for their support throughout the years, through many discussions, help with setting up experiments in the hardware lab, assistance with administrative matters, and the many other forms of support that made this work possible.

Lastly, I am thankful to my family.

Abstract

Cryptographic hardware is integral to modern digital infrastructures, ranging from embedded IoT devices to high-performance processors, where it protects sensitive information across numerous applications. Despite the strong theoretical security guarantees offered by cryptographic algorithms, their physical implementations remain vulnerable to side-channel attacks, which exploit unintended leakages such as power consumption or electromagnetic emissions to compromise secret data. In response, masking techniques have emerged as essential hardware countermeasures. These methods distribute sensitive data across multiple randomized shares, effectively mitigating such attacks. However, practical hardware implementations of masking often impose substantial costs, specifically in terms of area overhead, randomness consumption, and increased latency, particularly in resource-constrained settings.

This thesis addresses these challenges by investigating the design of efficient hardware masking techniques through two distinct yet complementary research tracks. The first track develops heuristic approaches, focusing on practical applicability and efficient resource usage based on empirical evaluation. We evaluate these designs through concrete hardware implementations and extensive leakage assessments, demonstrating their suitability for real-world scenarios.

The second track focuses on masking schemes with formally provable security guarantees, with latency as the principal design constraint. We introduce Time Sharing Masking (TSM), a novel masking scheme designed specifically for low-latency applications while satisfying robust formal security properties. We subsequently extend this concept to Higher-Order Time Sharing Masking (HO-TSM), achieving higher-order security while preserving low latency. In addition, we propose trade-off constructions that balance reductions in area and randomness with minimal impact on latency. Each construction is formally verified using automated tools to confirm compliance with established leakage models.

Overall, this thesis presents a systematic study of both practical and formal approaches to hardware masking, with a focus on improving efficiency under real-world constraints.

Beknopte samenvatting

Cryptografische hardware is een integraal onderdeel van moderne digitale infrastructuren, variërend van ingebedde IoT-apparaten tot krachtige processors, waar het gevoelige informatie in tal van toepassingen beschermt. Ondanks de sterke theoretische veiligheidsgaranties die cryptografische algoritmes bieden, blijven hun fysieke implementaties kwetsbaar voor zijkanaal-aanvallen, die misbruik maken van onbedoelde lekken, zoals stroomverbruik of elektromagnetische straling, om geheime gegevens te compromitteren. Als reactie hierop zijn maskeertechnieken in opkomst als essentiële hardware-tegenmaatregelen. Deze methoden verdelen gevoelige gegevens over meerdere willekeurige delen, waardoor dergelijke aanvallen op efficiënte wijze worden beperkt. Praktische hardware-implementaties van masking brengen echter vaak aanzienlijke kosten met zich mee, met name in termen van oppervlakteoverhead, de noodzaak voor willekeurige getallen en verhoogde latentie, vooral in omgevingen met beperkte middelen.

Dit proefschrift gaat deze uitdagingen aan door het ontwerp van efficiënte hardwaremaskeringstechnieken te onderzoeken via twee verschillende maar complementaire onderzoekstrajecten. Het eerste traject ontwikkelt heuristische benaderingen, waarbij de nadruk ligt op praktische toepasbaarheid en efficiënt gebruik van middelen op basis van empirische evaluatie. We evalueren deze ontwerpen aan de hand van concrete hardware-implementaties en uitgebreide lekbeoordelingen, waarmee we aantonen dat ze geschikt zijn voor praktijktoepassingen.

Het tweede onderzoekstraject richt zich op maskeringstechnieken met formeel aantoonbare veiligheidsgaranties, waarbij latentie de belangrijkste ontwerpbeperking is. We introduceren Time Sharing Masking (TSM), een nieuwe maskeringstechniek die speciaal ontworpen is voor toepassingen met lage latentie en tegelijkertijd robuuste formele veiligheidseigenschappen biedt. Vervolgens breiden we dit concept uit tot Higher-Order Time Sharing Masking (HO-TSM), waarmee een hogere veiligheid wordt bereikt zonder dat dit ten koste gaat van de

lage latentie. Daarnaast stellen we trade-off-constructies voor die een evenwicht bieden tussen vermindering van gebruikte oppervlakte en vermindering van de noodzaak aan willekeurige getallen, en dit met minimale impact op de latentie. Elke constructie wordt formeel geverifieerd met behulp van geautomatiseerde tools om de conformiteit met gevestigde lekkagemodellen te bevestigen.

In het algemeen presenteert dit proefschrift een systematische studie van zowel praktische als formele benaderingen van hardwaremaskering, met de nadruk op het verbeteren van de efficiëntie onder reële beperkingen.

List of Abbreviations

ADC	Analog-to-Digital Converter.
ANF	Algebraic Normal Form.
ASIC	Application-Specific Integrated Circuit.
AWDDL	Asynchronous Wave Dynamic Differential Logic.
BDD	Binary Decision Diagram.
BEL	Basic ELement (FPGA placement primitive).
CMOS	Complementary Metal–Oxide Semiconductor.
CPA	Correlation Power Analysis.
CPU	Central Processing Unit.
DES	Data Encryption Standard.
DOM	Domain-Oriented Masking.
DPA	Differential Power Analysis.
DRP	Dual-Rail Pre-Charge Logic.
EDA	Electronic Design Automation.
FF	Flip-Flop.
FPGA	Field-Programmable Gate Array.
FSM	Finite-State Machine.
GE	Gate Equivalent (area metric).

GHPC	Generic Hardware Private Circuits.
GHPC_{LL}	Low-Latency version of GHPC
GLM	Generic Low-Latency Masking.
HDL	Hardware Description Language.
HO-TSM	Higher-Order Time-Sharing Masking.
HPC	Hardware Private Circuits.
LFSR	Linear Feedback Shift Register.
LLTI	Low-Latency Threshold Implementation.
LMDPL	LUT-based Masked Dual-Rail with Pre-Charge Logic.
LUT	Look-Up Table.
MDPL	Masked Dual-Rail Pre-Charge Logic.
MIA	Mutual Information Analysis.
MUX	Multiplexer.
NAND	NAND Gate (logical conjunction with inversion).
NI	Non-Interference (formal security notion).
NOR	NOR Logic Gate.
OR	OR Gate (logical disjunction).
PINI	Probe-Isolating Non-Interference (formal security notion).
PLL	Phase-Locked Loop.
PRNG	Pseudorandom Number Generator.
RFID	Radio-Frequency Identification.
RNG	Random Number Generator.
RTL	Register-Transfer Level.
SAKURA	SAKURA Evaluation Board Series.

SCA	Side-Channel Attack.
SESYM	Self-Synchronized Masking.
SNI	Strong Non-Interference (formal security notion).
SPA	Simple Power Analysis.
SSA	Static Single Assignment (intermediate form).
TDES	Triple Data Encryption Algorithm.
TI	Threshold Implementation.
TRNG	True Random Number Generator.
TSM	Time-Sharing Masking.
TVLA	Test Vector Leakage Assessment.
WDDL	Wave Dynamic Differential Logic.
XOR	Exclusive OR.

Contents

Abstract iii

Beknopte samenvatting v

List of Abbreviations vii

Contents xi

List of Figures xiii

I Foundations and Context 1

1 Introduction 3
- 1.1 Thesis Contributions 4
- 1.2 Thesis Structure 5

2 Background and Theoretical Foundations 7
- 2.1 Side-Channel Attacks and Early Countermeasures 8
  - 2.1.1 Masking 8
  - 2.1.2 Hiding 9
- 2.2 Masking in Hardware and the Problem of Glitches 11
  - 2.2.1 The Problem of Glitches in Hardware 11
  - 2.2.2 Threshold Implementations (TI) 13
  - 2.2.3 Domain-Oriented Masking (DOM) 15
- 2.3 Efficiency Considerations in Hardware Masking: Area, Randomness, and Latency 17
- 2.4 Low-Latency Masking in Hardware 19
  - 2.4.1 Circuit-Level Approaches 20
  - 2.4.2 Scheme-Level Approaches 22
- 2.5 Formal Security Notions for Masking Schemes 23

2.5.1 The Probing Model 23
2.5.2 From Global Guarantees to Modular Security Notions . 24
2.5.3 Robust Probing Model for Hardware 29
2.6 Formal Verification Tools for Masked Hardware Designs 29
2.7 Conclusion 32

3 Research Problem and Thesis Scope 33
3.1 Limitations of Existing Approaches 34
3.1.1 Early Heuristic Techniques 34
3.1.2 Modern Provably Secure Schemes 36
3.2 Research Questions 37
3.3 Research Objectives and Methodology 39
3.3.1 Practical Security Track: Heuristic Masking Approach . 39
3.3.2 Provable Low-Latency Track: Formal Masking Approach 40
3.4 Thesis Structure 41

4 Metrics, Testing Methods, and Tools 43
4.1 Efficiency Metrics 43
4.2 Side-Channel Resistance Metrics 45
4.2.1 Test Vector Leakage Assessment (TVLA) 45
4.2.2 Attack-Based Metrics 47
4.3 Formal Verification 51
4.3.1 Manual Security Proofs 51
4.3.2 Tool-Assisted Formal Verification 53
4.4 Conclusion 54

5 Conclusion and Future Work 55

II Publications 83

6 Low-Cost First-Order Secure Boolean Masking in Glitchy Hardware 87

7 Low-Cost First-Order Secure Boolean Masking in Glitchy Hardware - Full Version* 95

8 Time Sharing - A Novel Approach to Low-Latency Masking 131

9 Higher-Order Time Sharing Masking 167

Curriculum Vitae 215

List of Figures

2.1 Wave Dynamic Differential Logic (WDDL) representation of an AND gate with complementary paths, including precharge and dual-rail evaluation stages. Adapted from [54]. 10
2.2 DOM-*indep* multiplier. 16

Part I

Foundations and Context

Chapter 1

Introduction

Cryptographic hardware has become deeply integrated into modern technology, with applications ranging from smartcards [1, 2] and Internet-of-Things (IoT) devices [3, 4, 5] to secure enclaves in modern processors [6]. Although modern cryptographic algorithms offer strong theoretical security, these assurances are based on the assumption that an attacker only has access to the algorithm's inputs and outputs, without any influence over or insight into the physical processes occurring inside the device [7, 8, 9]. This idealized setting corresponds to a black-box model, in which the adversary has oracle access to inputs and outputs only, with no observation or control of the internal computation.

In practice, however, an adversary can often observe physical leakages from a cryptographic device, such as power consumption [10, 11], electromagnetic emissions [12, 13], or execution timing [14], and exploit this side-channel information to compromise the cryptosystem. Allowing such observations places the problem in a grey-box model: the adversary retains the same I/O access and, in addition, can measure leakage correlated with intermediate values. Such side-channel attacks circumvent algorithmic security, demonstrating that even a theoretically unbreakable algorithm may be compromised if its physical implementation inadvertently discloses information [15, 16]. This reality has firmly established that algorithmic strength alone is insufficient when an adversary can observe physical leakages. Physical access to a cryptographic device fundamentally alters the threat landscape, introducing new attack surfaces that extend beyond the scope of classical cryptanalysis [17].

Defending against such side-channel attacks requires countermeasures implemented at the hardware [18] or software level [19]. One prominent strategy is masking [20], which involves splitting each secret value into multiple random

shares using a secret-sharing scheme and performing computations exclusively on these masked shares. This approach significantly reduces the statistical dependency between observable side-channel signals and the sensitive data. Because the device operates on these randomized shares rather than directly on the raw secret, any leaked intermediate values ideally appear indistinguishable from random noise to an attacker. However, masking does not come for free. It typically incurs significant implementation costs. Hardware designs with masking require additional circuitry to process multiple shares [21, 22], large amounts of fresh randomness to refresh masks [23, 24], and often extra clock cycles [25, 26] to manage more complex data flows. In short, while masking can provide strong protection against side-channel analysis, it does so at the expense of efficiency in terms of chip area, randomness usage, and execution speed.

Over the past two decades, extensive research on masking techniques has matured significantly. Today, masking is well-studied with clear theoretical proofs [27, 28, 29] and substantial empirical evidence supporting its effectiveness [25, 30]. Various masking schemes exist, each offering distinct trade-offs. At this stage, however, the central research challenge is optimizing overhead, specifically, reducing chip area, randomness consumption, and latency, without sacrificing security.

Despite steady improvements, a notable optimization gap remains: latency, defined as the number of clock cycles needed to produce a cryptographic result, has not kept pace with reductions in area and randomness usage. Historically, latency optimization was a secondary concern; early masking constructions prioritized provable security, accepting multi-cycle operations or deep pipelines as necessary trade-offs. Today, however, industries such as secure communications [31], automotive [32, 33], and IoT [34] increasingly require cryptographic engines with stringent throughput and response-time constraints. This shift has elevated latency optimization into a critical, yet largely unresolved, research problem.

1.1 Thesis Contributions

This thesis addresses the central challenge of achieving strong side-channel security in cryptographic hardware without incurring prohibitive implementation overhead, particularly latency. To this end, the contributions are structured into two complementary research tracks:

Contribution 1 – Practical Masking Framework (Efficiency Optimization). In the first track, specifically addressed in Chapters 6 and 7, the focus

lies on heuristic, empirically validated masking techniques aimed at achieving real-world efficiency. The goal of this track is to minimize overall implementation costs, concentrating on factors such as chip area, randomness consumption, and latency, while maintaining a satisfactorily high level of resistance to side-channel attacks. The emphasis is on practical design strategies that perform effectively in hardware, even in the absence of formal proofs of security. This approach deliberately challenges the overly conservative practices often adopted in traditional masking schemes, seeking instead to explore how far efficiency can be optimized without compromising security. The objective is to leverage engineering intuition and empirical feedback to develop masked implementations that are as efficient as possible, including the relaxation of design constraints where experimental evidence indicates acceptable safety margins. The security of the proposed constructions is demonstrated and validated through physical experiments, confirming their resilience against side-channel attacks up to a targeted level of complexity.

Contribution 2 – Provably Secure Low-Latency Gadgets. The second track, described in Chapters 8 and 9, focuses on masking techniques that provide provable security while aiming for low latency. This work relies on formal cryptographic methods and security proofs. We develop new masking schemes and gadgets that are formally proven secure and specifically designed to operate in fewer clock cycles than traditional approaches. These designs are thoroughly verified using automated formal verification tools. Additionally, we present case studies integrating these gadgets into complete cryptographic cores such as PRINCE [35] and AES [36], demonstrating clear efficiency improvements over existing low-latency solutions.

1.2 Thesis Structure

This thesis is written in a publication-based format. The chapters in the first half (Chapters 2 to 5) are structured to introduce and connect the research papers included in the latter part of the dissertation. In particular, the core contributions described in the latter half (Chapters 6 to 9) correspond to work that has been published in peer-reviewed conference and journal proceedings. For the reader's convenience, these publications are appended in full after the main chapters of the thesis. Cross-references between the narrative in the first half and the appended papers in the second half are clearly indicated where necessary. Specifically, Chapters 2 to 5 provide a connected story, bringing together results from the embedded papers and placing them within the broader hardware security context, while the appended papers at the end of the thesis present the complete details of those results as published.

In the first half, Chapter 2 traces the history of side-channel attacks and countermeasures and surveys the state of the art in side-channel countermeasures, with a particular focus on masking techniques and the formal security notions used to analyze their effectiveness. Chapter 3 then defines the scope of the research and formulates the guiding questions addressed by this thesis, outlining the core problems and the dual approach taken to solve them as described in Section 1.1. Chapter 4 defines and explains the performance and security evaluation metrics used throughout the thesis, outlining how implementation efficiency and side-channel resistance are quantitatively assessed. Chapter 5 provides a discussion of the insights gained from comparing the two design paradigms and offers perspectives on future research directions.

In the latter half, Chapters 6 and 7 present the proposed practical masking track. They introduce the secAND family of gadgets, detail the design of a first-order secure DES [37] implementation built from these gadgets, and describe the evaluation setup, including performance metrics and leakage testing methodology. Chapter 8 shifts to the provably secure track, introducing the Time-Sharing Masking (TSM) scheme for low-latency, first-order secure circuits and providing formal security proofs alongside implementation case studies. Chapter 9 extends this formal approach to higher-order security, describing the HO-TSM constructions, their security proofs, and variations that explore area-latency trade-offs with applications to AES.

Chapter 2

Background and Theoretical Foundations

In this chapter, we provide a comprehensive overview of early cryptographic side-channel attacks, their impact on hardware security, and the countermeasures developed to mitigate them. We begin with foundational research on side-channel attacks, followed by an examination of efficiency trade-offs in countermeasures, focusing on area, randomness, and latency. We discuss both practical approaches, validated through empirical testing, and formal security frameworks, which offer provable security guarantees. This chapter establishes the necessary context for the discussion of our research in subsequent chapters.

In evaluating countermeasures, we consider not only security but also implementation efficiency, measured primarily in terms of area, latency, and randomness [7, 38]. Here, *area* refers to the hardware resources required, typically quantified in gate equivalents (GE) for ASIC designs [39] or in logic elements (e.g. LUT, flip-flop, etc.) for FPGA implementations [40]. *Latency* , in the context of this thesis, denotes the number of clock cycles needed to complete a cryptographic operation, with lower latency enabling faster performance crucial for real-time applications. *Randomness consumption* captures the number of fresh random bits required by a countermeasure, as excessive randomness can increase design complexity and resource usage [41], especially in constrained environments. These metrics are central to the comparisons and analyses presented in this chapter and throughout the thesis.

2.1 Side-Channel Attacks and Early Countermeasures

Side-channel attacks exploit unintended information leakages, such as power consumption, electromagnetic emanations, and execution timing, from cryptographic devices. One of the most prominent examples is Differential Power Analysis (DPA) [11], where an adversary statistically analyzes power traces collected during device operation to deduce secret keys. These techniques are capable of breaking implementations of both symmetric and asymmetric cryptographic algorithms by correlating observed power consumption with internal logic computations.

The threat posed by side-channel attacks was first identified by Kocher et al. in the late 1990s [11], when they introduced *Differential Power Analysis (DPA)*, a simple, cost-effective, and non-invasive technique that exploits variations in power consumption to extract cryptographic secrets. Kocher et al. demonstrated that even standardized and mathematically secure algorithms such as DES could be compromised via physical leakages. Their work revealed a critical gap in conventional cryptographic assumptions: while algorithms operate on abstract secrets, real-world hardware leaks physical side-channel information. This insight fundamentally changed the field, highlighting that algorithmic strength alone is insufficient for secure implementations.

Kocher's findings spurred a wave of research into side-channel attacks/countermeasures [42, 43, 44, 45, 46] and marked the beginning of a new era in hardware security. From that point forward, preventing side-channel leakage became as essential as ensuring cryptographic soundness, laying the foundation for modern side-channel resistance techniques.

Initial countermeasures against side-channel attacks can be grouped into two main approaches. Fundamentally, these countermeasures seek to disrupt the correlation between secret data and observable side-channel signals. In this chapter, we introduce the two primary categories of countermeasures, namely *masking* and *hiding*, and trace the historical development of masking techniques, with particular emphasis on hardware implementations and the emergence of modern formal security notions.

2.1.1 Masking

Early works in the 2000s laid the foundation for masking, with Chari et al. [20], Goubin et al. [47], and others proposing masking as a principled countermeasure to mitigate side-channel leakage. Masking randomizes sensitive values by

splitting each secret variable into multiple *shares* using random masks. Rather than processing the secret value directly, the algorithm operates on its shares. For example, in *additive (Boolean) masking*, an 8-bit sensitive value x is split as $x = x_0 \oplus x_1$, where $x_0 = r$ is a random mask and $x_1 = x \oplus r$. Each individual share reveals no information about x, and computations are carried out on the masked representation, such that intermediate leakages remain independent of the unshared secret.

Masking increases the complexity of an attack because the adversary must combine leakages from multiple shares to reconstruct the secret, a method referred to as a *higher-order attack* [48, 49]. To protect against such attacks, the number of shares must be increased. More generally, a dth-order masking scheme requires at least $d + 1$ shares [50], ensuring that any combination of at most d shares is statistically independent of the secret.

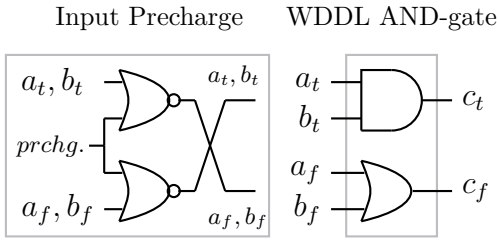
As a practical example, in a masked AES implementation, each byte of the secret key is stored as multiple shares. All operations, including the S-Box and MixColumns etc., are transformed into equivalent masked operations applied to these shares. In a first-order masked design, the leakage observed in a single power sample or on a single wire typically reflects the leakages from individual shares, such as $leak(x_0)$ or $leak(x_1)$, rather than any direct combination of the shares themselves. As a result, each share's leakage remains statistically independent of the secret value x, and a single observation does not reveal exploitable information about x. To bypass the masking, an adversary must combine leakage information from multiple measurements, for example through a higher-order attack such as second-order DPA [51, 52, 53].

Despite its effectiveness, masking typically incurs high implementation overhead. It requires the generation of fresh randomness and introduces additional logic to process each share securely, often resulting in increased area, power, and latency.

2.1.2 Hiding

Another early class of countermeasures explored in parallel to masking was *hiding*, which focuses on reducing or camouflaging the physical side-channel signal itself, thereby making side-channel analysis more difficult.

Techniques based on hiding generally fall into two categories: (1) signal reduction [55] and (2) noise injection [56, 57, 58]. Signal reduction focuses on designing hardware or software such that the power consumption is constant or balanced across operations. Dual-rail logic [59, 60] attempts to equalize the



(a) Logic gate diagram

a_t	b_t	a_f	b_f	c_t	c_f
0	0	1	1	0	1
0	1	1	0	0	1
1	0	0	1	0	1
1	1	0	0	1	0

(b) Truth table

Figure 2.1: Wave Dynamic Differential Logic (WDDL) representation of an AND gate with complementary paths, including precharge and dual-rail evaluation stages. Adapted from [54].

power consumption by ensuring that it is independent of the processed data, drawing the same amount of power in each cycle.

A well-known early example is Wave Dynamic Differential Logic (WDDL) [55], which implements each logic function using a pair of complementary gates to ensure constant switching activity. Figure 2.1a illustrates a complete WDDL AND gate, including the precharge stage and dual-rail logic. Each bit is encoded using two signals: a true rail (a_t, b_t) and a false rail (a_f, b_f). Every clock cycle is divided into a precharge phase, in which all rails are reset to zero, followed by an evaluation phase where exactly one rail switches depending on the input value. The AND gate processes the true inputs while the OR gate handles the false inputs, ensuring that either c_t or c_f switches, but never both. The truth table in Figure 2.1a confirms that only valid one-hot dual-rail combinations are permitted, with complementary outputs.

An important strength of WDDL [55] is that it is cryptographic-agnostic; the technique can be applied to arbitrary logic circuits, including cryptographic S-boxes, without requiring the designer to be aware of the specific algorithm being protected. Moreover, WDDL is inherently glitch-free by construction, as the dual-rail precharge mechanism ensures that all signals undergo exactly one controlled transition per cycle, independent of the input values.

However, Suzuki et al. [61] demonstrated that such dual-rail schemes suffer from the *early propagation effect*, where the timing of gate evaluations depends on the input, making the output leakage still exploitable [62, 63]. Additionally, hiding techniques are not easily supported by standard EDA flows and significantly increase both design complexity [64, 65] and area overhead [66, 67].

Noise-based techniques add random delays [68] or dummy operations [18] to

desynchronize the power trace. For instance, random instruction shuffling [69, 70] changes the execution order while maintaining throughput, or dedicated noise generators [71, 72] introduce unpredictable switching activity to obscure the actual computation. The goal is to reduce the correlation between secret data and power traces, thereby requiring an impractically large number of measurements to recover any leakage signal.

Despite their effectiveness, hiding techniques are not foolproof. Signal equalization cannot eliminate leakage entirely, and noise-based methods can be defeated by averaging over many traces. These approaches also introduce significant performance and area overhead. Conceptually, masking changes the data representation to reduce statistical dependence on secrets under certain assumptions about the leakage model and the physical implementation. In contrast, hiding techniques alter the physical leakage characteristics to make side-channel exploitation more difficult. Over the years, masking has become a systematic and widely studied approach in academic research because it enables rigorous mathematical analysis and security proofs within abstract leakage models, and it integrates naturally into cryptographic algorithm design. Nonetheless, in practical implementations, hiding remains an important class of countermeasures and is typically combined with masking to strengthen security [73, 74]. The remainder of this chapter will therefore focus on masking, while recognizing the complementary role of hiding in robust hardware security.

2.2 Masking in Hardware and the Problem of Glitches

Masking has been applied in both software and hardware, but its deployment in either domain presents unique challenges that must be addressed to ensure robust side-channel resistance. In particular, physical effects such as *glitches* can undermine the security guarantees of masking schemes when implemented in digital circuits. This section outlines the core problem of glitches in hardware and presents countermeasures developed to address them, notably Threshold Implementations (TI) [21, 75] and Domain-Oriented Masking (DOM) [22].

2.2.1 The Problem of Glitches in Hardware

To evaluate whether masked circuits ensure statistical independence between side-channel leakages and secret data, formal security models have been introduced to abstract physical leakage behavior. One of the most foundational among these is the *d-probing model*, introduced by Ishai et al. [76] in 2003. This

model captures the capabilities of an adversary who can observe up to d internal signals, known as *probes*, within a masked hardware circuit. Each probe reveals the value of a stable wire, representing an intermediate computation during circuit execution. A circuit is said to be d-probing secure if any set of at most d observed signals is statistically independent of the secret inputs. In other words, an attacker must observe more than d wires to infer any information about the sensitive data.

While the d-probing model effectively captures logical leakage under idealized conditions, it assumes that all internal signals are stable and unaffected by physical side effects. However, in real-world digital hardware, this assumption is often violated due to the presence of *glitches*. A glitch is a momentary, unintended transition on a signal before it stabilizes to its correct value [38]. Hardware is inherently parallel and subject to variations in gate and wire delays. These timing differences can cause gates to produce temporary incorrect outputs before settling, leading to transient behaviors known as glitches that are observable and exploitable.

Glitches arise primarily due to mismatched propagation delays on a gate's inputs [77]. In CMOS circuits, even when input signals change simultaneously at the logical level, they may arrive at different times due to electrical or structural delays. These effects are compounded in larger circuits, where glitches can propagate across multiple logic stages, a phenomenon referred to as *glitch propagation*. In the context of masking, this behavior is particularly dangerous. Although masking schemes are designed to prevent any gate from simultaneously processing multiple shares of the same secret, glitches can cause a gate's output to momentarily depend on multiple shares, thereby violating the security guarantee. This creates a leakage path that is not accounted for by the standard d-probing model.

Beyond glitches, *transitions* create leakage because in a synchronous design where a combinational block is driven by registers, a probe on any net after the block can depend on two successive register states. This is captured by a *transition-extended probing* view, where a probe may reference pairs of values across one clock boundary. Combined with glitch extension, a single probe on a combinational output can therefore correspond to both the stable wires in the combinational logic at the current cycle and the register values from the previous cycle that feed it. For masked designs, the combination of glitches and transitions must be carefully considered in looped, round-based architectures.

The problem of glitches in masked hardware implementations was first identified by Mangard et al. [78, 79] in 2005. Their work highlighted that naive application of masking is insufficient in hardware unless glitches are either prevented or explicitly accounted for in the security model. This realization led to the

development of masking strategies that are secure even in the presence of such transient behaviors [80, 21]. Among these, *Threshold Implementations (TI)* [21] emerged as a prominent approach, offering provable security under more realistic conditions.

2.2.2 Threshold Implementations (TI)

Threshold Implementations (TI) is a design methodology introduced by Nikova et al. [21] in 2006 to address the glitch problem for first-order secure hardware masking. The key idea is to structure the computation of masked shares in such a way that, even in the presence of glitches, any side-channel leakage remains independent of the underlying secrets. TI borrows concepts from secret sharing [81], where a sensitive variable is split into d shares, and the circuit is constructed such that any subset of up to $d-1$ shares remains statistically independent of the secret. In practice, TI requires that the function to be masked is decomposed into multiple sub-functions, and that each sub-function satisfies the following three properties:

- *Correctness:* The final combination of the output shares (typically via XOR) must yield the correct output of the original, unshared function. This ensures the masked computation is functionally equivalent to its unmasked counterpart.
- *Non-Completeness:* No sub-function may depend on all d shares of any single input. That is, each intermediate computation intentionally omits at least one share of each secret. This prevents glitches in a sub-function from combining all shares, which would otherwise risk reconstructing the sensitive value.
- *Uniformity:* The output shares must be uniformly distributed if the input shares are uniform. This avoids statistical biases that could lead to exploitable leakage in subsequent computations, especially in designs composed of multiple layers.

When these conditions are met, TI provides mathematically provable resistance against first-order DPA, even in the presence of glitches. To illustrate, consider the secure computation of a multiplication operation $z = x \cdot y$, where the inputs x and y are shared into four shares each:

$$x = x_0 \oplus x_1 \oplus x_2 \oplus x_3, \quad y = y_0 \oplus y_1 \oplus y_2 \oplus y_3.$$

The goal is to compute output shares z_i such that the final result $z = z_1 \oplus z_2 \oplus z_3 \oplus z_4$ satisfies correctness, while each sub-function computing z_i omits

at least one share of each input (non-completeness), and the output shares remain uniformly distributed given uniform inputs (uniformity). An example decomposition satisfying these properties, taken from [21], is as follows:

$$z_1 = (x_3 \oplus x_4)(y_2 \oplus y_3) \oplus y_2 \oplus y_3 \oplus y_4 \oplus x_2 \oplus x_3 \oplus x_4,$$

$$z_2 = (x_1 \oplus x_3)(y_1 \oplus y_4) \oplus y_1 \oplus y_3 \oplus y_4 \oplus x_1 \oplus x_3 \oplus x_4,$$

$$z_3 = (x_2 \oplus x_4)(y_1 \oplus y_4) \oplus y_2 \oplus x_2,$$

$$z_4 = (x_1 \oplus x_2)(y_2 \oplus y_3) \oplus y_1 \oplus x_1.$$

Each output share z_i is computed using a sub-function that intentionally omits at least one share of x and one share of y, thereby satisfying the non-completeness property. Furthermore, the structure of the equations is designed to ensure that, under uniformly random input shares, the output shares remain uniformly distributed, thereby achieving uniformity.

Minimizing the number of shares is crucial, as more shares lead to increased logic/area costs [82, 83]. Constructing a target function with the fewest shares while ensuring correctness, non-completeness, and uniformity is a key challenge. Achieving a correct, non-complete, and uniform sharing is widely perceived as a significant research problem [84, 85]. Several works have explored various S-Boxes [86, 87, 88, 89] and the minimum number of shares required, with uniform sharing being particularly difficult to achieve. For example, a 3-share realization is not always possible due to the uniformity constraint, and the Keccak S-Box, with its algebraic degree of 2, had no known uniform and incomplete 3-share Threshold Implementation [86, 90]. To address the loss of uniformity, fresh randomness can be injected via remasking or through techniques like "*changing of guards*" proposed by Daemen et al. [90], which was later used by Sukuwara et al. [83] for a 3-share uniform TI of the AES S-Box.

Following the introduction of Threshold Implementations (TI), several hardware-oriented masking schemes were proposed, but most were short-lived and are not widely considered today. In 2011, Prouff et al. [91] introduced a scheme based on Shamir's secret sharing to resist higher-order attacks. However, subsequent research [92] demonstrated that a straightforward implementation of this scheme could still fail in practice. In 2014, Bilgin et al. [93] extended TI to provide higher-order security, but Reparaz et al. [50] later identified vulnerabilities in this higher-order proposal and offered a fix. Despite these advances, designing higher-order TI circuits remains complex and inefficient: as the order increases, the number of required shares grows significantly. Furthermore, there is no straightforward method to scale a TI implementation to higher-order security,

highlighting the need for a generic, compact, and arbitrary-order hardware masking scheme.

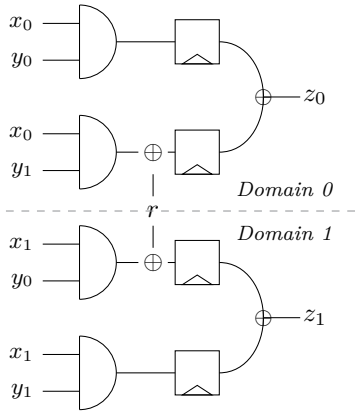
2.2.3 Domain-Oriented Masking (DOM)

Domain-Oriented Masking (DOM), introduced by Gross et al. in 2016 [22], offers glitch-resistant masking with a minimal number of shares ($d+1$ for d^{th}-order protection) and, importantly, enables scalability to arbitrary protection orders without requiring a full redesign for each order. In contrast to Threshold Implementations (TI), which operate at the algorithmic level, DOM is a gate-level technique that can transform an unprotected netlist of AND and XOR gates into a secure masked equivalent by replacing each gate with its masked counterpart. The core principle in DOM is to assign each share of a sensitive variable to one of $d+1$ independent domains. Linear operations such as XOR are performed independently within each domain, avoiding any interaction between shares of the same variable. Non-linear operations like AND, which inherently require interaction between shares, are handled using specialized masked gadgets that securely mix domains in a controlled fashion.

Depending on the relationship between the inputs, Domain-Oriented Masking defines two types of masked AND gates: DOM-*indep* and DOM-*dep*. A DOM multiplier consists of three steps: calculation, resharing, and integration. In this section, we focus on the DOM-*indep* multiplier, as DOM-*dep* is a special case that internally uses the DOM-*indep* structure and, moreover, DOM-*indep* has served as the foundation for many subsequent masking schemes.

For ease of understanding, we begin with the first-order case of the DOM-*indep* multiplier. Let the two input variables x and y be shared as $x = x_0 \oplus x_1$ and $y = y_0 \oplus y_1$. In the calculation step, share-wise multiplications are performed to compute the terms x_0y_0, x_0y_1, x_1y_0, and x_1y_1. These products are categorized as inner-domain terms (x_0y_0, x_1y_1), which involve shares from the same domain and are thus secure, and cross-domain terms (x_0y_1, x_1y_0), which mix shares from different domains. The distinction between DOM-*indep* and DOM-*dep* arises specifically in the treatment of these cross-domain terms. If the inputs x and y share the same randomness (e.g, $x = y$), then a single probe on a cross-domain term, such as $x_0y_1 = x_0x_1$, could leak information about both shares of the same variable, violating first-order security. DOM-*indep* avoids this by assuming that the inputs are independently shared using different random masks. Under this assumption, the cross-domain terms (x_0y_1, x_1y_0) are allowed, as they do not leak sensitive correlations.

To keep the number of output shares minimal, specifically $d+1$ for d^{th}-order security, the cross-domain terms must be integrated back into a domain.



(a) 1st-order security

Require: $x_{0..d}$, $y_{0..d}$
Ensure: $\{z_k\}$, with $\oplus_k z_k = (\oplus_i x_i) \cdot (\oplus_j y_j)$

1: $\forall i : \ in_i \leftarrow \{x_i y_i\}$ // *Inner-domain*
2: $\forall i < j : \ r_{i,j} \leftarrow \mathsf{rand}()$ // *Cross-domain*
$\{t_{i,j}\} \leftarrow \{x_i y_j \oplus r_{i,j}\}$
$\{t_{j,i}\} \leftarrow \{x_j y_i \oplus r_{i,j}\}$
3: $\forall k : \ z_k \leftarrow in_k \oplus \bigoplus_{j \neq k} t_{k,j}$
4: **return** z

(b) d^{th}-order security

Figure 2.2: DOM-*indep* multiplier.

However, to ensure security, these terms are first processed in the resharing step: they are refreshed using a common fresh random bit r, which ensures that they become statistically independent of the original input shares, and they are registered to prevent glitch propagation. In the final integration step, the reshared cross-domain terms are combined with the inner-domain terms to form the output shares z_0 and z_1. Due to the symmetric use of the fresh randomness r, their XOR results in the correct product $z = x \cdot y$, as illustrated in Figure 2.2a.

A dth-order secure DOM-*indep* multiplier follows the same structure as the first-order case, operating over $d+1$ domains. The full procedure is shown in Algorithm 2.2b. Each of the $d+1$ inner-domain terms $x_i y_i$ is computed independently, while the $d(d+1)$ cross-domain terms $x_i y_j$ for $i \neq j$ are refreshed symmetrically using random bits $r_{i,j}$. These refreshed terms are then integrated back into the $d+1$ output domains to produce the shares z_k, ensuring d^{th}-order security and correctness.

DOM-*dep* assumes that the multiplication inputs share the same randomness and therefore cannot compute $x \cdot y$ directly. To overcome this, it introduces a random blinding value z and splits the computation into two parts: first, it refreshes y as $y' = y \oplus z$ and computes $x \cdot y'$ using straightforward scalar multiplication; second, it computes $x \cdot z$ using a DOM-*indep* gadget. The final result is then obtained by XORing the two intermediate products:

$$x \cdot y = (x \cdot y') \oplus (x \cdot z).$$

While this approach masks the dependency between x and y, it comes at the

cost of additional randomness, logic, and register overhead. Moreover, Moos et al. [27] later demonstrated that DOM-*dep* does not achieve the claimed d^{th}-order security. Consequently, more robust primitives, such as HPC (Hardware Private Circuits) gadgets [94] discussed in Section 3.1.2, are required for securely multiplying dependent inputs. These alternatives, however, also incur higher implementation costs compared to DOM-*indep*.

Despite this, DOM-*indep* remains the most cost-efficient gate-level gadget and serves as the foundation for later constructions such as HPC. The core design challenge is determining whether a cheaper DOM-*indep* multiplier is sufficient, or whether stronger gadgets are necessary to guarantee d^{th}-order security, especially when composing AND gates in larger masked circuits. The following sections explore circuit-level composition strategies and discuss implementation cost and security.

By the mid-2010s, hardware masking had evolved from algorithmic techniques such as Threshold Implementations (TI) to gate-level frameworks like Domain-Oriented Masking (DOM). While DOM simplifies scaling to arbitrary protection orders, it introduces significant overhead by requiring fresh randomness for each masked gadget.

2.3 Efficiency Considerations in Hardware Masking: Area, Randomness, and Latency

Masking countermeasures inevitably trade implementation costs for resistance against side-channel adversaries. Although concrete costs depend on the target cipher, technology node, and masking scheme, three macro-level metrics consistently guide secure hardware design: *area*, *fresh randomness*, and *latency*. These dimensions are not independent; they influence one another, and effective design requires treating them in combination rather than in isolation. This section discusses each metric individually and explores their interdependencies. The precise definitions of these metrics and the methodology used to measure them are detailed in Section 4.1.

Area Overhead. Area overhead arises primarily from the need to replace each sensitive variable with at least $d + 1$ statistically independent *shares* to achieve d^{th}-order security. Every operation that previously handled a single value in the unprotected design must now process multiple shares. For linear operations, this duplication is straightforward, but non-linear functions require computing cross-products between shares, significantly increasing logic complexity. Additionally,

secure designs often insert pipeline registers between combinational stages to prevent glitch propagation. These registers not only increase the number of flip-flops but also require auxiliary control logic for clock gating or reset, further contributing to overall area consumption.

Randomness Requirements. Fresh random bits are essential to the security of masking. They are required both to initialize the input sharing and to randomize cross-domain interactions in non-linear gadgets [22]. Non-linear gates typically consume at least one fresh random bit per evaluation to prevent statistical dependencies that could violate probing security and this requirement increases with higher masking orders [22, 95]. Generating high-quality entropy on-chip introduces non-trivial silicon cost. True random number generators (TRNGs) rely on analog components and require calibration logic, while pseudo-random number generators (PRNGs) require state registers, feedback logic, and often operate in a dedicated clock domain [96, 97, 98]. Furthermore, the randomness source must sustain the peak consumption rate of the masked core; in lightweight systems, this can make the generator the dominant contributor to area and power overhead. To mitigate these challenges, designers have proposed two complementary strategies. First, they design masking schemes that allow the secure *reuse of randomness across independent gadgets*, thereby reducing the per-gadget entropy required during operation. Recent works [99, 100, 101, 102, 103, 24] have explored techniques for reusing or even eliminating randomness in first- and second-order masking schemes, demonstrating that careful sharing of masks across operations can reduce cost without compromising security. Second, designers focus on *efficient generation and distribution of randomness*, ensuring that high-quality mask bits can be supplied at the required rate without excessive hardware cost [104, 105, 106, 107]. In this context, lightweight stream ciphers such as Trivium have been used as practical PRNGs to generate fresh masks on-the-fly with minimal area overhead [108].

Latency. In unprotected hardware, a non-linear layer, such as an S-box, can typically be implemented within a single combinational stage. Masked variants, however, must decompose the same computation into multiple stages to prevent transient effects, such as glitches, from allowing multiple shares to momentarily converge. To mitigate this, registers are inserted between stages to obstruct glitch propagation and to synchronize the arrival times of share signals [109, 110]. When the core operates in an iterative round-by-round fashion, these additional cycles accumulate, resulting in a noticeable increase in end-to-end latency. In this thesis, latency is measured in clock cycles, representing the number of cycles required for a cryptographic operation to complete from input to corresponding output. A detailed discussion of how latency is defined and

measured, including examples for both entire ciphers and small components like S-boxes, is provided in Section 4.1.

While earlier efforts in hardware masking primarily focused on reducing area and randomness costs, recent research has increasingly shifted toward optimizing latency. Two general approaches have emerged. The first retrofits existing probing-secure designs by replacing pipeline registers with carefully timed combinational logic or latches that emulate synchronization behavior, for example techniques such as SESYM [111], self-timed masking [112], and glitch-stopping circuits [113]. The drawback of this approach is that it reduces the number of clock cycles at the cost of a lower maximum operating frequency. The second direction involves designing novel masking schemes from the ground up with latency minimization as the primary objective. However, this often comes with increased cost in the other two metrics, area and fresh randomness, due to more complex logic and additional entropy requirements. Several recent works [114, 115, 116, 117, 118] have explored this trade-off in detail. As a significant portion of this thesis focuses on such low-latency designs, we defer a detailed discussion to the chapters that follow. The next section takes a closer look at these developments and explores how low-latency designs are implemented in practice.

Reducing any single metric, area, randomness, or latency, inevitably places greater stress on the others. Each masking scheme or implementation technique attempts to optimize one or more of these dimensions while preserving security. Broadly, the literature follows two main directions: high-overhead designs that come with strong formal security guarantees, and more optimized constructions that rely on extensive empirical validation. This thesis explores both approaches.

2.4 Low-Latency Masking in Hardware

While the previous discussion covered efficiency trade-offs in hardware masking broadly, including area, randomness, and latency, it is important to highlight that latency optimization remains less mature than the other two. Although techniques for reducing area and randomness have been rigorously explored and refined over the past two decades, low-latency masking designs are still an evolving research frontier. A major practical drawback of masked hardware implementations is the additional latency introduced to secure computations. Designers typically insert extra register stages between combinational logic blocks to prevent transient glitches. While effective for ensuring security, these registers increase the number of clock cycles required to complete a masked operation, significantly raising latency for larger blocks such as AES S-boxes.

This overhead is undesirable in real-time or high-speed applications, where minimal delay is critical.

In contrast to classical digital design, where de-pipelining can straightforwardly reduce the cycle count, in masked hardware the register stages cannot be removed freely because they are integral to the security argument. In Threshold Implementations, this role is tied to non-completeness, which relies on register boundaries to prevent glitches from combining all shares (see Sec. 2.2.2). In Domain-Oriented Masking, the cross-products are first computed in separate domains and then compressed to outputs; registers are required to ensure that shares from different domains are combined only in a controlled and secure way (see Sec. 2.2.3). Simply removing these registers would undermine the conditions on which the security guarantees rely, so latency reduction in masked hardware requires more specialized strategies.

Over the past decade, researchers have proposed various hardware masking schemes aimed at reducing this latency penalty. Broadly speaking, low-latency masking has evolved along two directions. The first is *circuit-level*: keep the circuit structure the same and modify the implementation (logic style, clocking, input-arrival scheduling) to suppress or control glitches so larger blocks can run in one or a few cycles without adding extra registers. The second is *scheme-level*: build new masking schemes from the ground up so it requires as few register stages as possible; the scheme is defined to operate safely with minimal staging. This section reviews prominent low-latency masking techniques, highlighting their design principles, security models, and the trade-offs they introduce compared to conventional approaches.

2.4.1 Circuit-Level Approaches

The main obstacle to achieving single-cycle (fully unrolled) masked cryptographic circuits is the physical behavior of the circuit, specifically the glitch problem caused by transient signal races in combinational logic. A natural solution was to re-engineer the hardware at the logic level to be inherently glitch resistant. This gave rise to dual-rail pre-charge logic styles (DRP), such as Wave Dynamic Differential Logic (WDDL) [55, 119], briefly introduced in Section 2.1.2, and Masked Dual-Rail Pre-Charge Logic (MDPL) [67], DPLnoEE [120], AWDDL [121] etc., which ensure that every logic signal transitions in a controlled, monotonic fashion. In these styles, each bit is encoded on two wires, called the true rail and the false rail, and every clock cycle is split into two phases: a pre-charge phase, in which the rails reset to a neutral state, and an evaluation phase, in which exactly one rail switches. Because all signals undergo at most one transition in the evaluation phase,

glitching is eliminated. The result is a wave of computation that flows through the circuit without any timing races, effectively making the hardware glitch proof. This approach, pioneered in the mid-2000s, demonstrated that secure single-cycle operation was in principle possible, provided it is built from these specialized gates.

A significant refinement of this idea came with LUT-based Masked Dual-Rail with Pre-Charge Logic (LMDPL) in 2014 [122], which was subsequently improved in 2020 [123]. LMDPL integrated first-order Boolean masking directly into dual-rail gates by using FPGA look-up tables to implement logic functions. Like WDDL and MDPL, LMDPL ensured that every gate exhibited balanced, glitch-free switching behavior. However, LMDPL achieved this without requiring custom routing constraints between gates, which marked a crucial advantage. Even today, an AES engine built with LMDPL achieves low implementation costs and remains a benchmark for low-latency masked AES implementations.

The idea of precisely controlling signal timing to prevent glitches has continued to attract interest. Another investigation in this area was carried out by Schneider et al. [124] in 2016, who explored self-timed asynchronous circuit designs [125] that use handshake protocols. Instead of relying on a global clock, these designs incorporated acknowledge (Ack) signals to locally pause computations until preceding signals had stabilized. Schneider and Moradi implemented first-order masked versions of the PRINCE and Midori ciphers as part of their study. Their findings highlighted that although asynchronous techniques can indeed mitigate glitches, they introduce significant design and optimization challenges, which can offset the performance benefits expected from low-latency ciphers. Ultimately, the authors concluded that a return to simpler, synchronous designs may often be the more practical approach.

Nagpal et al. [111] introduced Self-Synchronized Masking (SESYM) in 2022, a generic technique that merges standard masking schemes with WDDL gates to achieve glitch-free, single-cycle masked hardware. SESYM was designed to be broadly applicable and supports higher-order masking, as demonstrated with Domain-Oriented Masking implementations of ASCON and AES. However, subsequent analysis revealed that imbalanced signal delays create timing vulnerabilities that compromise its security, a flaw demonstrated both theoretically and through practical hardware experiments [126, 127].

A recent fully synchronous approach is Glitch-Stopping Circuits without Registers, proposed by Zhang et al. [128] in 2024. It enables masked hardware designs without pipeline registers between logic stages. Instead, flip-flops are replaced by clock-gated barriers like clocked AND gates, 2-to-1 multiplexers, or latches, which block transient values effectively. Circuit operation is managed by adding additional auxiliary clocks with duty cycles of 25%, 50%, and 75%

etc., dividing each clock period into virtual cycles so only one combinational block evaluates at a time, preventing glitch propagation. Alternatively, they suggest that phase-shifted clocks from a phase-locked loop (PLL) or delay lines can achieve similar timing control.

While these techniques focus on modifying circuit logic styles or clocking to mitigate glitch propagation and reduce latency, another promising direction lies in algorithm-level optimizations. Rather than intricate hardware redesign, algorithm-level methods restructure cryptographic computations to achieve secure, low-latency implementations that are often simpler to implement in hardware, though they come with their own cost trade-offs.

2.4.2 Scheme-Level Approaches

Scheme-level masking techniques originally emerged without prioritizing low latency. However, recent extensions specifically address latency issues: for example, Threshold Implementation (TI) was extended into Low-Latency TI (LLTI) by Arribas et al. [116]. LLTI avoids using intermediate registers by replacing physically separated pipeline stages with a sequence of logical layers computed entirely within the same clock cycle. Instead of relying on registers to isolate stages and prevent glitches from leaking sensitive information, LLTI guarantees security through careful algorithmic design.

Domain-Oriented Masking (DOM) evolved into Generic Low-Latency Masking (GLM) [117]. Generic Low-Latency Masking (GLM) extends Domain-Oriented Masking by removing compression of shares back to a minimal number immediately within each masked gadget. Instead, GLM defers this compression to a final stage at the end of the computation. This means that after every non-linear operation, the number of shares temporarily increases, but registers that would normally separate stages can be removed entirely. By keeping the design fully combinational and performing a single compression step only once, GLM achieves low latency without compromising the underlying security guarantees of DOM.

Additionally, the more recent Generic Hardware Private Circuits (GHPC) introduced by Knichel et al. [129] explicitly provide formal security guarantees, the details and significance of which will be elaborated in Section 2.5. GHPC leverages Shannon decomposition to compute the target function over one share, while the other share serves to select the correct output via a secure multiplexer. Importantly, GHPC can achieve first-order masking with only a single register stage, regardless of how large or complex the target function is.

The main contributions of Chapters 8 and 9 align closely with the scheme-level masking approaches introduced in this subsection; therefore, we shall not discuss the technical details extensively here, as they will be covered in the corresponding chapters.

2.5 Formal Security Notions for Masking Schemes

Despite improvements in masking techniques, creating a securely masked circuit which is also efficient continues to be a difficult problem. As circuit designs grow more complex, it becomes harder to identify locations where masking becomes compromised and to correct them. Moreover, constructing higher-order masked implementations is non-trivial, which further complicates verification and debugging. Over the past ten years, there has been a growing emphasis on formal security notions, which offer a structured way to analyze and reason about the side-channel resistance of masked implementations. In this section, we begin with the basic probing model and its robust extension, which accounts for hardware effects such as glitches. We then introduce stronger notions, such as NI, SNI, and PINI, that ensure composability and resilience against physical defaults.

2.5.1 The Probing Model

Masking encodes a sensitive value into multiple shares, forcing the adversary to observe several internal signals to reconstruct the secret. The *probing model*, introduced by Ishai, Sahai, and Wagner (2003) [76], formalizes this principle. In this model, an adversary is allowed to place up to t probes on the internal wires of a circuit. Each probe reveals the value carried by that wire, typically a single bit. A masked circuit is said to be *t-probing secure* if the following condition is satisfied:

Probing Security (*t-probing*)

Any set of at most t probed wires yields no information about the sensitive data (i.e., secret inputs) of the circuit; in other words, the observed values are statistically independent of the secret.

This means an attacker gains no information unless more than t observations are collected. This condition implies the existence of a simulator that can reproduce the joint distribution of any t probed values using only t input shares, without

access to the actual secret. If such a simulation exists, then the real probes are indistinguishable from simulated ones, ensuring that no information about the secret is leaked. The results of Ishai et al. established that any function can be securely computed in a masked manner using $d = t + 1$ shares, such that any set of up to t probes reveals nothing about the secret. In the case of first-order probing security ($t = 1$), this implies that the value on any single wire is statistically independent of the secret. Higher-order probing security generalizes this guarantee to any set of t wires, ensuring that the joint distribution of up to t intermediate values is statistically independent of the inputs.

The foundational probing model became the baseline for evaluating masking. However, it also has some limitations:

1. The model assumes that each wire carries a stable value, ignoring physical characteristics of real circuits such as glitches, transitional effects, etc.
2. Each probe leaks independently: it reveals only its value.
3. While the model provides a criterion for evaluating security, it does not address how to construct circuits that satisfy it. In practice, achieving probing security requires careful design of the circuit's operations, typically using specialized masked components, referred to as *gadgets*, for both linear and non-linear operations. Furthermore, randomness must be injected at intermediate stages to eliminate statistical dependencies. Ensuring these conditions throughout the design is non-trivial.

These limitations have motivated refinements to the probing model and the development of stronger security notions, as discussed next.

2.5.2 From Global Guarantees to Modular Security Notions

Early works provided ad hoc proofs for small circuits/algorithms, demonstrating by hand their resistance to side-channel attacks [130, 131]. However, as circuits grow in size and complexity, manually analyzing all combinations of t probes to ensure t-probing security becomes infeasible. While the probing model serves as a security goal, it does not inherently guarantee that two individually t-probing secure circuits remain secure when composed. This challenge motivated the development of formal security notions that support modular and compositional reasoning. The objective is to construct complex secure circuits by composing smaller, well-understood components (commonly referred to as *gadgets*). The core idea is to prove that each gadget satisfies certain properties that, when combined, ensure the security of the entire composition. This approach significantly simplifies the overall verification effort.

Notation. In this section, we familiarize the reader with small toy examples to illustrate security notions. A secret value x is shared as two shares x_0 and x_1, satisfying $x = x_0 \oplus x_1$. We denote pairs of shares as vectors, for instance $a = (a_0, a_1)$ and $b = (b_0, b_1)$. The notation $\mathrm{Reg}(w)$ indicates that the signal w is stored in a register.

Non-Interference (NI) [132] and its stronger variant *Strong Non-Interference* (SNI) [133], were introduced for verifying masked software. The formal definition of t-NI is as follows:

Non-Interference (t-NI)

A gadget is said to be *t-NI* if any set of $t' < t$ intermediate probes can be simulated using at most t' shares of each of the gadget's inputs.

Informally, any information leaked from t' wires can be simulated using partial knowledge, specifically, t' shares from each input, which does not reveal the complete secret. In essence, t-NI is a strict condition that implies t-probing security. For example, a trivial linear gadget such as a masked XOR satisfies NI: each probe can be simulated by knowing the corresponding shares from each input.

The first-order DOM-*indep*(a, b) multiplier with shared inputs a and b, introduced in Section 2.2.3, satisfies both 1-NI and 1-probing security. As a counterexample, consider DOM-*indep*$(\mathrm{Reg}(a \oplus b), \mathrm{Reg}(b))$: it is 1-probing secure but not 1-NI, because a probe placed in the multiplier can reveal the multiplication $\mathrm{Reg}(a_0 \oplus b_0) \cdot \mathrm{Reg}(b_1)$, which would require two shares of b (i.e., b_0, b_1) to simulate, violating the requirement of 1-NI.

However, NI alone is too weak to ensure composability, as it does not guarantee statistical independence between the inputs and outputs of a gadget. In composed circuits, an output share of one gadget, carrying some dependency on an intermediate secret, may feed into another. A probe in the second gadget can then be combined with the output share from the first to break security.

To address this, *Strong Non-Interference* (t-SNI) was introduced as a stricter condition that ensures independence between a gadget's inputs and outputs, facilitating secure composition. It is defined as follows:

Strong Non-Interference (t-SNI)

A gadget is said to be *t-SNI* if, for every set of probes consisting of t_1 probes on the internal wires and t_2 probes on the outputs (with $t_1 + t_2 \leq t$), the joint distribution of these probes can be simulated using at most t_1 input shares of each input.

Intuitively, even if an adversary observes some output shares in addition to internal wires, they learn nothing beyond what is already revealed by internal probes. In practice, SNI gadgets serve as probe propagation stopping points, i.e., a probe on the output of an SNI gadget can be simulated without access to the inputs, thereby preventing input dependencies from propagating across gadget boundaries.

A toy example of an SNI gadget is $\mathrm{Reg}(\mathrm{DOM} - \mathit{indep}(a, b))$. Here, the outputs of the multiplier are registered, meaning any probe on the output can be simulated without knowledge of the input shares. This is because the random bit r, introduced inside the multiplier, ensures that the output is statistically independent of the inputs at the point of observation.

If every component of a circuit satisfies the SNI property, then the entire circuit is guaranteed to be t-probing secure. However, achieving SNI often incurs additional overhead. For instance, SNI gadgets typically require fresh randomness to decorrelate their outputs, and simple operations like masked XOR are not SNI by default, therefore require output SNI refreshes, which adds cost.

Early composition strategies thus faced a trade-off: one could either enforce strong but expensive properties like SNI on every operation to enable modular composition, or use more efficient gadgets satisfying only NI, at the cost of requiring a more complex global analysis of their interactions and the careful insertion of output refreshes to ensure that the composed circuit achieves t-probing security [134]. Furthermore, the original definition of SNI applied only to gadgets producing a single shared output. Extending it to gadgets with multiple outputs required additional tweaks.

Probe-Isolating Non-Interference (PINI) was introduced by Cassiers et al.(2018) [135] as a security notion aimed at enabling trivial composition. It was explicitly designed to allow gadgets to be composed without any special ordering or insertion of refreshing operations, while being more efficient than SNI. The formal definition is as follows:

Probe-Isolating Non-Interference (t-PINI)

A gadget is said to be *t-PINI* if, for every set of probes consisting of t_1 probes on internal wires and t_2 probes on the output wires (with $t_1+t_2 \leq t$), the joint distribution of these probes can be simulated using at most t_1+t_2 **same share indices** from each input.

A gadget satisfying t-PINI guarantees that any set of t probes, whether on internal wires or outputs, can be simulated using only the same t share indices for each input. This imposes a stricter constraint than NI or SNI. While NI and SNI allow flexibility in choosing share indices per input, PINI enforces *isolation* by requiring that the same set of indices be used across all inputs.

The first-order DOM-*indep*(a, b) gadget is not 1-PINI because a probe on the internal product $a_0 \cdot b_1$ reveals one share of a and one share of b with different indices (0 and 1), violating the same-index requirement of PINI. To address this, consider masking a with fresh randomness: let $c = a + r_{\text{mask}}$, where r_{mask} is a random bit that refreshes both a_0 and a_1. Then, the gadget DOM-*indep*$(\text{Reg}(c), b)$ satisfies 1-PINI. Since c is now independent of the original shares of a, any probe within the multiplier reveals information involving only same-index shares, preserving the PINI condition.

PINI and SNI differ in how they treat probes on outputs. SNI enforces strong independence by disallowing any probes on the output wires to be simulated with input knowledge. In contrast, PINI permits simulation of output probes using input shares, but still mandates that the same share indices are used. This distinction is evident in simple gadgets such as masked XOR, which satisfies t-PINI but is only t-NI, not t-SNI. Additionally, the PINI definition naturally extends to gadgets with multiple outputs. In this case, the t_2 probes on the outputs must correspond to the same set of share indices across *all* outputs of the gadget. From a composability perspective, any network of t-PINI gadgets is itself t-PINI without requiring additional conditions, and is also t-probing secure by construction.

PINI enables trivial composition in the presence of glitches. However, in round-based constructions with a single-register feedback loop, where the output of a gadget is fed back as input in the next cycle, the two consecutive inputs of the same physical gadget are not independent. Under transitions, this invalidates the trivial PINI composition rule. To address this, OPINI was introduced in [136], providing a strengthened notion that restores modular reasoning in such settings.

Output Probe–Isolating Non-Interference (t-OPINI)

A gadget is said to be *t-OPINI* if, for every set of probes consisting of t_1 probes on internal wires and t_2 probes on the output wires (with $t_1 + t_2 \leq t$), there exist at most t_1 *additional output-share indices* such that the joint distribution of these probes, *together with the gadget outputs at those additional indices*, can be simulated using at most $t_1 + t_2$ **same share indices** from each input.

To put the formal notions in perspective: probing security serves as the baseline definition, while NI, SNI, and PINI are engineering principles that enable the practical construction of large t-probing secure circuits. The development of these compositional notions has advanced the maturity of provable security in masking.

However, this progress is not without drawbacks. While these notions make it easier to compose probing-secure circuits, they introduce significant implementation overhead. Satisfying compositional properties often requires increased circuit area, higher latency, and the use of fresh randomness, even in cases where such strong guarantees may not be necessary. For example, gadgets are often required to use unique fresh randomness to ensure that their internal and output values are simulatable, which is critical for formal proofs.

This approach tends to be overly conservative: to simplify composition, strong gadgets assume the worst-case *interactions* and enforce strong security properties uniformly across all components, even when local analysis might suggest that weaker guarantees suffice. For example, PINI enforces that all probes must target the same share indices across all inputs, even in cases where such alignment is not necessary for security. While this strict condition enables trivial composition, it can impose avoidable constraints and lead to inefficient implementations. As a result, the price paid for composability is a substantial increase in implementation cost.

In contrast, building efficient masked circuits typically demands manual optimization and careful analysis of the entire design. One is faced with a trade-off: either invest effort in optimizing and verifying each part of the circuit by hand, or rely on generic composable gadgets with strong guarantees, accepting the associated overhead. Striking the right balance between efficiency and ease of construction remains an open research problem.

2.5.3 Robust Probing Model for Hardware

The security notions discussed in the earlier subsections, probing security, NI, SNI, and PINI, were originally formulated under the *standard probing model*, which assumes that each wire carries a stable value and each probe reveals one such stable signal.

However, this abstraction fails to capture certain physical effects inherent to hardware. For instance, *glitches*, defined in Section 2.2.1, transient logic changes due to gate delays, can cause a single probe to observe multiple switching events in rapid succession. Similarly, *transitional leakages* [136, 137] between successive cycles may encode both the previous and new values on a single wire during signal changes, violating the core assumption of one stable value per probe.

To address these physical defaults, Faust et al.[138] introduced the *robust probing model* in 2018. This model extends the classical probing framework by explicitly accounting for such effects. A single probe in this model may correspond not just to a wire's value at one time but to a set of values induced by glitches over time. For example, probing the output of a combinational gate also implicitly considers the input wires and their transitions, recursively extending back to the last synchronization point (e.g., flip-flops or registers).

If a circuit is secure in the robust probing model, often called *glitch-robust*, then even in the presence of such transient behaviors, an adversary limited to t probes learns nothing about the secret inputs. This refinement led to the development of new gadget designs, along with glitch-aware extensions of the NI, SNI, and PINI notions, ensuring composability and provable security even in the presence of physical effects.

2.6 Formal Verification Tools for Masked Hardware Designs

Designing a masked circuit by hand while ensuring that it satisfies security notions such as NI, SNI, PINI, or robust probing security is highly error-prone and does not scale well for complex cryptographic algorithms. In response to this challenge, the late 2010s saw the development of several automated tools that assist in the design and validation of secure hardware. These tools can be broadly classified into two categories: (1) generation tools [139, 140, 141], which construct masked implementations from unmasked specifications or gate-level netlists by applying composition rules or assembling pre-verified gadgets; and (2)

verification tools, which analyze a given masked implementation to determine whether it satisfies the desired security properties.

The usefulness of these tools stems from the compositional nature of modern security notions. Properties such as SNI and PINI guarantee that circuits constructed from secure components remain secure under composition, thereby enabling automated generation from a library of provably secure gadgets. As a result, several frameworks have been developed to build circuits that are provably secure in the probing and robust probing models, significantly reducing manual effort. Nonetheless, automated generation tools are not directly relevant to this thesis, as they are not employed in the presented work. Instead, the focus lies on verification tools, which are utilized in Chapters 8 and 9 to validate the security of the masked designs discussed in those chapters.

`MaskVerif`, introduced by Barthe et al. [142] in 2015, is an automated verification tool designed to formally analyze whether a given masked software or hardware implementation satisfies t-probing security. In later versions, it was extended to support stronger compositional notions such as t-NI and t-SNI. The tool accepts input in the form of pseudocode written in Static Single Assignment (SSA) form or Verilog netlists and verifies the implementation under the specified security notion. If the implementation is secure, `MaskVerif` produces a formal certificate that no attacks exist up to the tested security order. Otherwise, it returns a concrete set of signals (or probes) that constitute a violation of the security property. Importantly for hardware, `MaskVerif` supports analysis under the robust probing model: it can handle glitches in combinational circuits and transitional leakages in sequential designs. This feature made it one of the first tools to provide efficient and rigorous verification of masking in realistic hardware settings.

Nonetheless, like many formal tools, its scalability can become a limitation. The verification time increases significantly with the circuit size and the probing order being tested. Moreover, early versions of `MaskVerif` were reported to be overly conservative, occasionally flagging secure designs as insecure (i.e., producing false negatives) [143]. Such inaccuracies are problematic in verification workflows, where soundness is typically more critical than efficiency: a verifier must have confidence in the correctness of the result, even if runtime is high. Despite these limitations, `MaskVerif` remains a foundational tool in the formal verification of masking and has influenced the design of many subsequent tools and frameworks.

SILVER (Statistical Independence and Leakage VERification), introduced by Knichel et al. [143] in 2020, is a formal verification tool that adopts a symbolic approach based on Boolean function analysis. It represents internal signals of a circuit using Reduced Ordered Binary Decision Diagrams (ROBDDs), which

encode Boolean expressions compactly and allow reasoning about the statistical dependence between secret inputs and intermediate values. The tool accepts gate-level netlists after synthesis, including circuits with registers, making it applicable to practical hardware designs. SILVER supports a wide range of formal security notions, including t-probing, t-NI, t-SNI, and t-PINI, under both the standard and robust probing models.

The core idea behind SILVER is to build symbolic expressions for every signal in terms of the secret and random inputs, then use BDDs to construct joint distributions of up to t probed values. By checking the resulting distribution for statistical independence, the tool determines whether the circuit satisfies the target security notion. Unlike simulation-based tools, which rely on statistical tests and sampling, SILVER provides exact results due to its symbolic formulation. Its logic-level analysis also enables it to naturally account for hardware-level effects such as glitches.

SILVER has been recognized for its simplicity, high accuracy, and broad functionality. Its symbolic framework avoids false positives and false negatives, a known limitation in earlier tools such as `MaskVerif`. It is also supported by a well-maintained public repository [144], which contributes to its practical usability. However, its main drawback lies in scalability. The size of BDDs can grow rapidly with circuit complexity, particularly for large Boolean expressions such as S-boxes or circuits with many inputs. As a result, SILVER is most effective when applied to small circuits such as gadgets or individual S-boxes, and becomes impractical for full cryptographic designs or higher-order masking.

While formal methods such as `MaskVerif` and SILVER offer strong mathematical guarantees, they can become computationally expensive for large circuits that do not adhere to a composable gadget structure. PROLEAD, introduced by Müller et al. [145] in 2022, is a more recent hardware leakage detection tool that adopts a simulation-driven approach to evaluate masked hardware implementations. Rather than attempting exhaustive formal proofs, PROLEAD trades off formal rigor for scalability by focusing solely on verifying t-probing security in the robust probing model. It does not support compositional notions such as NI, SNI, or PINI.

PROLEAD operates at the gate-level by performing smart logic simulations across different combinations of random mask values, and checking for statistical independence between the secret inputs and any set of d simulated probes. Crucially, it targets the robust probing model and accounts for both glitches and transitions by modeling the circuit's logic over time.

The main advantages of PROLEAD are threefold. First, it requires no power model or analog simulation; it works purely at the logic level, under the

assumption that statistical independence of logic values implies the absence of physical leakage. Second, it scales to complete cipher implementations, such as AES or other large circuits, which are often infeasible for exhaustive formal verification tools like SILVER. This makes PROLEAD particularly valuable for evaluating hand-optimized designs that may deviate from composable structures to save area or randomness, cases where formal tools may fail to verify security. Third, it can detect flaws arising from the combined effects of glitches and sequential behaviors (transitions). In essence, PROLEAD performs a systematic leakage assessment through logic-level simulation, mimicking what a physical evaluator might test in a lab environment, but in a scalable and fully virtual manner.

Several other tools and frameworks, such as `Rebecca` [146], `IronMask` [147], and others [148, 149, 150] , exist in the literature but are not discussed in this section, as they are either not relevant to the scope of this thesis or have not been used in any of its associated publications. In contrast, SILVER and PROLEAD represent the current benchmark tools in the domain, as they address many of the limitations observed in earlier approaches. PROLEAD is particularly effective for broad, real-world validation of masked designs under robust leakage models, while SILVER provides strong formal guarantees by exhaustively proving properties using BDDs, giving designers confidence that, under the analyzed notion, the circuit is secure. Each tool has its respective strengths, and they are in fact complementary. In practice, a designer may use SILVER during the development phase to formally verify critical modules and apply PROLEAD to the final integrated design to double-check for potential leakage across the full implementation.

2.7 Conclusion

This chapter has established the foundational concepts and theoretical models necessary to understand side-channel vulnerabilities and the countermeasures used to mitigate them. With this background in place, the next chapter presents the overarching structure of this thesis, outlining the key research directions pursued and the methodology adopted to balance practical efficiency with formal security guarantees.

Chapter 3

Research Problem and Thesis Scope

In the previous chapter, we reviewed the early landscape of side-channel attacks and masking countermeasures, as well as the formal security notions that bring structure to these countermeasures. We also discussed verification tools that support the validation of security properties under these models.

Despite this progress, a central challenge remains unresolved: how to design masked hardware that is both secure and highly efficient. Existing countermeasures tend to optimize either for formal security or for implementation cost, but rarely succeed in achieving both simultaneously. On one end of the spectrum, provably secure designs incur high overhead, particularly in randomness, area, and latency. On the other hand, practical approaches reduce these costs at the expense of formal guarantees. This trade-off is especially problematic in real-world scenarios, where constrained devices require cryptographic operations to be secure, fast, compact, and lightweight.

This thesis addresses this problem by pursuing two complementary research paths:

1. The first path explores whether practical side-channel resistance can be achieved through cost-efficient designs that do not necessarily satisfy strong probing or compositional formal models. Many such designs, including early masking schemes like TI and DOM, were developed before the widespread adoption of formal security notions. Although these schemes often demonstrate strong empirical resistance, they were not

originally constructed with formal proofs or composability in mind. This direction prioritizes practical efficiency while ensuring that empirical security remains robust, and is further explored in Chapter 4.

2. The second path focuses on provably secure masking under the robust probing model, with an emphasis on minimizing *latency*. While prior work in formal masking has largely focused on reducing area and randomness costs, latency has historically received less attention, as it was not a pressing requirement when formal models were first introduced. Recently, however, the demand for high-speed, formally secure cryptographic hardware, particularly from industry, has brought latency to the forefront as an important optimization target. This path is developed in Chapters 5 and 6.

The remainder of this chapter defines the core research problem in detail, analyzes the limitations of existing approaches along these two dimensions, formulates the key research questions, and outlines the scope, objectives, and methodology of the thesis. It thereby establishes the structure for the two lines of contribution presented in the chapters that follow.

3.1 Limitations of Existing Approaches

Existing masking techniques can be broadly categorized into two groups: early heuristic schemes that prioritize implementation efficiency, and modern provably secure schemes that focus primarily on rigorously satisfying formal security models. Each category has its own advantages and drawbacks, which motivate the two divergent research paths explored in this thesis.

3.1.1 Early Heuristic Techniques

Techniques such as Threshold Implementations (TI) and Domain-Oriented Masking (DOM) were developed to address hardware-specific pitfalls like glitches without necessarily relying on strong formalism. These approaches typically achieve low area and randomness overhead and follow a common strategy: decompose complex non-linear functions into smaller sub-circuits (with low algebraic degree) that can be more easily masked, and use pipeline registers and online randomness to prevent glitch propagation between sub-circuits . This "divide-and-mask" methodology has indeed enabled secure and efficient first-order implementations in practice[25, 151, 22].

However, these heuristic schemes come with significant limitations in terms of generality and scalability to higher-order security.

- **Manual, ad hoc design style:** These techniques require designing each masked component (e.g., S-boxes or elementary multipliers) in a specialized way [152, 151, 153, 154]. The designer must manually ensure properties such as non-completeness and uniformity for each component, and then carefully combine multiple such components into a larger circuit. This process is non-trivial, as it often involves inserting pipeline registers at the right locations and refreshing intermediate values with fresh online randomness to prevent leakage. As a result, the security of the overall design is not guaranteed by construction but instead relies on careful, component-wise engineering. This lack of straightforward composability makes the design process laborious and error-prone, often requiring a significant redesign or a ground-up transformation of the circuit to securely mask an otherwise unprotected implementation.
- **Scalability to higher order:** Threshold Implementations, for instance, inherently require more shares as the protection order increases [75, 155]. Specifically, they often require $td+1$ shares, where t is the algebraic degree of the function being masked. In practice, maintaining output uniformity under this model can require even more shares, and constructing such uniform decompositions is a well-known hard problem [156, 157, 158]. This rapidly increases circuit complexity. While higher-order DOM is relatively simpler in terms of share count, recent studies [27] have shown that published higher-order DOM-based designs are prone to local weaknesses or composability issues when evaluated under the robust probing model. This implies that an adversary might still be able to combine multiple probes to recover sensitive information. Moreover, as the security order increases, these schemes incur steep overheads in both area and randomness [159]. In summary, such heuristic schemes are often optimal for first-order protection but tend to scale poorly to higher orders.
- **Lack of formal proofs:** Perhaps the most fundamental limitation is that early heuristic schemes do not provide formal security or composability guarantees. They are typically secure when considered in isolation (e.g., a gadget on its own), but are not automatically secure when composed arbitrarily [27]. For example, a DOM-indep gadget achieves glitch resistance only if its two inputs are independently shared. If the same input is mistakenly fed (e.g., computing $a \wedge a$), the security breaks down [95], since the gadget satisfies only NI and not PINI. The literature contains several cases [160, 161, 162, 163, 164, 29] where masked implementations believed to be secure were later shown to contain vulnerabilities due to

such subtle composition issues. This was not due to negligence but rather due to the absence of such notions, until recently [165, 166, 95, 29, 167, 168], of a general theory ensuring that if each part is secure, the whole composition remains secure. As a result, every new masked design must be carefully analyzed and verified in its entirety. For large circuits constructed in an ad hoc manner, this type of holistic verification is difficult to perform with current verification tools and methodologies [149, 29, 143, 148, 169, 150, 142, 170].

In summary, early heuristic approaches tend to trade rigorous security guarantees for better performance and smaller circuit size. Their main drawback is that, without a provable security model, one must rely on empirical validation, which cannot exhaustively cover all possible scenarios. This introduces a risk that an unforeseen interaction or a specific higher-order probe combination may compromise the intended security of the design.

3.1.2 Modern Provably Secure Schemes

Modern masking schemes, in contrast to early heuristic approaches, provide rigorous guarantees that a masked circuit will resist side-channel attacks by design. These schemes are typically constructed under formal models such as the robust probing model, and satisfy compositional security notions like Probe-Isolating Non-Interference (PINI). The cost, however, is that these schemes have traditionally been far less efficient than heuristic approaches in terms of area, randomness, and latency.

- **Granularity and its impact on latency:** A prominent line of work in this category is the Hardware Private Circuits (HPC) framework [95], which introduced the first PINI-compatible gadgets targeting basic logic functions. For example, Cassiers et al. [95] proposed HPC1 and HPC2 gadgets, which are masked two-input AND gates proven secure under the robust probing model. While arbitrary circuits can, in principle, be built by composing such atomic gadgets, each HPC1 or HPC2 gadget requires two clock cycles to securely compute a single non-linear operation. Consequently, straightforward composition of such gadgets leads to significant performance degradation, with latency increasing proportionally to the circuit depth.

  To address this limitation, Knichel et al. [118] introduced HPC3, a low-latency gadget capable of completing a masked non-linear operation in a single clock cycle. This was achieved by trading off additional randomness—roughly doubling the fresh random bits required compared

to HPC2—in exchange for shallower pipelining. While this significantly improved latency, until recently all PINI-compliant gadgets remained limited in scope, targeting only small atomic operations such as AND gates [102, 171]. As a result, protecting larger functions such as 4-bit or 8-bit S-boxes still required composing many such small gadgets.

A notable advancement came with the introduction of Generic Hardware Private Circuits (GHPC) by Knichel et al. [172], which proposed a general methodology to transform arbitrary Boolean functions into first-order PINI-compliant gadgets using a form of Shannon decomposition. This allowed constructing a single, secure, and trivially composable gadget for an entire S-box. However, GHPC remains limited to first-order security and does not currently extend to higher orders.

- **Randomness and area costs:** Provably secure gadgets require additional registers and logic to ensure secure computation across separate shares and to properly synchronize intermediate signals. To maintain statistical independence between internal computations, these gadgets introduce fresh randomness at specific points in the design. For example, HPC1 and HPC2 each incur a two-cycle latency and require additional pipeline registers, which increases area overhead. HPC3 reduces the latency to one cycle by sacrificing efficiency in randomness, requiring approximately twice the number of fresh random bits as HPC2.

  In larger designs, the cumulative randomness demand becomes a significant burden, particularly for on-chip entropy sources or PRNGs [108, 173, 174]. Similarly, area costs grow substantially. For instance, a first-order AES S-box constructed using HPC-style gadgets is typically less efficient than its DOM or TI-based counterpart. Even algorithm-level provable constructions like GHPC face this problem: attempting to implement complex functions in a single clock cycle causes a blow-up in both randomness and area. For example, GHPC_{LL}, a low-latency GHPC instantiation for AES, requires approximately 64 kGE and over 2000 fresh random bits per evaluation, which is impractical for most real-world applications. As a result, GHPC-based methods are typically limited to smaller designs such as SKINNY [175] or PRINCE [35], where the complexity remains manageable.

3.2 Research Questions

Given the limitations and gaps identified in the previous sections, this thesis is guided by two central research questions.

Research Question 1 (Practical Security Path): *How can masking schemes be made more efficient in practice without strictly adhering to formal security models?*

This question focuses on the possibility of designing countermeasures that do not satisfy all the assumptions of compositional or formal frameworks, but still offer strong protection against real-world side-channel attacks. The goal is to develop masking techniques that significantly increase the difficulty for an attacker, for example by requiring more traces or higher-order analysis, while keeping implementation costs low in terms of area, randomness usage, and latency.

This direction embraces a heuristic and relaxed view of security, aiming for designs that are secure in practice even if they are not formally proven. The main challenge lies in defining what constitutes sufficient practical security and in ensuring that optimizations or simplifications do not unintentionally introduce vulnerabilities. Achieving this requires thorough empirical evaluation and careful reasoning about potential attack vectors. This path also involves identifying which assumptions from formal models are overly conservative, and determining when they can be safely relaxed without compromising real-world security.

Research Question 2 (Provably Secure Low-Latency Path): *Is it possible to realize a more efficient, provably secure masking scheme under the robust probing model that satisfies strong composability guarantees while minimizing overheads, particularly in terms of latency?*

This track targets the development of new masking gadgets or techniques that satisfy formal security notions such as SNI and PINI, while significantly reducing the number of clock cycles required per operation compared to state-of-the-art HPC-based designs. It also considers reducing area and randomness overheads relative to GHPC-based approaches. The goal is to explore whether low-latency masked operations can be achieved without compromising compositional guarantees, including first- and higher-order security.

Successfully answering this question would demonstrate that formal security and implementation efficiency are not inherently at odds. It would show that it is possible to design masked cryptographic hardware that is both provably secure and practically efficient.

Together, these two research questions guide the rest of this thesis. RQ1 focuses on a more applied, empirical approach to masking, while RQ2 emphasizes a formally grounded, yet implementation-conscious, direction.

3.3 Research Objectives and Methodology

To answer the research questions posed earlier, this thesis follows two complementary research tracks, each with distinct objectives and methodologies.

3.3.1 Practical Security Track: Heuristic Masking Approach

This track aims to develop masking techniques that minimize hardware overhead, particularly in terms of chip area, randomness consumption, and latency, while still providing sufficient protection against practical side-channel attacks. It deliberately operates outside strict formal security models to explore what can be gained in terms of efficiency. Specific objectives include:

- Designing a basic building block, `secAND`, in two realizations with distinct goals: `secAND-FF`, which minimizes area and eliminates fresh randomness; and `secAND-PD`, which targets latency by removing the gadget's internal register stage (avoiding pipelining) and introducing intentional, controlled delays within the gadget to enforce a safe signal ordering under glitches. The latter reduces the cycle count while accepting a longer critical path.
- Realizing a complete first-order masked implementation of the DES [37] block cipher using `secAND` as a core primitive.
- Validating the empirical security of the implementation through practical leakage evaluation techniques such as Test Vector Leakage Assessment (TVLA) [176].
- Quantifying the efficiency and security trade-off by measuring area, performance, and the trace count at which leakage becomes detectable.

The ultimate objective of this track is to provide a proof of concept that demonstrates how hardware masking can achieve meaningful empirical security while maintaining low implementation cost. This is shown by applying the technique to a non-trivial cryptographic primitive, namely DES, and comparing its performance and observed leakage against more conventional designs such as those based on Domain-Oriented Masking (DOM).

The methodology is iterative and driven by experiments. We begin with the theoretical design of `secAND`, conducting an informal analysis to identify potential leakage paths, such as timing correlations between shares. The design is then implemented at the hardware description level in Verilog and prototyped on an FPGA platform.

Security evaluation is central to this methodology. We perform TVLA tests and other empirical analyses to detect first-order leakage. If leakage is observed, the design is refined, for instance by introducing fresh randomness at certain stages or by adjusting pipeline registers to prevent timing correlations. Implementation metrics serve as feedback throughout the process. After each iteration, the prototype is synthesized to measure area and evaluated for timing and randomness usage. These results are compared against baseline implementations to quantify the efficiency improvements and the security trade-offs.

The outcome of this track is a thoroughly tested masked implementation that, while not supported by a formal proof, offers *strong empirical evidence of security* and clearly quantifiable advantages in terms of implementation cost.

3.3.2 Provable Low-Latency Track: Formal Masking Approach

This track follows the *scheme-level* direction: the masking is redesigned at the gadget or function level so that secure operation requires as few register stages as possible. Within this scope, the focus is on masking solutions that fully comply with formal security definitions (robust probing model, PINI composition, etc.), targeting first- or higher-order security as required. The objective is to optimize latency and performance overhead to a greater extent. The specific goals are:

- Proposing a new masking framework that can compute non-linear functions in fewer cycles than existing approaches while still being provably secure. For example, a first-order secure small S-Box (4-bit).
- Extending efficiently to larger designs, ensuring that a large cryptographic function like an AES S-Box can be built efficiently.
- Formally verifying and evaluating the security and performance overhead, whether they satisfy proving that new gadgets satisfy security notions and measuring implementation metrics.

The methodology is more design-theoretic and verification-driven. We start from the requirements of the robust probing model and composition notions such as SNI, PINI that our design must satisfy. We prove on paper and using formal verification tools that the gadgets meet SNI, PINI. In addition to manual proofs, we leverage formal verification tools specialized for netlists to verify under security notions in glitch-extended models.

Once the gadget designs are validated for security, we implement them in hardware and integrate them into a larger architecture. Even though the design

is provably secure, we perform an experimental sanity check by running TVLA tests to confirm that no unexpected physical effects violate the assumptions. This dual approach of formal verification and physical testing provides a higher degree of confidence. Formally verified for security and demonstrated to meet performance targets, thereby showing that provable security and efficiency can indeed coexist.

3.4 Thesis Structure

The remainder of this dissertation is organized around three core research contributions, beginning with a practical countermeasure and followed by two chapters that develop and extend a provably secure low-latency masking methodology.

- **Chapters 4 & 5 – Practical First-Order Masking in Glitchy Hardware:** These chapters address the first research question by presenting the `secAND` family of gates and their use in a complete first-order masked DES core. We describe how safe input sequencing removes the need for fresh randomness, and detail two implementations: a flip-flop-based version and a variant using deliberate path delay to counteract glitches. The design is evaluated on FPGA for side-channel leakage and implementation efficiency, demonstrating strong empirical security at minimal hardware cost.
- **Chapter 6 – Time-Sharing Masking (TSM): Provable Low-Latency Foundations:** This chapter opens the formal research track by introducing Time-Sharing Masking (TSM), a PINI-compliant design strategy that achieves first-order security with single-cycle latency. We formally prove its security in the robust probing model and present case studies on a complete PRINCE core and an AES S-box to highlight its efficiency gains in area and randomness compared to prior methods.
- **Chapter 7 – Higher-Order TSM and Area–Latency Trade-Offs:** Building on the previous chapter, this contribution generalizes TSM to arbitrary masking orders, resulting in Higher-Order TSM (HO-TSM). The scheme achieves a latency equal to the order d. We prove that HO-TSM satisfies glitch-extended PINI and SNI properties and demonstrate its applicability with second-order masked AES S-boxes and a complete first-order secure AES-128 core. In addition, we introduce and evaluate an alternative construction that reduces the area of TSM and HO-TSM at the cost of a single-cycle increase in latency, thereby exploring the trade-offs

between area and latency. Formal verification with SILVER, maskVerif, and PROLEAD, together with empirical TVLA testing, confirms that the theoretical security guarantees hold in practice.

Chapter 4

Metrics, Testing Methods, and Tools

This chapter serves as a reference guide to help interpret the experimental results and formal proofs presented in the following chapters. We collect and explain the key performance metrics used to evaluate the efficiency of masked implementations, describe the security tests conducted, outline the adversary models considered, and detail the verification techniques applied throughout this thesis. The goal is to ensure that when metrics such as area or latency are reported, a result of leakage analysis is shown, or an adversarial assumption is referenced in the latter half of this thesis, the reader can easily recall what these terms mean and how they are evaluated. The sections that follow define the efficiency metrics (area, latency, randomness), describe the HDL synthesis and measurement setups, outline the methodology for side-channel leakage testing, formalize attacker models and security notions, and explain how formal security proofs and verification tools are used.

4.1 Efficiency Metrics

Area Overhead. We quantify hardware area using standard conventions for both ASIC and FPGA implementations. For ASICs (Application-Specific Integrated Circuits), area is typically reported in *Gate Equivalents (GE)* a unit meant to be technology-independent, where 1 GE corresponds to the area of a two-input NAND gate with minimum drive strength in the selected technology. In practice, the total synthesized silicon area is divided by the

area of a single 2-input NAND gate to yield the result in GE [39, 177]. This normalization is intended to enable fair comparisons across different technology nodes. For example, comparing designs synthesized in 90 nm versus 65 nm without distortion from process scaling. Authors typically specify the target library and process node (e.g., "UMC 180 nm standard cell library") alongside GE values to provide context [117, 87].

For FPGA (Field-Programmable Gate Array) targets, area is expressed in terms of Look-Up Tables (LUTs) and Flip-Flops (FFs). We report area using units appropriate to the platform: kGE for ASICs and LUT/FF counts for FPGAs. Unless otherwise noted, the reported area excludes the logic for random number generation (e.g., PRNG), focusing solely on the core cryptographic implementation. This is consistent with common practice, as randomness generation is typically treated as a separate module.

Randomness Consumption. Many masking countermeasures require *fresh randomness*, meaning newly generated random values for each execution of a cryptographic operation. This ensures that sensitive intermediate values are masked with independent random bits during every evaluation. In contrast, *reusable randomness* refers to techniques where the same random mask or bits are used across multiple operations to save hardware resources. However, such reuse can introduce statistical dependencies that an adversary might exploit, and must therefore be handled with care.

In hardware, randomness is typically provided by a random number generator, either a TRNG or a PRNG, which supplies random bits to the masking logic as needed. To quantify randomness consumption, one can measure the number of fresh random bits required per operation. At the lowest level, this refers to the bits consumed by an elementary gadget, such as a masked AND gate or a masked S-box. At the next level, one can sum the randomness across all gadgets within a single round of a cryptographic algorithm (e.g., one AES round). Finally, the total randomness required per encryption is the accumulation of fresh bits over all rounds.

High randomness requirements are a practical bottleneck, particularly in round-based architectures where multiple S-boxes are evaluated in parallel and require fresh random bits each clock cycle. This places a significant burden on the PRNG, which must sustain high throughput to meet the online demand. For instance, a first-order masked cipher can require several thousand fresh random bits per encryption, which explains why many designs explicitly aim to minimize randomness consumption as a key optimization goal [101, 100].

Latency. We measure latency in the hardware context as the number of clock cycles required for an operation to produce a result from the time the inputs arrive. For a cryptographic core, latency typically refers to the number of cycles per encryption for a block cipher. For smaller components such as an S-box, it is common practice to define latency in terms of register stages. That is, the latency of an AES S-box is typically reported as the number of register stages within the S-box. A single-cycle S-box contains one internal register stage and produces one output per cycle once pipelined.

Another important timing metric is the *critical path delay*, which refers to the worst-case propagation delay through combinational logic between two register stages. The critical path determines the maximum clock frequency $f_{\max}$ at which the circuit can operate, typically reported in MHz and calculated as the inverse of the critical path delay after synthesis.

There is an inherent trade-off between latency and critical path. Pushing more computation into a single clock cycle can reduce latency but typically lengthens the critical path and lowers $f_{\max}$. For example, our secure AND gadget targeting practical security, secAND-PD (introduced in Chapter 7), computes a masked AND in a single clock cycle by introducing deliberate path delays to stagger the arrival of input shares. This achieves a single-cycle latency but reduces the maximum operating frequency. Therefore, it is often necessary to consider both the cycle count and the resulting $f_{\max}$ when comparing implementations. More clock cycles with higher operating frequency can, in some cases, offer better overall performance.

4.2 Side-Channel Resistance Metrics

In addition to implementation efficiency, cryptographic hardware must be evaluated for its resistance to side-channel attacks. This section outlines the standard methodologies used to assess side-channel leakage, with a focus on statistical testing techniques.

4.2.1 Test Vector Leakage Assessment (TVLA)

Test Vector Leakage Assessment (TVLA) [178, 179, 176] is a widely adopted methodology for detecting the presence of side-channel leakage in cryptographic implementations. It is based on Welch's t-test, a statistical hypothesis test used to determine whether two sets of measurements have significantly different means. TVLA can be performed with two configurations: non-specific, which

does not target any particular intermediate value but rather detects statistically significant data-dependent variations in general, and specific, which tests leakage hypotheses on known sensitive intermediates [75]. Due to its simplicity and efficiency, especially in the fixed-versus-random configuration, non-specific TVLA has become a standard preliminary assessment method in both academia and industry.

Trace Acquisition. The TVLA procedure begins with the acquisition of power traces from the device under test. A standard approach is the fixed-versus-random strategy, where two groups of traces are collected:

- Group 1 uses a fixed input (commonly a constant plaintext).
- Group 2 uses random inputs (a different random plaintext per encryption).

In our experiments, the secret key remains constant in both cases, ensuring that the only input-dependent variation arises from the plaintext. This isolates the influence of input data on the device's power consumption and facilitates the detection of leakage.

Welch's *t*-Test. For each time sample in the trace waveform, a t-value is computed as follows:

$$t = \frac{\overline{X}_1 - \overline{X}_2}{\sqrt{\frac{s_1^2}{n_1} + \frac{s_2^2}{n_2}}}$$

where $\overline{X}_1$ and $\overline{X}_2$ are the mean values of the fixed and random input trace sets, s_1 and s_2 are their respective standard deviations, and n_1, n_2 denote the number of traces in each group. The null hypothesis H_0 assumes that the two means are equal, indicating no data-dependent leakage. The alternative hypothesis H_1 suggests a statistically significant difference, implying leakage.

Pass/Fail Decision. The test passes if the absolute t-value $|t|$ remains below a chosen threshold across all time samples. Otherwise, it fails. A commonly used threshold is $|t| > 4.5$, corresponding to a confidence level of approximately 99.999% under large sample assumptions. A failure suggests the presence of detectable leakage, prompting further analysis or mitigation.

Higher-Order Extensions. First-order TVLA detects leakage correlated with the first statistical moment (i.e., the mean). For unprotected implementations, large t-statistic spikes above the threshold often appear with relatively few traces, indicating immediate and strong leakage. In contrast, a first-order masked implementation is expected to pass this test, maintaining t-values within the [-4.5, 4.5] range even after a large number of traces.

However, even if first-order leakage is not present, higher-order leakage may still exist. Such leakage can arise either from higher statistical moments within a single time sample (univariate leakage) or from joint dependencies among multiple time samples (multivariate leakage), where no single sample alone reveals information, but combinations of samples do. To detect this, TVLA can be extended to higher orders through preprocessing [180]. A common approach is to compute the centered product of two time samples from the same trace, including the possibility of combining a sample with itself. This captures second-order effects both via variance in single samples and via covariance between different samples. The same Welch's t-test is then applied to these derived values. If the resulting t-values exceed the threshold, the device is considered to leak at second order. This technique naturally generalizes. Third-order leakage can be tested by combining three samples, and so on.

The primary role of TVLA is to act as a conservative screening mechanism. If a device passes TVLA up to a certain order, one concludes that there is no statistically significant evidence of leakage of that order, within the limitations of the sample size and noise. If the test fails, meaning the t-values exceed the threshold, the implementation is considered to leak. In such cases, evaluators typically proceed to diagnose the source of leakage or attempt an exploitative side-channel attack to assess the severity and exploitability of the information leak.

4.2.2 Attack-Based Metrics

While TVLA is a detection-oriented methodology that assesses whether any measurable leakage exists, attack-based metrics aim to evaluate the exploitability of this leakage. Specifically, they quantify how much effort an adversary would need to exert in order to extract secret information from the device. In practice, these approaches complement leakage detection and offer a more application-oriented perspective.

Several common attack techniques include Simple Power Analysis (SPA) [11], Differential Power Analysis (DPA) [11], Correlation Power Analysis (CPA) [42], and Mutual Information Analysis (MIA) [181]. These methods attempt to recover the secret key from side-channel measurements and thereby provide

a quantifiable measure of vulnerability. The outcome of such evaluations is typically expressed in terms of success rate, number of traces required, or probability of successful key recovery under defined conditions [182, 7].

Simple Power Analysis (SPA). [11] SPA is the most basic form of side-channel attack and involves visually inspecting power traces to identify secret-dependent variations. In the context of hardware masking, this may involve observing trace features that correlate with the processing of sensitive values, such as unmasked intermediates or operations where masking is applied incorrectly or inconsistently.

SPA typically requires prior knowledge of the hardware design, including the timing of specific computations and the locations in the power trace where their effects are likely to appear. The attacker must know what to look for in the waveform and when critical masked or unmasked operations are expected to occur. Without such structural insight, SPA becomes difficult to apply, especially for securely masked implementations.

Differential Power Analysis (DPA). [11] DPA builds upon SPA by applying statistical techniques to extract key-dependent information from power traces where leakage is not immediately visible. Introduced by Kocher et al., DPA exploits correlations between power consumption and internal data values that depend on the secret key.

In a typical DPA attack, the adversary guesses a portion of the secret key and uses it to compute an intermediate value based on known inputs. Power traces are then divided into groups according to a predicted property of this intermediate, such as whether a specific bit is 0 or 1. The average trace of each group is computed, and the difference between them is analyzed. If the key guess is correct, a visible peak may appear at the point in time when the corresponding computation occurs. This indicates consistent statistical variation linked to the guessed value. If the guess is incorrect, the grouping is essentially random and no meaningful pattern emerges.

DPA does not require detailed knowledge of the hardware structure but does rely on access to inputs and a model of how data-dependent values propagate. It is therefore not fully black-box, though it remains widely applicable. Its strength lies in the ability to work with large sets of traces and detect first-order leakage, even when individual traces do not reveal clear information.

Correlation Power Analysis (CPA). [42] CPA is a more refined and widely used variant of DPA that employs correlation coefficients as the distinguishing metric instead of simple differences. Rather than partitioning traces into just two groups, CPA uses a hypothetical leakage model, for example, assuming that the device's power consumption is proportional to the Hamming weight of an intermediate value.

For each possible subkey value, the attacker computes a prediction of the intermediate, such as the S-box output of AES given the guessed subkey and the known plaintext. Based on this, the attacker then derives a predicted leakage, typically the Hamming weight of the intermediate. The Pearson correlation coefficient is then computed between the vector of predicted leakages and the actual measured power at each sample point in the trace.

The correct key hypothesis is expected to yield a higher correlation at the point in time when the targeted intermediate is processed, since the predicted leakage aligns with the actual data-dependent variation. In practice, the CPA procedure produces correlation traces for each subkey guess, and the guess with the most significant correlation peak is selected as the likely correct key.

CPA is designed to quantify linear dependence between power consumption and a secret-dependent variable. It offers a more sensitive and robust metric than basic DPA, often requiring fewer traces. This is because it leverages multi-valued predictions rather than relying on a binary partition, thus extracting more information from the leakage.

Mutual Information Analysis (MIA). [181] Mutual Information Analysis is a side-channel attack technique based on information theory. Unlike CPA, which assumes a linear relationship between a predicted leakage and measured power, MIA estimates the mutual information $I(X;Z)$ between a hypothetical secret-dependent variable X (such as the Hamming weight of an intermediate) and the observed side-channel measurements Z.

The mutual information quantifies the reduction in uncertainty of one variable given knowledge of the other. In this context, higher mutual information indicates stronger statistical dependence between the predicted leakage and the actual measurements. In a typical MIA attack, the adversary evaluates $I(\text{intermediate};\text{power})$ for each key hypothesis. The correct key is expected to maximize this value, as it aligns the hypothetical leakage with the true leakage behavior of the device.

One of the main advantages of MIA is its generality. It does not require assumptions about linearity or Gaussian noise, and can detect arbitrary statistical dependencies (including non-linear relationships that CPA might

miss). This makes it particularly useful in scenarios where the leakage function is unknown or where standard models like Hamming weight are not appropriate.

However, this generality comes at a cost. In practice, estimating mutual information is computationally more intensive than computing correlation coefficients. MIA also typically requires more traces to achieve the same level of statistical confidence, as density estimation (e.g., using histograms) introduces additional variability. Despite these challenges, MIA is a valuable evaluation tool in side-channel analysis, particularly when the goal is to confirm the absence of any exploitable leakage rather than just the absence of linear correlation.

Generally, leakage detection techniques such as Test Vector Leakage Assessment (TVLA) and attack-based evaluations are considered complementary in side-channel security analysis. A common methodology is to begin with TVLA as a quick statistical scan to detect the presence of exploitable leakage at a given order. For example, if an implementation passes first-order TVLA, it is typically unnecessary to attempt a first-order attack such as CPA or DPA, as the absence of detectable leakage suggests that such attacks would either fail or require an unrealistic effort.

In the context of heuristic masking, as explored in Chapters 6 & 7, which represents the practical security track of this thesis, the objective is not to achieve formal guarantees but to make key recovery attacks significantly harder in practice. In this chapter, the goal of the evaluation is to show that first-order TVLA does not detect leakage until a very high number of traces are used, indicating that any potential leakage is too small to be exploited easily. In typical evaluation workflows, such a scenario would justify an attack-based analysis, such as CPA, to quantify how difficult it is to extract the key and to assess the severity of the leakage. By contrast, we do not employ MIA here: being order-agnostic, it can flag dependence arising at any order, whereas our evaluation scope is explicitly first-order. While no such attacks are performed in this thesis, their conceptual role remains important in understanding the level of protection that heuristic masking provides under practical conditions.

In contrast, the constructions in Chapters 8 & 9, are designed to satisfy formal first- and second-order security notions under the robust probing model (Section 2.5.3). These designs are expected to pass first- and second-order TVLA by construction, and the evaluation is limited to confirming that this is indeed the case. Since these constructions come with provable guarantees, attack-based evaluation is neither required nor meaningful at the orders for which security is formally established. To make such guarantees concrete, the next section discusses how these security properties are formally defined and verified, and how formal tools are used to validate that the implementations meet the required security notions.

4.3 Formal Verification

In Section 2.5, we introduced formal security notions such as probing security, non-interference (NI), strong non-interference (SNI), and probe-isolating non-interference (PINI). These notions serve as abstractions that enable secure composition of smaller gadgets into larger masked circuits. For designs to be considered provably secure, it is necessary to formally establish that they satisfy one or more of these notions.

Orthogonal to practical, empirical validation through leakage detection lies the approach of formal verification. When we claim that a construction is formally secure, we rely not only on its theoretical structure but also on concrete proofs and automated tools that validate compliance with a given security notion. This section provides an accessible overview of how such security proofs are structured and how formal verification tools are used to support the claims made in this thesis. This discussion is intended as a high-level introduction for readers who may be unfamiliar with formal security proofs in hardware masking; those seeking a more rigorous treatment are referred to the cited literature for technical details and complete definitions.

4.3.1 Manual Security Proofs

For new masking schemes or gadgets introduced in this thesis (particularly in Chapters 8 and 9), we frequently provide simulation-based proofs [183, 184, 91, 185, 168] of security. A simulation-based proof is a formal argument that shows, for every set of observations (or probes) an adversary may obtain, one can construct a simulator that can produce the same joint distribution of values observed at the probes using only limited, non-sensitive information. If such a simulator can be constructed and satisfies the constraints posed by the security notion in question, this establishes the formal security of the design under the considered security notion.

Throughout this thesis, we rely on well-established formal notions such as *Non-Interference* (NI), *Strong Non-Interference* (SNI), and *Probe-Isolating Non-Interference* (PINI) to reason about the security of masked hardware constructions. Each of these notions imposes specific constraints on what information a simulator is allowed to access when reconstructing the values seen by an adversary's probes. Under NI, a total of $t = t_1 + t_2$ probes, of which t_1 are placed on internal wires and t_2 on outputs, must be simulatable using at most $t_1 + t_2$ input shares, taken across all inputs. SNI tightens this requirement by forbidding the simulator from accessing any input shares to simulate output probes; that is, only t_1 input shares may be used to simulate the entire set of t

$(=t_1 + t_2)$ probes. PINI further restricts the simulation condition by requiring that the input shares used must not only be limited in number but must also share the same indices across all inputs. In other words, the simulator may access up to t_1 shares, but only those that belong to the same index positions in each input. These increasingly strict constraints play a central role in how we structure the proofs that follow.

In the accompanying formal proofs across Chapters 8 and 9, the simulation-based security argument consistently follows a certain structure:

1. **Formal gadget definition.** Each proof begins by rigorously specifying the masked gadget under consideration. This includes a detailed description of the inputs and outputs, the use of fresh randomness, and the sequence of combinational operations and register stages that define the circuit's structure.

2. **Probing configuration.** We define the adversary's capabilities in terms of the security order targeted by the proof. For instance, a second-order security claim implies that the adversary may place at most two probes anywhere within the gadget.

3. **Probe classification and grouping.** Since exhaustively analyzing all possible probe placements is often infeasible, we group probes into equivalence classes based on their structural role within the gadget. These classes typically align with distinct stages in the circuit, such as combinational logic, register boundaries, randomness refresh layers, or final outputs. The key property of this classification is that any probe within the same class can be simulated using equivalent information. This allows us to construct the simulator for a representative probe in each class, thereby covering all relevant cases while keeping the proof tractable.

4. **Simulator construction and indistinguishability argument.** For each equivalence class of probes, we begin by identifying the minimal set of input shares required to simulate the corresponding values. Once this set is determined, we construct a simulator that uses only these permitted shares to generate values that are statistically indistinguishable from the distribution observed by the adversary at the probed positions. A key enabler of this simulation is the injection of fresh random masks throughout the gadget. These masks break statistical dependencies between intermediate signals and the underlying secrets, thereby simplifying the simulator's task. Once the simulation is shown to succeed, we verify that the identified share set adheres to the constraints imposed by the target security notion—such as a bound on the number of accessible shares (NI), disallowing input shares for simulating outputs (SNI), or enforcing

index-based isolation (PINI). This completes the argument that the gadget satisfies the required formal security definition.

Together, these steps establish a complete simulation-based argument that demonstrates compliance with the required formal security notion. While this section outlines the overarching methodology, the specific proofs throughout Chapters 8 and 9 follow this structure in detail, adapting it as needed for each construction.

4.3.2 Tool-Assisted Formal Verification

Beyond manual proofs, many of the constructions in this thesis are verified using automated tools that implement the definitions of NI, SNI, and PINI in a symbolic or structural form. These tools allow us to exhaustively check that the required simulation constraints hold for all permitted probe configurations. The tools themselves were introduced earlier in Section 2.6, where their features and limitations were discussed in detail.

The overall workflow begins with a high-level description of a masked gadget or small circuit, typically written in Verilog. This design is then synthesized into a gate-level netlist using a standard-cell library. The netlist is subsequently analyzed by a verification tool. The desired security notion (e.g., probing security, PINI, etc.) and target order d are provided as configuration parameters. The tool checks exhaustively whether any tuple of d probes in the provided netlist violates the specified security property. If the design passes, the tool confirms that no violating probe combination was found. If the design fails, it returns a counterexample, typically a set of signal names that, when probed together, violate the notion. These counterexamples are useful during development, especially when testing early versions of new masking techniques that may contain subtle flaws.

While simulation-based proofs provide strong analytical assurances, they are manually constructed and subject to human error. Tool-based validation, by contrast, enables automated checking at the netlist level. It may, however, be limited by scalability. Using both approaches together improves confidence that a design is secure both in theory and in its actual implementation.

4.4 Conclusion

This chapter introduced the key metrics, methods, and tools used in the remainder of the thesis. We clarified how performance is measured, described our approach to evaluating side-channel leakage, and explained the framework for formal security proofs and automated verification. These concepts will be referenced frequently in subsequent chapters, providing the basis for clearly interpreting results and ensuring consistency throughout.

With this reference in place, the next chapters focus on the main contributions of the thesis. We recommend reading Chapters 6 to 9 before proceeding to the final conclusion in Chapter 5, as these chapters present the core research contributions and findings on which the concluding insights are based.

Chapter 5

Conclusion and Future Work

This chapter concludes the thesis by summarizing the main findings, reflecting on their significance and limitations, and suggesting directions for future research. At the outset of this thesis, we identified two main research directions worth pursuing: achieving practical security through heuristic methods, and attaining provable security with an emphasis on efficiency and low latency.

Focusing on the first of these, in Chapters 6 and 7, we explored engineering techniques to improve practical masking implementations. Rather than relying solely on rigid theoretical constructions, we adopted an iterative, experimental approach, developing and refining methods through extensive trial and error. A particular aspect of this work that we find noteworthy is the demonstration of a clear progression in security for the `secAND-PD` scheme, where we observed measurable reductions in leakage as the lengths of signal delays were increased. Unlike traditional masking designs, which often follow a linear path of design, formal proof or construction based on a set of rules, and single-shot implementation to verify security, our methodology embraced an adaptive learning curve, continuously testing and adjusting the design until it met the desired security targets. This pragmatic approach highlights the value of engineering intuition and experimental feedback in achieving practical side-channel security.

However, the process was not without significant challenges. One of the more demanding aspects was discovering how to horizontally chain lookup tables to create sufficient signal delays in a deterministic manner. This required working at a very low-level, including setting `BEL`, `LOC` constraints for placement and performing considerable manual design effort. Unlike typical masking schemes, which often remain at a high level of abstraction and can afford to ignore such

low-level implementation details, our approach necessitated engaging deeply with the physical design aspects of FPGA architecture. Additionally, we observed that increasing the delay lengths not only led to reductions in side-channel leakage but also introduced trade-offs, such as a decrease in the maximum achievable frequency (f_{max}), which is typically undesirable in high-performance implementations.

In Chapters 8 and 9, we introduced a novel low-latency masking scheme that provides formal security guarantees and is applicable to functions with high algebraic degree while maintaining fixed latency, specifically, single-cycle latency for first-order security and two cycles for second-order security and so on. Our results demonstrate that this approach offers significant efficiency improvements over existing low-latency techniques, particularly for large functions such as the AES S-box. Although a performance gap remains between our low-latency masking approach and traditional schemes that prioritize minimizing area and randomness, our work narrows this gap and contributes valuable insights into the trade-offs involved in low-latency design. To further explore these trade-offs, we introduced a design variant in Chapter 9 that increases latency by one cycle but offers reductions in area and randomness requirements. Additionally, in Chapter 9, we presented a higher-order extension of our scheme, which, however, introduces some additional implementation costs. Achieving more efficient higher-order masking with low latency remains an open and promising research direction; for example, designing an efficient single-cycle, second-order masked AES S-box continues to be a compelling challenge for future work.

All of our constructions in Chapters 8 and 9 were designed to comply with the *robust d-probing model*, which, while offering strong formal guarantees, is often considered overly restrictive and not always necessary for achieving practical security. This view is supported by the authors of the *Robust-but-Relaxed d-probing model* [127, 126], who show that relaxing certain assumptions can preserve real-world security while enabling efficient masked implementations that would otherwise fail under the strict robust d-probing requirements. Beyond these approaches, several other leakage models have been proposed and actively studied in recent years [186, 187, 131], each introducing different assumptions about the adversary's capabilities.

Nevertheless, a gap remains between what formal security notions aim to guarantee and what is practically necessary for secure hardware. These circuits ultimately operate in real-world cryptographic devices, where physical effects such as coupling and layout-induced leakage introduce complexities that are not yet fully understood [188, 189, 190, 191, 192]. While formal security notions impose valuable structure and rigor on circuit design, they cannot fully capture all physical realities. Bridging the gap between what is provably secure and what is practically secure remains an open and significant research challenge.

Finally, this thesis has highlighted the limitations of existing formal verification tools. While each tool brings certain strengths, they also come with shortcomings that often require workarounds to perform thorough analyses of the implementations under examination. Although it may be an ambitious goal, it is important to acknowledge that no current tool simultaneously offers all four desirable properties: (i) efficiency (fast results), (ii) accuracy (sound and complete analysis), (iii) scalability (applicability to large circuits), and (iv) functionality (support for a wide range of security notions). Developing such unified tools remains an open research challenge and would represent a significant contribution to the field.

Bibliography

[1] Y. Eslami, A. Sheikholeslami, P. Gulak, S. Masui, and K. Mukaida. "An area-efficient universal cryptography processor for smart cards". In: *IEEE Transactions on Very Large Scale Integration (VLSI) Systems* 14.1 (Jan. 2006), pp. 43–56. ISSN: 1063-8210. DOI: 10.1109/tvlsi.2005.863188. URL: https://doi.org/10.1109/tvlsi.2005.863188.

[2] D. Naccache and D. M'Raihi. "Cryptographic smart cards". In: *IEEE Micro* 16.3 (June 1996), pp. 14, 16–24. ISSN: 0272-1732. DOI: 10.1109/40.502402. URL: https://doi.org/10.1109/40.502402.

[3] V. A. Thakor, M. A. Razzaque, and M. R. A. Khandaker. "Lightweight Cryptography Algorithms for Resource-Constrained IoT Devices: A Review, Comparison and Research Opportunities". In: *IEEE Access* 9 (2021), pp. 28177–28193. ISSN: 2169-3536. DOI: 10.1109/access.2021.3052867. URL: https://doi.org/10.1109/access.2021.3052867.

[4] K.-K. R. Choo, S. Gritzalis, and J. H. Park. "Cryptographic Solutions for Industrial Internet-of-Things: Research Challenges and Opportunities". In: *IEEE Transactions on Industrial Informatics* 14.8 (Aug. 2018), pp. 3567–3569. ISSN: 1551-3203. DOI: 10.1109/tii.2018.2841049. URL: https://doi.org/10.1109/tii.2018.2841049.

[5] M. N. Khan, A. Rao, and S. Camtepe. "Lightweight Cryptographic Protocols for IoT-Constrained Devices: A Survey". In: *IEEE Internet of Things Journal* 8.6 (Mar. 2021), pp. 4132–4156. ISSN: 2327-4662. DOI: 10.1109/jiot.2020.3026493. URL: https://doi.org/10.1109/jiot.2020.3026493.

[6] V. Costan and S. Devadas. *Intel SGX Explained*. Cryptology ePrint Archive, Paper 2016/086. 2016. URL: https://eprint.iacr.org/2016/086.

[7] S. Mangard, E. Oswald, and T. Popp. "Power analysis attacks - revealing the secrets of smart cards". In: 2007. URL: https://api.semanticscholar.org/CorpusID:44510491.

[8] J. Katz and Y. Lindell. *Introduction to Modern Cryptography, Second Edition*. 2nd. Chapman & Hall/CRC, 2014. ISBN: 1466570261.

[9] M. Rostami, F. Koushanfar, and R. Karri. "A Primer on Hardware Security: Models, Methods, and Metrics". In: *Proceedings of the IEEE* 102.8 (2014), pp. 1283–1295. ISSN: 0018-9219. DOI: 10.1109/jproc.2014.2335155. URL: https://doi.org/10.1109/jproc.2014.2335155.

[10] T. S. Messerges, E. A. Dabbish, and R. H. Sloan. "Investigations of power analysis attacks on smartcards". In: *Proceedings of the USENIX Workshop on Smartcard Technology on USENIX Workshop on Smartcard Technology*. WOST'99. Chicago, Illinois: USENIX Association, 1999, p. 17.

[11] P. Kocher, J. Jaffe, and B. Jun. "Differential Power Analysis". In: *Advances in Cryptology — CRYPTO' 99*. Ed. by M. Wiener. Berlin, Heidelberg: Springer Berlin Heidelberg, 1999, pp. 388–397. ISBN: 978-3-540-48405-9.

[12] K. Gandolfi, C. Mourtel, and F. Olivier. "Electromagnetic Analysis: Concrete Results". In: *Cryptographic Hardware and Embedded Systems — CHES 2001*. Ed. by Ç. K. Koç, D. Naccache, and C. Paar. Berlin, Heidelberg: Springer Berlin Heidelberg, 2001, pp. 251–261. ISBN: 978-3-540-44709-2.

[13] J.-J. Quisquater and D. Samyde. "ElectroMagnetic Analysis (EMA): Measures and Counter-measures for Smart Cards". In: *Smart Card Programming and Security*. Ed. by I. Attali and T. Jensen. Berlin, Heidelberg: Springer Berlin Heidelberg, 2001, pp. 200–210. ISBN: 978-3-540-45418-2.

[14] P. C. Kocher. "Timing Attacks on Implementations of Di(cid:14)e-Hellman, RSA, DSS, and Other Systems". In: 1996. URL: https://api.semanticscholar.org/CorpusID:15475583.

[15] Y. Zhou and D. Feng. "Side-Channel Attacks: Ten Years After Its Publication and the Impacts on Cryptographic Module Security Testing". In: *IACR Cryptol. ePrint Arch.* (2005), p. 388. URL: http://eprint.iacr.org/2005/388.

[16] M. Randolph and W. Diehl. "Power Side-Channel Attack Analysis: A Review of 20 Years of Study for the Layman". In: *Cryptography* 4.2 (2020), p. 15. ISSN: 2410-387X. DOI: 10.3390/cryptography4020015. URL: https://doi.org/10.3390/cryptography4020015.

[17] F.-X. Standaert, G. Piret, and J.-J. Quisquater. "Cryptanalysis of block ciphers: A survey". In: *UCL Crypto Group* (2003).

[18] M.-L. Akkar and C. Giraud. "An Implementation of DES and AES, Secure against Some Attacks". In: *Cryptographic Hardware and Embedded Systems — CHES 2001*. Ed. by Ç. K. Koç, D. Naccache, and C. Paar. Berlin, Heidelberg: Springer Berlin Heidelberg, 2001, pp. 309–318. ISBN: 978-3-540-44709-2.

[19] N. Veyrat-Charvillon, M. Medwed, S. Kerckhof, and F.-X. Standaert. "Shuffling against Side-Channel Attacks: A Comprehensive Study with Cautionary Note". In: *Advances in Cryptology – ASIACRYPT 2012*. Ed. by X. Wang and K. Sako. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, pp. 740–757. ISBN: 978-3-642-34961-4.

[20] S. Chari, C. S. Jutla, J. R. Rao, and P. Rohatgi. "Towards Sound Approaches to Counteract Power-Analysis Attacks". In: *Advances in Cryptology — CRYPTO' 99*. Ed. by M. Wiener. Berlin, Heidelberg: Springer Berlin Heidelberg, 1999, pp. 398–412. ISBN: 978-3-540-48405-9.

[21] S. Nikova, C. Rechberger, and V. Rijmen. "Threshold Implementations Against Side-Channel Attacks and Glitches". In: *Lecture Notes in Computer Science*. Springer Berlin Heidelberg, 2006, pp. 529–545. ISBN: 9783540494966. DOI: 10.1007/11935308_38. URL: https://doi.org/10.1007/11935308_38.

[22] H. Gross, S. Mangard, and T. Korak. "Domain-Oriented Masking". In: *Proceedings of the 2016 ACM Workshop on Theory of Implementation Security*. ACM. ACM, Oct. 2016, pp. 3–3. DOI: 10.1145/2996366.2996426. URL: https://doi.org/10.1145/2996366.2996426.

[23] S. Faust, C. Paglialonga, and T. Schneider. "Amortizing Randomness Complexity in Private Circuits". In: *Lecture Notes in Computer Science*. Springer International Publishing, 2017, pp. 781–810. ISBN: 9783319706931. DOI: 10.1007/978-3-319-70694-8_27. URL: https://doi.org/10.1007/978-3-319-70694-8_27.

[24] J. Feldtkeller, D. Knichel, P. Sasdrich, A. Moradi, and T. Güneysu. "Randomness Optimization for Gadget Compositions in Higher-Order Masking". In: *IACR Transactions on Cryptographic Hardware and Embedded Systems* (Aug. 2022), pp. 188–227. ISSN: 2569-2925. DOI: 10.46586/tches.v2022.i4.188-227. URL: https://doi.org/10.46586/tches.v2022.i4.188-227.

[25] T. De Cnudde, O. Reparaz, B. Bilgin, S. Nikova, V. Nikov, and V. Rijmen. "Masking AES With d+1 Shares in Hardware". In: *Proceedings of the 2016 ACM Workshop on Theory of Implementation Security*. ACM. ACM, Oct. 2016, pp. 43–43. DOI: 10.1145/2996366.2996428. URL: https://doi.org/10.1145/2996366.2996428.

[26] L. De Meyer, O. Reparaz, and B. Bilgin. “Multiplicative Masking for AES in Hardware”. In: *IACR Transactions on Cryptographic Hardware and Embedded Systems* (Aug. 2018), pp. 431–468. ISSN: 2569-2925. DOI: 10.46586/tches.v2018.i3.431-468. URL: https://doi.org/10.46586/tches.v2018.i3.431-468.

[27] T. Moos, A. Moradi, T. Schneider, and F.-X. Standaert. “Glitch-Resistant Masking Revisited”. In: *IACR Transactions on Cryptographic Hardware and Embedded Systems* (Feb. 2019), pp. 256–292. ISSN: 2569-2925. DOI: 10.46586/tches.v2019.i2.256-292. URL: https://doi.org/10.46586/tches.v2019.i2.256-292.

[28] E. Prouff and M. Rivain. “Masking against Side-Channel Attacks: A Formal Security Proof”. In: *Lecture Notes in Computer Science*. Springer Berlin Heidelberg, 2013, pp. 142–159. ISBN: 9783642383472. DOI: 10.1007/978-3-642-38348-9_9. URL: https://doi.org/10.1007/978-3-642-38348-9_9.

[29] R. Bloem, H. Gross, R. Iusupov, B. Könighofer, S. Mangard, and J. Winter. “Formal Verification of Masked Hardware Implementations in the Presence of Glitches”. In: *Lecture Notes in Computer Science*. Springer International Publishing, 2018, pp. 321–353. ISBN: 9783319783741. DOI: 10.1007/978-3-319-78375-8_11. URL: https://doi.org/10.1007/978-3-319-78375-8_11.

[30] S. Nikova, V. Rijmen, and M. Schläffer. “Secure Hardware Implementation of Nonlinear Functions in the Presence of Glitches”. In: *Journal of Cryptology* 24.2 (Oct. 2010), pp. 292–321. ISSN: 0933-2790. DOI: 10.1007/s00145-010-9085-7. URL: https://doi.org/10.1007/s00145-010-9085-7.

[31] A.-E. M. Taha, A. M. Rashwan, and H. S. Hassanein. “Secure Communications for Resource-Constrained IoT Devices”. In: *Sensors* 20.13 (2020), p. 3637. ISSN: 1424-8220. DOI: 10.3390/s20133637. URL: https://doi.org/10.3390/s20133637.

[32] B. Lampe and W. Meng. “Intrusion Detection in the Automotive Domain: A Comprehensive Review”. In: *IEEE Communications Surveys & Tutorials* 25.4 (2023), pp. 2356–2426. ISSN: 1553-877X. DOI: 10.1109/comst.2023.3309864. URL: https://doi.org/10.1109/comst.2023.3309864.

[33] P. Tran and D. C. Nguyen. “Advanced Lightweight Cryptography for Automotive Security: Surveys, Challenges and Solutions”. In: SCITEPRESS - Science and Technology Publications, Jan. 2022, pp. 304–311. DOI: 10.5220/0011109500003194. URL: https://doi.org/10.5220/0011109500003194.

[34] A. Goulart, A. Chennamaneni, D. Torre, B. Hur, and F. Y. Al-Aboosi. "On Wide-Area IoT Networks, Lightweight Security and Their Applications—A Practical Review". In: *Electronics* 11.11 (2022), p. 1762. ISSN: 2079-9292. DOI: 10.3390/electronics11111762. URL: https://doi.org/10.3390/electronics11111762.

[35] J. Borghoff, A. Canteaut, T. Güneysu, E. B. Kavun, M. Knezevic, L. R. Knudsen, G. Leander, V. Nikov, C. Paar, C. Rechberger, P. Rombouts, S. S. Thomsen, and T. Yalçın. "PRINCE – A Low-Latency Block Cipher for Pervasive Computing Applications". In: *Advances in Cryptology – ASIACRYPT 2012*. Ed. by X. Wang and K. Sako. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, pp. 208–225. ISBN: 978-3-642-34961-4.

[36] National Institute of Standards and Technology (NIST). *Advanced Encryption Standard (AES)*. FIPS Publication 197. Nov. 2001. URL: https://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.197.pdf.

[37] National Bureau of Standards. *Data Encryption Standard (DES)*. FIPS PUB 46. U.S. Department of Commerce. Jan. 1977.

[38] S. Brown and Z. Vranesic. *Fundamentals of Digital Logic with VHDL Design with CD-ROM*. 3rd ed. USA: McGraw-Hill, Inc., 2008. ISBN: 0077221435.

[39] H. Kaeslin. *Digital Integrated Circuit Design: From VLSI Architectures to CMOS Fabrication*. 1st. USA: Cambridge University Press, 2008. ISBN: 0521882672.

[40] V. Pedroni. *Digital Electronics and Design with VHDL*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2008. ISBN: 0123742706.

[41] V. Fischer. "A Closer Look at Security in Random Number Generators Design". In: *Constructive Side-Channel Analysis and Secure Design*. Ed. by W. Schindler and S. A. Huss. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, pp. 167–182. ISBN: 978-3-642-29912-4.

[42] E. Brier, C. Clavier, and F. Olivier. "Correlation Power Analysis with a Leakage Model". In: *Lecture Notes in Computer Science*. Springer Berlin Heidelberg, 2004, pp. 16–29. ISBN: 9783540226666. DOI: 10.1007/978-3-540-28632-5_2. URL: https://doi.org/10.1007/978-3-540-28632-5_2.

[43] J.-S. Coron. "Resistance Against Differential Power Analysis For Elliptic Curve Cryptosystems". In: *Lecture Notes in Computer Science*. Springer Berlin Heidelberg, 1999, pp. 292–302. ISBN: 9783540666462. DOI: 10.1007/3-540-48059-5_25. URL: https://doi.org/10.1007/3-540-48059-5_25.

[44] L. Goubin and J. Patarin. “DES and Differential Power Analysis The “Duplication” Method”. In: *Lecture Notes in Computer Science*. Springer Berlin Heidelberg, 1999, pp. 158–172. ISBN: 9783540666462. DOI: 10.1007/3-540-48059-5_15. URL: https://doi.org/10.1007/3-540-48059-5_15.

[45] T. K., A. Moonmoon, and V. I. “A dynamic and differential CMOS logic with signal independent power consumption to withstand differential power analysis on smart cards”. In: *Proceedings of the 28th European Solid-State Circuits Conference* (2002).

[46] T. S. Messerges. “Using Second-Order Power Analysis to Attack DPA Resistant Software”. In: *Lecture Notes in Computer Science*. Springer Berlin Heidelberg, 2000, pp. 238–251. ISBN: 9783540414551. DOI: 10.1007/3-540-44499-8_19. URL: https://doi.org/10.1007/3-540-44499-8_19.

[47] L. Goubin and J. Patarin. “DES and Differential Power Analysis The “Duplication” Method”. In: *Cryptographic Hardware and Embedded Systems*. Ed. by Ç. K. Koç and C. Paar. Berlin, Heidelberg: Springer Berlin Heidelberg, 1999, pp. 158–172. ISBN: 978-3-540-48059-4.

[48] T. S. Messerges. “Using Second-Order Power Analysis to Attack DPA Resistant Software”. In: *Cryptographic Hardware and Embedded Systems — CHES 2000*. Ed. by Ç. K. Koç and C. Paar. Berlin, Heidelberg: Springer Berlin Heidelberg, 2000, pp. 238–251. ISBN: 978-3-540-44499-2.

[49] J.-S. Coron, E. Prouff, and M. Rivain. “Side Channel Cryptanalysis of a Higher Order Masking Scheme”. In: *Cryptographic Hardware and Embedded Systems - CHES 2007*. Ed. by P. Paillier and I. Verbauwhede. Berlin, Heidelberg: Springer Berlin Heidelberg, 2007, pp. 28–44. ISBN: 978-3-540-74735-2.

[50] O. Reparaz, B. Bilgin, S. Nikova, B. Gierlichs, and I. Verbauwhede. “Consolidating Masking Schemes”. In: *Advances in Cryptology – CRYPTO 2015*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2015, pp. 764–783. ISBN: 9783662479889. DOI: 10.1007/978-3-662-47989-6_37. URL: https://doi.org/10.1007/978-3-662-47989-6_37.

[51] J. Waddle and D. Wagner. “Towards Efficient Second-Order Power Analysis”. In: *Lecture Notes in Computer Science*. Springer Berlin Heidelberg, 2004, pp. 1–15. ISBN: 9783540226666. DOI: 10.1007/978-3-540-28632-5_1. URL: https://doi.org/10.1007/978-3-540-28632-5_1.

[52] E. Oswald, S. Mangard, C. Herbst, and S. Tillich. “Practical Second-Order DPA Attacks for Masked Smart Card Implementations of Block Ciphers”. In: *Lecture Notes in Computer Science*. Springer Berlin Heidelberg, 2006, pp. 192–207. ISBN: 9783540310334. DOI: 10.1007/11605805_13. URL: https://doi.org/10.1007/11605805_13.

[53] F.-X. Standaert, N. Veyrat-Charvillon, E. Oswald, B. Gierlichs, M. Medwed, M. Kasper, and S. Mangard. “The World Is Not Enough: Another Look on Second-Order DPA”. In: *Lecture Notes in Computer Science*. Springer Berlin Heidelberg, 2010, pp. 112–129. ISBN: 9783642173721. DOI: 10.1007/978-3-642-17373-8_7. URL: https://doi.org/10.1007/978-3-642-17373-8_7.

[54] D. Suzuki, M. Saeki, and T. Ichikawa. “DPA Leakage Models for CMOS Logic Circuits”. In: *Cryptographic Hardware and Embedded Systems – CHES 2005*. Ed. by J. R. Rao and B. Sunar. Berlin, Heidelberg: Springer Berlin Heidelberg, 2005, pp. 366–382. ISBN: 978-3-540-31940-5.

[55] K. Tiri and I. Verbauwhede. “A logic level design methodology for a secure DPA resistant ASIC or FPGA implementation”. In: *Proceedings Design, Automation and Test in Europe Conference and Exhibition*. Vol. 1. IEEE Comput. Soc, None, 246–251 Vol.1. DOI: 10.1109/date.2004.1268856. URL: https://doi.org/10.1109/date.2004.1268856.

[56] T. Güneysu and A. Moradi. “Generic Side-Channel Countermeasures for Reconfigurable Devices”. In: *Cryptographic Hardware and Embedded Systems – CHES 2011*. Ed. by B. Preneel and T. Takagi. Berlin, Heidelberg: Springer Berlin Heidelberg, 2011, pp. 33–48. ISBN: 978-3-642-23951-9.

[57] D. Das, S. Maity, S. B. Nasir, S. Ghosh, A. Raychowdhury, and S. Sen. “High efficiency power side-channel attack immunity using noise injection in attenuated signature domain”. In: *2017 IEEE International Symposium on Hardware Oriented Security and Trust (HOST)*. IEEE, 2017, pp. 62–67. DOI: 10.1109/hst.2017.7951799. URL: https://doi.org/10.1109/hst.2017.7951799.

[58] W. Yu, O. A. Uzun, and S. Köse. “Leveraging on-chip voltage regulators as a countermeasure against side-channel attacks”. In: *Proceedings of the 52nd Annual Design Automation Conference*. DAC ’15. San Francisco, California: ACM, 2015, pp. 1–6. ISBN: 9781450335201. DOI: 10.1145/2744769.2744866. URL: https://doi.org/10.1145/2744769.2744866.

[59] J.-L. Danger, S. Guilley, S. Bhasin, and M. Nassar. “Overview of Dual rail with Precharge logic styles to thwart implementation-level attacks on hardware cryptoprocessors”. In: *2009 3rd International Conference*

on Signals, Circuits and Systems (SCS) (2009), pp. 1–8. URL: https://api.semanticscholar.org/CorpusID:8523478.

[60] M. Bucci, L. Giancane, R. Luzzi, and A. Trifiletti. "Three-Phase Dual-Rail Pre-charge Logic". In: *Cryptographic Hardware and Embedded Systems - CHES 2006*. Ed. by L. Goubin and M. Matsui. Berlin, Heidelberg: Springer Berlin Heidelberg, 2006, pp. 232–241. ISBN: 978-3-540-46561-4.

[61] D. Suzuki and M. Saeki. "Security Evaluation of DPA Countermeasures Using Dual-Rail Pre-charge Logic Style". In: *Cryptographic Hardware and Embedded Systems - CHES 2006*. Ed. by L. Goubin and M. Matsui. Berlin, Heidelberg: Springer Berlin Heidelberg, 2006, pp. 255–269. ISBN: 978-3-540-46561-4.

[62] T. Popp, M. Kirschbaum, T. Zefferer, and S. Mangard. "Evaluation of the Masked Logic Style MDPL on a Prototype Chip". In: *Cryptographic Hardware and Embedded Systems - CHES 2007*. Ed. by P. Paillier and I. Verbauwhede. Berlin, Heidelberg: Springer Berlin Heidelberg, 2007, pp. 81–94. ISBN: 978-3-540-74735-2.

[63] S. Guilley, S. Chaudhuri, L. Sauvage, T. Graba, J.-L. Danger, P. Hoogvorst, V.-N. Vong, M. Nassar, and F. Flament. "Shall we trust WDDL?" In: *Future of Trust in Computing: Proceedings of the First International Conference Future of Trust in Computing 2008*. Ed. by D. Gawrock, H. Reimer, A.-R. Sadeghi, and C. Vishik. Wiesbaden: Vieweg+Teubner, 2009, pp. 208–215. ISBN: 9783834807946. DOI: 10.1007/978-3-8348-9324-6_22. URL: https://doi.org/10.1007/978-3-8348-9324-6_22.

[64] K. Tiri and I. Verbauwhede. "Place and Route for Secure Standard Cell Design". In: *Smart Card Research and Advanced Applications VI*. Ed. by J.-J. Quisquater, P. Paradinas, Y. Deswarte, and A. A. El Kalam. Boston, MA: Springer US, 2004, pp. 143–158. ISBN: 978-1-4020-8147-7.

[65] K. Tiri and I. Verbauwhede. "A digital design flow for secure integrated circuits". In: *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 25.7 (2006), pp. 1197–1208. DOI: 10.1109/TCAD.2005.855939.

[66] J.-L. Danger, S. Guilley, S. Bhasin, and M. Nassar. "Overview of Dual rail with Precharge logic styles to thwart implementation-level attacks on hardware cryptoprocessors". In: *2009 3rd International Conference on Signals, Circuits and Systems (SCS)*. IEEE, 2009, pp. 1–8. DOI: 10.1109/icscs.2009.5412599. URL: https://doi.org/10.1109/icscs.2009.5412599.

[67] T. Popp and S. Mangard. “Masked Dual-Rail Pre-charge Logic: DPA-Resistance Without Routing Constraints”. In: *Cryptographic Hardware and Embedded Systems – CHES 2005*. Ed. by J. R. Rao and B. Sunar. Berlin, Heidelberg: Springer Berlin Heidelberg, 2005, pp. 172–186. ISBN: 978-3-540-31940-5.

[68] M. Bucci, R. Luzzi, M. Guglielmo, and A. Trifiletti. “A Countermeasure against Differential Power Analysis based on Random Delay Insertion”. In: *2005 IEEE International Symposium on Circuits and Systems (ISCAS)*. IEEE, None, pp. 3547–3550. DOI: 10.1109/iscas.2005.1465395. URL: https://doi.org/10.1109/iscas.2005.1465395.

[69] C. Herbst, E. Oswald, and S. Mangard. “An AES Smart Card Implementation Resistant to Power Analysis Attacks”. In: *Lecture Notes in Computer Science*. Springer Berlin Heidelberg, 2006, pp. 239–252. ISBN: 9783540354376. DOI: 10.1007/11767480_16. URL: https://doi.org/10.1007/11767480_16.

[70] N. Veyrat-Charvillon, M. Medwed, S. Kerckhof, and F.-X. Standaert. “Shuffling against Side-Channel Attacks: A Comprehensive Study with Cautionary Note”. In: *Lecture Notes in Computer Science*. Springer Berlin Heidelberg, 2012, pp. 740–757. ISBN: 9783642349607. DOI: 10.1007/978-3-642-34961-4_44. URL: https://doi.org/10.1007/978-3-642-34961-4_44.

[71] T. Güneysu and A. Moradi. “Generic Side-Channel Countermeasures for Reconfigurable Devices”. In: *Lecture Notes in Computer Science*. Springer Berlin Heidelberg, 2011, pp. 33–48. ISBN: 9783642239502. DOI: 10.1007/978-3-642-23951-9_3. URL: https://doi.org/10.1007/978-3-642-23951-9_3.

[72] G. Ratanpal, R. Williams, and T. Blalock. “An on-chip signal suppression countermeasure to power analysis attacks”. In: *IEEE Transactions on Dependable and Secure Computing* 1.3 (July 2004), pp. 179–189. ISSN: 1545-5971. DOI: 10.1109/tdsc.2004.25. URL: https://doi.org/10.1109/tdsc.2004.25.

[73] M. Rivain, E. Prouff, and J. Doget. “Higher-Order Masking and Shuffling for Software Implementations of Block Ciphers”. In: *Lecture Notes in Computer Science*. Springer Berlin Heidelberg, 2009, pp. 171–188. ISBN: 9783642041372. DOI: 10.1007/978-3-642-04138-9_13. URL: https://doi.org/10.1007/978-3-642-04138-9_13.

[74] I. Levi, D. Bellizia, D. Bol, and F.-X. Standaert. “Ask Less, Get More: Side-Channel Signal Hiding, Revisited”. In: *IEEE Transactions on Circuits and Systems I: Regular Papers* 67.12 (Dec. 2020), pp. 4904–4917. ISSN: 1549-8328. DOI: 10.1109/tcsi.2020.3005338. URL: https://doi.org/10.1109/tcsi.2020.3005338.

[75] B. Bilgin, B. Gierlichs, S. Nikova, V. Nikov, and V. Rijmen. "Higher-Order Threshold Implementations". In: *Lecture Notes in Computer Science*. Springer Berlin Heidelberg, 2014, pp. 326–343. ISBN: 9783662456071. DOI: 10.1007/978-3-662-45608-8_18. URL: https://doi.org/10.1007/978-3-662-45608-8_18.

[76] Y. Ishai, A. Sahai, and D. Wagner. "Private Circuits: Securing Hardware against Probing Attacks". In: *Advances in Cryptology - CRYPTO 2003*. Ed. by D. Boneh. Berlin, Heidelberg: Springer Berlin Heidelberg, 2003, pp. 463–481. ISBN: 978-3-540-45146-4.

[77] J. M. Rabaey. *Digital integrated circuits: a design perspective*. USA: Prentice-Hall, Inc., 1996. ISBN: 0131786091.

[78] S. Mangard, T. Popp, and B. M. Gammel. "Side-Channel Leakage of Masked CMOS Gates". In: *Topics in Cryptology – CT-RSA 2005*. Ed. by A. Menezes. Berlin, Heidelberg: Springer Berlin Heidelberg, 2005, pp. 351–365. ISBN: 978-3-540-30574-3.

[79] S. Mangard, N. Pramstaller, and E. Oswald. "Successfully Attacking Masked AES Hardware Implementations". In: *Cryptographic Hardware and Embedded Systems – CHES 2005*. Ed. by J. R. Rao and B. Sunar. Berlin, Heidelberg: Springer Berlin Heidelberg, 2005, pp. 157–171. ISBN: 978-3-540-31940-5.

[80] W. Fischer and B. M. Gammel. "Masking at Gate Level in the Presence of Glitches". In: *Cryptographic Hardware and Embedded Systems – CHES 2005*. Ed. by J. R. Rao and B. Sunar. Berlin, Heidelberg: Springer Berlin Heidelberg, 2005, pp. 187–200. ISBN: 978-3-540-31940-5.

[81] A. Shamir. "How to share a secret". In: *Communications of the ACM* 22.11 (Nov. 1979), pp. 612–613. ISSN: 0001-0782. DOI: 10.1145/359168.359176. URL: https://doi.org/10.1145/359168.359176.

[82] B. Bilgin, B. Gierlichs, S. Nikova, V. Nikov, and V. Rijmen. "Trade-Offs for Threshold Implementations Illustrated on AES". In: *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 34.7 (2015), pp. 1188–1200. ISSN: 0278-0070. DOI: 10.1109/tcad.2015.2419623. URL: https://doi.org/10.1109/tcad.2015.2419623.

[83] T. Sugawara. "3-Share Threshold Implementation of AES S-box without Fresh Randomness". In: *IACR Transactions on Cryptographic Hardware and Embedded Systems* 2019.1 (Nov. 2018), pp. 123–145. DOI: 10.13154/tches.v2019.i1.123-145. URL: https://tches.iacr.org/index.php/TCHES/article/view/7336.

[84] B. Bilgin, S. Nikova, V. Nikov, V. Rijmen, and G. Stütz. "Threshold Implementations of All 3 ×3 and 4 ×4 S-Boxes". In: *Cryptographic Hardware and Embedded Systems - CHES 2012 - 14th International Workshop, Leuven, Belgium, September 9-12, 2012. Proceedings*. Ed. by E. Prouff and P. Schaumont. Vol. 7428. Lecture Notes in Computer Science. Springer, 2012, pp. 76–91. DOI: 10.1007/978-3-642-33027-8_5. URL: https://doi.org/10.1007/978-3-642-33027-8%5C_5.

[85] E. Piccione, S. Andreoli, L. Budaghyan, C. Carlet, S. Dhooghe, S. Nikova, G. Petrides, and V. Rijmen. *An Optimal Universal Construction for the Threshold Implementation of Bijective S-boxes*. Cryptology ePrint Archive, Paper 2022/1141. 2022. URL: https://eprint.iacr.org/2022/1141.

[86] B. Bilgin, J. Daemen, V. Nikov, S. Nikova, V. Rijmen, and G. Van Assche. "Efficient and First-Order DPA Resistant Implementations of Keccak". In: *Smart Card Research and Advanced Applications*. Ed. by A. Francillon and P. Rohatgi. Cham: Springer International Publishing, 2014, pp. 187–199. ISBN: 978-3-319-08302-5.

[87] A. Moradi, A. Poschmann, S. Ling, C. Paar, and H. Wang. "Pushing the Limits: A Very Compact and a Threshold Implementation of AES". In: *Lecture Notes in Computer Science*. Springer Berlin Heidelberg, 2011, pp. 69–88. ISBN: 9783642204647. DOI: 10.1007/978-3-642-20465-4_6. URL: https://doi.org/10.1007/978-3-642-20465-4_6.

[88] D. Božilov, M. Knežević, and V. Nikov. "Optimized threshold implementations: securing cryptographic accelerators for low-energy and low-latency applications". In: *Journal of Cryptographic Engineering* 12.1 (Nov. 2021), pp. 15–51. ISSN: 2190-8508. DOI: 10.1007/s13389-021-00276-5. URL: https://doi.org/10.1007/s13389-021-00276-5.

[89] B. Begül. *Threshold implementations*. Doctoral dissertation. 2015. DOI: 10.3990/1.9789036538916. URL: http://dx.doi.org/10.3990/1.9789036538916.

[90] J. Daemen. *Changing of the Guards: a simple and efficient method for achieving uniformity in threshold sharing*. Cryptology ePrint Archive, Paper 2016/1061. 2016. URL: https://eprint.iacr.org/2016/1061.

[91] M. Rivain and E. Prouff. "Provably Secure Higher-Order Masking of AES". In: *Cryptographic Hardware and Embedded Systems, CHES 2010*. Ed. by S. Mangard and F.-X. Standaert. Berlin, Heidelberg: Springer Berlin Heidelberg, 2010, pp. 413–427. ISBN: 978-3-642-15031-9.

[92] J.-S. Coron, E. Prouff, M. Rivain, and T. Roche. *Higher-Order Side Channel Security and Mask Refreshing*. Cryptology ePrint Archive, Paper 2015/359. 2015. URL: https://eprint.iacr.org/2015/359.

[93] B. Bilgin, B. Gierlichs, S. Nikova, V. Nikov, and V. Rijmen. "Higher-Order Threshold Implementations". In: *Advances in Cryptology – ASIACRYPT 2014*. Ed. by P. Sarkar and T. Iwata. Berlin, Heidelberg: Springer Berlin Heidelberg, 2014, pp. 326–343. ISBN: 978-3-662-45608-8.

[94] G. Cassiers, B. Gregoire, I. Levi, and F.-X. Standaert. "Hardware Private Circuits: From Trivial Composition to Full Verification". In: *IEEE Transactions on Computers* 70.10 (2021), pp. 1677–1690. ISSN: 0018-9340. DOI: 10.1109/tc.2020.3022979. URL: https://doi.org/10.1109/tc.2020.3022979.

[95] G. Cassiers, B. Gregoire, I. Levi, and F.-X. Standaert. "Hardware Private Circuits: From Trivial Composition to Full Verification". In: *IEEE Transactions on Computers* 70.10 (Oct. 2021), pp. 1677–1690. ISSN: 0018-9340. DOI: 10.1109/tc.2020.3022979. URL: https://doi.org/10.1109/tc.2020.3022979.

[96] M. Bakiri, C. Guyeux, J.-F. Couchot, and A. K. Oudjida. "Survey on hardware implementation of random number generators on FPGA: Theory and experimental analyses". In: *Computer Science Review* 27 (2018), pp. 135–153. ISSN: 1574-0137. DOI: 10.1016/j.cosrev.2018.01.002. URL: https://doi.org/10.1016/j.cosrev.2018.01.002.

[97] C. K. Koc. *Cryptographic Engineering*. 1st. Springer Publishing Company, Incorporated, 2008. ISBN: 0387718168.

[98] B. Sunar, W. J. Martin, and D. R. Stinson. "A Provably Secure True Random Number Generator with Built-In Tolerance to Active Attacks". In: *IEEE Transactions on Computers* 56.1 (2007), pp. 109–119. ISSN: 0018-9340. DOI: 10.1109/tc.2007.250627. URL: https://doi.org/10.1109/tc.2007.250627.

[99] A. Rezaei Shahmirzadi and A. Moradi. "Second-Order SCA Security with almost no Fresh Randomness". In: *IACR Transactions on Cryptographic Hardware and Embedded Systems* (July 2021), pp. 708–755. ISSN: 2569-2925. DOI: 10.46586/tches.v2021.i3.708-755. URL: https://doi.org/10.46586/tches.v2021.i3.708-755.

[100] T. Beyne, S. Dhooghe, A. Ranea, and D. Šijačić. "A Low-Randomness Second-Order Masked AES". In: *Lecture Notes in Computer Science*. Springer International Publishing, 2022, pp. 87–110. ISBN: 9783030992767. DOI: 10.1007/978-3-030-99277-4_5. URL: https://doi.org/10.1007/978-3-030-99277-4_5.

[101] S. Dhooghe, A. R. Shahmirzadi, and A. Moradi. "Second-Order Low-Randomness d + 1 Hardware Sharing of the AES". In: *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications*

Security. ACM. ACM, Nov. 2022, pp. 815–828. DOI: 10.1145/3548606.3560634. URL: https://doi.org/10.1145/3548606.3560634.

[102] D. Knichel and A. Moradi. "Composable Gadgets with Reused Fresh Masks". In: *IACR Transactions on Cryptographic Hardware and Embedded Systems* (June 2022), pp. 114–140. ISSN: 2569-2925. DOI: 10.46586/tches.v2022.i3.114-140. URL: https://doi.org/10.46586/tches.v2022.i3.114-140.

[103] A. Rezaei Shahmirzadi and A. Moradi. "Re-Consolidating First-Order Masking Schemes". In: *IACR Transactions on Cryptographic Hardware and Embedded Systems* (Dec. 2020), pp. 305–342. ISSN: 2569-2925. DOI: 10.46586/tches.v2021.i1.305-342. URL: https://doi.org/10.46586/tches.v2021.i1.305-342.

[104] S. K. Park and K. W. Miller. "Random number generators: good ones are hard to find". In: *Communications of the ACM* 31.10 (Oct. 1988), pp. 1192–1201. ISSN: 0001-0782. DOI: 10.1145/63039.63042. URL: https://doi.org/10.1145/63039.63042.

[105] T. E. Tkacik. "A Hardware Random Number Generator". In: *Lecture Notes in Computer Science*. Springer Berlin Heidelberg, 2003, pp. 450–453. ISBN: 9783540004097. DOI: 10.1007/3-540-36400-5_32. URL: https://doi.org/10.1007/3-540-36400-5_32.

[106] K. Tsoi, K. Leung, and P. Leong. "Compact FPGA-based true and pseudo random number generators". In: *11th Annual IEEE Symposium on Field-Programmable Custom Computing Machines, 2003. FCCM 2003.* IEEE Comput. Soc. IEEE Comput. Soc, None, pp. 51–61. DOI: 10.1109/fpga.2003.1227241. URL: https://doi.org/10.1109/fpga.2003.1227241.

[107] S. B., M. W., and R. S. Douglas. "A Provably Secure True Random Number Generator with Built-In Tolerance to Active Attacks". In: *IEEE transactions on computers* (2007). DOI: 10.1109/TC.2007.4.

[108] G. Cassiers, L. Masure, C. Momin, T. Moos, A. Moradi, and F.-X. Standaert. "Randomness Generation for Secure Hardware Masking – Unrolled Trivium to the Rescue". In: *IACR Communications in Cryptology* (July 2024). ISSN: 3006-5496. DOI: 10.62056/akdkp2fgx. URL: https://doi.org/10.62056/akdkp2fgx.

[109] S. Nikova, V. Rijmen, and M. Schläffer. "Secure Hardware Implementation of Nonlinear Functions in the Presence of Glitches". In: *Journal of Cryptology* 24.2 (2010), pp. 292–321. ISSN: 0933-2790. DOI: 10.1007/s00145-010-9085-7. URL: https://doi.org/10.1007/s00145-010-9085-7.

[110] A. Askeland, S. Dhooghe, S. Nikova, V. Rijmen, and Z. Zhang. "Guarding the First Order: The Rise of AES Maskings". In: *Smart Card Research and Advanced Applications: 21st International Conference, CARDIS 2022, Birmingham, UK, November 7–9, 2022, Revised Selected Papers*. Birmingham, United Kingdom: Springer International Publishing, 2023, pp. 103–122. ISBN: 9783031253188. DOI: 10.1007/978-3-031-25319-5_6. URL: https://doi.org/10.1007/978-3-031-25319-5_6.

[111] R. Nagpal, B. Gigerl, R. Primas, and S. Mangard. "Riding the Waves Towards Generic Single-Cycle Masking in Hardware". In: *IACR Transactions on Cryptographic Hardware and Embedded Systems* (Aug. 2022), pp. 693–717. ISSN: 2569-2925. DOI: 10.46586/tches.v2022.i4.693-717. URL: https://doi.org/10.46586/tches.v2022.i4.693-717.

[112] M. Simões, L. Bossuet, N. Bruneau, V. Grosso, P. Haddad, and T. Sarno. "Self-timed Masking: Implementing Masked S-Boxes Without Registers". In: *Lecture Notes in Computer Science*. Springer International Publishing, 2023, pp. 146–164. ISBN: 9783031253188. DOI: 10.1007/978-3-031-25319-5_8. URL: https://doi.org/10.1007/978-3-031-25319-5_8.

[113] Z. Zhang, S. Petkova-Nikova, and V. Nikov. "Glitch-Stopping Circuits: Hardware Secure Masking without Registers". In: *Proceedings of the 2024 on ACM SIGSAC Conference on Computer and Communications Security*. ACM. ACM, Dec. 2024, pp. 3406–3420. DOI: 10.1145/3658644.3670335. URL: https://doi.org/10.1145/3658644.3670335.

[114] N. Müller, T. Moos, and A. Moradi. "Low-Latency Hardware Masking of PRINCE". In: *Lecture Notes in Computer Science*. Springer International Publishing, 2021, pp. 148–167. ISBN: 9783030899141. DOI: 10.1007/978-3-030-89915-8_7. URL: https://doi.org/10.1007/978-3-030-89915-8_7.

[115] P. Sasdrich, B. Bilgin, M. Hutter, and M. E. Marson. "Low-Latency Hardware Masking with Application to AES". In: *IACR Transactions on Cryptographic Hardware and Embedded Systems* (Mar. 2020), pp. 300–326. ISSN: 2569-2925. DOI: 10.46586/tches.v2020.i2.300-326. URL: https://doi.org/10.46586/tches.v2020.i2.300-326.

[116] V. Arribas, Z. Zhang, and S. Nikova. "LLTI: Low-Latency Threshold Implementations". In: *IEEE Transactions on Information Forensics and Security* 16 (2021), pp. 5108–5123. ISSN: 1556-6013. DOI: 10.1109/tifs.2021.3123527. URL: https://doi.org/10.1109/tifs.2021.3123527.

[117] H. Gross, R. Iusupov, and R. Bloem. "Generic Low-Latency Masking in Hardware". In: *IACR Transactions on Cryptographic Hardware and Embedded Systems* (May 2018), pp. 1–21. ISSN: 2569-2925. DOI: 10.

46586/tches.v2018.i2.1-21. URL: https://doi.org/10.46586/tches.v2018.i2.1-21.

[118] D. Knichel and A. Moradi. "Low-Latency Hardware Private Circuits". In: *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security*. ACM. ACM, Nov. 2022, pp. 1799–1812. DOI: 10.1145/3548606.3559362. URL: https://doi.org/10.1145/3548606.3559362.

[119] K. Tiri, M. Akmal, and I. Verbauwhede. "A dynamic and differential CMOS logic with signal independent power consumption to withstand differential power analysis on smart cards". In: *Proceedings of the 28th European Solid-State Circuits Conference*. 2002, pp. 403–406.

[120] S. Bhasin, S. Guilley, F. Flament, N. Selmane, and J.-L. Danger. "Countering early evaluation". In: *Proceedings of the 5th Workshop on Embedded Systems Security*. WESS '10. Scottsdale, Arizona: ACM, 2010, pp. 1–8. ISBN: 9781450300780. DOI: 10.1145/1873548.1873554. URL: https://doi.org/10.1145/1873548.1873554.

[121] A. Moradi and V. Immler. "Early Propagation and Imbalanced Routing, How to Diminish in FPGAs". In: *Cryptographic Hardware and Embedded Systems – CHES 2014*. Ed. by L. Batina and M. Robshaw. Berlin, Heidelberg: Springer Berlin Heidelberg, 2014, pp. 598–615. ISBN: 978-3-662-44709-3.

[122] A. J. Leiserson, M. E. Marson, and M. A. Wachs. "Gate-Level Masking under a Path-Based Leakage Metric". In: *Cryptographic Hardware and Embedded Systems – CHES 2014*. Ed. by L. Batina and M. Robshaw. Berlin, Heidelberg: Springer Berlin Heidelberg, 2014, pp. 580–597. ISBN: 978-3-662-44709-3.

[123] P. Sasdrich, B. Bilgin, M. Hutter, and M. E. Marson. "Low-Latency Hardware Masking with Application to AES". In: *IACR Transactions on Cryptographic Hardware and Embedded Systems* 2020.2 (Mar. 2020), pp. 300–326. DOI: 10.13154/tches.v2020.i2.300-326. URL: https://tches.iacr.org/index.php/TCHES/article/view/8553.

[124] A. Moradi and T. Schneider. "Side-Channel Analysis Protection and Low-Latency in Action". In: *Advances in Cryptology – ASIACRYPT 2016*. Ed. by J. H. Cheon and T. Takagi. Berlin, Heidelberg: Springer Berlin Heidelberg, 2016, pp. 517–547. ISBN: 978-3-662-53887-6.

[125] S. Moore, R. Anderson, R. Mullins, G. Taylor, and J. J. Fournier. "Balanced self-checking asynchronous logic for smart card applications". In: *Microprocessors and Microsystems* 27.9 (2003), pp. 421–430. ISSN: 0141-9331. DOI: https://doi.org/10.1016/S0141-9331(03)00092-9.

URL: https://www.sciencedirect.com/science/article/pii/S0141933103000929.

[126] N. Müller, D. Lammers, and A. Moradi. "A Deep Analysis of two Glitch-Free Hardware Masking Schemes SESYM and LMDPL". In: *IACR Transactions on Cryptographic Hardware and Embedded Systems* 2024.3 (July 2024), pp. 76–98. ISSN: 2569-2925. DOI: 10.46586/tches.v2024.i3.76-98. URL: https://doi.org/10.46586/tches.v2024.i3.76-98.

[127] N. Müller and A. Moradi. "Robust but Relaxed Probing Model". In: *IACR Transactions on Cryptographic Hardware and Embedded Systems* 2024.4 (Sept. 2024), pp. 451–482. ISSN: 2569-2925. DOI: 10.46586/tches.v2024.i4.451-482. URL: https://doi.org/10.46586/tches.v2024.i4.451-482.

[128] Z. Zhang, S. Petkova-Nikova, and V. Nikov. "Glitch-Stopping Circuits: Hardware Secure Masking without Registers". In: *Proceedings of the 2024 on ACM SIGSAC Conference on Computer and Communications Security*. CCS '24. Salt Lake City, UT, USA: ACM, 2024, pp. 3406–3420. ISBN: 9798400706363. DOI: 10.1145/3658644.3670335. URL: https://doi.org/10.1145/3658644.3670335.

[129] D. Knichel, P. Sasdrich, and A. Moradi. "Generic Hardware Private Circuits: Towards Automated Generation of Composable Secure Gadgets". In: *IACR Transactions on Cryptographic Hardware and Embedded Systems* 2022.1 (Nov. 2021), pp. 323–344. DOI: 10.46586/tches.v2022.i1.323-344. URL: https://tches.iacr.org/index.php/TCHES/article/view/9299.

[130] E. Prouff, C. Giraud, and S. Aumônier. "Provably Secure S-Box Implementation Based on Fourier Transform". In: *Cryptographic Hardware and Embedded Systems - CHES 2006*. Ed. by L. Goubin and M. Matsui. Berlin, Heidelberg: Springer Berlin Heidelberg, 2006, pp. 216–230. ISBN: 978-3-540-46561-4.

[131] E. Prouff and M. Rivain. "Masking against Side-Channel Attacks: A Formal Security Proof". In: *Advances in Cryptology – EUROCRYPT 2013*. Ed. by T. Johansson and P. Q. Nguyen. Berlin, Heidelberg: Springer Berlin Heidelberg, 2013, pp. 142–159. ISBN: 978-3-642-38348-9.

[132] G. Barthe, S. Belaïd, F. Dupressoir, P.-A. Fouque, B. Grégoire, and P.-Y. Strub. "Verified Proofs of Higher-Order Masking". In: *Advances in Cryptology – EUROCRYPT 2015*. Ed. by E. Oswald and M. Fischlin. Berlin, Heidelberg: Springer Berlin Heidelberg, 2015, pp. 457–485. ISBN: 978-3-662-46800-5.

[133] G. Barthe, S. Belaïd, F. Dupressoir, P.-A. Fouque, B. Grégoire, P.-Y. Strub, and R. Zucchini. *Strong Non-Interference and Type-Directed Higher-Order Masking*. Cryptology ePrint Archive, Paper 2015/506. 2015. URL: https://eprint.iacr.org/2015/506.

[134] S. Belaïd, D. Goudarzi, and M. Rivain. "Tight Private Circuits: Achieving Probing Security with the Least Refreshing". In: *Advances in Cryptology – ASIACRYPT 2018*. Ed. by T. Peyrin and S. Galbraith. Cham: Springer International Publishing, 2018, pp. 343–372. ISBN: 978-3-030-03329-3.

[135] G. Cassiers and F.-X. Standaert. "Trivially and Efficiently Composing Masked Gadgets With Probe Isolating Non-Interference". In: *IEEE Transactions on Information Forensics and Security* 15 (2020), pp. 2542–2555. DOI: 10.1109/TIFS.2020.2971153.

[136] G. Cassiers and F.-X. Standaert. "Provably Secure Hardware Masking in the Transition- and Glitch-Robust Probing Model: Better Safe than Sorry". In: *IACR Transactions on Cryptographic Hardware and Embedded Systems* 2021.2 (Feb. 2021), pp. 136–158. DOI: 10.46586/tches.v2021.i2.136-158. URL: https://tches.iacr.org/index.php/TCHES/article/view/8790.

[137] N. Müller, D. Knichel, P. Sasdrich, and A. Moradi. "Transitional Leakage in Theory and Practice: Unveiling Security Flaws in Masked Circuits". In: *IACR Transactions on Cryptographic Hardware and Embedded Systems* 2022.2 (Feb. 2022), pp. 266–288. DOI: 10.46586/tches.v2022.i2.266-288. URL: https://tches.iacr.org/index.php/TCHES/article/view/9488.

[138] S. Faust, V. Grosso, S. M. D. Pozo, C. Paglialonga, and F.-X. Standaert. *Composable Masking Schemes in the Presence of Physical Defaults and the Robust Probing Model*. Cryptology ePrint Archive, Paper 2017/711. 2017. URL: https://eprint.iacr.org/2017/711.

[139] D. Knichel, A. Moradi, N. Müller, and P. Sasdrich. "Automated Generation of Masked Hardware". In: *IACR Transactions on Cryptographic Hardware and Embedded Systems* (Nov. 2021), pp. 589–629. ISSN: 2569-2925. DOI: 10.46586/tches.v2022.i1.589-629. URL: http://dx.doi.org/10.46586/tches.v2022.i1.589-629.

[140] L. Wu, Y. Fan, B. Preneel, W. Wang, and M. Wang. "Automated Generation of Masked Nonlinear Components:" in: *Lecture Notes in Computer Science*. Springer Nature Switzerland, 2024, pp. 319–339. ISBN: 9783031614859. DOI: 10.1007/978-3-031-61486-6_19. URL: http://dx.doi.org/10.1007/978-3-031-61486-6_19.

[141] G. Cassiers, B. Gigerl, S. Mangard, C. Momin, and R. Nagpal. "Compress: Generate Small and Fast Masked Pipelined Circuits". In: *IACR Transactions on Cryptographic Hardware and Embedded Systems* 2024.3 (July 2024), pp. 500–529. DOI: 10.46586/tches.v2024.i3.500-529. URL: https://ojs.ub.rub.de/index.php/TCHES/article/view/11686.

[142] G. Barthe, S. Belaïd, G. Cassiers, P.-A. Fouque, B. Grégoire, and F.-X. Standaert. "maskVerif: Automated Verification of Higher-Order Masking in Presence of Physical Defaults". In: *Lecture Notes in Computer Science*. Springer International Publishing, 2019, pp. 300–318. ISBN: 9783030299583. DOI: 10.1007/978-3-030-29959-0_15. URL: https://doi.org/10.1007/978-3-030-29959-0_15.

[143] D. Knichel, P. Sasdrich, and A. Moradi. "SILVER – Statistical Independence and Leakage Verification". In: *Lecture Notes in Computer Science*. Springer International Publishing, 2020, pp. 787–816. ISBN: 9783030648367. DOI: 10.1007/978-3-030-64837-4_26. URL: https://doi.org/10.1007/978-3-030-64837-4_26.

[144] P. S. David Knichel and A. Moradi. *SILVER - Statistical Independence and Leakage Verification*. 2025. URL: https://github.com/Chair-for-Security-Engineering/SILVER (visited on 07/14/2025).

[145] M. Nicolai and M. A. "PROLEAD A Probing-Based Hardware Leakage Detection Tool". In: *IACR Trans. Cryptogr. Hardw. Embed. Syst.* (2022).

[146] R. Bloem, H. Gross, R. Iusupov, B. Könighofer, S. Mangard, and J. Winter. "Formal Verification of Masked Hardware Implementations in the Presence of Glitches". In: *Advances in Cryptology – EUROCRYPT 2018*. Ed. by J. B. Nielsen and V. Rijmen. Cham: Springer International Publishing, 2018, pp. 321–353. ISBN: 978-3-319-78375-8. URL: https://github.com/isec-tugraz/rebecca.

[147] S. Belaïd, D. Mercadier, M. Rivain, and A. R. Taleb. "IronMask: Versatile Verification of Masking Security". In: *2022 IEEE Symposium on Security and Privacy (SP)*. IEEE. May 2022, pp. 142–160. DOI: 10.1109/sp46214.2022.9833600. URL: http://dx.doi.org/10.1109/sp46214.2022.9833600.

[148] V. Arribas, S. Nikova, and V. Rijmen. "VerMI: Verification Tool for Masked Implementations". In: *2018 25th IEEE International Conference on Electronics, Circuits and Systems (ICECS)*. IEEE. IEEE, Dec. 2018, pp. 381–384. DOI: 10.1109/icecs.2018.8617841. URL: https://doi.org/10.1109/icecs.2018.8617841.

[149] F. Zhou, H. Chen, and L. Fan. "Prover - Toward More Efficient Formal Verification of Masking in Probing Model". In: *IACR Transactions on Cryptographic Hardware and Embedded Systems* 2025.1 (Dec. 2024), pp. 552–585. ISSN: 2569-2925. DOI: 10.46586/tches.v2025.i1.552-585. URL: https://doi.org/10.46586/tches.v2025.i1.552-585.

[150] V. Hadžić and R. Bloem. "CocoAlma: A Versatile Masking Verifier". In: *Formal Methods in Computer-Aided Design* (Oct. 2021). DOI: 10.34727/2021/ISBN.978-3-85448-046-4_9. URL: https://repositum.tuwien.at/handle/20.500.12708/18618.

[151] B. Bilgin, B. Gierlichs, S. Nikova, V. Nikov, and V. Rijmen. "Trade-Offs for Threshold Implementations Illustrated on AES". In: *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 34.7 (July 2015), pp. 1188–1200. ISSN: 0278-0070. DOI: 10.1109/tcad.2015.2419623. URL: https://doi.org/10.1109/tcad.2015.2419623.

[152] B. Bilgin, S. Nikova, V. Nikov, V. Rijmen, N. Tokareva, and V. Vitkup. "Threshold implementations of small S-boxes". In: *Cryptography and Communications* 7.1 (Sept. 2014), pp. 3–33. ISSN: 1936-2447. DOI: 10.1007/s12095-014-0104-7. URL: https://doi.org/10.1007/s12095-014-0104-7.

[153] B. Bilgin, J. Daemen, V. Nikov, S. Nikova, V. Rijmen, and G. Van Assche. "Efficient and First-Order DPA Resistant Implementations of Keccak". In: *Lecture Notes in Computer Science*. Springer International Publishing, 2014, pp. 187–199. ISBN: 9783319083018. DOI: 10.1007/978-3-319-08302-5_13. URL: https://doi.org/10.1007/978-3-319-08302-5_13.

[154] B. Bilgin, S. Nikova, V. Nikov, V. Rijmen, and G. Stütz. "Threshold Implementations of All 3 ×3 and 4 ×4 S-Boxes". In: *Lecture Notes in Computer Science*. Springer Berlin Heidelberg, 2012, pp. 76–91. ISBN: 9783642330261. DOI: 10.1007/978-3-642-33027-8_5. URL: https://doi.org/10.1007/978-3-642-33027-8_5.

[155] S. Dhooghe, S. Nikova, and V. Rijmen. "Threshold Implementations in the Robust Probing Model". In: *Proceedings of ACM Workshop on Theory of Implementation Security Workshop*. ACM. ACM, Nov. 2019, pp. 30–37. DOI: 10.1145/3338467.3358949. URL: https://doi.org/10.1145/3338467.3358949.

[156] J. Daemen. "On Non-uniformity in Threshold Sharings". In: *Proceedings of the 2016 ACM Workshop on Theory of Implementation Security*. ACM. ACM, Oct. 2016, pp. 41–41. DOI: 10.1145/2996366.2996374. URL: https://doi.org/10.1145/2996366.2996374.

[157] J. Daemen. “Changing of the Guards: A Simple and Efficient Method for Achieving Uniformity in Threshold Sharing”. In: *Lecture Notes in Computer Science*. Springer International Publishing, 2017, pp. 137–153. ISBN: 9783319667867. DOI: 10.1007/978-3-319-66787-4_7. URL: https://doi.org/10.1007/978-3-319-66787-4_7.

[158] T. Sugawara. “3-Share Threshold Implementation of AES S-box without Fresh Randomness”. In: *IACR Transactions on Cryptographic Hardware and Embedded Systems* (Nov. 2018), pp. 123–145. ISSN: 2569-2925. DOI: 10.46586/tches.v2019.i1.123-145. URL: https://doi.org/10.46586/tches.v2019.i1.123-145.

[159] A. Moradi and A. Wild. *Assessment of Hiding the Higher-Order Leakages in Hardware - what are the achievements versus overheads?* Cryptology ePrint Archive, Paper 2015/597. 2015. URL: https://eprint.iacr.org/2015/597.

[160] R. Oscar. “A note on the security of Higher-Order Threshold Implementations”. In: *IACR Cryptology ePrint Archive* (2015).

[161] J.-S. Coron, E. Prouff, M. Rivain, and T. Roche. “Higher-Order Side Channel Security and Mask Refreshing”. In: *Lecture Notes in Computer Science*. Springer Berlin Heidelberg, 2014, pp. 410–424. ISBN: 9783662439326. DOI: 10.1007/978-3-662-43933-3_21. URL: https://doi.org/10.1007/978-3-662-43933-3_21.

[162] A. Beckers, L. Wouters, B. Gierlichs, B. Preneel, and I. Verbauwhede. “Provable Secure Software Masking in the Real-World”. In: *Lecture Notes in Computer Science*. Springer International Publishing, 2022, pp. 215–235. ISBN: 9783030997656. DOI: 10.1007/978-3-030-99766-3_10. URL: https://doi.org/10.1007/978-3-030-99766-3_10.

[163] J.-S. Coron, C. Giraud, E. Prouff, S. Renner, M. Rivain, and P. K. Vadnala. “Conversion of Security Proofs from One Leakage Model to Another: A New Issue”. In: *Lecture Notes in Computer Science*. Springer Berlin Heidelberg, 2012, pp. 69–81. ISBN: 9783642299117. DOI: 10.1007/978-3-642-29912-4_6. URL: https://doi.org/10.1007/978-3-642-29912-4_6.

[164] F. Uhle, F. Stolz, and A. Moradi. “Another Evidence to not Employ Customized Masked Hardware”. In: *IACR Transactions on Cryptographic Hardware and Embedded Systems* 2024.4 (Sept. 2024), pp. 133–155. ISSN: 2569-2925. DOI: 10.46586/tches.v2024.i4.133-155. URL: https://doi.org/10.46586/tches.v2024.i4.133-155.

[165] L. De Meyer, B. Bilgin, and O. Reparaz. “Consolidating Security Notions in Hardware Masking”. In: *IACR Transactions on Cryptographic Hardware and Embedded Systems* (May 2019), pp. 119–147. ISSN: 2569-2925. DOI: 10.46586/tches.v2019.i3.119-147. URL: https://doi.org/10.46586/tches.v2019.i3.119-147.

[166] S. Faust, V. Grosso, S. Merino Del Pozo, C. Paglialonga, and F.-X. Standaert. “Composable Masking Schemes in the Presence of Physical Defaults & the Robust Probing Model”. In: *IACR Transactions on Cryptographic Hardware and Embedded Systems* (Aug. 2018), pp. 89–120. ISSN: 2569-2925. DOI: 10.46586/tches.v2018.i3.89-120. URL: https://doi.org/10.46586/tches.v2018.i3.89-120.

[167] T. De Cnudde, M. Ender, and A. Moradi. “Hardware Masking, Revisited”. In: *IACR Transactions on Cryptographic Hardware and Embedded Systems* (May 2018), pp. 123–148. ISSN: 2569-2925. DOI: 10.46586/tches.v2018.i2.123-148. URL: https://doi.org/10.46586/tches.v2018.i2.123-148.

[168] G. Cassiers and F.-X. Standaert. “Trivially and Efficiently Composing Masked Gadgets With Probe Isolating Non-Interference”. In: *IEEE Transactions on Information Forensics and Security* 15 (2020), pp. 2542–2555. ISSN: 1556-6013. DOI: 10.1109/tifs.2020.2971153. URL: https://doi.org/10.1109/tifs.2020.2971153.

[169] N. Müller and A. Moradi. “PROLEAD”. In: *IACR Transactions on Cryptographic Hardware and Embedded Systems* (Aug. 2022), pp. 311–348. ISSN: 2569-2925. DOI: 10.46586/tches.v2022.i4.311-348. URL: https://doi.org/10.46586/tches.v2022.i4.311-348.

[170] Q. L. Meunier, E. Pons, and K. Heydemann. “LeakageVerif: Efficient and Scalable Formal Verification of Leakage in Symbolic Expressions”. In: *IEEE Transactions on Software Engineering* 49.6 (June 2023), pp. 3359–3375. ISSN: 0098-5589. DOI: 10.1109/tse.2023.3252671. URL: https://doi.org/10.1109/tse.2023.3252671.

[171] V. Hadžic and R. Bloem. “Efficient and Composable Masked AES S-Box Designs Using Optimized Inverters”. In: *IACR Transactions on Cryptographic Hardware and Embedded Systems* 2025.1 (Dec. 2024), pp. 656–683. ISSN: 2569-2925. DOI: 10.46586/tches.v2025.i1.656-683. URL: https://doi.org/10.46586/tches.v2025.i1.656-683.

[172] D. Knichel, P. Sasdrich, and A. Moradi. “Generic Hardware Private Circuits”. In: *IACR Transactions on Cryptographic Hardware and Embedded Systems* (Nov. 2021), pp. 323–344. ISSN: 2569-2925. DOI: 10.46586/tches.v2022.i1.323-344. URL: https://doi.org/10.46586/tches.v2022.i1.323-344.

[173] S. Picek, B. Yang, V. Rozic, J. Vliegen, J. Winderickx, T. De Cnudde, and N. Mentens. "PRNGs for Masking Applications and Their Mapping to Evolvable Hardware". In: *Lecture Notes in Computer Science*. Springer International Publishing, 2017, pp. 209–227. ISBN: 9783319546681. DOI: 10.1007/978-3-319-54669-8_13. URL: https://doi.org/10.1007/978-3-319-54669-8_13.

[174] V. Grosso, F.-X. Standaert, and E. Prouff. "Low Entropy Masking Schemes, Revisited". In: *Lecture Notes in Computer Science*. Springer International Publishing, 2014, pp. 33–43. ISBN: 9783319083018. DOI: 10.1007/978-3-319-08302-5_3. URL: https://doi.org/10.1007/978-3-319-08302-5_3.

[175] C. Beierle, J. Jean, S. Kölbl, G. Leander, A. Moradi, T. Peyrin, Y. Sasaki, P. Sasdrich, and S. M. Sim. "The SKINNY Family of Block Ciphers and its Low-Latency Variant MANTIS". In: *IACR Cryptol. ePrint Arch.* 2016 (2016), p. 660. URL: https://api.semanticscholar.org/CorpusID:1279633.

[176] G. Goodwill, B. Jun, J. Jaffe, and P. Rohatgi. *A Testing Methodology for Side-Channel Resistance Validation*. NIST Non-Invasive Attack Testing Workshop. Available at https://csrc.nist.gov/CSRC/media/Events/Non-Invasive-Attack-Testing-Workshop/documents/08_Goodwill.pdf. 2011.

[177] N. Weste and D. Harris. *CMOS VLSI Design: A Circuits and Systems Perspective*. 4th. USA: Addison-Wesley Publishing Company, 2010. ISBN: 0321547748.

[178] G. Goodwill, B. Jun, J. Jaffe, and P. Rohatgi. *A testing methodology for side channel resistance validation*. http://csrc.nist.gov/newsevents/non-invasive-attack-testing-workshop/08Goodwill.pdf. 2011.

[179] J. Cooper, E. De Mulder, G. Goodwill, J. Jaffe, G. Kenworthy, and P. Rohatgi. *Test vector leakage assessment (TVLA) methodology in practice*. http://icmc-2013.org/wp/wp-content/uploads/2013/09/goodwillkenworthtestvector.pdf. 2013.

[180] T. Schneider and A. Moradi. "Leakage Assessment Methodology". In: *Cryptographic Hardware and Embedded Systems – CHES 2015*. Berlin, Heidelberg: Springer-Verlag, 2022, pp. 495–513. ISBN: 978-3-662-48323-7. DOI: 10.1007/978-3-662-48324-4_25. URL: https://doi.org/10.1007/978-3-662-48324-4_25.

[181] B. Gierlichs, L. Batina, P. Tuyls, and B. Preneel. "Mutual Information Analysis". In: *Cryptographic Hardware and Embedded Systems – CHES 2008*. Ed. by E. Oswald and P. Rohatgi. Berlin, Heidelberg: Springer Berlin Heidelberg, 2008, pp. 426–442. ISBN: 978-3-540-85053-3.

[182] F.-X. Standaert, T. G. Malkin, and M. Yung. "A Unified Framework for the Analysis of Side-Channel Key Recovery Attacks". In: *Advances in Cryptology - EUROCRYPT 2009*. Ed. by A. Joux. Berlin, Heidelberg: Springer Berlin Heidelberg, 2009, pp. 443–461. ISBN: 978-3-642-01001-9.

[183] Y. Lindell. "How to Simulate It – A Tutorial on the Simulation Proof Technique". In: *Tutorials on the Foundations of Cryptography: Dedicated to Oded Goldreich*. Ed. by Y. Lindell. Cham: Springer International Publishing, 2017, pp. 277–346. ISBN: 978-3-319-57048-8. DOI: 10.1007/978-3-319-57048-8_6. URL: https://doi.org/10.1007/978-3-319-57048-8_6.

[184] S. Belaïd, F. Benhamouda, A. Passelègue, E. Prouff, A. Thillard, and D. Vergnaud. "Randomness Complexity of Private Circuits for Multiplication". In: *Advances in Cryptology – EUROCRYPT 2016*. Ed. by M. Fischlin and J.-S. Coron. Berlin, Heidelberg: Springer Berlin Heidelberg, 2016, pp. 616–648. ISBN: 978-3-662-49896-5.

[185] J.-S. Coron, E. Prouff, M. Rivain, and T. Roche. "Higher-Order Side Channel Security and Mask Refreshing". In: *Fast Software Encryption*. Ed. by S. Moriai. Berlin, Heidelberg: Springer Berlin Heidelberg, 2014, pp. 410–424. ISBN: 978-3-662-43933-3.

[186] A. Duc, S. Dziembowski, and S. Faust. "Unifying Leakage Models: From Probing Attacks to Noisy Leakage." In: *Advances in Cryptology – EUROCRYPT 2014*. Ed. by P. Q. Nguyen and E. Oswald. Berlin, Heidelberg: Springer Berlin Heidelberg, 2014, pp. 423–440. ISBN: 978-3-642-55220-5.

[187] S. Dziembowski, S. Faust, and M. Skorski. "Noisy Leakage Revisited". In: *Advances in Cryptology - EUROCRYPT 2015*. Ed. by E. Oswald and M. Fischlin. Berlin, Heidelberg: Springer Berlin Heidelberg, 2015, pp. 159–188. ISBN: 978-3-662-46803-6.

[188] Z. Chen, S. Haider, and P. Schaumont. "Side-Channel Leakage in Masked Circuits Caused by Higher-Order Circuit Effects". In: *Advances in Information Security and Assurance*. Ed. by J. H. Park, H.-H. Chen, M. Atiquzzaman, C. Lee, T.-h. Kim, and S.-S. Yeo. Berlin, Heidelberg: Springer Berlin Heidelberg, 2009, pp. 327–336. ISBN: 978-3-642-02617-1.

[189] G. O. Dyrkolbotn, K. Wold, and E. Snekkenes. “Security Implications of Crosstalk in Switching CMOS Gates”. In: *Information Security*. Ed. by M. Burmester, G. Tsudik, S. Magliveras, and I. Ilić. Berlin, Heidelberg: Springer Berlin Heidelberg, 2011, pp. 269–275. ISBN: 978-3-642-18178-8.

[190] A. Moradi. “Side-Channel Leakage through Static Power”. In: *Cryptographic Hardware and Embedded Systems – CHES 2014*. Ed. by L. Batina and M. Robshaw. Berlin, Heidelberg: Springer Berlin Heidelberg, 2014, pp. 562–579. ISBN: 978-3-662-44709-3.

[191] L. Zussa, I. Exurville, J.-M. Dutertre, J.-B. Rigaud, B. Robisson, A. Tria, and J. Clédière. “Evidence of an information leakage between logically independent blocks”. In: *Proceedings of the Second Workshop on Cryptography and Security in Computing Systems*. CS2 ’15. Amsterdam, Netherlands: Association for Computing Machinery, 2015, pp. 25–30. ISBN: 9781450331876. DOI: 10.1145/2694805.2694810. URL: https://doi.org/10.1145/2694805.2694810.

[192] T. De Cnudde, B. Bilgin, B. Gierlichs, V. Nikov, S. Nikova, and V. Rijmen. “Does Coupling Affect the Security of Masked Implementations?” In: *Constructive Side-Channel Analysis and Secure Design*. Ed. by S. Guilley. Cham: Springer International Publishing, 2017, pp. 1–18. ISBN: 978-3-319-64647-3.

Part II

Publications

List of publications

Journal publications

1. Dilip Kumar S. V. , Siemen Dhooghe, Josep Balasch, Benedikt Gierlichs and Ingrid Verbauwhede, "Time Sharing – A Novel Approach to Low-Latency Masking". *IACR Transactions on Cryptographic Hardware and Embedded Systems*, 2024(3), pp. 249–272. doi:10.46586/tches.v2024.i3.249-272.

2. Dilip Kumar S. V. , Siemen Dhooghe, Josep Balasch, Benedikt Gierlichs and Ingrid Verbauwhede, "Higher-Order Time Sharing Masking". *IACR Transactions on Cryptographic Hardware and Embedded Systems*, 2025(2), pp. 235–267. doi:10.46586/tches.v2025.i2.235-267.

3. S. V. Dilip Kumar, Josep Balasch, Benedikt Gierlichs and Ingrid Verbauwhede, "Low-Cost First-Order Secure Boolean Masking in Glitchy Hardware - full version*". *IEEE Transactions on Information Forensics and Security*, vol. 20, pp. 2437–2449, 2025. doi:10.1109/TIFS.2025.3541442.

International Conferences with Proceedings

4. S. V. Dilip Kumar, Josep Balasch, Benedikt Gierlichs and Ingrid Verbauwhede, "Low-Cost First-Order Secure Boolean Masking in Glitchy Hardware". *2023 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, Antwerp, Belgium, 2023, pp. 1–2. doi:10.23919/DATE56975.2023.10136920.

Chapter 6

Low-Cost First-Order Secure Boolean Masking in Glitchy Hardware

Publication data

Dilip Kumar S.V., Josep Balasch, Benedikt Gierlichs and Ingrid Verbauwhede, "Low-Cost First-Order Secure Boolean Masking in Glitchy Hardware," *2023 Design, Automation & Test in Europe Conference & Exhibition (DATE), Antwerp, Belgium, 2023*, pp. 1-2.

Contributions

Principal author.

Low-Cost First-Order Secure Boolean Masking in Glitchy Hardware

Dilip Kumar S. V.[1], Josep Balasch[2], Benedikt Gierlichs[1], Ingrid Verbauwhede[1]

[1]COSIC, KU Leuven, Belgium
[2]e-Media Research Lab, STADIUS, KU Leuven, Belgium

Abstract. We describe how to securely implement the logical AND of two bits in hardware in the presence of glitches without the need for fresh randomness. As a case study, we design, implement and evaluate a DES core using our AND gate. Our goal is an overall practically relevant tradeoff between area, latency, randomness cost and security. We focus on first-order secure Boolean masking and we do not aim for provable security. The resulting DES engine shows no evidence of first-order leakage in a non-specific leakage assessment with 50M traces.

1 Introduction

Over the last few decades, a lot of attention has been dedicated to researching and developing fast and efficient cryptographic implementations that are secure against power analysis attacks. Masking is a well-known technique that can be used to protect both hardware and software implementations. Its core idea is to split the data being processed by an implementation into random shares, effectively eliminating its correlation with the device's power consumption. Modern masking techniques, such as Threshold Implementation (TI) [3] and Domain-oriented Masking (DOM) [2], have been designed to address the problem caused by glitches. In contrast to classical Boolean masking, they eliminate the propagation of glitches through register layers and maintain the uniformity of the intermediate values by injecting fresh randomness. As a result, they achieve *provable* security against first-order attacks. But provable security comes with higher costs. Protected implementations using modern masking schemes require more resources in terms of area, latency and randomness than classical Boolean masking. In hardware, masking is commonly applied at the gate level. As logic gates are used as a fundamental building block in gate-level masking, any cost reduction in building a masked logic gate benefits the overall cost of a masked circuit significantly. To this end, in this work, we develop a low-cost Boolean masked AND gate suitable for hardware implementations which requires no

fresh randomness. Additionally, we discuss guidelines for building low-cost circuits using our proposed low-cost AND gadget. And as a case study, we design and implement a masked DES encryption engine that provides *practical* security in the presence of glitches. And finally, we evaluate the performance of our design both in terms of cost (area, latency, randomness) and first-order side-channel leakage on an FPGA platform.

2 Low-Cost Masked AND2 Gadget

We start our work from the masked AND2 gadget proposed by Biryukov et al. [1] for software implementations. To compute $z = x \cdot y$, where $x = x_0 \oplus x_1$, $y = y_0 \oplus y_1$ and $z = z_0 \oplus z_1$:

$$\begin{aligned} z_0 &= (x_0 \cdot y_0) \oplus (x_0 + \overline{y_1}) \\ z_1 &= (x_1 \cdot y_0) \oplus (x_1 + \overline{y_1}) \end{aligned} \tag{1}$$

From now on, we will refer to this gadget as `secAND2`. A remarkable property of this gadget is that it does not require fresh randomness to be secure. Instead, the uniformity of the output is achieved by reusing the randomness of the inputs. This characteristic becomes critical when implementing circuits that combine several terms, for instance, through addition. It can lead to a decrease in security if the added terms are not independent. For this reason, the outputs of some of `secAND2` gadgets are selectively refreshed. Adding security measures only when necessary in turn decreases the cost of fresh randomness required to build circuits.

A further well-known problem arises when implementing such a gadget in hardware as glitches can happen. Glitches on the output of a logic gate are created by different arrival times of its input signals. It is impossible to predict the order in which the inputs arrive in a large circuit. To implement `secAND2` securely, we take control over the order of and the delay between the input signals, and send the inputs in a safe sequence such that there are no glitches and thus no leakage of any information about the sensitive inputs or intermediate values. For `secAND2`, as long as either y_0 or y_1 arrive last, we observe no leakage. These results can be explained from the `secAND2` equations in (1). It is easy to see that z_0 depends on x_0, y_0 and y_1 whereas z_1 depends on x_1, y_0 and y_1. Therefore, by delaying the arrival of y_0 or y_1 we can achieve a *temporary* non-completeness property (refer to the non-completeness property of Threshold Implementations) for both output bits during the evaluation. For our purpose of creating a secure low-cost masked AND gate, we use a flip flop (FF) to delay one of the shares y_0/y_1, see Figure 1. Products of more than just two variables, can

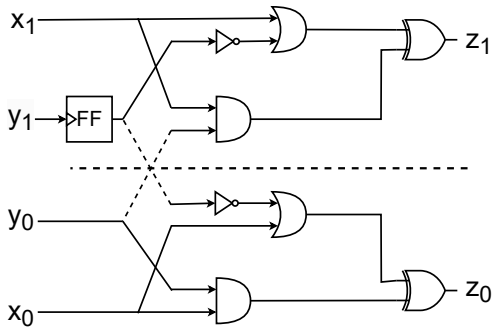


Figure 1: `secAND2` gate with internal FF.

be computed by cascading multiple `secAND2` gadgets. With careful placement of inputs and interconnections between the cascaded `secAND2` gadgets, we can obtain a construction with no additional (i.e. external) FFs. Thereby ensuring a small area footprint.

3 Case Study : DES

We chose the Data Encryption Standard (DES) as our target algorithm because it is the main building block of TDES or 3DES, which is still widely used today. The main difficulty in protecting the DES implementation lies in the S-boxes, which are the only non-linear components. A hardware-friendly way to implement them is to represent them as four 4-bit permutations (the so-called *mini S-boxes*) and a multiplexer (MUX). Each mini S-box/MUX can be represented through equations in Algebraic Normal Form (ANF) that consist only of AND and XOR gates. These equations can be split into two stages: AND stage and XOR stage. In the AND stage, the inputs of the mini S-box (or MUX) are multiplied using our `secAND2` gadget. Then, in the XOR stage, the outputs of the AND stage are combined to produce the outputs of the mini S-Box (or MUX).

Our final DES implementation is constructed by adapting a round-based architecture to incorporate first-order Boolean masking. The performance results are summarized in Table 1. We also provide the numbers from the work in [4], reporting a 3DES implementation protected with DOM, in the table.

Note that the number of cycles reported in [4] is scaled down to compare with our DES implementation.

Table 1: Utilization results of DES implementations.

Version		**ASIC** [GEs]	**FPGA** [FFs]	**FPGA** [LUT6s]	**Randomness** [bits/round]	**Cycles**
this work		14488*	819	2129	14	7*16+3
without recycling		-	-	-	14*8 = 112	7*16+3
[4]	DOM-indep	13800†	-	-	22*8 = 176	5*16+4
	DOM-dep	22400†	-	-	22*3*8=528	5*16+4

† Calculated using a 28 nm RVT standard cell library from Global Foundries.
* Calculated using NanGate 45nm Open Cell Library.

From the area results, we can see that our design is more compact than the DOM-dep version in [4] but slightly larger than the DOM-indep version. Our design uses 30 `secAND2` gates per DES S-Box, compared to 22 DOM-indep gates in [4]. While `secAND2` gates are by design smaller than the DOM-indep gates, the additional modules in our implementation (e.g. refreshing gadgets, input/output S-box registers, etc.) compensate the gain to some extent. An important remark here is that our GE results include a masked key schedule whereas the results in [4] include the cost of an unmasked key schedule. The overhead caused by this module further explains the difference between both designs. In terms of randomness usage, our design requires less bits per round as we reuse randomness across the 8 different S-boxes. This decision does not have an impact on the first-order security (see below) and hence we opt to use it in our reference implementation. If our design did not recycle randomness, or if the designs in [4] did, then our random bits requirements would still be lower. We also note that there is no security evaluation of the DOM-indep version in [4]. They evaluate only the DOM-dep version which consumes 3 random bits per refreshing. Finally, our implementation takes two more cycles per round compared to [4].

We lastly evaluate the side-channel security of our protected DES implementation using a SAKURA-G board equipped with a Spartan-6 FPGA. We use the non-specific Test Vector Leakage Assessment (TVLA) methodology, and we perform three fixed vs. random tests (50 million traces each) with three different fixed plaintexts. Figure 2 shows from top to bottom: a power trace, first, second and third-order univariate t-test results, for one of the fixed plaintexts. As we can see there is no evidence of first-order leakage. There are very few and minor crossings of the 4.5 threshold in the first-order t-test values, but they are not consistent across the three different plaintexts, i.e. the threshold is not exceeded at the same time indexes, as is required by the TVLA methodology in order to deem a device leaking.

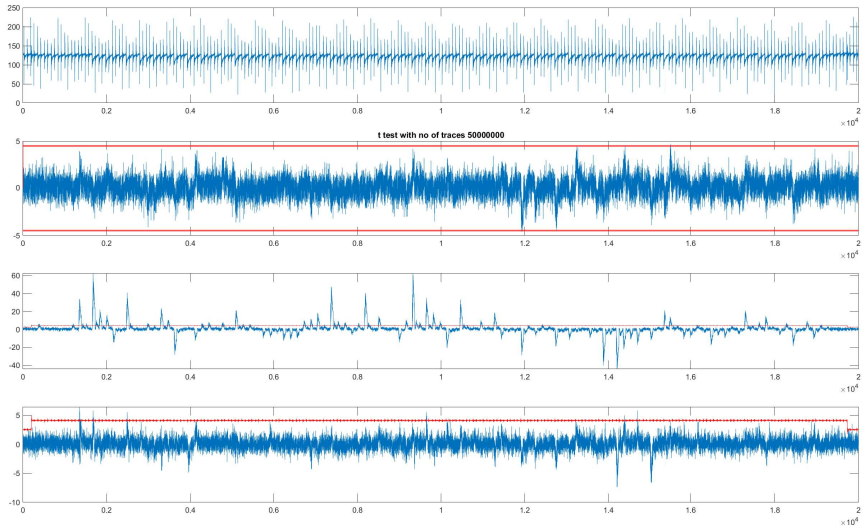


Figure 2: TVLA FvR result using 50 Million Traces.

Acknowledgments

This research has been funded in part by KU Leuven Startfinanciering STG/20/047, by CyberSecurity Research Flanders VR20192203, by VLAIO ICON CS IDIoTS HBC.2020.2789, by the Research Council KU Leuven C16/15/058, by the European Commission through Belfort ERC Advanced Grant 101020005 695305, through HORIZON-CL3-2021-CS-01-02 ORSHIN 101070008, and through H2020 Twinning SAFEST 952252.

References

[1] Biryukov, A., Dinu, D., Corre, Y. L., and Udovenko, A. Optimal first-order boolean masking for embedded iot devices. In *Smart Card Research and Advanced Applications - CARDIS 2017* (2017), T. Eisenbarth and Y. Teglia, Eds., vol. 10728 of *LNCS*, Springer, pp. 22–41.

[2] Gross, H., Mangard, S., and Korak, T. Domain-oriented masking: Compact masked hardware implementations with arbitrary protection order. In *Theory of Implementation Security - TIS@CCS 2016* (2016), B. Bilgin, S. Nikova, and V. Rijmen, Eds., ACM, p. 3.

[3] Nikova, S., Rechberger, C., and Rijmen, V. Threshold implementations against side-channel attacks and glitches. In *Information and Communications Security - ICICS 2006* (2006), P. Ning, S. Qing, and N. Li, Eds., vol. 4307 of *LNCS*, Springer, pp. 529–545.

[4] Sasdrich, P., and Hutter, M. Protecting triple-des against DPA - A practical application of domain-oriented masking. In *Constructive Side-Channel Analysis and Secure Design - COSADE 2018* (2018), J. Fan and B. Gierlichs, Eds., vol. 10815 of *LNCS*, Springer, pp. 207–226.

Chapter 7

Low-Cost First-Order Secure Boolean Masking in Glitchy Hardware - Full Version*

Publication Data

Dilip Kumar S.V., Josep Balasch, Benedikt Gierlichs, and Ingrid Verbauwhede, "Low-Cost First-Order Secure Boolean Masking in Glitchy Hardware – full version," *IEEE Transactions on Information Forensics and Security*, vol. 20, pp. 2437–2449, 2025. doi: 10.1109/TIFS.2025.3541442.

Contributions

Principal author.

Clarifications

Glitch security of `secAND2` (single use). Each output is expressed as the XOR of two branches:

$$z_0 = (x_0 \cdot y_0) \ \oplus \ (x_0 + \overline{y_1}), \qquad z_1 = (x_1 \cdot y_0) \ \oplus \ (x_1 + \overline{y_1}).$$

Structurally, x_0 (resp. x_1) acts as a select signal that chooses between two data paths driven by y_0 and y_1. When $x_0 = 1$, the OR branch $x_0 + \overline{y_1}$ is forced to 1 (independent of y_1), leaving only the AND branch $x_0 \cdot y_0$ data-dependent; when $x_0 = 0$, the AND branch is forced to 0 and only the OR branch depends on y_1. The same holds symmetrically for z_1 with x_1.

Glitch security therefore reduces to an *arrival-order* constraint: x_0 and x_1 must stabilize before the last of $\{y_0, y_1\}$, and one y-share must arrive last. With x_i fixed early, exactly one branch per output is active while the other is constant, so transient races cannot cause both y_0 and y_1 to influence the same output simultaneously. The final y arrival produces only a clean, single update on the selected branch, preventing any mixing of the two y shares in the presence of glitches.

Transition security of `secAND2` (consecutive uses). When reusing the same instance, the next input must not interact with residual state from the previous computation. We enforce this with an *overwrite-before-select* policy together with the same last-arrival rule. In the `secAND2-PD` realization, input timing is scheduled in three steps within a cycle: first, one share of the new y updates, overwriting any dependence on the prior y; second, the x shares settle (the select is now applied to fresh y); third, the remaining y share arrives last, preserving single-branch activity at the moment of the final update. In the `secAND2-FF` realization, the internal state is explicitly cleared before applying new inputs; the reset is asserted in the reverse order so that one y share is reset first and then the remaining three shares, ensuring no new x transition can toggle against an old y. After reset, the new inputs are applied with the same last-arrival policy as in single use.

Low-Cost First-Order Secure Boolean Masking in Glitchy Hardware - Full Version*

Dilip Kumar S. V.[1], Josep Balasch[2], Benedikt Gierlichs[1], Ingrid Verbauwhede[1]

[1]COSIC, KU Leuven, Belgium
[2]e-Media Research Lab, STADIUS, KU Leuven, Belgium

Abstract. We describe how to securely implement the masked logical AND of two bits in hardware in the presence of glitches without the need for fresh randomness, and we provide guidelines for the composition of circuits. As a case study, we design, implement, and evaluate masked DES cores. We focus on first-order secure Boolean masking and do not aim for provable security. Our goal is a practically relevant trade-off between area, latency, randomness cost, and security. We provide two low-cost solutions. Our first solution focuses on strong security while simultaneously aiming for low implementation costs. The resulting DES engine shows no evidence of first-order leakage in a non-specific leakage assessment with 50M traces. Our second solution follows the opposite approach: we focus on lowering implementation costs, latency to be specific, while not sacrificing much on security. Our low-latency DES engine exhibits signs of first-order leakage only after approximately 15M traces.

1 Introduction

Over the last few decades, much attention has been dedicated to researching and developing fast and efficient cryptographic implementations that are secure against power analysis attacks [15]. Masking [5, 11] is a well-known technique that can be used to protect both hardware and software implementations. Its core idea is to split the data being processed by an implementation into random shares, effectively eliminating its correlation with the device's power consumption.

In this work, we focus on first-order Boolean masking, where each sensitive (intermediate) value x is randomly split into two shares x_0 and x_1 such that $x = x_0 \oplus x_1$. First-order masked implementations can, in theory, be broken with higher-order attacks, which combine leakage of multiple (or all) shares to derive sensitive values. We nevertheless focus on first-order masking because performing a successful higher-order attack can be made very difficult by adding

noise (the number of traces needed increases exponentially in the attack order, with the noise factor in the basis) [5].

In hardware, masking is commonly applied at the gate level. As logic gates are used as a fundamental building block in gate-level masking, any cost reduction in building a masked logic gate significantly benefits the overall cost of a masked circuit. A significant hurdle in hardware masking is to overcome the effect of glitches, i.e. undesired signal transitions in the circuit, as they are known to temporarily reveal unmasked sensitive values [16].

A methodology for implementing a masked circuit in hardware requires at least a masked AND gadget, a masked XOR gadget and rules for the composition of gadgets to build a masked circuit. A masked XOR is easy because one can simply apply the XOR to each share separately. A masked AND is more difficult as the computation needs to involve all shares of all variables at some point. One needs to be very careful not to reveal any unmasked sensitive intermediate values, as demonstrated in many previous works. Composition is also difficult because circuit effects, uniformity of inputs and outputs, and their dependencies need to be tracked and corrected as necessary.

Modern hardware masking techniques, such as Threshold Implementation (TI) [19] and Domain-oriented Masking (DOM) [12], have been designed to address the problem caused by glitches. In contrast to classical Boolean masking, they control the propagation of glitches through register layers and maintain the uniformity of the intermediate values by injecting fresh randomness. As a result, they achieve *provable* security against first-order attacks. Threshold Implementation is shown to be provable secure [8] under the glitch-robust probing model [9]. Although DOM does not enjoy a security proof, it has shown many times to be secure in practice. Some of them further generalize to higher-orders. However provable security comes with higher costs. Protected implementations using modern masking schemes require a lot more resources in terms of area, latency, and randomness than classical Boolean masking [21].

In this work, we develop a low-cost Boolean masked AND gate to build masked circuits that provide *practical* security in the presence of glitches. Our contributions are as follows:

- Starting from the software-oriented masked AND construction by Biryukov et al. [4], we derive a low-cost AND gadget suitable for hardware implementations which requires no fresh randomness. We propose two solutions to prevent glitches by controlling the arrival time of input operands.
- We provide guidelines for composition and exemplary circuits for securely

computing the logical AND of more than two terms and circuits with AND and XOR gadgets. We pay particular attention to the need to remask and explain when and how to do it.

- We design and implement two masked DES encryption engines building on the proposed low-cost AND gadget and guidelines for composition. We add security measures only where needed for practical security.
- We evaluate the performance of our designs both in terms of cost (area, latency, randomness) and first-order side-channel leakage on an FPGA platform.

2 Low-Cost Masked AND2 Gadget

If a regular AND2 computes $z = x \cdot y$, a straightforward masked AND2 could for instance compute $z_0 = x_0 \cdot y_0 \oplus x_0 \cdot y_1$ and $z_1 = x_1 \cdot y_0 \oplus x_1 \cdot y_1$ such that $z = z_0 \oplus z_1$. This would, however, not be secure because $z_0 = x_0 \cdot (y_0 \oplus y1) = x_0 \cdot y$ depends on unmasked y, and similar for z_1. A simple solution to this problem, as first proposed by Trichina [21] consists in the introduction of a fresh random bit r in the equations:

$$\begin{aligned} z_0 &= r \oplus (x_0 \cdot y_0) \oplus (x_0 \cdot y_1) \oplus (x_1 \cdot y_1) \oplus (x_1 \cdot y_0) \\ z_1 &= r \end{aligned} \tag{1}$$

This construction is secure only if the order of evaluation is from left to right. A well-known problem arises when implementing such a gadget in hardware, because the order of evaluation is unknown and glitches in the combinational circuit can happen. Previous work, such as TI and DOM, provide solutions for this problem. They require fresh random bits, too. The cost in terms of the number of random bits is an important criterion when comparing masked implementations. Gross et al. [13] propose an AND2 gadget and rules for composition which allow implementing, e.g. an entire masked AES-128 using only two bits of randomness. The security of their approach was proven in the t-probing model [14]. However, in hardware, this approach leads to a significant penalty in latency. The software implementation provided by Gross et al. was found to be insecure [1]. Like Gross et al. [13] we start our work from the masked AND2 gadget proposed by Biryukov et al. [4] for software implementations:

$$\begin{aligned} z_0 &= (x_0 \cdot y_0) \oplus (x_0 + \overline{y_1}) \\ z_1 &= (x_1 \cdot y_0) \oplus (x_1 + \overline{y_1}) \end{aligned} \tag{2}$$

$\cdot, \oplus, +$ denote AND, XOR and OR, respectively. We refer to this gadget as `secAND2` from now on. A remarkable property of this gadget is that it does not require fresh randomness to be secure. Yet, due to the lack of a fresh mask, the output is not independent of the input, which needs to be considered during composition. Another advantage of the `secAND2` gadget over the one by Trichina is that it requires fewer elementary logic operations (AND, XOR, etc.) and will thus lead to a faster implementation in software or a smaller implementation in hardware. While Gross et al. aimed for provable security in the presence of glitches with minimal randomness requirements, we strive for an overall practically relevant tradeoff between area, latency, randomness cost and security.

2.1 Secure Hardware Implementation of `secAND2`

A straightforward ASIC implementation of `secAND2` using a common standard cell library, i.e. using AND2, XOR2, OR2 and INV gates, will be insecure due to glitches in the circuit. A similar problem occurs when implementing the logic equations on FPGAs using Look Up Tables (LUT). We have verified this by performing leakage assessment tests on a Spartan6 FPGA. Our results clearly show that programming the equations for the outputs of `secAND2` (z_0 and z_1) directly into LUTs leaks, which we attribute to glitches.

Glitches on the output of a logic gate are created by different arrival times of its input signals. Predicting the order in which the inputs arrive in a large circuit is impossible. But if we have control over the order of and the delay between the arrival of input signals, it might be possible to send the inputs in a safe sequence such that there are no glitches and thus no leakage of any information about the sensitive inputs or intermediate values. In the next subsection, we investigate the existence of such safe sequences for `secAND2`.

2.2 Identifying Safe Input Sequences

We experiment with the issue of glitches on the same Spartan6 FPGA by forcibly sending the inputs of `secAND2` (x_0, x_1, y_0, y_1) at different time instances. Sending one input after another can be executed in a controlled manner with the help of registers by connecting them directly to the inputs of `secAND2`. First, we reset all four registers to 0. Then, we update one register at a time over four consecutive clock cycles with the desired input sequence. We exhaust all $4! = 24$ possible input sequences and observe any potential leakage using TVLA methodology [10] for each sequence across the four clock cycles by collecting half a million traces. To improve the signal-to-noise ratio (SNR), we replicated

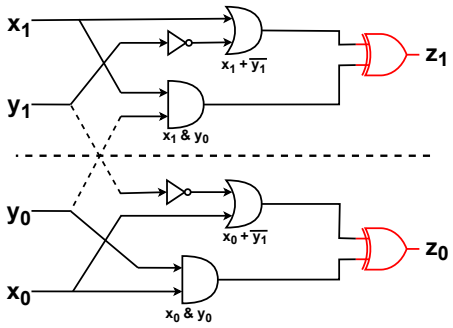


Figure 1: `secAND2` gate schematic.

multiple parallel instances of `secAND2` on the FPGA, each receiving identical inputs. Inputs x and y are independently shared with uniformly distributed random bits.

Clock Cycle	1	2	3	4	
Input	*	*	*	x_1 or x_0	$\rightarrow$ Sequence leaks
Input	*	*	*	y_1 or y_0	$\rightarrow$ Sequence does not leak

Table 1: Leakage behaviour of `secAND2` for different input sequences. '*' denotes any of the remaining input shares.

The results of this experiment are summarized in Table 1. In short, we observe leakage in sequences where either x_0 or x_1 arrive in the last clock cycle, but not in sequences where either y_0 or y_1 arrive last. These results can be explained from the `secAND2` equations in (2). Our `secAND2` gadget is not non-complete with respect to y (refer to the non-completeness property of TI), as the equations involve both shares y_0 and y_1. The equation for z_0 depends on x_0, y_0 and y_1 whereas z_1 depends on x_1, y_0 and y_1. Assume x_0 arrives in the last cycle, as all input registers are initially reset, if the input x_0 is 1, then the XOR gate outputting z_0(highlighted in red in Figure 1) toggles from $\overline{y_1}$ to $y_0 \oplus 1$, a hamming distance of $y_0 \oplus y_1$. Therefore, if a glitch occurs, the late arrival of x_0 can reveal information about the unshared input $y(= y_0 \oplus y_1)$ and similar for x_1. By forcing y_0 or y_1 to arrive last, we essentially make x_0 and x_1 arrive early. We observe no leakage in sequences where y_0 or y_1 arrive last because x_0 and x_1 do not evaluate on the combined value of y_0 and y_1, which would leak the unshared input $y(= y_0 \oplus y_1)$. Looking at the `secAND2` circuit in Figure 1, in the first three clock cycles, no signal or gate has enough information to be able to leak anything about either sensitive unshared inputs, x and y. And as a result, we can achieve a *temporary* non-completeness property for both

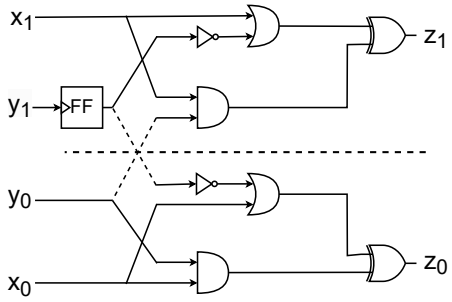


Figure 2: `secAND2` gate with internal FF or `secAND2-FF`.

output bits during the evaluation in the first three clock cycles. In the fourth clock cycle, only a single input bit arrives straight from a register. Any signal in the `secAND2` circuit, see Figure 1, will toggle at most once. In other words, glitches cannot occur in the last clock cycle. Thus, no sensitive information can be leaked even though `secAND2` is no longer non-complete. The Hamming Distance of the outputs, z_0 and z_1, before and after the fourth clock cycle does not depend on either sensitive inputs, x or y. Therefore, the final cycle does not leak either sensitive input, and any sequence that ends with y_0 or y_1 can securely compute a product of two shared variables.

Although we have identified safe sequences, we have to address a few issues related to our initial assumption and proposed solution. We began with the assumption that the four registers connected to `secAND2` to provide inputs are reset to 0. But this is hardly the case in practice. Our `secAND2` gadgets are typically expected to be reused for computation as part of a cryptographic circuit. It might not always be feasible to reset the input registers between computations, for example, if the circuit is pipelined. Additionally, this approach would significantly increase latency as each `secAND2` evaluation would take four clock cycles to compute instead of one. Lastly, a masked cryptographic circuit typically contains several AND gates connected to one another; sending the inputs in four clock cycles for every multiplication would require extra registers to temporarily buffer the intermediate values, which also increases the area cost. In what follows, we propose two solutions to tackle these problems.

2.3 Solution 1: `secAND2` with a flip-flop (`secAND2-FF`)

To create a secure low-cost masked AND gate, any circuit in which one of the two shares of y, i.e. y_0 or y_1, arrives last will guarantee no leakage. This can be achieved by delaying the processing of either of the two inputs, for example, y_1. Our `secAND2` could hence be constructed as illustrated in Figure 2, using an internal delay flip flop (FF). The flip-flop delays the input y_1 and ensures secure computation. This optimization reduces the number of clock cycles to calculate a multiplication from four to two. We shall refer to this faster two-cycle `secAND2` as `secAND2-FF` from now on. We also verified its security with leakage assessment experiments.

In subsection 2.2, we explained that the order in which inputs arrive could determine whether the computation is secure or not and that late arrival of x_0 (or x_1) has the potential to reveal information about the unshared input $y(= y_0 \oplus y_1)$. Suppose we compute two multiplications consecutively on the same `secAND2-FF` gadget: let the inputs for the first multiplication be (m_0, m_1, n_0, n_1) and the inputs for the second multiplication (a_0, a_1, b_0, b_1). If we did not reset the inputs between the multiplications and a_0 arrives before b_0 and b_1, the existing inputs of the `secAND2-FF`, n_0 and n_1 would remain unchanged when a_0 arrives. If $m_0 = 0$ and $a_0 = 1$, then the output z_0(in Figure 2) toggles from $\overline{n_1}$ to $n_0 \oplus 1$, or if $m_0 = 1$ and $a_0 = 0$, then the output z_0 toggles from $n_0 \oplus 1$ to $\overline{n_1}$. Hence, a_0 would leak information about the previous computation, $n(= n_0 \oplus n_1)$. Our first solution, `secAND2-FF`, reduces latency, but it must be reset between successive computations.

2.4 Solution 2: `secAND2` with path delay (`secAND2-PD`)

We propose our second solution to address the issues of `secAND2`. We eliminate the need for resetting between consecutive multiplications while also reducing the latency. Using a flip-flop as a delay element in our previous solution guarantees that one of the inputs arrives late. Instead of using a flip-flop, we now propose using path delay to achieve the same result, for instance, by making one of the input signals travel through a longer path so it arrives late. This solution follows a more practical approach and comes with certain constraints, such as placement and routing, which might not be as straightforward to implement. We shall later explain in more detail in Section 5 that this solution can indeed be achieved in practice. Using path delay as a delay element, instead of a flip-flop, eliminates the critical need to reduce the number of cycles required to send the inputs. Unlike our previous solution, we could send our inputs one after another, each input with a different amount of delay, while not increasing the cycle count of our implementation. In fact, we could compute `secAND2` in a

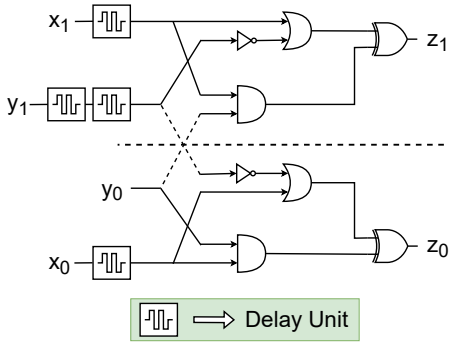


Figure 3: `secAND2` gate with path delay or `secAND2-PD`.

single clock cycle. This may of course increase the critical path delay and thus reduce the circuit's maximum clock frequency.

Consequently, this approach of sending each input with a different amount of delay would help us compute consecutive multiplications without the need to reset the inputs between computations. We propose such a cost-efficient delayed sequence in Figure 3, which we shall call `secAND2-PD`. Each input is either delayed by zero, one, or two `DelayUnits`. We refer to a replicable amount of delay as a `DelayUnit`, and we shall later, in Section 5, explain how this can be realized in practice. Input y_0 is not delayed and arrives first in order to protect against information leakage about the previous computation. It is followed by the delayed x_0 and x_1. And finally, y_1 arrives as the last input as explained above.

Referring to our previous example, in Section 2.3, b_0 arrives before a_0 (or a_1) does. Therefore a_0 (or a_1) cannot leak information about $n(= n_0 \oplus n_1)$ from the previous computation, as n_0 is replaced by b_0. And the final input b_1 arrives after a_0 and a_1, thereby protecting information leakage about the current computation, i.e. a_0 (or a_1) cannot leak information about $b(= b_0 \oplus b_1)$. In conclusion, `secAND2-PD` does not require an input reset and also decreases the latency of our `secAND2` gadget to a single cycle.

3 Composing Secure Masked Circuits

`secAND2-FF` and `secAND2-PD` gadgets can be used as a building block to securely implement more complex masked circuits. In this section, we provide guidelines on two important steps which are generally required to build circuits. We first

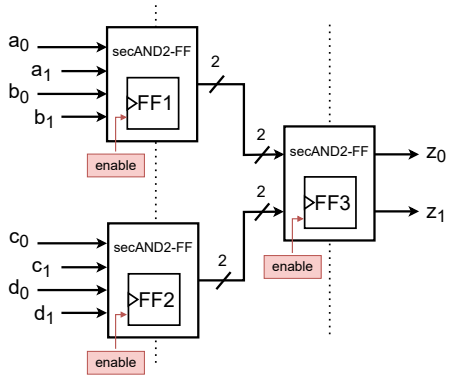


Figure 4: Product of four masked variables using `secAND2-FF`.

show how to compute products of more than two variables securely. And then, we explain how to add *dependent* variables securely.

3.1 Computing Product Terms using `secAND2-FF`

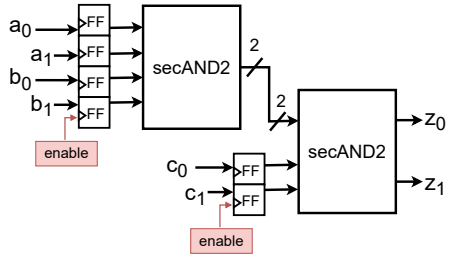


Figure 5: `secAND2` with input registers.

As we start building a circuit, it is common to have situations where we need to compute a product of more than two variables. To illustrate, we compute a product of four variables, $z = a \cdot b \cdot c \cdot d$, which we assume here to be independently shared. Implementing this expression securely can be done with the circuit in Figure 4, which evaluates $z = \texttt{secAND2-FF}(\texttt{secAND2-FF}(a, b), \texttt{secAND2-FF}(c, d))$ using three `secAND2-FF` gadgets and has a latency of three clock cycles.

By carefully controlling when we sample the internal FFs, we can achieve a secure construction with no additional (i.e. external) FFs, which also helps us keep the area footprint low. All the inputs arrive in the first clock cycle, and in the second clock cycle, the enable signal corresponding to flip-flops FF1

Expression	Delay Sequence
Product of 3 variables $z = a \cdot b \cdot c$	$c_0 \to b_0 \to a_0, a_1 \to b_1 \to c_1$
Product of 4 variables $z = a \cdot b \cdot c \cdot d$	$d_0 \to c_0 \to b_0 \to a_0, a_1 \to b_1 \to c_1 \to d_1$

Table 2: Delay sequence for a product of 3 or 4 variables.

and FF2 is set to high. The enable signal controls when the FF samples the input. FF1 and FF2 sample b_1 and d_1 respectively, therefore `secAND2-FF`(a, b) and `secAND2-FF`(c, d) are computed securely. The enable signal of FF3 remains disabled during the second clock cycle, so the `secAND2-FF` computing $z =$ `secAND2-FF`(`secAND2-FF`(a, b), `secAND2-FF`(c, d)) is inactive. And in the third and final clock cycle, the enable signal of FF3 is toggled to high, thereby securely completing the computation of output (z_0, z_1) in three clock cycles.

In the general case, implementing a product of n independent variables requires $n - 1$ `secAND2-FF` gadgets arranged into $log_2(n)$ layers, such that all different sub-products are cascaded. The latency of the circuit becomes $log_2(n) + 1$ cycles.

In specific cases, it might be advantageous to take the internal FFs out of the `secAND2-FF` gadgets and instead place them at the beginning. For instance, when we compute low-degree products. The flip-flop inside `secAND2-FF` serves the purpose of delaying one of the input signals. Equivalently, we can replace `secAND2-FF` with a `secAND2` and place registers before the gate to buffer the input shares, as shown in Figure 5. We can then use a Finite State Machine (FSM) to control when the FFs sample, thus guaranteeing a safe arrival sequence of the input operands to the `secAND2` gadgets. Unlike internal FFs inside `secAND2-FF` gadgets, which solely belong to that gadget, the input registers we now use can be commonly shared by multiple `secAND2` gadgets. For instance, consider the two multiplications $a * b * c$ and $a * b * d$. The input registers used to store a_0, a_1, b_0, b_1, can be shared for the two multiplications. This is usually the case when we are computing polynomials. We would have to compute several different products with common inputs. The resulting circuit can have a slightly larger area due to extra input FFs, but it can be beneficial for evaluation purposes. It allows us, for instance, to test and compare different input sequences or to reset the FFs at any given time.

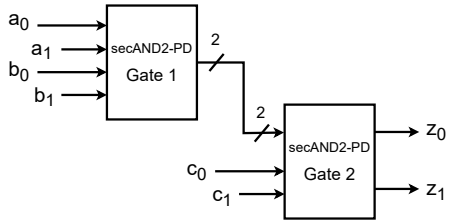


Figure 6: Product of three masked variables using `secAND2-PD`.

3.2 Computing Product Terms using `secAND2-PD`

Our construction for computing a product of more than two variables using `secAND2-PD` resembles a chain-like structure in contrast to the tree structure we illustrated in the previous subsection. This decision was made for practical reasons, as it helped implement hardware delays easier. In our experience, it is relatively easy to enforce delays on inputs of `secAND2-PD` that arrive directly from registers. But in a typical tree structure which is organized in layers, the outputs of `secAND2-PD` gadgets of one layer are fed as inputs to the next layer of `secAND2-PD` gadgets. Based on our experience, it was not as easy to enforce delays on outputs of `secAND2-PD` gadgets. A construction for a product of three variables, $z = a \cdot b \cdot c$, which we assume here to be independently shared, is shown in Figure 6. An appropriately delayed input sequence, as shown in Table 2, can be used to compute the product of three variables in a single clock cycle. To create the delayed sequence, we use a `DelayUnit`, like the one we used in Section 2.4. c_0 is not delayed, b_0 is delayed by one `DelayUnit`, a_0 and a_1 are delayed by two `DelayUnits`, b_1 is delayed by three `DelayUnits` and c_1 is delayed by four `DelayUnits`. The reasoning behind the input sequence remains the same as before, a_0 (and a_1) not only has the potential to leak information about values $b(= b_0 \oplus b_1)$ in Gate 1, see Figure 6, but also $c(= c_0 \oplus c_1)$ in Gate 2. Hence, c_0 is sent first to protect against information leakage about the value of $c(= c_0 \oplus c_1)$ from any previous computation and c_1 arrives last to protect against leakage about the current computation. The rest of the sequence, between c_0 and c_1, is identical to what was discussed in subsection 2.4. Similarly, we can construct an input sequence for a product of four variables, see Table 2.

To generalize, implementing a product of n independent variables requires $n-1$ `secAND2-PD` gadgets arranged into $n-1$ layers. In theory, with a proper input sequence, a product of any number of variables can be computed in a single clock cycle. It is up to the designer to realise such a sequence in practice. In this work, we successfully and securely computed a product of three variables in a single clock cycle. We have not explored computing the product of more

than three variables in a single cycle, as it was not needed for our secure DES implementation.

3.3 Addition of Product Terms

Masked AND and XOR gates are fundamental to building a masked circuit. So far, we concentrated on masking AND gates. But it is also essential to ensure no loss of security during XOR. Both our `secAND2-FF` and `secAND2-PD` gadgets do not consume fresh randomness. Instead, the uniformity of the output is achieved by reusing the randomness of the inputs. This characteristic becomes critical when implementing circuits that combine several terms, for instance, through addition. It can lead to decreased security if the added terms are not independent.

Consider the function $f = x \oplus y \oplus x \cdot y$, where the product term $z = x \cdot y$ is computed with either a `secAND2-FF` or `secAND2-PD` gadget. In this situation, the masked output z is not independent of x and y, leading to a data-dependent distribution of the masked inputs of the XOR plane. Securing this function, as well as any other which combines dependent shares, requires selectively *refreshing* the (intermediate) dependent variables. Figure 7 depicts a circuit to compute f securely. It requires 1 bit of randomness m to refresh the shares of z and guarantee a uniform output distribution.

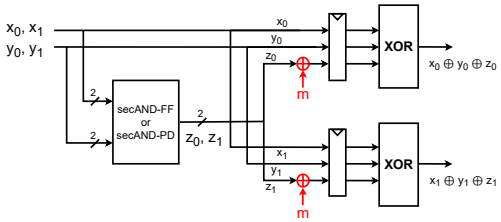


Figure 7: $f = x \oplus y \oplus x \cdot y$ (secure).

4 Case Study: DES

In this section, we use both `secAND2-FF` and `secAND2-PD` gadgets as fundamental building blocks to build a secure cryptographic circuit, following the rules for composition defined in Section 3. We choose the Data Encryption Standard (DES) as our target algorithm because it is the main building block of Triple-DES(TDES), which is still widely used today.

4.1 Deconstructing DES

We begin with the (unprotected) design, which is a classical round-based DES architecture including key schedule. The main difficulty in protecting the DES implementation lies in the S-Boxes, which are the only non-linear components. The DES cipher has 8 different S-boxes in the substitution layer, each taking six input bits $(x_0, x_1, x_2, x_3, x_4, x_5)$ and returning four output bits (y_1, y_2, y_3, y_4). A hardware-friendly way to implement them is to represent them as four 4-bit permutations (the so-called *mini S-boxes*) and a multiplexer (MUX). As we follow a bottom-up approach, we represent each mini S-box through equations in Algebraic Normal Form (ANF) that consist only of AND and XOR terms. Equation 3 shows the representation of a mini S-box.

$$\begin{aligned} y_1 &= 1 \oplus x_1 \oplus x_2 \oplus x_1x_2 \oplus x_2x_3 \oplus x_1x_2x_3 \oplus x_4 \oplus x_2x_3x_4 \\ y_2 &= 1 \oplus x_1 \oplus x_2 \oplus x_1x_3 \oplus x_2x_4 \oplus x_3x_4 \oplus x_1x_3x_4 \\ y_3 &= 1 \oplus x_1x_2 \oplus x_3 \oplus x_1x_3 \oplus x_2x_3 \oplus x_1x_2x_3 \oplus x_4 \oplus x_1x_4 \\ &\quad \oplus x_2x_4 \oplus x_1x_2x_4 \oplus x_3x_4 \\ y_4 &= x_1 \oplus x_3 \oplus x_1x_4 \oplus x_2x_4 \oplus x_1x_3x_4 \end{aligned} \tag{3}$$

As visible in the equations, there are at most six distinct terms of degree 2 and four terms of degree 3. Additionally, all four mini S-boxes of an S-box share common inputs, which means it is sufficient to compute the ten possible product terms only once. The equations in ANF form can be split into two stages: AND stage and XOR stage. The AND stage can be implemented with 10 AND gates. In the XOR stage, the product terms from the AND stage are XORed together with the inputs to form the outputs of the mini S-boxes, as shown in Equation 3. Since the outputs of the AND stage are not independent of the inputs, it is necessary to add a refresh layer in front of the XOR layer, as motivated in Section 3.3. The outputs of the 10 AND gates in the AND stage are refreshed with 10 bits of fresh randomness before the XOR layer. It is possible to further optimize the refresh step by selectively refreshing only some of the ten terms instead of refreshing all of them while maintaining uniformity, but we leave this optimization for future work.

The purpose of the MUX is to select which one of the four mini S-boxes to output. Similar to the mini S-boxes, computations can also be split into three stages. Equation 4 is in ANF and shows how each output bit of the 4×1 MUXes is computed. First, we calculate the four multiplications of the select bits, x_0 and x_5, that is: $x_0x_5, x_0\overline{x_5}, \overline{x_0}x_5, \overline{x_0}\ \overline{x_5}$. Second, each of these products is multiplied with the respective output of the mini S-boxes. Since the MUX receives 16 inputs from the four mini S-boxes, there are 16 multiplications in

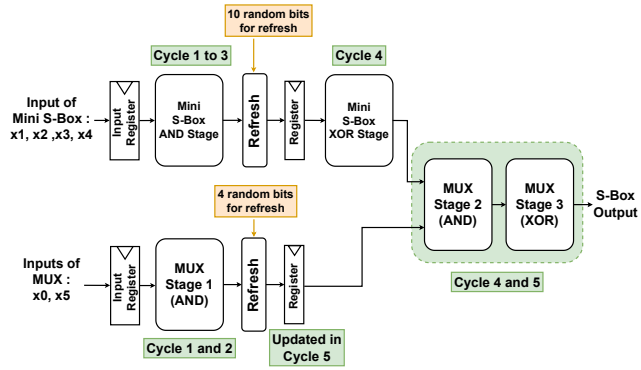
the second stage. Lastly, in the third stage, the outputs of the second stage are XORed together to produce the four S-Box output bits.

$$\begin{aligned} y_1 &= (x_0x_5)y_1^{(1)} \oplus (x_0\overline{x_5})y_1^{(2)} \oplus (\overline{x_0}x_5)y_1^{(3)} \oplus (\overline{x_0}\ \overline{x_5})y_1^{(4)} \\ y_2 &= (x_0x_5)y_2^{(1)} \oplus (x_0\overline{x_5})y_2^{(2)} \oplus (\overline{x_0}x_5)y_2^{(3)} \oplus (\overline{x_0}\ \overline{x_5})y_2^{(4)} \\ y_3 &= (x_0x_5)y_3^{(1)} \oplus (x_0\overline{x_5})y_3^{(2)} \oplus (\overline{x_0}x_5)y_3^{(3)} \oplus (\overline{x_0}\ \overline{x_5})y_3^{(4)} \\ y_4 &= (x_0x_5)y_4^{(1)} \oplus (x_0\overline{x_5})y_4^{(2)} \oplus (\overline{x_0}x_5)y_4^{(3)} \oplus (\overline{x_0}\ \overline{x_5})y_4^{(4)} \end{aligned} \tag{4}$$

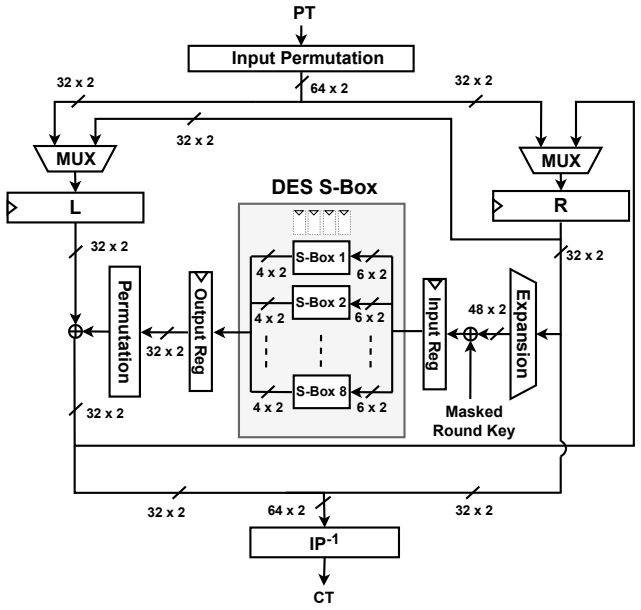
Finally, we add the MUX and the four mini S-box circuits to construct the DES S-box. Our final DES implementation is constructed by adapting the round-based architecture to incorporate first-order Boolean masking. All original variables (plaintext, key, round state) are split into two shares and operations are performed in the masked domain. Datapaths for linear operations are simply duplicated to operate on individual shares. Our design also includes a masked key schedule that runs parallel to the DES operation. The substitution layer is adapted to incorporate our protected S-Boxes.

4.2 Building DES with `secAND2-FF`

Figure 8a visually represents all stages of the S-Box using `secAND2-FF`. The AND stage of the mini S-boxes has product terms with a maximum degree of 3. Hence this stage can be computed in 3 clock cycles using ten `secAND2-FF` gadgets. As the mini S-boxes are composed of low-degree polynomials, we follow the approach we suggested in section 3.1. We remove the internal FFs from `secAND2-FF` and place an input register layer. The calculation of the mini S-box outputs takes four clock cycles, three for the AND stage and one for the XOR stage. The calculation of the MUX AND Stage 1 (variables of the form (x_0x_5)) is done in parallel and takes two cycles. Both outputs are then sent to the MUX AND Stage 2. As the inputs for the MUX AND Stage 2 come from two parts of the S-Box, they must be synchronized to guarantee a safe multiplication sequence. This is why a register is placed after the MUX AND Stage 1. Recall that we explained the need to refresh dependent variables in Section 3.3. Therefore, we need to refresh the output of the MUX AND Stage 2 to maintain uniformity before going through the MUX XOR Stage 3. To save costs, we move the refresh stage inside the MUX and safely apply it directly after the MUX AND Stage 1. By doing so, we save one clock cycle in the overall S-box computation while also reducing the randomness requirements, as fewer terms need to be refreshed. Given that a register is already placed after the MUX AND Stage 1 (to ensure a safe sequence for the MUX AND Stage 2),



(a) S-box architecture



(b) High-level DES architecture (protected)

Figure 8: Building DES using `secAND2-FF`.

this optimization also saves some flip-flops. The final S-box architecture using `secAND2` with input register layer has a latency of 5 clock cycles, and it is the one we use in our evaluation and experiments.

The final DES architecture is shown in Figure 8b. After adding an S-box output register and a state register (L and R) between successive rounds, the design

yields a final latency of 7 cycles per round. Whether the S-box output register can be removed and reduce the latency to 6 cycles, without affecting the security of the implementation, is a question we leave for future work.

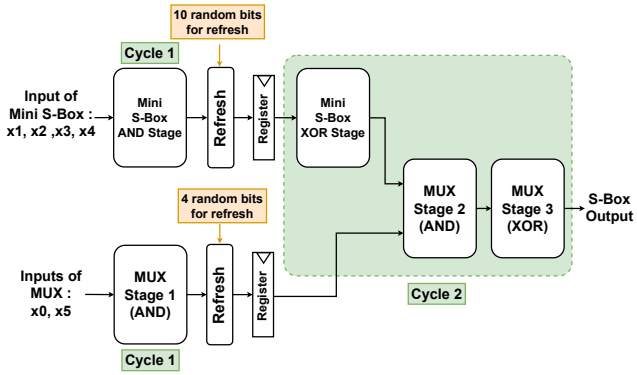
4.3 Building DES with `secAND2-PD`

Using `secAND2-PD`, every AND stage in the circuit can be executed in a single cycle. The S-box architecture for the design using `secAND2-PD` is shown in Figure 9a. We already discussed how to compute a product of up to three inputs in Section 3.2. Hence the mini S-box AND stage can be implemented with 10 `secAND2-PD` instances and with a reduced latency of one clock cycle. For the Mini S-box AND Stage and the MUX Stage 1, the path delays are applied to the inputs $(x_0, x_1, x_2, x_3, x_4, x_5)$. For the MUX Stage 2, the delays are applied to the refreshed outputs of MUX Stage 1 and the linear Mini S-box XOR Stage outputs. The result is an S-box architecture with a latency of 2 clock cycles.

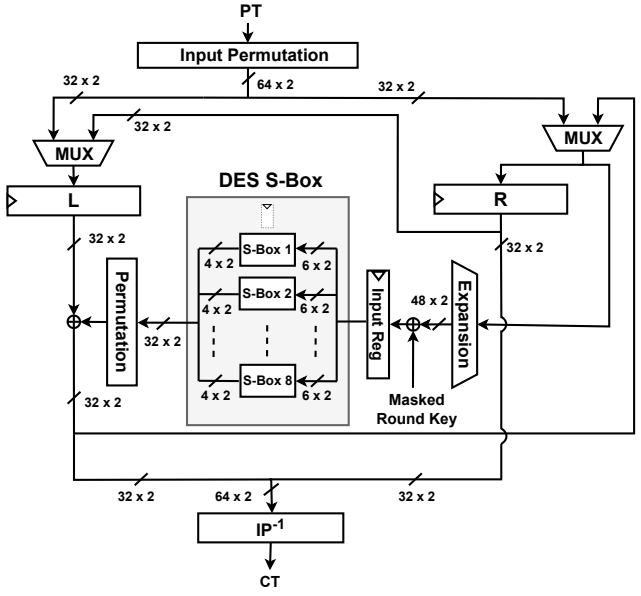
Our focus for DES design using `secAND2-PD` is to reduce implementation costs, latency to be specific. Therefore, we decided not to increase latency by adding any additional register stages outside the S-box. For the previous design using `secAND2-FF`, the input register is crucial for security as it is used to control the order of the input sequence. Because of this, the state register (R), see Figure 8b, is updated first and then the state register (R) is used to update the input register over the next few clock cycles, to compute the S-Box safely. But for `secAND2-PD`, the inherent path delay takes care of the input sequence. And for that reason, the output of the S-box is directly connected to the input register, see Figure 9b, instead of going through the state register, which saves a clock cycle. The state register is updated in parallel, and the S-box output register is also removed. Hence, the latency of our design with path delay is reduced to 2 clock cycles per round.

5 Implementing path delays in hardware

Glitches are a well-known characteristic of CMOS circuits that make it difficult to implement masking securely in hardware. Glitches are unintended transitions at the output of gates in a circuit due to the delayed arrival of some input signals. To solve this, in Section 2.4, we proposed using path delay to control the order in which inputs arrive. This section explains the steps we took to realize path delay in hardware. We use a SAKURA-G board with a Xilinx SPARTAN-6 FPGA to implement our protected DES implementation, using Xilinx ISE software. We use Look Up Tables (LUT) as a basic delay element, by



(a) S-box architecture



(b) High-level DES architecture (protected)

Figure 9: Building DES using `secAND2-PD`.

connecting the input of the LUT directly to its output. The LUT acts as a buffer for the input and provides the path delay we need. Several LUTs are chained together to form a basic unit, which we refer to as a *`DelayUnit`*. For path delay to be an effective countermeasure, it needs to be replicable. If we were to let ISE design tools perform the placement and routing of our basic delay elements

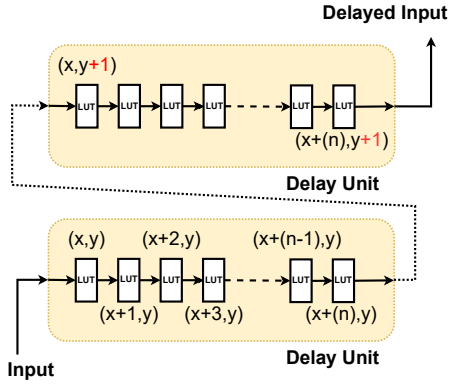


Figure 10: An input is delayed by two `DelayUnits`. Each `DelayUnit` is of size n LUTs.

(LUT), the amount of delay would vary depending on where the LUTs are placed on the FPGA. This would result in an inconsistent outcome, and the amount of delay would be hard to quantify. For that reason, we manually perform the placement of our delay elements by defining constraints. FPGA resources such as LUTs, flip-flops and multiplexers are grouped together in SLICEs to create configurable logic blocks. Every SLICE of a SPARTAN-6 FPGA contains four LUTs, and by manually choosing the location of the SLICE, we can control the placement of our delay elements. To create a `DelayUnit`, we place the delay elements horizontally by incrementing the x coordinate of the SLICE, as shown in Figure 10. If a signal has to be delayed by more than one `DelayUnit`, multiple `DelayUnits` are stacked vertically using location constraints on the FPGA. As a result, we obtain a deterministic layout with a delay amount that can be controlled and quantified.

Our next step is to investigate the optimal size for a `DelayUnit`, i.e. the number of LUTs chained together, to achieve practical security. As a general rule, the more the delay, the better it is for security. A large amount of delay would guarantee that the arrival of inputs is well separated in time. But it is also important to note that there is a trade-off between cost and security. Creating a large delay chain increases the utilization cost of our implementation and also limits the maximum operating frequency of our implementation. As the focus of our work is to find a trade-off between low-cost and practical security, we gradually increase the size of our delay chain until we find a good balance.

We tested several sizes of the `DelayUnit` through leakage assessment experiments and found a `DelayUnit` of size 10 LUTs to be optimal. Please refer to Section 7

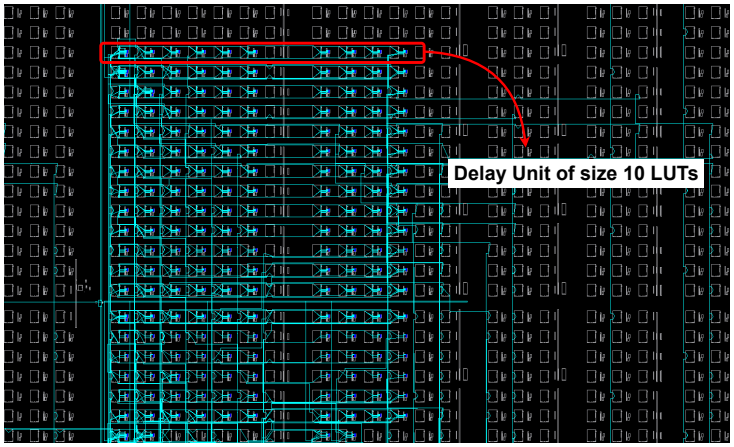


Figure 11: `DelayUnits` for an S-box.

where we validate this claim with detailed leakage test results for different sizes of the `DelayUnit`. Figure 11 shows the placement and routing of the `DelayUnits` that correspond to one of the S-boxes of the protected DES design using `secAND2-PD` on a SPARTAN-6 FPGA. The placement and routing were performed by Xilinx ISE software with the location constraints we assigned to individual delay elements. Figure 12 shows a zoomed-out view covering all the S-boxes of DES. Each red rectangle represents one S-box, hence 8 in total.

6 Implementation Results

The results of our full DES implementations are summarized in Table 3. We include area reports from both ASIC and FPGA synthesis. The ASIC design results are gathered using NanGate 45nm Open Cell Library [18] and synthesized with Synopsys DC Compiler 2017.09. We put elementary `secAND2` and XOR gates in modules and compile with `-exact_map` option to prevent the modules from being optimized. The FPGA design results are obtained from implementing the design on a Spartan-6 FPGA, the same model as used in the evaluation in Section 7, using the Xilinx ISE software. Note that to prevent optimization of modules and interaction between shares, we synthesize with "Keep Hierarchy" constraint set to on. Lastly, we provide in the table also the numbers from the work in [20], reporting a TDES implementation protected with DOM. Note that the number of cycles as reported in [20] is scaled down to compare with a DES implementation (the original cycle count for TDES is 5*48+4).

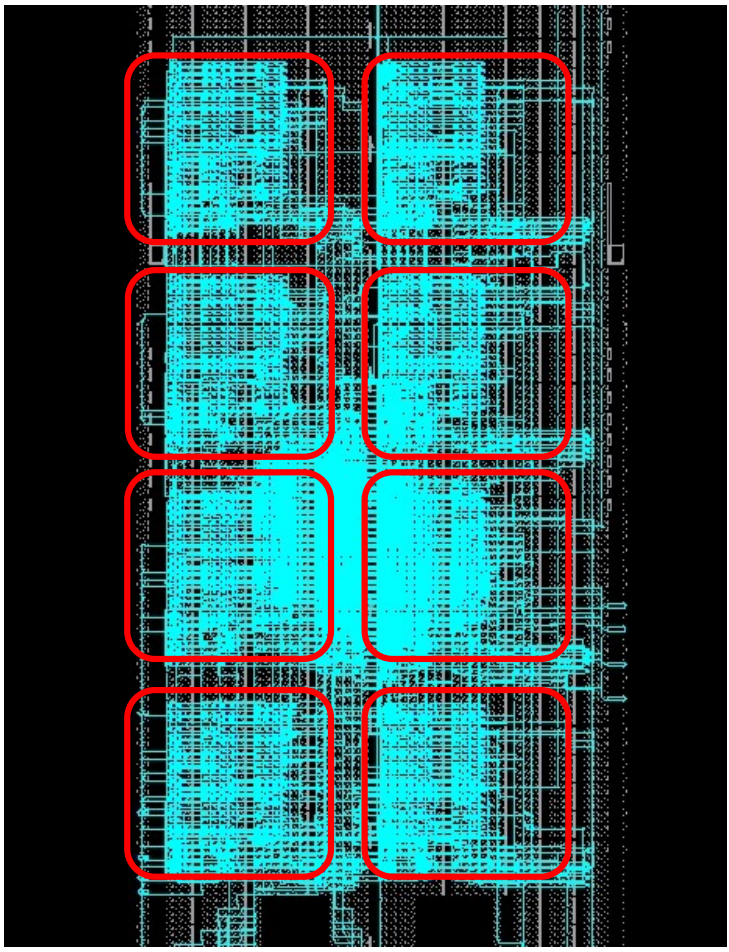


Figure 12: All `DelayUnits` for DES using `secAND2-PD`.

Table 3: Utilization results of full DES implementations including the masked key schedule.

Version		**ASIC** [GEs]	**FPGA** [FF/ LUT]	**Rand**[§]	**Cycle**[§]	**Max Freq.**[‖] [MHz]
`secAND2-FF`		14488	819/ 2129	14	7	183
`secAND2-PD`		52273	678/ 7122	14	2	21
[20][#]	DOM-indep	13800[†]	-	176	5	-
	DOM-dep	22400[†]	-	528	5	-

[†] Calculated using a 28 nm RVT standard cell library from Global Foundries.
[#] Both versions from [20] do not mask the key schedule. [§] per round
[‖] Obtained from Xilinx ISE timing report

6.1 Utilization for `secAND2-FF` version

From the area results, we can see that our design using `secAND2-FF` is more compact than the DOM-dep version in [20] but slightly larger than the DOM-indep version. Our design uses 30 `secAND2` gates per DES S-Box, compared to 22 DOM-indep gates in [20]. While `secAND2` gates are by design smaller than the DOM-indep gates, the additional modules in our implementation (e.g. refreshing gadgets, input/output S-box registers, etc.) compensate for the gain to some extent. A non-negligible difference here is, however, that our GE results include the masked key schedule which adds around 900 GEs to the total area cost. This is not the case in [20], in which the numbers include the cost of an unmasked key schedule. The overhead caused by this module further explains the difference between both designs.

In terms of randomness usage, our design involves fewer bits, 14 bits per round, as we opt to reuse randomness across the eight different S-boxes. We note this decision does not have an impact on the first-order security (see Section 7), and hence we opt to use it in our reference implementation. In case our design would not recycle randomness, or in case the work in [20] did, then our numbers, 14*8 = 112 bits per round, would still be lower than both DOM-dep and DOM-indep. Additionally, we only expect a slight increase in area due to additional FFs required to store the 112 bits of randomness. We also note that in [20] there is no evaluation of the security of the DOM-indep version, but rather of the DOM-dep version which consumes 3 random bits per AND gate.

Lastly, our implementation takes 2 more cycles per round compared to [20]. While all designs feature a protected S-box with a latency of 5 cycles, the two additional input/output S-box registers in our design increase the overall round latency to 7 cycles. The input S-box register is a design choice, which facilitates control over the sequence of the input operands to the mini S-box stage and allows us flexible resetting of the flip-flops. Whether the output S-box register can be removed without affecting the security of the implementation is a question we leave for future work.

6.2 Utilization for `secAND2-PD` version

The latency for our design using `secAND2-PD` is two cycles per round, much less than compared to [20]. Our randomness usage remains the same, 14 bits per round, for both versions using `secAND2-FF` and `secAND2-PD`.

The usage of `DelayUnit` in our design using `secAND2-PD` complicates the area comparisons between designs. We use LUTs as delay elements in our FPGA

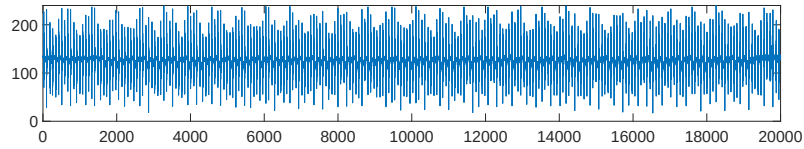


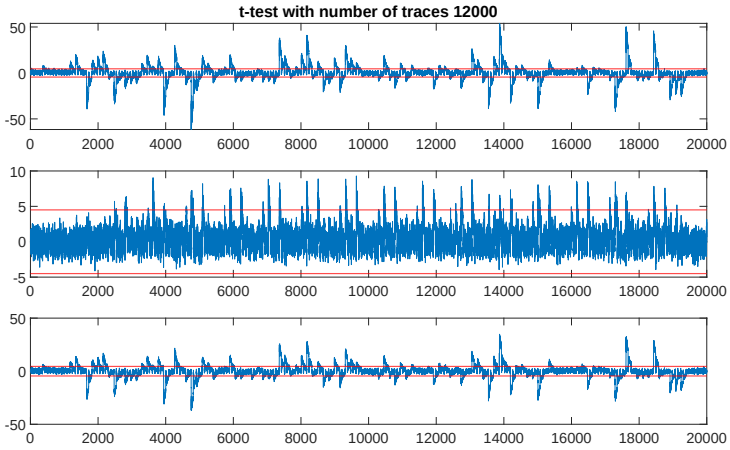
Figure 13: Power trace for protected DES design using `secAND2-FF` (seven cycles per round).

implementation. In Section 5, we explained that we find the optimal size for `DelayUnit` through multiple iterations and by using leakage results as feedback. However, this process of tweaking the implementation to find the optimal size would be very expensive and also impractical for ASIC implementations. Therefore, it is a challenging task to provide accurate ASIC implementation costs for our design using `secAND2-PD`. Instead, we follow a different approach to provide a reasonable area estimate. For our ASIC design, we use inverters (NOT gates) as our delay elements to buffer signals and measure the size of `DelayUnit` by the number of inverters they are composed of. We first synthesize a circuit without any delay elements, which is equivalent to a protected DES implementation using `secAND2-PD` but with `DelayUnit` of size 0 LUTs. And then estimate the path delay of the critical path of the circuit. We set the size of the `DelayUnit` as the number of inverters required to achieve a delay of the critical path multiplied by a factor of 3, to accommodate a safety margin. From further synthesis runs, we estimated the size of `DelayUnit` for the desired amount of delay to be 120 inverters. Table 3 shows the total area including all `DelayUnits`, each of size 120 inverters. If we exclude the `DelayUnits`, the area of the remaining circuit only adds up to *12592 GEs*. Having a generous amount of 120 inverters per `DelayUnit` contributes to the rest of the area utilization. Whether the size of `DelayUnit` can be reduced without affecting the security is hard to predict.

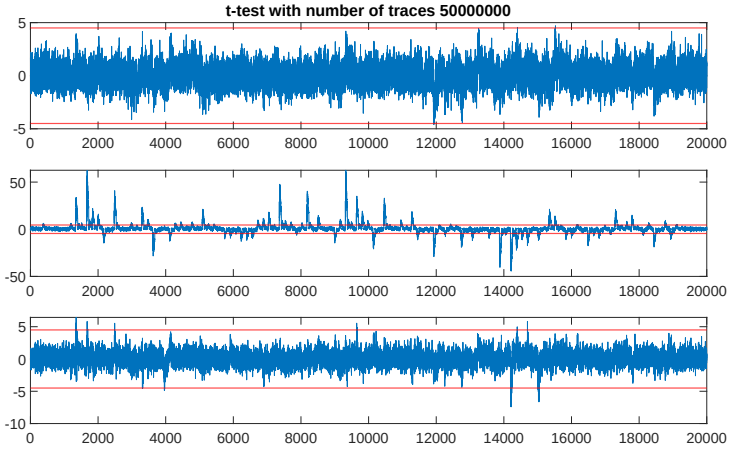
We would like to emphasize that the purpose of utilization results for our design using `secAND2-PD` is only to provide a realistic estimate and not a new state-of-the-art DES implementation. As it is difficult to provide an accurate number that would guarantee the security of an ASIC implementation without testing it.

7 Security Evaluation

We use a SAKURA-G board to evaluate the side-channel security of our protected DES implementations on the target SPARTAN-6 FPGA. We measure



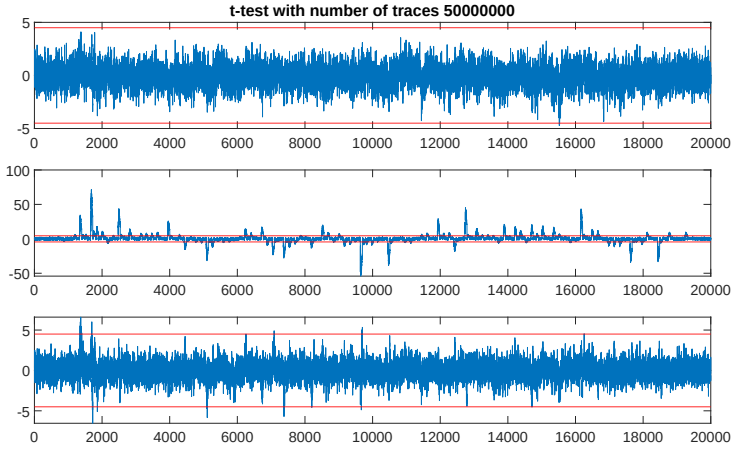
(a) PRNG off (12 thousand traces)



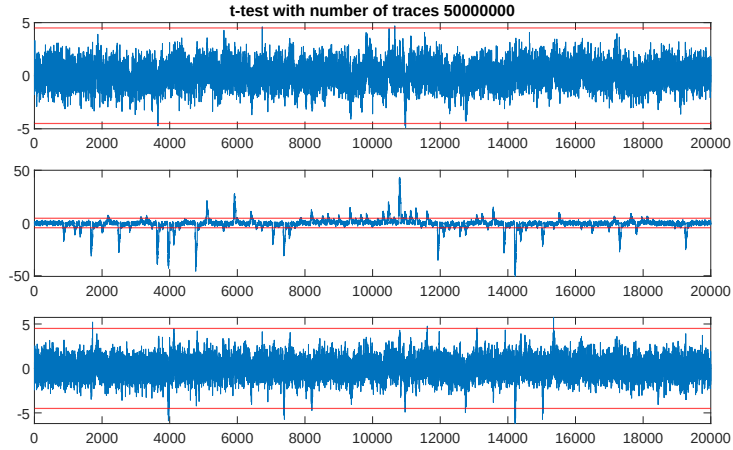
(b) Fixed Plaintext 1 (50 Million traces)

Figure 13: Leakage assessment results for protected DES design using `secAND2-FF` (Part 1).

the (onboard amplified) power consumption of the target FPGA on SMA connector J3 with a Tektronix DPO7254C oscilloscope sampling at 500MS/s and operating at 3 MHz. The power measurements cover the entire DES operation. We use the non-specific Test Vector Leakage Assessment (TVLA) methodology as explained in [2] and originally described in [10]. We focus on univariate analysis since our protected DES core processes both shares in parallel. We perform fixed vs. random tests where we provide either a fixed or



(c) Fixed Plaintext 2 (50 Million traces)



(d) Fixed Plaintext 3 (50 Million traces)

Figure 13: (Continued) Leakage assessment results for protected DES design using `secAND2-FF` (Part 2).

a random plaintext to the masked DES core, in a random sequence. The DES key is fixed for all experiments but masked before every DES operation. We use only 14 random bits per round, thus recycling them for all 8 S-boxes in parallel.

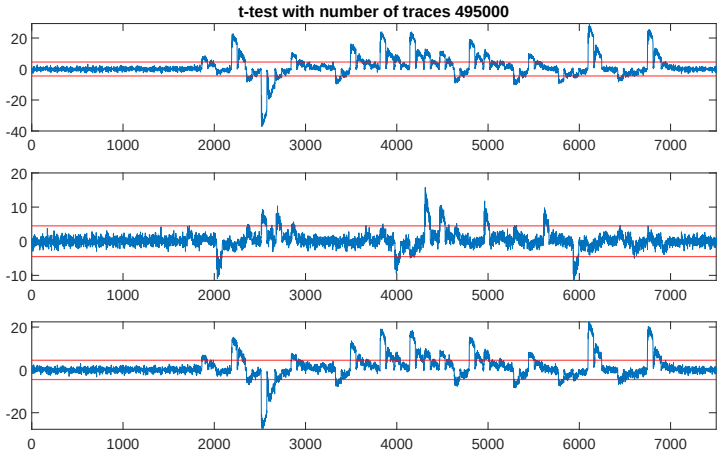
7.1 Leakage assessment of protected DES design using `secAND2-FF`

Figure 13 shows the power trace (raw oscilloscope ADC output) covering the entire DES operation. Figures 13b, 13c, 13d display the results we obtained in three tests using three different fixed plaintexts for the design using `secAND2-FF`. We measured and analyzed 50 million traces for each of these tests. Each Figure shows from top to bottom: first-order univariate t-test result, second-order univariate t-test result and third-order univariate t-test result. Red lines indicate the commonly applied threshold at ± 4.5. As we can see, there is no evidence of first-order leakage. There are very few and minor crossings of the ± 4.5 threshold in the first-order t-test values, but they are not consistent across the three different plaintexts, i.e. the threshold is not exceeded at the same time indexes, as is required by the TVLA methodology in order to deem an implementation leaking. More importantly, we have clear evidence of second-order leakage, with the second-order t-test reaching values as large as 60. Therefore an adversary would likely be better off using a second-order attack against our protected DES implementation, as similarly argued in [3], which require exponentially more traces by adding noise, as discussed earlier. Alternatively, one can simply prevent an attacker from acquiring the required number of traces with protocol-level countermeasures such as re-keying.

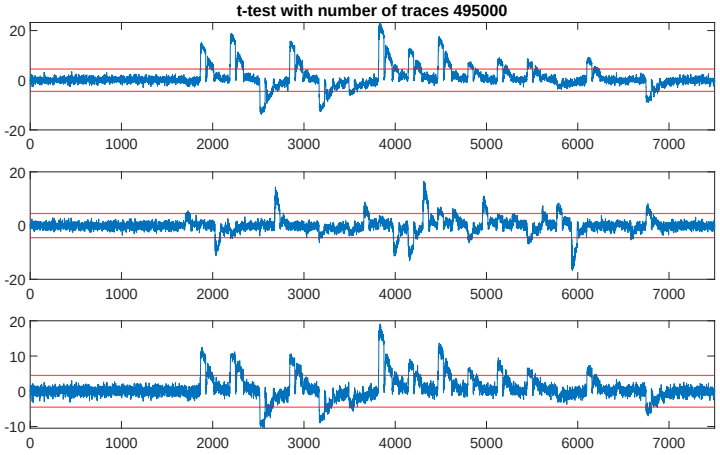
For completeness, we provide a result of the same leakage assessment performed with the PRNG switched off, which means all random bits used for the initial masking and used by the masked DES core are zero. Figure 13a shows that with as little as 12 000 traces we obtain very significant peaks in the first-order t-test, which confirms that our setup works well.

7.2 Finding the optimal `DelayUnit` size for `secAND2-PD`

In Section 5, we explained that in practice, the size of a `DelayUnit` is important for security. In this subsection, we provide evidence to support our claim. We implement several different versions of our protected DES using `secAND2-PD`, which we explained in Section 4.3. The only difference between the versions is the size of `DelayUnit`. We can confidently say that any difference in leakage assessment results we might observe is directly attributed to the change in the `DelayUnit` size, as all other aspects of the implementation remain the same. For our first version, we start with the smallest possible size of a `DelayUnit` consisting of a single LUT. And for successive versions, we gradually increase the size of `DelayUnit`. We perform fixed vs. random tests for all versions. To



(a) Size: 1 LUT (Half a million traces)

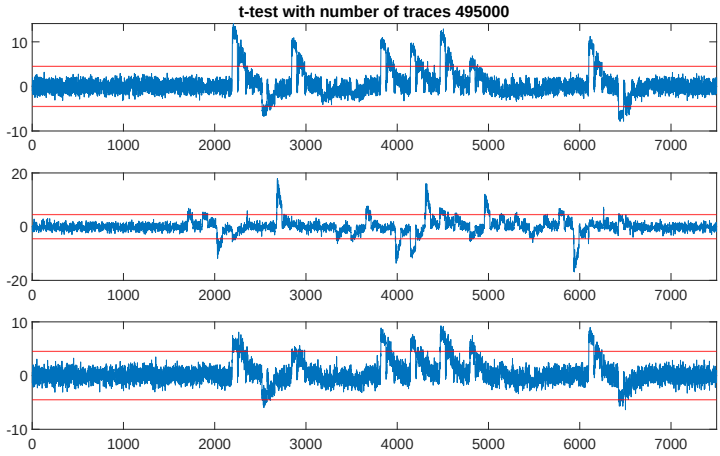


(b) Size: 3 LUT (Half a million traces)

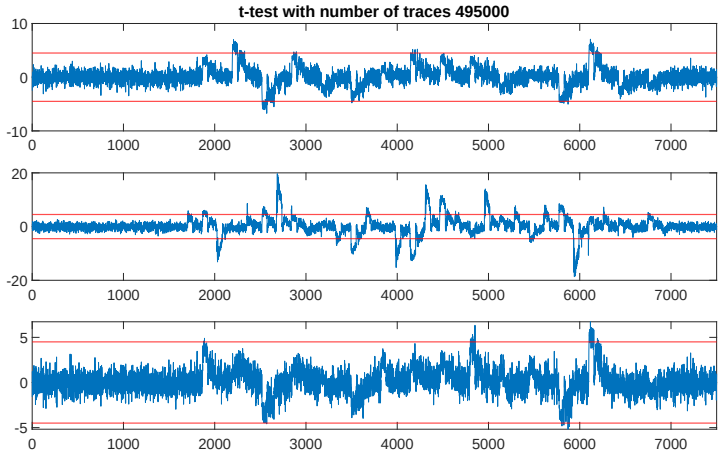
Figure 14: Leakage assessment results for different `DelayUnit` sizes for `secAND2-PD` (Part 1).

make a fair comparison of the test results, we use the same fixed plaintext and collect the same number of traces for all versions.

Figure 15 shows the power trace covering the entire protected DES using `secAND2-PD`. Figure 14 shows the leakage assessment results for all versions. We collect half a million traces for all versions with the same fixed plaintext.



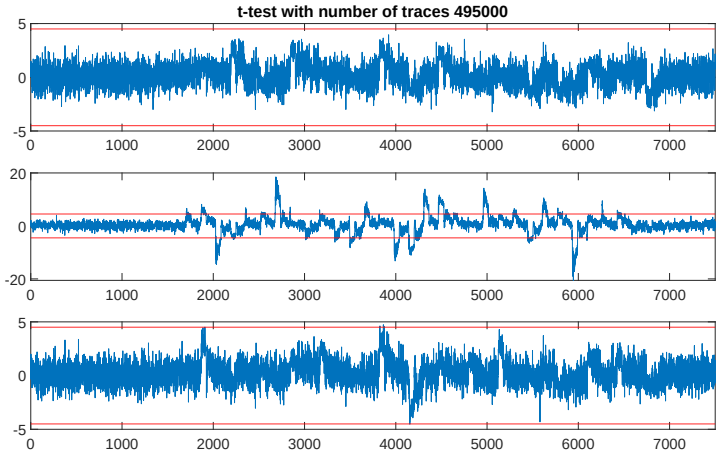
(c) Size: 5 LUT (Half a million traces)



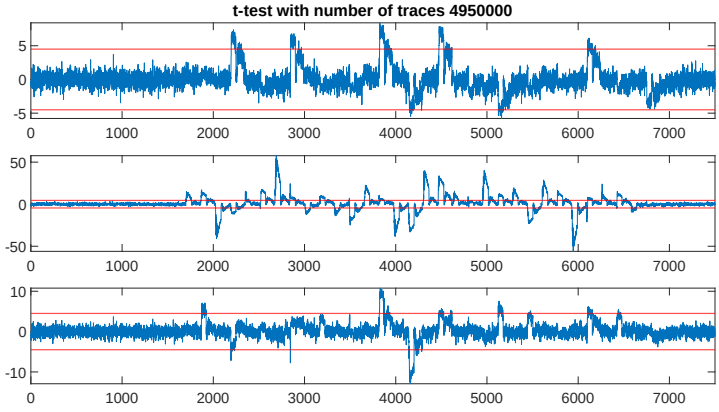
(d) Size: 6 LUT (Half a million traces)

Figure 14: (Continued) Leakage assessment results for different `DelayUnit` sizes for `secAND2-PD` (Part 2).

Each subfigure shows from top to bottom: first-order univariate t-test result, second-order univariate t-test result and third-order univariate t-test result. We see pronounced leakage for the design using the smallest `DelayUnit` size, see Figure 14a. A simple explanation would be that a `DelayUnit` consisting of just a single LUT is not enough to have any significant impact on the path delay. Therefore it cannot guarantee the delayed sequence required for secure



(e) Size: 7 LUT (Half a million traces)



(f) Size: 7 LUT (Five million traces)

Figure 14: (Continued) Leakage assessment results for different `DelayUnit` sizes for `secAND2-PD` (Part 3).

computation of `secAND2-PD`, hence the leakage. From the rest of the figures, we can notice that there is a pattern with the first-order leakage as we increase `DelayUnit` size. We can see a decrease in first-order leakage as we increase the size. And for a `DelayUnit` of size 7 LUTs, see Figure 14e, we can no longer see any first-order leakage with half a million traces. But upon collecting more traces, we eventually started noticing first-order leakage, which indicates that

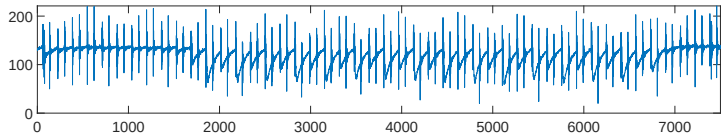


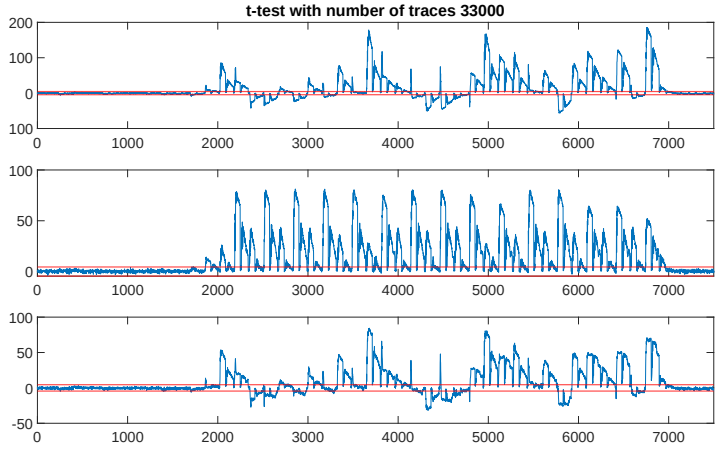
Figure 15: Power trace for protected DES design using `secAND2-PD` (two cycles per round).

a size of 7 LUTs is still not the optimal size. Figure 14f shows the leakage assessment results with 5 million traces. So we increased the size further and found the `DelayUnit` size of 10 LUTs to be optimal. And from our experiments, sizes of more than 10 LUTs do not contribute to any significant improvement in first-order leakage and only result in increased utilization costs. The critical path for our protected DES using `secAND2-PD` lies in the mini S-Box AND stage, see Figure 9a. The mini S-Box AND stage has product terms with a maximum degree of 3 and the critical path of `secAND2-PD` is determined by the input share which is delayed the most. To compute a product of three variables, the input share c_1 is delayed by 4 `DelayUnits`, see Table 2. From the Xilinx ISE timing report, with a `DelayUnit` size of 10 LUTs, the FPGA implementation of our protected DES design with `secAND2-PD` has a maximum operating frequency of 21 MHz. In comparison, the FPGA implementation of our protected DES using `secAND2-FF` results in a maximum frequency of 183 MHz.

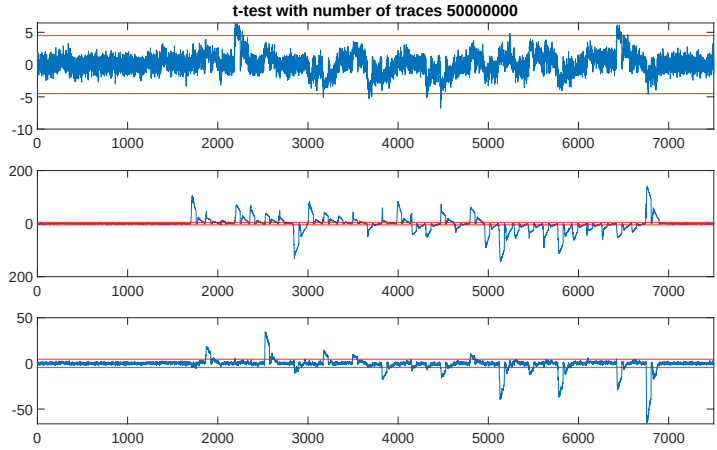
In the next subsection, we present the leakage assessment results of our final version of protected DES with `DelayUnit` size of 10 LUTs.

7.3 Leakage assessment of protected DES design using `secAND2-PD`

Figure 16 shows the leakage test results with the same three fixed plaintexts we used to evaluate the protected DES with `secAND2-FF`. The sanity check with PRNG off in Figure 16d shows that with 33,000 traces we obtain very significant peaks in the first-order t-test. Comparing the t-test results with PRNG on to our previous results using `secAND2-FF`, we can see that the first-order t-statistic does indeed exceed the threshold of ± 4.5. We think there are two possible explanations for this observation. The first explanation is that the `DelayUnit` size of 10 LUTs does not provide enough delay. The security of `secAND2-PD` relies on the input sequence. We demonstrated in Section 7.2 that increasing the amount of delay will separate the arrival time of the inputs, which is necessary for a safe computation. Therefore, a `DelayUnit` size of more than 10 LUTs



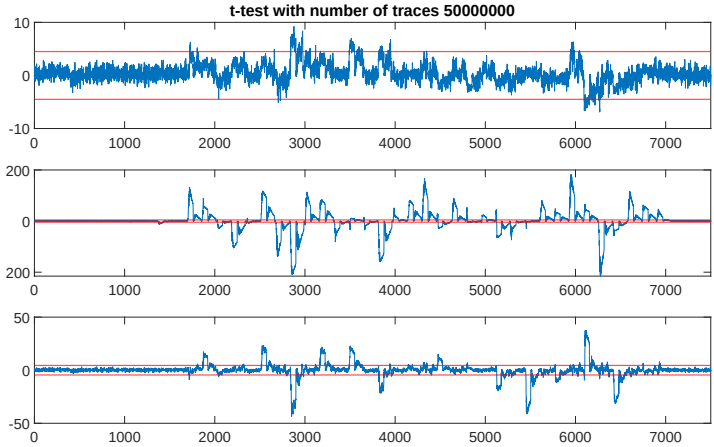
(a) PRNG off (33 thousand traces)



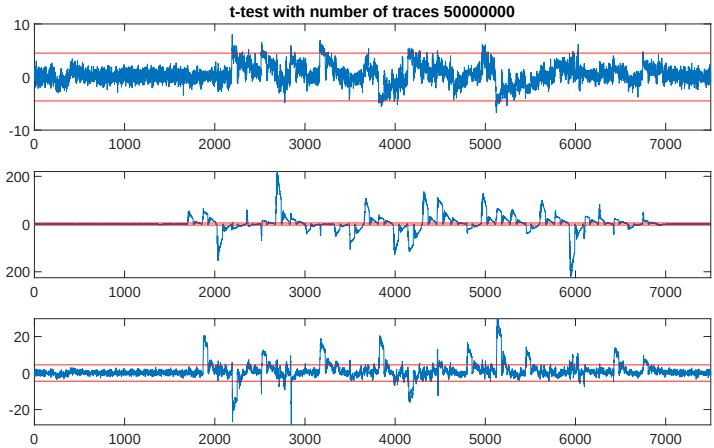
(b) Fixed Plaintext 1 (50 Million traces)

Figure 16: Leakage assessment results for protected DES design using `secAND2-PD`. (Part 1)

should be more secure. But we conducted further experiments, and we did not notice much improvement from increasing the `DelayUnit` size, which leads us to our second explanation. For *d+1*-implementations with *d=1*, i.e. 2-share implementations, multiple prior works, [17], have noted first-order leakage even if the implementations are first-order probing secure [7]. Provable secure implementations do not always translate to secure masked implementations



(c) Fixed Plaintext 2 (50 Million traces)



(d) Fixed Plaintext 3 (50 Million traces)

Figure 16: (Continued) Leakage assessment results for protected DES design using `secAND2-PD`. (Part 2)

due to the presence of physical effects such as *coupling* [6]. These effects are particularly relevant to our design because of the long path delays we create in our circuit for `secAND2-PD`, which in turn creates a lot of opportunity for coupling.

8 Conclusion

We describe two approaches to designing and implementing a low-cost first-order secure Boolean masked DES encryption engine in hardware in the presence of glitches. While provable security is certainly important and informative in this domain, it can but does not always lead to implementations that are secure in practice. This is typical because the model that was used to prove security does not entirely capture physical reality. We therefore do not aim for provable security but for practical security by requiring no first-order leakage up to a number of measurements where we see strong evidence of higher-order leakage. Admittedly our design and implementation ideas are strongly inspired by approaches like TI and DOM. But we add security measures only where practical leakage assessment pointed out an issue, and not to make the theory work. We construct two first-order Boolean masked DES cores with our two AND gates, `secAND2-FF` and `secAND2-PD`.

Using `secAND2-FF`, our design shows no evidence of first-order leakage with 50M traces. The only related work we found is by Sasdrich and Hutter [20]. Taking the difference in the key schedule into account, our area number is comparable to their DOM-indep result and we need less randomness per S-box than their cheapest variant, DOM-indep. They perform leakage assessment only for the significantly more costly (area and randomness) DOM-dep implementation. Our implementation is moderately slower, 115 clock cycles compared to 84 clock cycles, which is our trade-off choice. Using `secAND2-PD`, we substantially reduce the number of clock cycles compared to DOM-indep and DOM-dep implementations, while the randomness cost remains the same. Our `secAND2-PD` gadget, uses LookUpTables (LUTs), which is an FPGA component, to create path delays. Hence, a fair comparison of our area with the reported ASIC numbers of [20] is hardly possible. However, we emphasize that the purpose of our `secAND2-PD` design is to provide a hardware-oriented solution for masking. Recently, there has been an increasing interest in low-latency masking techniques. Some of the latest approaches focus on preventing glitch propagation by substituting registers with dual-rail encodings and asynchronous logic. However, while achieving low latency, these methods often face limitations in lower maximum frequencies. Our `secAND2-PD` gadget provides an alternative solution tailored for applications such as smart cards or RFID, which do not require fast clock frequencies. Considering there is not much research on practical solutions for masking on hardware, we demonstrate the effectiveness of using LUT-based path delay as an interesting new research direction.

Acknowledgments

This work was supported in part by the Flemish Government through the Cybersecurity Research Program with grant number VOEWICS02, by the European Commission through the Horizon 2020 research and innovation program under grant agreement Belfort ERC Advanced Grant 101020005 695305, and through the Horizon Europe research and innovation program under grant agreement HORIZON-CL3-2021-CS-01-02 101070008 ORSHIN.

References

[1] Beckers, A., Wouters, L., Gierlichs, B., Preneel, B., and Verbauwhede, I. Provable secure software masking in the real-world. In *Constructive Side-Channel Analysis and Secure Design - COSADE 2022* (2022), J. Balasch and C. O'Flynn, Eds., vol. 13211 of *LNCS*, Springer, pp. 215–235.

[2] 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.

[3] Bilgin, B., Gierlichs, B., Nikova, S., Nikov, V., and Rijmen, V. A more efficient AES threshold implementation. In *Progress in Cryptology - AFRICACRYPT 2014* (2014), D. Pointcheval and D. Vergnaud, Eds., vol. 8469 of *LNCS*, Springer, pp. 267–284.

[4] Biryukov, A., Dinu, D., Corre, Y. L., and Udovenko, A. Optimal first-order boolean masking for embedded iot devices. In *Smart Card Research and Advanced Applications - CARDIS 2017* (2017), T. Eisenbarth and Y. Teglia, Eds., vol. 10728 of *LNCS*, Springer, pp. 22–41.

[5] Chari, S., Jutla, C. S., Rao, J. R., and Rohatgi, P. Towards sound approaches to counteract power-analysis attacks. In *Advances in Cryptology - CRYPTO '99* (1999), M. J. Wiener, Ed., vol. 1666 of *LNCS*, Springer, pp. 398–412.

[6] 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.

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

[8] Dhooghe, S., Nikova, S., and Rijmen, V. Threshold implementations in the robust probing model. In *Proceedings of ACM Workshop on Theory of Implementation Security, TIS@CCS 2019, London, UK, November 11, 2019* (2019), B. Bilgin, S. Petkova-Nikova, and V. Rijmen, Eds., ACM, pp. 30–37.

[9] 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.

[10] Goodwill, G., Jun, B., Jaffe, J., and Rohatgi, P. A testing methodology for side channel resistance validation. `http://csrc.nist.gov/newsevents/non-invasive-attack-testing-workshop/08Goodwill.pdf`, 2011.

[11] Goubin, L., and Patarin, J. DES and differential power analysis (the "duplication" method). In *Cryptographic Hardware and Embedded Systems - CHES'99* (1999), Ç. K. Koç and C. Paar, Eds., vol. 1717 of *LNCS*, Springer, pp. 158–172.

[12] Gross, H., Mangard, S., and Korak, T. Domain-oriented masking: Compact masked hardware implementations with arbitrary protection order. In *Theory of Implementation Security - TIS@CCS 2016* (2016), B. Bilgin, S. Nikova, and V. Rijmen, Eds., ACM, p. 3.

[13] Gross, H., Stoffelen, K., Meyer, L. D., Krenn, M., and Mangard, S. First-order masking with only two random bits. In *Theory of Implementation Security - TIS@CCS 2019* (2019), B. Bilgin, S. Petkova-Nikova, and V. Rijmen, Eds., ACM, pp. 10–23.

[14] 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 *LNCS*, Springer, pp. 463–481.

[15] Kocher, P. C., Jaffe, J., and Jun, B. Differential power analysis. In *Advances in Cryptology - CRYPTO '99* (1999), M. J. Wiener, Ed., vol. 1666 of *LNCS*, Springer, pp. 388–397.

[16] Mangard, S., Popp, T., and Gammel, B. M. Side-channel leakage of masked CMOS gates. In *Topics in Cryptology - CT-RSA 2005* (2005), A. Menezes, Ed., vol. 3376 of *LNCS*, Springer, pp. 351–365.

[17] Müller, N., Moos, T., and Moradi, A. Low-latency hardware masking of PRINCE. In *Constructive Side-Channel Analysis and Secure Design - 12th International Workshop, COSADE 2021, Lugano, Switzerland, October 25-27, 2021, Proceedings* (2021), S. Bhasin and F. D. Santis, Eds., vol. 12910 of *Lecture Notes in Computer Science*, Springer, pp. 148–167.

[18] NANGATE. The nangate 45nm open cell library. `https://www.nangate.com`.

[19] Nikova, S., Rechberger, C., and Rijmen, V. Threshold implementations against side-channel attacks and glitches. In *Information and Communications Security - ICICS 2006* (2006), P. Ning, S. Qing, and N. Li, Eds., vol. 4307 of *LNCS*, Springer, pp. 529–545.

[20] Sasdrich, P., and Hutter, M. Protecting triple-des against DPA - A practical application of domain-oriented masking. In *Constructive Side-Channel Analysis and Secure Design - COSADE 2018* (2018), J. Fan and B. Gierlichs, Eds., vol. 10815 of *LNCS*, Springer, pp. 207–226.

[21] Trichina, E. Combinational Logic Design for AES SubByte Transformation on Masked Data. `http://eprint.iacr.org/2003/236`, 2003.

Chapter 8

Time Sharing - A Novel Approach to Low-Latency Masking

Publication Data

Dilip Kumar S.V., Siemen Dhooghe, Josep Balasch, Benedikt Gierlichs, and Ingrid Verbauwhede, "Time Sharing – A Novel Approach to Low-Latency Masking," *IACR Transactions on Cryptographic Hardware and Embedded Systems*, 2024(3), pp. 249–272. https://doi.org/10.46586/tches.v2024.i3.249-272.

Contributions

Principal author.

Verification and Security Scope

Verification with SILVER. In this chapter, "formal verification" means automated checks with SILVER for PINI/SNI under the robust probing model. As a scheme, TSM is proven first-order glitch-extended PINI (Sec. 3.3). For

implementation-level instances (e.g., the PRINCE S-Box), we used SILVER; we provide no pen-and-paper proof for these gadgets. We did not use SILVER on large composite circuits such as a full AES S-Box. These implementation results are tool-based evidence only; they are *not* formal proofs and do *not* constitute a formal verification of the full designs.

Scope under transitions (round-based PRINCE). Our proof in Sec. 3.3 establishes first-order security of TSM in the glitch-extended probing model. The round-based PRINCE implementation in Sec. 5.2 is built with one round per cycle using the internal registers of the TSM S-Boxes as the only registers of the design. While this construction is secure under the glitch-extended model, we do not claim security in the glitch+transition setting, since that would require conformity with the stronger OPINI notion. Accordingly, all claims in this chapter are restricted to the glitch-extended probing model.

Time Sharing - A Novel Approach to Low-Latency Masking

Dilip Kumar S. V.[1], Siemen Dhooghe[1], Josep Balasch[2],
Benedikt Gierlichs[1], Ingrid Verbauwhede[1]

[1]COSIC, KU Leuven, Belgium
[2]e-Media Research Lab, STADIUS, KU Leuven, Belgium

Abstract. We present a novel approach to small area and low-latency first-order masking in hardware. The core idea is to separate the processing of shares in time in order to achieve non-completeness. Resulting circuits are proven first-order glitch-extended PINI secure. This means the method can be straightforwardly applied to mask arbitrary functions without constraints which the designer must take care of. Furthermore we show that an implementation can benefit from optimization through EDA tools without sacrificing security. We provide concrete results of several case studies. Our low-latency implementation of a complete PRINCE core shows a 32% area improvement (44% with optimization) over the state-of-the-art. Our PRINCE S-Box passes formal verification with a tool and the complete core on FPGA shows no first-order leakage in TVLA with 100 million traces. Our low-latency implementation of the AES S-Box costs roughly one third (one quarter with optimization) of the area of state-of-the-art implementations. It shows no first-order leakage in TVLA with 250 million traces.

1 Introduction

Implementing secure cryptographic algorithms in a computer system without compromising their promised security has always been challenging. Early research demonstrating the vulnerabilities of cryptographic implementations by Kocher *et al.* [21] showed that it is possible to find the secrets that a computer processes by monitoring its execution time and thereby highlighted the need to build secure implementations. This led to the consolidation of side-channel analysis as a field of study that attempts to gain information from the implementation of a chip or computer system by monitoring its physical effects rather than exploiting a weakness of the implemented algorithm. Along with timing analysis [21], power analysis [22] and electromagnetic analysis [12, 31] represent some of the best-known side-channel attacks. Power analysis, in

particular, is perhaps the most popular due to its low setup cost, non-invasive nature, and devastating effectiveness.

In the past few decades, there has been a great deal of research on securing cryptographic implementations against side-channel attacks. Chari *et al.* [9] as well as Goubin and Patarin [14] independently proposed a generic countermeasure called masking that splits the data being processed into random shares to thwart power analysis attacks. The idea behind the countermeasure is to eliminate the correlation between the secret data and the data being processed, since the device's power consumption depends on the latter, which is now random. But processing multiple shares also comes with overheads in terms of implementation area, execution time, online randomness, etc. Along with securing implementations, it has also been important to minimize these overheads. Masking proved to be quite successful for securing software implementations, but it was later found that masked hardware implementations still leak information about the secrets due to glitches [24]. Several modern masking techniques such as Threshold Implementations (TI) [29], Consolidating Masking Schemes (CMS) [33], and Domain Oriented Masking (DOM) [16] were proposed to securely mask hardware in the presence of glitches. They were quite successful in creating secure and efficient hardware implementations with low area and randomness usage. Overhead reductions are typically achieved by decomposing complex non-linear functions, such as an S-Box, into smaller sub-circuits with low algebraic degrees that can be masked efficiently. The composition of the sub-circuits requires careful use of register stages to prevent glitch propagation and re-masking intermediate values to maintain uniformity.

Due to the recent advent of IoT devices, which are very accessible to an attacker, there is a need for embedded real-time applications to have fast data processing, such as memory encryption, to ensure security. As a consequence, there is a new motivation to design masking schemes suitable for low-latency implementations. One of the first generic approaches called GLM was proposed by Groß *et al.* and was used to design low-latency S-Boxes in [15]. GLM, built upon DOM, reduces latency by eliminating register stages required for share compression after non-linear operations. Skipping share compression exponentially increases the share count after every non-linear operation, drastically increasing the overall area and randomness utilization. This especially makes the approach impractical for masking large functions such as higher-degree S-Boxes. Other research into low-latency masking includes LLTI [1] based on TI and other methods involving asynchronous circuits [25, 27].

Although masking techniques are typically proven secure in the t-probing model [17], most are not generic and are not trivial to compose with other design elements. In other words, converting any unprotected circuit to a protected one is not straightforward and is usually laborious. Recently in [8],

Cassiers *et al.* introduced a new security notion called Probe Isolating Non-Interference (PINI), which allows for trivial composition. Any PINI gadget is directly composable with other (linear and non-linear) PINI gadgets, without significant overheads. In [6], the authors propose two small multiplication gadgets called HPC1 and HPC2 that can be composed in the glitch-extended probing model, introduced by Faust *et al.* [11], to create more complex circuits. Later in [18], Knichel *et al.* proposed the HPC3 gadget specifically intended for low-latency applications. Although one can build any circuit with these gadgets, the latency of the circuit grows with the algebraic degree of the function. To the best of our knowledge, there exists only one algorithm-level approach (which does not simply compose elementary gadgets) to generate first-order PINI secure circuits, namely GHPC [20]. Despite being PINI secure, their low latency version GHPC_{LL} also suffers from the high area and randomness overheads for larger functions, like GLM. A single-cycle AES S-Box using GHPC_{LL} costs 64.1 kGEs and 2048 bits of randomness, similar to the cost of GLM. But GHPC_{LL} has the advantage that it is proven to be composable secure while GLM is not.

Contributions. We present a new masking method for low-latency applications that is first-order PINI composable secure, and - more importantly - brings substantially less overhead than other composable low-latency masking schemes. Our contributions are the following:

- We present a masking method that secures any function against first-order attacks and uses only a single register stage, thus executes in a single clock cycle.
- We provide a formal description and follow up with a proof that shows any circuit secured by our approach is first-order glitch-extended PINI secure.
- Compared to previously published algorithm-level approaches such as GLM [15] and GHPC [20] that implement single-cycle Boolean functions, our method shows a substantial improvement both in terms of area as well as online randomness required, for realistically complex circuits.
- We apply our proposed method to produce a masked first-order secure PRINCE implementation that executes in one cycle per round and show the improvements in the utilization results. We demonstrate the security of our PRINCE S-Box with a formal verification tool and show that the complete PRINCE core on FPGA exhibits no first-order leakage in TVLA with 100 million traces.
- We apply our proposed method to mask a more complex function, *i.e.* the AES S-Box, in order to demonstrate its potential for efficient

implementations. We show significant improvements in utilization costs and demonstrate that our method scales well especially when masking larger functions. Our AES S-Box on FPGA shows no first-order leakage in TVLA with 250 million traces.

Outline. The remainder of this article is organized as follows. In Section 2, we briefly introduce the notation and recall relevant background. In Section 3, we start explaining our idea with a toy example and design a simple masked AND gate. Next, we detail our method with a formal description which can then be applied to any Boolean function. We prove that our method is first-order PINI glitch-extended composable secure. In Section 4 we discuss advantages of our approach. In Section 5, we present our first case study by applying our method to produce a masked, first-order secure, low-latency PRINCE core. In Section 6, we present our second case study by applying our method to a more complex and challenging function, the AES S-Box. In Section 7, we verify through leakage assessment tests that our theoretical proof does indeed translate to practically secure implementations of a complete PRINCE core and an AES S-Box. Finally, Section 8 presents our conclusions and avenues for future work.

2 Preliminaries

In this section we briefly introduce the notation and recall relevant background.

2.1 Notation

Boolean masking splits each bit $x \in \mathbb{F}_2$ into n uniform random shares x_i such that $x = x_0 \oplus \ldots \oplus x_{n-1}$. The storage of a variable in a register is denoted by curly brackets $\{\cdot\}$.

2.2 Probing Model

In the probing model, introduced by Ishai, Sahai, and Wagner [17], an adversary $\mathcal{A}$ is allowed to observe a set of at most t (predefined) wires of a circuit at each execution of the masking. The security of a given implementation is proven by showing that a simulator $\mathcal{S}$ can perfectly simulate any set of at most t probes without any knowledge of the input shares $(x_0, \ldots, x_{n-1})$. A circuit ensuring this condition for any set of size t is said to be t-probing secure.

The probing model provides a way to prove the security of a circuit. However, this algorithmic circuit does not trivially map to practice where, due to leakage effects, an adversary can gain more information than what a probe typically captures. The main leakage effect to be considered is that of glitches. In this work, we model glitches by bundling groups of wires over which a glitch could carry information from one wire to another. Whereas one of the adversary's probes normally results in the value of a single wire, a glitch-extended probe allows obtaining the values of all wires in a bundle. This extension of the probing model has been discussed in the work of Reparaz et al. [33] and formalized by Faust et al. [11]. 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."

2.3 Related Work

Due to the growing interest in low-latency masking, several masking techniques have been developed in recent years which are specifically focused on reducing latency. Some techniques relevant to our work are GLM [15], GHPC [20] and LMDPL [35]. Among these, the constructions for GHPC and GLM are most comparable to our technique, while LMDPL employs a distinct dual-rail precharge logic.

GLM is a low-latency masking approach proposed by Groß *et al.* that can be applied to protect any security-sensitive circuit [15]. GLM is based on the Domain Oriented-Masking (DOM) scheme [16], which was introduced to create low-area and low randomness designs. DOM is a gate-level masking technique that uses masked AND gates (DOM multipliers) to build and secure more complex circuits. A masked circuit built with DOM is split into independent circuits called "domains" based on the share index of variables. Non-linear operations compute on all shares of a variable requiring communication between domains and are called "cross-domain terms". For a secure computation, the cross-domain terms are refreshed and stored in registers before they are compressed and merged with inner-domain terms to limit the number of shares and to reduce area.

While DOM optimizes for area and randomness, GLM trades area and randomness for reduced latency. The register stages in DOM multipliers increase latency in a design. GLM reduces latency by eliminating these stages. However, constructing a low-latency circuit by eliminating these register stages introduces complications, leading to increased area and randomness requirements for the circuit. First, the number of output shares increases after every non-linear

operation as there is no share compression due to the lack of register stages. The cross-domain terms cannot be merged with the inner-domain terms, increasing the number of shares. Furthermore, as the non-linear logic depth increases, the number of shares of the intermediate values in the circuit also increases exponentially, resulting in a significant increase in area. Second, removing the registers causes the circuit to be susceptible to variable collisions. GLM requires the inputs to non-linear gates to be independently shared. If the circuit violates this condition, the colliding variables must be duplicated with multiple shared instances of the same variable with independent sharings. To resolve collisions, it might also be necessary to duplicate the entire fan-in circuitry causing the collision. All of these fixes increase the area and randomness overhead of the circuit. As a final step, secure share compression is performed by refreshing the shares with randomness and storing them in registers before the compression, which also increases the cost of area and randomness since many shares need to be refreshed and stored.

GHPC, introduced by Knichel *et al.* [20], is a low-latency masking technique that uses Shannon Decomposition to transform arbitrary Boolean functions into secure PINI composable gadgets. For simplicity, we will illustrate the technique by applying it to a 4×4 function, $F(x, y, z, w) : \mathbb{F}_2^4 \to \mathbb{F}_2^4$. The technique is applied independently to each coordinate function, decomposing it into cofactors by fixing the input shares within a single share domain. Consider $f(x_0{+}x_1, y_0{+}y_1, z_0{+}z_1, w_0{+}w_1) : \mathbb{F}_2^4 \to \mathbb{F}_2$ as the shared representation of one of the four coordinate functions of F, where the subscript denotes the share domain of the inputs. The function f is decomposed into 16 cofactors by considering all combinations of inputs from the second share domain, *i.e.* $\{x_1, y_1, z_1, w_1\} \in \{0,1\}^4$. For example, if $\{x_1, y_1, z_1, w_1\} = \{0, 1, 1, 0\}$, then the corresponding cofactor would be $f(x_0, \overline{y_0}, \overline{z_0}, w_0)$. The resulting Shannon cofactors only depend on the inputs from the first share domain, *i.e.* $\{x_0, y_0, z_0, w_0\}$. A secure implementation of F with GHPC necessitates two register layers. A secure low-latency implementation of F with GHPC_{LL} reduces this to one register layer at the cost of more randomness. For each coordinate function, in the first phase, the 16 cofactors are calculated, refreshed, and registered using inputs from the first share domain. In the second phase, the inputs from the second share domain serve as selection bits to choose the correct cofactor out of the sixteen for output. In GHPC, the number of cofactors, registers, and randomness required is determined by the number of inputs and not the algebraic degree of the function.

3 Time Sharing Masking

We introduce our novel approach to securely first-order mask any (vectorial) Boolean function in hardware with a single register layer. We will refer to it using the acronym TSM, short for Time Sharing Masking, in the remainder of the paper. We begin the explanation with a toy example, applying TSM to a single AND gate, in Section 3.1. In Section 3.2, we write out TSM formally so it applies to any Boolean function, in particular also vectorial Boolean functions. Finally, in Section 3.3, we prove that TSM is first-order glitch-extended PINI composable secure.

3.1 Preliminary Example

Let x and y be the two inputs of the AND gate that computes $z = x \cdot y$. And let $(x_0, x_1), (y_0, y_1)$ be their sharings such that $x = x_0 + x_1$ and $y = y_0 + y_1$. A key aspect of TSM is to separate *in time* the processing of $share_0$ inputs from the processing of $share_1$ inputs with the help of a register layer, see Figure 1. Before the computation begins, we refresh the inputs with two random bits r_3, r_4 for the masked AND gate to be composable secure (see Section 3.3):

$$x'_0 = x_0 + r_3, \qquad x'_1 = x_1 + r_3, \qquad y'_0 = y_0 + r_4, \qquad y'_1 = y_1 + r_4.$$

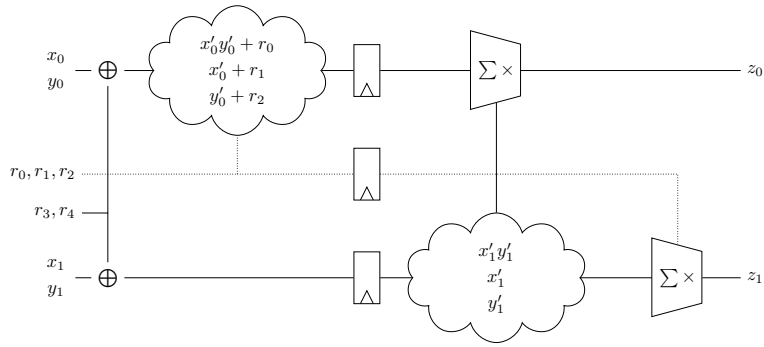


Figure 1: Application of TSM to a single AND gate.

Then, in the first phase, all cross product combinations of $share_0$, *i.e.*, $(x'_0, y'_0, x'_0y'_0)$, are computed, refreshed, and stored in registers. In the second phase, all cross product combinations of $share_1$, *i.e.*, $(x'_1, y'_1, x'_1y'_1)$, are computed. Finally, in order to produce the output of the AND gate, products of masked combinations of $share_0$ and combinations of $share_1$ are summed up, see Eq. (1).

$$
\begin{aligned}
z_0 &= \left\{x'_0 y'_0 + r_0\right\} + \left\{x'_0 + r_1\right\} \cdot \left\{y'_1\right\} + \left\{y'_0 + r_2\right\} \cdot \left\{x'_1\right\} \\
z_1 &= \left\{r_0\right\} + \left\{r_1\right\} \cdot \left\{y'_1\right\} + \left\{r_2\right\} \cdot \left\{x'_1\right\} + \left\{x'_1\right\} \cdot \left\{y'_1\right\}
\end{aligned}
\tag{1}
$$

The computation requires five fresh random bits and eight registers to store the intermediate shares. The area and randomness utilization for computing a single AND gate is high, but we use this toy example only for illustration. TSM should be mainly applied to mask more complex non-linear functions as a whole, and not individual AND gates. The advantages of this approach will become more apparent in the remainder of the paper.

3.2 Formal Description

We provide a description of TSM working on an arbitrary (vectorial) Boolean function. The outline is presented in Figure 2.

Specifically in this section, we change the notation to denote bits in a word by square brackets ($x[0], ..., x[k-1]$) instead of using different letters (*e.g.*, x, y in the previous section). We denote $x \in \mathbb{F}_2^k$ a k-bit word where its two-share Boolean masking is denoted by $\bar{x} = (x_0, x_1) \in \mathbb{F}_2^{2k}$ such that $x_0 + x_1 = x$ with $x_0 = (x_0[0], ..., x_0[k-1])$ and $x_1 = (x_1[0], ..., x_1[k-1])$ the notation of the share-words in separate bits. This change allows a simpler, more compact presentation of what follows next.

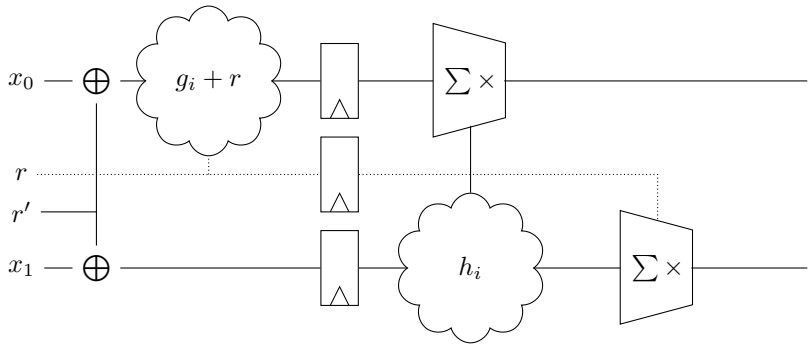


Figure 2: Application of TSM to an arbitrary (vectorial) Boolean function described by the functions g_i and h_i.

We explain the TSM method in a constructive manner where we first rewrite in Eq. (2) the algebraic normal form of a shared function as the sum of non-complete terms where each term is the multiplication between share domain 0 and share domain 1. We then rewrite this equation to Eq. (3) by adding fresh randomness allowing us to safely form the two output shares outlined in Eq. (4). Finally, the inputs of the gadget are first re-masked to ensure composable security.

We start informally, where we first rewrite the equations of a shared monomial. Namely, note that for the product of the bits $x[j]$ for some set of indices $j \in J \subset \{0, ..., k-1\}$

$$\begin{aligned}\prod_{j\in J} x[j] &= \prod_{j\in J}(x_0[j] + x_1[j]) = \sum_{i\in\mathbb{F}_2^k}\prod_{j\in J} x_{i[j]}[j] \\ &= \sum_{i\in\mathbb{F}_2^k}\prod_{j\in J \text{ s.t. } i[j]=0} x_0[j] \prod_{j\in J \text{ s.t. } i[j]=1} x_1[j]\,.\end{aligned}$$

In words, each monomial can be split as the sum of non-complete terms and each of these terms can be split as the multiplication of shares from domain 0 and shares from domain 1.

Consider an arbitrary Boolean function

$$f : \mathbb{F}_2^k \to \mathbb{F}_2 : x = (x[0], ..., x[k-1]) \mapsto f(x[0], .., x[k-1])\,.$$

We denote its two-share masking by $\bar{F} : \mathbb{F}_2^{2k} \to \mathbb{F}_2^2 : \bar{x} \mapsto (F_0(\bar{x}), F_1(\bar{x}))$ such that $F_0(\bar{x}) + F_1(\bar{x}) = f(x_0 + x_1)$. The above insight can be applied to each monomial in the algebraic normal form of f. We thus say that there exist functions g_i and h_i such that

$$f(x_0 + x_1) = \sum_{I\in\mathcal{P}_k} g_{\pi(I)}((x_0[i])_{i\in I}) h_{\pi(I)}((x_1[i])_{i\in\Omega/I})\,, \tag{2}$$

where we denote $(x_0[i])_{i\in I}$ as the set of all bits $x_0[i]$ for i in I. We also denote by $\mathcal{P}_k$ the power set of the indices $\Omega = \{0, ..., k-1\}$, namely all possible sets of indices in Ω. It is clear that $|\mathcal{P}_k| = 2^k$. The sets in $\mathcal{P}_k$ are numbered and indicated by the function π.

The functions g_i, h_i in Eq. (2) work only on share domain 0 and 1, respectively. To go back to the masked AND gate example from Section 3.1, the functions g_i, h_i are the following

$$g_0(x_0, y_0) = x_0 y_0 \qquad g_1(x_0) = x_0 \qquad g_2(y_0) = y_0 \qquad g_3(\emptyset) = 1$$

$$h_0(\emptyset) = 1 \qquad h_1(y_1) = y_1 \qquad h_2(x_1) = x_1 \qquad h_3(x_1, y_1) = x_1 y_1 \,.$$

Multiplying and adding the above terms, for $\mathcal{P}_2 = ((0,1),(0),(1),\emptyset)$, we get

$$\begin{aligned} xy &= \sum_{I \in \mathcal{P}_2} g_{\pi(I)}((x_0[i])_{i\in I}) h_{\pi(I)}((x_1[i])_{i \in \Omega/I}) \\ &= g_0(x_0, y_0)h_0(\emptyset) + g_1(x_0)h_1(y_1) + g_2(y_0)h_2(x_1) + g_3(\emptyset)h_3(x_1, y_1) \\ &= x_0y_0 + x_0y_1 + x_1y_0 + x_1y_1 \,. \end{aligned}$$

The above sharing is already correct, however, it misses randomness for its security.

We thus further adapt Eq. (2) by adding randomness. Namely, by adding 2^k random bits $r = (r_0, ..., r_{2^k-1})$, we get

$$\begin{aligned} f(x_0 + x_1) = &\sum_{I \in \mathcal{P}_k} (g_{\pi(I)}((x_0[i])_{i\in I}) + r_{\pi(I)}) h_{\pi(I)}((x_1[i])_{i \in \Omega/I}) \\ &+ \sum_{I \in \mathcal{P}_k} r_{\pi(I)} h_{\pi(I)}((x_1[i])_{i \in \Omega/I}) \,. \end{aligned} \tag{3}$$

By re-masking, we can split the computation in two parts (read two phases), the computation and refreshing of $g_i(\cdot)$ on the first shares, and the computation and recombination of $h_i(\cdot)$ on the second shares. This is also depicted in Figure 2. In this figure, we also observe that the shares x_0, x_1 are first refreshed with the randomness $r' \in \mathbb{F}_2^k$. This is done in order to make TSM composable secure as proven in Section 3.3.

Finally, the two shares $F_0(\bar{x}), F_1(\bar{x})$ are composed as follows

$$\begin{aligned} F_0(\bar{x}) = &\sum_{I \in \mathcal{P}_k} (g_{\pi(I)}((x_0[i])_{i\in I}) + r_{\pi(I)}) h_{\pi(I)}((x_1[i])_{i \in \Omega/I}) \\ F_1(\bar{x}) = &\sum_{I \in \mathcal{P}_k} r_{\pi(I)} h_{\pi(I)}((x_1[i])_{i \in \Omega/I}) \,. \end{aligned} \tag{4}$$

Since the functions g_i and h_i (or their product) consist of all terms up to degree k, any Boolean function can be made from these g_i and h_i. This is extended for vectorial functions ($\mathbb{F}_2^k \to \mathbb{F}_2^\ell$) by re-using the g_i and h_i functions for each coordinate function. While this can make the output shares non-uniform (in the extreme case, two coordinate functions are equal), the security comes from the register layer being filled with uniquely re-masked values and from each function working only on one share domain at a time. This is made formal in the next section where we show that TSM is PINI composable secure.

3.3 Security

We prove that TSM is first-order probing secure and that it is, moreover, composable first-order secure in the Probe-Isolating Non-Interference (PINI) framework by Cassiers *et al.* [8]. Namely, we show that any circuit secured with TSM allows for trivial composition. Since TSM is designed to work over hardware, we use the glitch-extended probing model by Faust *et al.* [11] to extend the PINI framework into the glitch-extended PINI framework. This PINI security is particularly important since it allows for the composition between gadgets without the need to place additional registers between them. Since all maskings of linear layers (where the linear function is applied share-wise) are PINI secure, we can trivially secure linear functions without adding additional registers or additional randomness.

Before starting the proof that the approach delivers PINI secure solutions, we need to introduce the necessary concepts to introduce PINI security. We start by providing the notion of simulation.

Definition 1 (Simulatability [8])**.** Let $P = \{p_1, ..., p_\ell\}$ be a set of ℓ probes of a gadget C and C_P the tuple of values of the probes for an execution of C. Let $I = \{(i_1, j_1), ..., (i_k, j_k)\} \subset \{0, ..., d-1\} \times \{0, ..., m-1\}$ be a set of input wires of C. A simulator is a randomized function $\mathcal{S} : \mathbb{F}_q^k \to \mathbb{F}_q^\ell$. The set of probes P can be simulated with the set of input wires I if there exists a simulator $\mathcal{S}$ such that for any inputs $x_{*,*}$, the distributions $C_P(x_{*,*})$ and $\mathcal{S}(x_{i_1,j_1}, ..., x_{i_k,j_k})$ are equal, where the probability is over the random coins in C and $\mathcal{S}$.

The above definition defines the security game in terms of a simulation game. This framework is extended to PINI security where we define which information is given to the simulator.

Definition 2 (PINI [8])**.** Let G be a gadget over d shares and P a set of t_0 (glitch-extended) probes on wires of G (called internal probes). Let A be a set of t_1 share indices. G is t-PINI if for all P and A such that $t_0 + t_1 \leq t$, there exists a set B of at most t_0 share indices such that probes on the set of wires $P \cup y_{A,*}$ can be simulated with the wires $x_{A \cup B}$, with $x_{i,*}$ denoting all inputs of share i and $y_{i,*}$ denoting all outputs of share i.

Given the above definition of PINI, we show that any circuit secured by the TSM method from Section 3 is composable secure. Intuitively, the reason TSM is composable secure is due to each registered value (in the single register stage of the method) being re-masked by unique randomness.

Theorem 1. *Any circuit secured by TSM (Section 3) is first-order glitch-extended PINI.*

Proof. Denoting the k-bit input shares x_i and the output shares y_i. Looking at Definition 2 for $t = 1$ (considering glitch-extended probes), we find that we need to prove two cases. Namely,

- for $t_0 = 0$ and $t_1 = 1$, in which case we need to show that the output shares y_i can be simulated using the input shares x_i for $i \in \{0, 1\}$.
- for $t_0 = 1$ and $t_1 = 0$, in which case we need to show that a single intermediate probe can be simulated using either x_0 or x_1.

We begin with the first case, we have to prove that y_i can be simulated using x_i. We split up the proof depending on i.

- For y_0, the output is calculated from the values g_i re-masked by r and by values h_i which operate on the second shares x_1 re-masked by r'. Since g_i is re-masked by r and x_1 is re-masked by r', a simulator can sample r and r' and perfectly simulate the values y_0 as uniform randomness (in particular, the simulator does not need the values x_0).
- For y_1, since it is created using only x_1 and randomness r', a simulator can perfectly simulate y_1 given x_1 and by uniformly random sampling r' (in fact, due to r' the simulation would also work from scratch in which case the simulator can simulate the probed values as uniform randomness).

For the proof of the second case where we simulate an intermediate probe, we consider only probes in the first phase of the circuit, since probes on the second phase were already considered in the previous case. However, for probes on the first phase, it is clear that these can be perfectly simulated since the computation is done share-wise (a probe either only sees values from x_0 or from x_1) in which case the simulator simply gets either the zero or the one shares and performs the computation following the algorithm.

Since both cases are proven, the masking is first-order glitch-extended PINI. □

As a result, since the TSM circuit is first-order glitch-extended PINI secure, it can be composed with any other PINI gadget (which includes all linear operations too) without adding extra register stages or randomness following the proofs of composable security from the original work [8].

4 Advantages of TSM

In this section we mainly outline the general efficiency of TSM and contrast it with the first-order case of GLM and GHPC_{LL}. In general, we emphasize the advantages of TSM through a comparison of area, considering both the number of registers and logic, as well as the randomness required for masking a $\mathbb{F}_2^k \to \mathbb{F}_2^k$ function with an algebraic degree of $k-1$ (which represents the highest algebraic degree for a k-bit permutation).

4.1 Registers and Randomness Cost

In Section 3.2, we described our approach by applying it to an arbitrary Boolean function $f : \mathbb{F}_2^k \to \mathbb{F}_2$. The Boolean function is computed as a combination of the functions g_i and h_i. Importantly, the functions g_i and h_i solely depend on the k shared inputs. We emphasize that TSM can be extended to vectorial Boolean functions with multiple outputs ($\mathbb{F}_2^k \to \mathbb{F}_2^k$) because all coordinate functions share the same k inputs. The intermediate registers which store the refreshed results of the g_i functions, the random bits, and the second share inputs can be commonly used to calculate the shared outputs of all coordinate functions without increasing the register and randomness cost. In other words, the number of intermediate registers and randomness remains constant irrespective of the number of outputs.

TSM requires at most $2^k - 2$ registers to store the results of the g_i functions since $|\mathcal{P}_k| = 2^k$ and there is no degree k term (removing one register) and we do not store a constant term (removing the second register). TSM then requires at most another $2^k - 2$ registers to store the random bits r. Finally, TSM requires k registers to store the second share inputs. This gives a total of at most $2^{k+1} + k - 4$ registers. Similarly, for the randomness, TSM requires at most $2^k - 2$ bits to refresh the g_i functions and another k bits for r' in Figure 2.

We compare these numbers with GLM and GHPC_{LL} in Table 1. We note that both TSM and GLM can be more efficient than what is reported in the table, depending on the function to which the method is applied. Namely, we report the *worst case metrics* such that any function of degree $k-1$ can be implemented with the given register and randomness costs.

We observe roughly a factor $k/2$ improvement in the number of registers and a factor k in random bits over both GLM and GHPC_{LL}. As previously mentioned, this improvement is a direct result of re-using the registered values for each coordinate function (of the k outputs).

Name	# Registers	# Register Layers	# Random Bits
TSM	$2^{k+1}+k-4$	1	2^k+k-2
GLM [15]	$2^k k$	1	$2^k k$
GHPC_{LL} [20]	$2^k k+k$	1	$2^k k$

Table 1: Comparison for a $\mathbb{F}_2^k \to \mathbb{F}_2^k$ function of algebraic degree $k-1$.

If we compare TSM with GLM for masking an AES S-Box, we see significant, concrete savings in registers and random bits when implementing a higher algebraic degree function with many outputs. Groß *et al.* [15] report the cost of masking an AES S-Box with a single register layer to be $16 \cdot 2^7 (= 2048)$ registers and $16 \cdot 2^7 (= 2048)$ random bits. To compare these numbers with TSM, we fill in the value $k = 8$ in Table 1. TSM requires only 516 registers and only 262 random bits per AES S-Box.

In Section 6, we discuss the implementation of the AES S-Box with TSM in more detail and provide concrete numbers for the area cost, including combinational logic.

4.2 Combinational logic

Without loss of generality, let us consider the PRINCE S-box, a 4×4 function $S : \mathbb{F}_2^4 \to \mathbb{F}_2^4$ for illustration. We show the equations for the function S in Algebraic Normal Form (ANF) in Eq. (5). We denote (a, b, c, d) as the four input bits and f^0, f^1, f^2, f^3 as the coordinate functions which produce the four output bits.

$$\begin{aligned} f^0 &= 1 + dc + b + cb + dcb + a + da + ba, \\ f^1 &= 1 + db + cb + dcb + ca + cba, \\ f^2 &= d + dc + a + da + ca + dca + cba, \\ f^3 &= 1 + c + cb + dcb + a + dca + ba + dba. \end{aligned} \tag{5}$$

Before delving into the benefit of TSM, let us briefly discuss how the function S would be masked using GLM. In the first stage, every coordinate f^0, f^1, f^2, f^3 is split into eight share domains. Cubic terms, such as bcd, are split into eight shared multiplication terms, $b_0c_0d_0, b_0c_0d_1, ..., b_1c_1d_1$. One multiplication term is assigned to each of the eight shared domains of the coordinate functions. Quadratic terms, such as bc, are split into four shared multiplication terms $b_0c_0, b_0c_1, b_1c_0, b_1c_1$ and are distributed among four of the eight shared domains of the coordinate functions. The share domains are then refreshed with fresh randomness and are registered. In the second stage, share compression is

performed to reduce the number of shares from eight to two. In summary, the first stage involves expanding the number of shares, followed by the second stage, where the shares are compressed. On the other hand, to mask the function S, GHPC applies "Shannon decomposition", an identity that splits any Boolean function into parts called cofactors, to each coordinate function f^0, f^1, f^2, f^3. The coordinate functions are independently expanded into 16 cofactors by setting one share of the inputs a,b,c, and d to either 0 or 1. A common characteristic between GLM and GHPC, which may be regarded as a potential drawback, is that every coordinate function is treated as a separate entity even though they commonly share the same inputs.

Applying TSM to the function S, in the first stage, all inputs are remasked, then all cross-products of the $share_0$ inputs are computed, *i.e.*, $(a_0, b_0, a_0b_0, ..., b_0c_0d_0)$, and finally those are refreshed and stored in the register layer. In the second stage, the cross-products of the $share_1$ inputs are computed, and they are then multiplied and summed with the masked cross-products of the $share_0$ inputs to produce the outputs of the coordinate functions.

TSM allows to reduce the cost of combinatorial logic by efficient reuse in several ways. First, we can deduplicate identical terms across coordinate functions, i.e. compute them only once and then reuse them. For example, dc is needed to compute f_0 and f_2 but there is no need to compute dc twice. Overall this allows to reduce the number of distinct terms to compute from 20 to 14. The decrease in logic becomes more prominent with an increase of the number of coordinate functions. This also reduces the number of random bits needed for refreshing in phase 1, and the number of registers.

Second, we can reuse already computed lower degree terms to compute higher degree terms. For example, we can compute dc and reuse it for computing dcb. Eq. (6) shows the sharing of the coordinate function f^0 and Eq. 7 shows in the first three lines the straightforward computation for $(b)_0$, $(cb)_0$ and $(dcb)_0$. In the fourth line it shows a more efficient computation of $(dcb)_0$ by reusing the already computed $(cb)_0$ which results in a reduction of the number of logic gates, thereby lowering the area. The decrease in logic gates becomes more prominent with an increase in the algebraic degree of the terms, particularly when masking higher algebraic degree functions such as the AES S-Box.

$$\begin{aligned} f_0^0 &= 1 + (dc)_0 + (b)_0 + (cb)_0 + (dcb)_0 + (a)_0 + (da)_0 + (ba)_0, \\ f_1^0 &= (dc)_1 + (b)_1 + (cb)_1 + (dcb)_1 + (a)_1 + (da)_1 + (ba)_1. \end{aligned} \tag{6}$$

$$
\begin{aligned}
(b)_0 &= \left\{b'_0 + r_1\right\} \\
(bc)_0 &= \left\{b'_0c'_0 + r_7\right\} + \left\{c'_0 + r_2\right\} \cdot \left\{b'_1\right\} + \left\{b'_0 + r_1\right\} \cdot \left\{c'_1\right\} \\
(bcd)_0 =& \left\{b'_0c'_0d'_0 + r_{13}\right\} + \left\{b'_0c'_0 + r_7\right\} \cdot \left\{d'_1\right\} + \left\{b'_0d'_0 + r_8\right\} \cdot \left\{c'_1\right\} \\
&+ \left\{c'_0d'_0 + r_9\right\} \cdot \left\{b'_1\right\} + \left\{b'_0 + r_1\right\} \cdot \left\{c'_1\right\} \cdot \left\{d'_1\right\} \\
&+ \left\{d'_0 + r_3\right\} \cdot \left\{b'_1\right\} \cdot \left\{c'_1\right\} + \left\{c'_0 + r_2\right\} \cdot \left\{d'_1\right\} \cdot \left\{b'_1\right\} \\
(bcd)_0 =& (bc)_0 \cdot \left\{d'_1\right\} + \left\{b'_0c'_0d'_0 + r_{13}\right\} + \left\{b'_0d'_0 + r_8\right\} \cdot \left\{c'_1\right\} \\
&+ \left\{c'_0d'_0 + r_9\right\} \cdot \left\{b'_1\right\} \quad + \left\{d'_0 + r_3\right\} \cdot \left\{b'_1\right\} \cdot \left\{c'_1\right\}
\end{aligned}
\tag{7}
$$

Optimization during Logic Synthesis

Importantly, since the combinational logic in phase 1 (before the register layer) and the combinational logic in phase 2 (after the register layer) is non-complete, it is safe to allow (or even, one should enforce) the logic optimization through Electronic Design Automation (EDA) tools, without the need to carefully place logic in distinct modules. The only kind of optimization which must not be allowed is register re-timing, as that may move combinational logic across the register stage which may lead to a violation of non-completeness. Our case studies in the following sections include the impact of logic optimization on area, maximum frequency and security.

5 Case Study: Application of TSM to PRINCE

For our first case study, we chose to implement a masked version of the PRINCE block cipher [3] with TSM as a proof of concept. PRINCE is one of the few ciphers designed primarily for low-latency applications. With the rise of embedded devices with real-time requirements such as fast encryption or decryption, PRINCE fills the void of fast, lightweight block ciphers.

PRINCE is a 64-bit block cipher with a 128-bit key and consists of 12 rounds. The key is split into two 64-bit keys. One half is used to produce whitening keys (k_0, k'_0) and the other half (k_1) is used in $PRINCE_{core}$ for the round key addition, see Figure 3. Each round of PRINCE consists of a key addition, a substitution layer (S, S^{-1}), a linear layer $(SR, SR^{-1}, M, M', M^{-1})$ and the

addition of a round constant $(RC_0, RC_1, \ldots, RC_{11})$. For more details about the cipher, we refer the reader to the original paper [3].

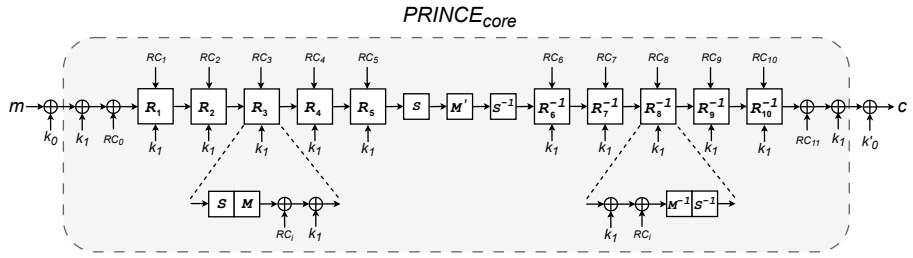


Figure 3: Schematic view of 12 rounds of PRINCE. Regular rounds R_i consist of an S-Box layer (S) and a linear layer M, and inverse rounds R_i^{-1} consist of an inverse S-Box layer (S^{-1}) and an inverse linear layer M^{-1}.

5.1 Masked PRINCE S-Box

The difficulty of masking a cipher implementation typically lies with its non-linear elements. The non-linear substitution layer (S)/inverse substitution layer (S^{-1}) of PRINCE consists of 4-bit S-Boxes of maximum algebraic degree three. We have already introduced the PRINCE S-box in Section 4. Our masked S-Box with a latency of one cycle is shown in Figure 4. Adhering to the guidelines for efficient implementation outlined in Section 4, we re-use the common degree-2 and degree-3 terms across coordinate functions, thus reducing the number of terms to compute, refresh and store from 20 to 14. Additionally, we capitalize on the shared outputs of degree-2 terms to calculate degree-3 terms.

Masking a single PRINCE S-Box with TSM requires 18 fresh random bits (4 for initial remasking and 14 for refreshing) and 32 intermediate registers $(14 + 14 + 4)$. In Table 2, we compare the utilization results of our masked PRINCE S-Box with other relevant PINI secure low-latency designs from the literature.

The synthesis results are gathered using the NanGate 45nm Open Cell Library [28]. We use Synopsys DC Compiler v2021.06 for Synthesis and provide results for two different sets of options. The first set of options (`compile -exact_map`) aims for a direct mapping of our design to logic. The second set of options (`compile_ultra -no_autoungroup -no_boundary_optimization`) aims for maximum logic optimization while making sure that no logic is moved across the register layer, see Section 4.2.

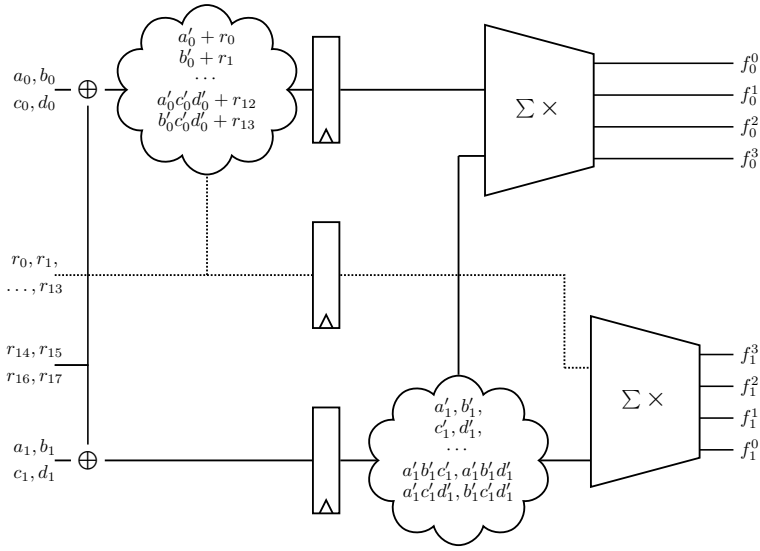


Figure 4: Architecture of our masked PRINCE S-Box.

Although the `compile_ultra` option should not be carelessly used for masked implementations, because typically careful placement of logic in distinct modules is required, it can be safely done with TSM, as long as register retiming is not permitted. Later in Section 5.4 we demonstrate that TSM indeed remains secure with either set of these options, without careful placement of logic in distinct modules, and show that it becomes insecure if register retiming is enabled. The area numbers were gathered by synthesizing our designs with a target frequency of 100 MHz.

Table 2: Utilization results of relevant first-order masked PINI secure PRINCE S-Boxes.

Design	Method	Area (GE)	Rand (bits)	Cycles
This work	**TSM**	**538**[a] / **453**[b]	**18**	**1**
[20]	GHPC_{LL}	987	64	1
[20]	GHPC_{LL}-AND	445	24	2
[20]	GHPC	1384	4	2

[a] `compile -exact_map`
[b] `compile_ultra -no_autoungroup -no_boundary_optimization`

Our single-cycle masked PRINCE S-Box has both lower area as well as randomness cost when compared to the GHPC_{LL} design proposed by Knichel *et*

al. [18]. The other designs need more than one single cycle. The exact sharings of our PRINCE S-Box are given in the HDL source code[1].

5.2 Round-Based Architecture

As this work aims to achieve low latency, the substitution layer consisting of 16 S-Boxes is implemented in parallel. We looked at the existing literature for efficient round-based architectures for PRINCE and chose to implement a first-order secure version of the unprotected architecture presented by Moradi *et al.* [25]. The PRINCE S-Box and its inverse (S^{-1}) are affine equivalents. By carefully constructing the architecture for the cipher, it is possible to minimize the additional circuitry needed to implement the inverse substitution layer. We can use affine transformations and re-use existing S-Box circuitry to compute S^{-1}. The inverse S-Box can be computed by applying input and output transformations to the S-Box as follows: $S^{-1} = A \circ S \circ A$. The input and output transformations applied to the S-Box are identical, A: `5764FDCE1320B98A`.

Our construction for the first-order secure PRINCE implementation using TSM is shown in Figure 5. The optimized single-cycle design requires only one substitution layer and uses the input/output transformation to calculate the inverse substitution layer. This reduces the area of our implementation. The matrix M' is an involution, and the other matrices M and M^{-1} used in the $PRINCE_{core}$ can be derived from M' through nibble shuffling operations SR and SR^{-1}, *i.e.* $M = SR \circ M'$ and $M^{-1} = M' \circ SR^{-1}$. Hence, we avoid implementing three matrices to save some area. All linear operations are duplicated for the two shares. Our key and round constants are not shared and are applied to only one of the two shares. Overall sixteen copies of the shared S-Box, see Figure 4, are implemented for parallel execution. The register stage is integrated within the shared S-Box. There are no additional state registers outside the S-Box, making our implementation's latency one cycle per round and thus 12 cycles for PRINCE.

5.3 Efficiency and Comparison

The utilization results of our implementation are summarized in Table 3. The maximum frequency was obtained by synthesizing our designs while iteratively increasing the target frequency until the slack became negative. We have further included all first-order secure PRINCE designs with a low latency of one cycle per round available in the literature, to the best of our knowledge.

[1] `https://github.com/KULeuven-COSIC/TSM`

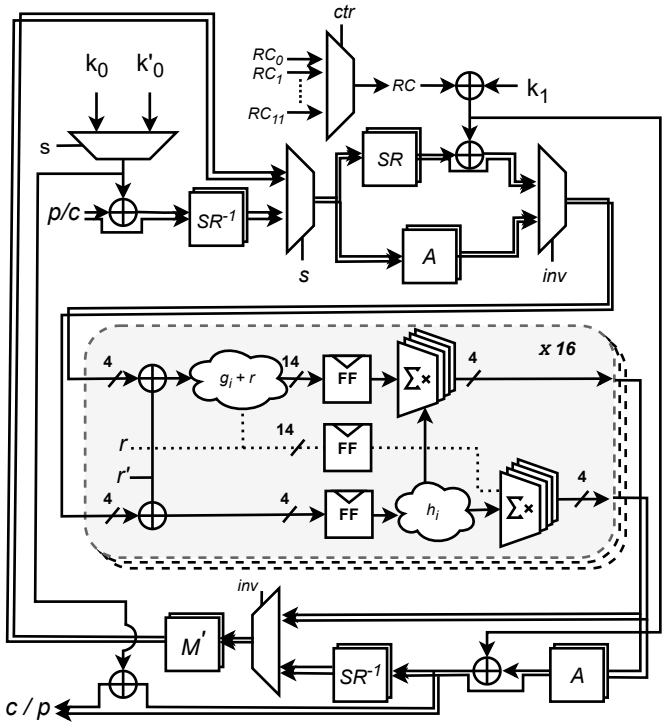


Figure 5: Architecture of our masked PRINCE design.

From the area results, we can see that our design has the smallest area footprint by a good margin. With the `compile` option, it is about 32% smaller than the GLM design presented by Muller *et al.* [26], which had the smallest area footprint among all existing low-latency designs before this work. With the `compile_ultra` option, the area becomes about 44% smaller than GLM. The other low-latency designs ([1], [4], [26]) are all based on Threshold Implementations (TI). As TI relies on non-completeness for its security, all these designs process data which are split into more than two shares. The increased share count results in a larger area. The second smallest design in Table 3, from [26], is based on GLM and requires implementing both the substitution and the inverse substitution layer. The security of GLM relies upon the condition that there are no variable collisions, *i.e.*, all the inputs must be independently shared. We refer the reader to the original GLM paper by Groß *et al.* [15], where they discuss variable collisions and why it is necessary for the inputs to the S-Box to have independent sharings. The authors of [26] argue that the affine transformations, required to compute the inverse S-Box, cause some of the inputs

of the S-Box to be dependent and to violate non-completeness. This limitation of GLM implies that both the regular substitution layer consisting of S and its inverse consisting of S^{-1} must be implemented to avoid collisions. As a result, the additional circuit for S^{-1} increases their design area. However, in contrast, TSM does not require independent sharings of the inputs, because of the initial remasking with r' in Figure 2. With our method, affine transformations can be used to calculate S^{-1}. This advantage helps us achieve the lowest area compared to all other designs, with a one cycle per round latency. We note that like in the designs from Muller *et al.* [26], which we mainly compare our design to, our design also does not mask the key. This ensures our area comparisons to be fair. The table also highlights the positive impact of logic optimization on the maximum clock frequency of our design.

Table 3: Utilization results of relevant first-order masked PRINCE implementations.

Design	Method	Area (GE)	f_{max} (Mhz)	Rand (bpc)*	Cycles
This Work	**TSM**	**13685**[a]	**485**[a]	**288**	**12**
		10926[b]	**610**[b]		
[26]	GLM	20046	-	128	12
[1]	LLTI	25857	488	48	12
[26]	4-share TI	26158	-	0	12
[4]#	TI	41628	335	48	12
[26]	5-share TI	42158	-	0	12

[a] `compile -exact_map`
[b] `compile_ultra -no_autoungroup -no_boundary_optimization`
* bits per cycle
# Resynthesized with the NANGATE45 standard cell library by the authors of [1]

In terms of randomness usage, our design uses more bits per cycle compared to other designs. However, we remind the reader of the results from Section 4.1. For masking any arbitrary function, *i.e.*, when comparing worst case metrics, TSM consumes less randomness than GLM. But we would like to note that the efficiency can be better depending on the concrete function that is masked. In the paper by Muller *et al.* [26], their GLM based design consumes 256 random bits per cycle, which is comparable to our design's requirement of 288 bits per cycle. However, to reduce the costs, they re-use randomness over S-Boxes, which reduces it to 128 bits per cycle. Such optimizations come at the cost of losing the composable security of the general approach. Since composable security was one of the goals of this work, we leave further optimizations specifically

for PRINCE, as well as randomness optimizations of a PINI secure general approach, as future work.

Our area estimation does not include the area overhead of PRNGs required to generate randomness, but this is consistent with the literature in general and the other designs included in Table 3 in particular. We use AES-128 in Output Feedback (OFB) mode to generate the randomness, which might be inefficient for the high randomness demands of low-latency applications. But we see random bit generation as a closely related yet orthogonal problem, and did not try to optimize this part. In the recent work by Cassiers *et al.* [7], the authors investigate several PRNGs and determine unrolled implementations of stream ciphers Trivium [5] and Bivium [32] to be the best candidates for efficiently generating random bits. To provide an estimate of the PRNG area overhead, generating 288 random bits per cycle for our design would require approximately 6.5 kGEs and 10.2 kGEs using unrolled implementations of Bivium and Trivium, respectively.

5.4 Formal Verification

In addition to the general security proof for TSM presented in Section 3.3, we validate the security of our TSM PRINCE S-box using the formal verification tool SILVER [19]. The tool verifies whether a masked implementation is secure and composable under various security notions such as probing security, Non-Interference (NI), Strong Non-Interference (SNI), and Probe-Isolating Non-Interference (PINI). We synthesized our PRINCE S-Box Verilog code to generate netlists with both the `compile` and `compile_ultra` options we discussed in Section 5.3. As shown in Table 4, the generated netlists for both options pass all security tests, which guarantees that our PRINCE S-Box is indeed probing and composable secure.

In order to increase confidence, we generate and verify an additional netlist of our S-Box after intentionally enabling register retiming, which is a performance optimization technique that allows combinational elements of our S-Box to be moved forward or backward across registers. The security of our masking technique relies on separating computations on shares with a register layer. Any movement of combinational logic across registers would very likely violate non-completeness. We set the `set_optimize_registers` condition to `true` and generate a netlist with the `compile_ultra -retime` option. The resulting netlist fails all robust probing tests. We conclude that our S-Box is secure when compiled with `compile_ultra` as long as register retiming remains disabled.

Table 4: Verification of our PRINCE S-Box with SILVER using different compiler options.

Compiler Option	Probing		NI		SNI		PINI		Uni.
	std.	rob.	std.	rob.	std.	rob.	std.	rob.	
`compile -exact_map`	✓	✓	✓	✓	✓	✓	✓	✓	✓
`compile_ultra -no_autoungroup -no_boundary_optimization`	✓	✓	✓	✓	✓	✓	✓	✓	✓
`set_optimize_registers true compile_ultra -retime`	✓	✗	✓	✗	✓	✗	✓	✗	✓

6 Case Study: Application of TSM to the AES S-Box

For our second case study, we examine the AES S-Box, a larger function with a higher algebraic degree, to demonstrate the efficiency of TSM. As outlined in Section 4, the main benefit of TSM comes from the ability to consolidate the coordinate functions of an S-Box and reuse the intermediate registered values among them. As one would expect, for larger S-Boxes with many coordinate functions, such as AES, the benefits of TSM should be more evident compared to other low-latency masking techniques that independently mask each coordinate function with unique sharings. In this section, we use TSM to create a low-latency AES S-Box and compare our implementation costs with other relevant low-latency masking techniques.

An unprotected AES S-Box processes eight input bits and generates eight output bits, denoted as $S : \mathbb{F}_2^8 \rightarrow \mathbb{F}_2^8$. We represent the eight input bits as (a, b, c, d, e, f, g, h) and the eight outputs using individual coordinate functions, each denoted as $f : \mathbb{F}_2^8 \rightarrow \mathbb{F}_2$. The coordinate functions written in their Algebraic Normal Form (ANF) include all terms up to degree 7. In total, there are 254 terms, calculated as the sum of binomial coefficients for each degree from 1 to 7: $\sum_{i=1}^{7} \binom{8}{i} = 254$.

6.1 Masked AES S-Box

At the high level, our strategy for masking the AES S-Box closely resembles the general TSM approach. In the initial phase, we calculate all cross-products of the $share_0$ inputs up to degree 7, *i.e.* excluding the degree 8 term $a_0b_0c_0d_0e_0f_0g_0h_0$, and refresh them with 254 bits of randomness. In the following phase, we calculate the cross-products of the $share_1$ inputs and combine them with the masked cross-products of the $share_0$ inputs to produce the S-Box outputs. When applying TSM to a large S-Box such as AES, the register cost forms only

a small fraction of the total area. The majority of the total area is attributed to the logic computation of coordinate functions in the second phase. Consequently, optimizing logic becomes a crucial factor. We therefore apply the ideas explained in Section 4.2, and in particular extend them to all higher degree (degree-4,-5,-6 and -7) terms. An example with the sharings for the degree-4 term $abcd$ is shown in Eq. (8). We use the already computed degree-3 and degree-2 terms: acd, bcd, cd to construct the sharing of $abcd$. By following a similar strategy, for degree-5 terms, we use the outputs of the degree-4 and degree-3 terms. For degree-6 terms, we use the outputs of the degree-5, degree-4, and degree-3 terms. Finally, for degree-7 terms, we use the outputs of the degree-6, degree-5, and degree-4 terms. For additional details, we provide our HDL source code[2] with the precise sharings of our complete AES S-Box.

$$\begin{aligned}
(abcd)_0 = {} & (acd)_0 \cdot \left\{b'_1\right\} + (bcd)_0 \cdot \left\{a'_1\right\} + (cd)_0 \cdot \left\{a'_1\right\} \cdot \left\{b'_1\right\} \\
& + \left\{a'_0 b'_0 c'_0 d'_0 + r_{14}\right\} \quad + \left\{a'_0 b'_0 c'_0 + r_{10}\right\} \cdot \left\{d'_1\right\} \\
& + \left\{a'_0 b'_0 d'_0 + r_{11}\right\} \cdot \left\{c'_1\right\} + \left\{a'_0 b'_0 + r_4\right\} \cdot \left\{c'_1\right\} \cdot \left\{d'_1\right\} \\
\hline
(abcd)_1 = {} & (acd)_1 \cdot \left\{b'_1\right\} + (bcd)_1 \cdot \left\{a'_1\right\} + (cd)_1 \cdot \left\{a'_1\right\} \cdot \left\{b'_1\right\} + \left\{r_{14}\right\} \\
& + \left\{r_{10}\right\} \cdot \left\{d'_1\right\} \quad + \left\{r_{11}\right\} \cdot \left\{c'_1\right\} + \left\{r_4\right\} \cdot \left\{c'_1\right\} \cdot \left\{d'_1\right\}
\end{aligned} \tag{8}$$

6.2 Efficiency and Comparison

We summarize the utilization results for our masked S-Box in Table 5 and compare them with other state-of-the-art low-latency masking schemes. We synthesized our design with the same compiler options and switches we used for PRINCE in Section 5.1. All designs were synthesized with a target clock frequency of 100 MHz to get the area results. To find the maximum frequency of our designs, we follow the same strategy of gradually increasing the frequency until we observe a negative slack.

With `compile`, our AES S-Box has an area of 20.5 kGEs, which is approximately one-third of the area of GLM and GHPC_{LL}. We attribute the lower area cost to the two aspects of TSM we discussed in Section 4. First, the sharings for the 254 terms are commonly reused among all 8 coordinate functions. Second, reusing already computed lower degree terms for computing higher degree terms substantially reduces the area for logic. Using the `compile_ultra` compiler

[2] `https://github.com/KULeuven-COSIC/TSM`

option further reduces the logic area, resulting in a much smaller design with just below 14.5 kGEs. Also here we note the positive impact of logic optimization on the maximum clock frequency. Still our S-Box is larger than the S-Box in LMDPL. But it is noteworthy that the AES S-Box in LMDPL is based on dual-rail logic, which requires a pre-charge phase between consecutive evaluations. A comparison of a single AES S-box in TSM and in LMDPL is thus not straightforward.

Table 5: Utilization results of relevant first-order masked AES S-Box implementations.

Design	Method	Area (GE)	f_{max} (MHz)	Rand (bits)	Latency (Cycles)
This Work	**TSM**	**20523**[a]	**375**[a]	**262**	**1**
		14352[b]	**535**[b]		
[35]	LMDPL	3480	400	36	1
[15]	GLM	60730	356	2048	1
[20]	GHPC_{LL}	64111	-	2048	1
[23]	LMDPL	2830	-	36	2
[15]	GLM	6740	584	416	2
[1]	4-share LLTI	25780	277	0	2
[1]	4-share TI	58410	40	0	2
[20]	GHPC	77145	-	8	2

[a] `compile -exact_map`
[b] `compile_ultra -no_autoungroup -no_boundary_optimization`

Comparing randomness usage, our S-Box requires roughly eight times less randomness when compared to both GLM and GHPC_{LL}. The reduction can be directly linked to the consolidation of coordinate functions and the reuse of the 254 shared terms. Given that the AES S-Box produces eight output bits, we observe an eight-fold decrease. LMDPL requires less randomness but again a comparison of a single S-Box is difficult.

As for the maximum frequency, it is not an easy comparison as most designs use a different standard cell library. Our design has a maximum frequency similar to GLM and LMDPL.

6.3 Formal Verification

We were not able to formally verify the netlists of our AES S-Box, primarily due to the limitations of SILVER, which uses Binary Decision Diagrams

(BDD) to represent Boolean functions and generate probability distributions for verification and statistical independence checks. Since our masked AES S-Box is significantly larger than the masked PRINCE S-Box, the tool terminates with a message indicating that the unique table storing BDD nodes has been filled. As an alternative to formal verification, we experimentally assess the leakage of our AES S-Box on an FPGA, see the next section.

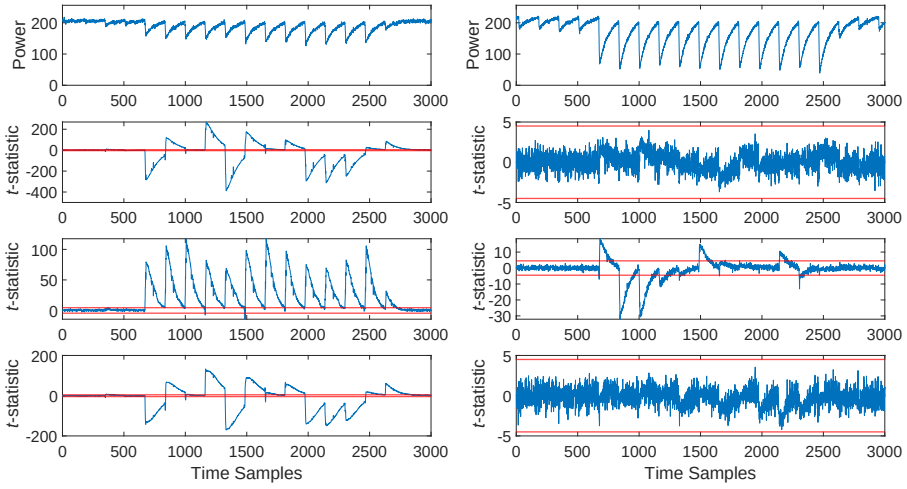
7 Practical Evaluation

We evaluate our first-order protected designs by implementing and checking for side-channel leakage on FPGA. We use a SAKURA-G board [30], featuring two Xilinx SPARTAN-6 FPGAs, and Xilinx ISE v14.7. As commonly done, we implement our protected design (complete PRINCE or AES S-Box) on the target FPGA where it receives pre-shared inputs from the control FPGA and returns shared outputs to the control FPGA in order to avoid any IO leakage. We use the control FPGA to interface with a PC and to orchestrate the sequence of inputs to the target FPGA. We use an (unprotected) AES-128 encryption core in Output Feedback (OFB) mode as a PRNG on the control FPGA. In case of the complete PRINCE design, the target FPGA contains a wrapper module that implements PRNGs to generate online randomness. The seed for these PRNGs is generated on the control FPGA and transferred to the target FPGA for every PRINCE encryption/decryption. The protected PRINCE core receives random bits from the PRNGs in the wrapper as needed in every clock cycle.

We measure the (on-board amplified) power consumption of the target FPGA on SMA connector J3 with a Tektronix DPO7254C oscilloscope sampling at 500MS/s. We operate the board at a very slow clock frequency of 3 MHz to ensure that there is no overlap between the power patterns of consecutive clock cycles. The clock signal is stable and provided by an external function generator synchronized with the oscilloscope to enable synchronous sampling. This is done to ensure the best possible alignment of the traces to reduce sampling noise. The power measurements cover the entire 12 rounds for the protected PRINCE, and one S-Box computation for AES. The top rows of Figures 6 and 7 show sample power traces (raw oscilloscope ADC output).

We use the non-specific Test Vector Leakage Assessment (TVLA) methodology as explained by Bilgin *et al.* [2] and originally described in [13, 10]. Concretely we use the fast computation method described by Reparaz *et al.* [34]. We focus on univariate analysis since we claim no first-order leakage. We perform fixed vs. random tests where we provide either a fixed or a random input to the

masked PRINCE core resp. to the masked AES S-Box, in a random sequence. The unprotected keys are fixed for all experiments.



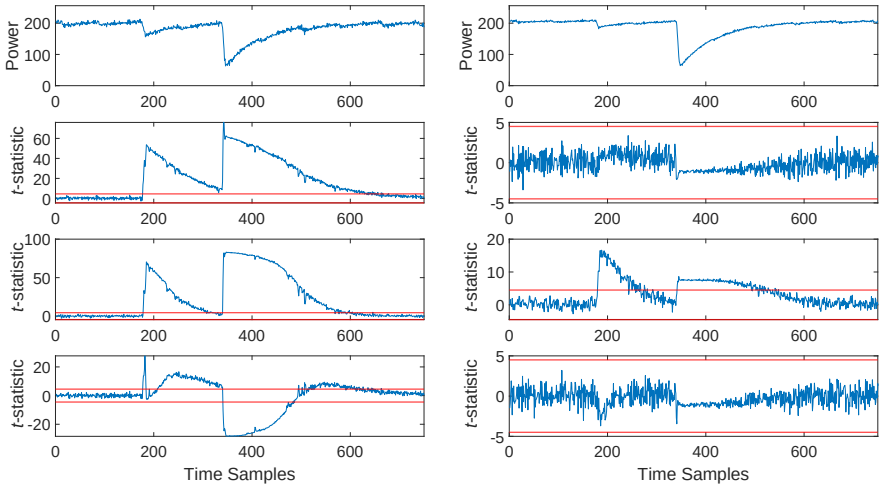
(a) PRNGs set to *OFF* with 100k traces.

(b) PRNGs set to *ON* with 100 million traces.

Figure 6: TVLA results for masked PRINCE core. Top to bottom: power trace, the first-, second-, and third-order univariate t-test result. The red lines indicate the ±4.5 threshold.

As a safety check, we first provide a result of the leakage assessment performed with the PRNGs switched off, which means all random bits used by the masked PRINCE core, by the masked AES S-Box, and for initial masking, are zero. Figure 6a and Figure 7a show for 100k traces, from top to bottom: the first-order univariate t-test result, second-order univariate t-test result, and third-order univariate t-test result. Red lines indicate the commonly applied threshold at ±4.5. We obtain very significant peaks in the first-order t-test, which confirms that our setup works well.

Figure 6b and Figure 7b show the TVLA results with the PRNGs switched ON. We measured and analyzed 100 million traces of the complete PRINCE core and 250 million traces of the AES S-Box. As we can see from the first-order t-test results in the second rows, the t-test value does not exceed the threshold of ±4.5 at any time sample, therefore, indicating that our designs do not exhibit first-order leakage. The processing of a non-linear function must combine information from all shares at some point. TSM uses two shares and thus we expect to observe second-order leakage, which is indeed evident in the second-order t-test



(a) PRNGs set to *OFF* with 100k traces.

(b) PRNGs set to *ON* with 250 million traces.

Figure 7: TVLA results for AES S-Box. Top to bottom: power trace, the first-, second-, and third-order univariate t-test result. The red lines indicate the ±4.5 threshold.

results. We show this result to increase confidence in the correctness of our measurement and processing. One can further expect to observe third-order leakage, because the data in TSM's register layer is essentially in three shares. However, we do not observe leakage in the third-order t-test results, one possible explanation is that more traces would be required.

8 Conclusion

We introduced a novel approach for low-latency masking that ensures security by separating the processing of the different shares over time with a register layer. TSM is straightforward to apply to any arbitrary (vectorial) Boolean function and does not come with constraints that a designer must be careful about to avoid compromising security. We prove first-order glitch-extended PINI security and demonstrate excellent implementation results in terms of low overheads when implementing a complete PRINCE core and an AES S-box.

More specifically, the masking method scales in register and randomness cost on the number of inputs and the algebraic degree of the function, but it amortizes

this cost for each output bit. The overhead reduction of TSM comes from the ability to consolidate masked component functions and use the intermediate values commonly among them.

We demonstrated the effectiveness of our approach by implementing a first-order secure low-latency PRINCE that has 32% resp. 44% area improvement over the state-of-the-art. We formally validated the security of our PRINCE S-Box implementation with SILVER, and we provide results of practical leakage assessment on the complete PRINCE core which show no first-order leakage with 100 million traces.

Our masked AES S-box costs roughly one-third resp. one-quarter of the area of state-of-the-art implementations, and has much lower randomness cost as well. We deliberately exclude LMDPL from the comparison because a direct comparison to TSM seems difficult. The AES S-Box in LMDPL is smaller and requires less random bits, at first sight, but it requires dual-rail logic and a precharge phase between consecutive executions, which will become important when building a circuit which executes over more than one clock cycle. We provide results of practical leakage assessment of our AES S-Box which show no first-order leakage with 250 million traces.

As a future work, it would be interesting to reduce the randomness cost of TSM without trading off latency and security. Using an additional register layer (on the output), it is possible to re-use the random bits r over all gadgets. For our PRINCE implementation, this would mean 14 random bits can be re-used among all 16 S-Boxes. However, without such an additional register layer, the reduction of randomness seems less trivial. Another direction for future work is to extend and generalize TSM to higher-order security. It would also be interesting to implement more algorithms with TSM in order to compare the overhead reductions with other existing low-latency designs.

Acknowledgments

This research has been funded in part by KU Leuven Startfinanciering STG/20/047, by CyberSecurity Research Flanders VR20192203, by the European Commission through Belfort ERC Advanced Grant 101020005 695305, and through Horizon Europe RIA grant HORIZON-CL3-2021-CS-01-02 101070008 ORSHIN.

References

[1] Arribas, V., Zhang, Z., and Nikova, S. LLTI: low-latency threshold implementations. *IEEE Trans. Inf. Forensics Secur. 16* (2021), 5108–5123.

[2] Bilgin, B., Gierlichs, B., Nikova, S., Nikov, V., and Rijmen, V. Higher-order threshold implementations. In *Advances in Cryptology - ASIACRYPT 2014 - 20th International Conference on the Theory and Application of Cryptology and Information Security, Kaoshiung, Taiwan, R.O.C., December 7-11, 2014, Proceedings, Part II* (2014), P. Sarkar and T. Iwata, Eds., vol. 8874 of *Lecture Notes in Computer Science*, Springer, pp. 326–343.

[3] Borghoff, J., Canteaut, A., Güneysu, T., Kavun, E. B., Knezevic, 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 (full version). *IACR Cryptol. ePrint Arch.* (2012), 529.

[4] Bozilov, D., Nikov, V., and Rijmen, V. Design trade-offs in threshold implementations. In *26th IEEE International Conference on Electronics, Circuits and Systems, ICECS 2019, Genoa, Italy, November 27-29, 2019* (2019), IEEE, pp. 751–754.

[5] Cannière, C. D. Trivium: A stream cipher construction inspired by block cipher design principles. In *Information Security, 9th International Conference, ISC 2006, Samos Island, Greece, August 30 - September 2, 2006, Proceedings* (2006), S. K. Katsikas, J. López, M. Backes, S. Gritzalis, and B. Preneel, Eds., vol. 4176 of *Lecture Notes in Computer Science*, Springer, pp. 171–186.

[6] 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.

[7] Cassiers, G., Masure, L., Momin, C., Moos, T., Moradi, A., and Standaert, F.-X. Randomness generation for secure hardware masking - unrolled trivium to the rescue. Cryptology ePrint Archive, Paper 2023/1134, 2023. `https://eprint.iacr.org/2023/1134`.

[8] 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.

[9] Chari, S., Jutla, C. S., Rao, J. R., and Rohatgi, P. Towards sound approaches to counteract power-analysis attacks. In *Advances in Cryptology - CRYPTO '99, 19th Annual International Cryptology Conference, Santa Barbara, California, USA, August 15-19, 1999, Proceedings* (1999), M. J. Wiener, Ed., vol. 1666 of *Lecture Notes in Computer Science*, Springer, pp. 398–412.

[10] Cooper, J., De Mulder, E., Goodwill, G., Jaffe, J., Kenworthy, G., and Rohatgi, P. Test vector leakage assessment (TVLA) methodology in practice. `http://icmc-2013.org/wp/wp-content/uploads/2013/09/goodwillkenworthtestvector.pdf`, 2013.

[11] 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.

[12] Gandolfi, K., Mourtel, C., and Olivier, F. Electromagnetic analysis: Concrete results. In *Cryptographic Hardware and Embedded Systems - CHES 2001, Third International Workshop, Paris, France, May 14-16, 2001, Proceedings* (2001), Ç. K. Koç, D. Naccache, and C. Paar, Eds., vol. 2162 of *Lecture Notes in Computer Science*, Springer, pp. 251–261.

[13] Goodwill, G., Jun, B., Jaffe, J., and Rohatgi, P. A testing methodology for side channel resistance validation. `http://csrc.nist.gov/newsevents/non-invasive-attack-testing-workshop/08Goodwill.pdf`, 2011.

[14] Goubin, L., and Patarin, J. DES and differential power analysis (the "duplication" method). In *Cryptographic Hardware and Embedded Systems, First International Workshop, CHES'99, Worcester, MA, USA, August 12-13, 1999, Proceedings* (1999), Ç. K. Koç and C. Paar, Eds., vol. 1717 of *Lecture Notes in Computer Science*, Springer, pp. 158–172.

[15] Gross, H., Iusupov, R., and Bloem, R. Generic low-latency masking in hardware. *IACR Trans. Cryptogr. Hardw. Embed. Syst. 2018*, 2 (2018), 1–21.

[16] 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 2016 Vienna, Austria, October, 2016* (2016), B. Bilgin, S. Nikova, and V. Rijmen, Eds., ACM, p. 3.

[17] Ishai, Y., Sahai, A., and Wagner, D. A. Private circuits: Securing hardware against probing attacks. In *Advances in Cryptology - CRYPTO 2003, 23rd Annual International Cryptology Conference, Santa Barbara, California, USA, August 17-21, 2003, Proceedings* (2003), D. Boneh, Ed., vol. 2729 of *Lecture Notes in Computer Science*, Springer, pp. 463–481.

[18] 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.

[19] 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.

[20] Knichel, D., Sasdrich, P., and Moradi, A. Generic hardware private circuits towards automated generation of composable secure gadgets. *IACR Trans. Cryptogr. Hardw. Embed. Syst. 2022*, 1 (2022), 323–344.

[21] 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.

[22] Kocher, P. C., Jaffe, J., and Jun, B. Differential power analysis. In *Advances in Cryptology - CRYPTO '99, 19th Annual International Cryptology Conference, Santa Barbara, California, USA, August 15-19, 1999, Proceedings* (1999), M. J. Wiener, Ed., vol. 1666 of *Lecture Notes in Computer Science*, Springer, pp. 388–397.

[23] 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.

[24] Mangard, S., Popp, T., and Gammel, B. M. Side-channel leakage of masked CMOS gates. In *Topics in Cryptology - CT-RSA 2005, The Cryptographers' Track at the RSA Conference 2005, San Francisco, CA, USA, February 14-18, 2005, Proceedings* (2005), A. Menezes, Ed., vol. 3376 of *Lecture Notes in Computer Science*, Springer, pp. 351–365.

[25] 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 - 22nd International Conference on the Theory and Application of Cryptology and Information Security, Hanoi, Vietnam, December 4-8, 2016, Proceedings, Part I* (2016), J. H. Cheon and T. Takagi, Eds., vol. 10031 of *Lecture Notes in Computer Science*, pp. 517–547.

[26] Müller, N., Moos, T., and Moradi, A. Low-latency hardware masking of PRINCE. In *Constructive Side-Channel Analysis and Secure Design - 12th International Workshop, COSADE 2021, Lugano, Switzerland, October 25-27, 2021, Proceedings* (2021), S. Bhasin and F. D. Santis, Eds., vol. 12910 of *Lecture Notes in Computer Science*, Springer, pp. 148–167.

[27] 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.

[28] NANGATE. The nangate 45nm open cell library. `https://www.nangate.com`.

[29] Nikova, S., Rechberger, C., and Rijmen, V. Threshold implementations against side-channel attacks and glitches. In *Information and Communications Security, 8th International Conference, ICICS 2006, Raleigh, NC, USA, December 4-7, 2006, Proceedings* (2006), P. Ning, S. Qing, and N. Li, Eds., vol. 4307 of *Lecture Notes in Computer Science*, Springer, pp. 529–545.

[30] Project, S. H. S. Sakura-G: Side-channel AttacK User Reference Architecture. `https://satoh.cs.uec.ac.jp/SAKURA/hardware/SAKURA-G.html`, 2013.

[31] 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.

[32] Raddum, H. Cryptanalytic results on trivium. estream, ecrypt stream cipher project, report 2006/039, 2006, 2006.

[33] Reparaz, O., Bilgin, B., Nikova, S., Gierlichs, B., and Verbauwhede, I. Consolidating masking schemes. In *Advances in Cryptology - CRYPTO 2015* (2015), R. Gennaro and M. Robshaw, Eds., vol. 9215 of *Lecture Notes in Computer Science*, Springer, pp. 764–783.

[34] Reparaz, O., Gierlichs, B., and Verbauwhede, I. Fast leakage assessment. In *Cryptographic Hardware and Embedded Systems - CHES 2017 - 19th International Conference, Taipei, Taiwan, September 25-28, 2017, Proceedings* (2017), W. Fischer and N. Homma, Eds., vol. 10529 of *Lecture Notes in Computer Science*, Springer, pp. 387–399.

[35] 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.

Chapter 9

Higher-Order Time Sharing Masking

Publication Data

Dilip Kumar S.V., Siemen Dhooghe, Josep Balasch, Benedikt Gierlichs, and Ingrid Verbauwhede, "Higher-Order Time Sharing Masking," *IACR Transactions on Cryptographic Hardware and Embedded Systems*, 2025(2), pp. 235–267. https://doi.org/10.46586/tches.v2025.i2.235-267.

Contributions

Principal author.

Verification Note

In this chapter, "formal verification" means automated checks with SILVER for PINI/SNI under the robust probing model. As a scheme, HO-TSM is proven d-th-order glitch-extended PINI and SNI (see Sec. 3.3). The latency–area trade-off construction is likewise proven PINI and SNI (see Sec. 4.3). At the implementation level, we used SILVER on small HO-TSM instances (e.g.,

second-/third-order AND gadgets) and on small examples of the latency–area trade-off construction; we provide no pen-and-paper proofs for these gadgets. We did not use SILVER on large composite circuits such as a full AES S-Box. These implementation results are tool-based evidence only; they are *not* formal proofs and do *not* constitute a formal verification of the full designs.

Higher-Order Time Sharing Masking

Dilip Kumar S. V.[1], Siemen Dhooghe[1], Josep Balasch[2],
Benedikt Gierlichs[1], Ingrid Verbauwhede[1]

[1]COSIC, KU Leuven, Belgium
[2]e-Media Research Lab, STADIUS, KU Leuven, Belgium

Abstract. At CHES 2024, Time Sharing Masking (TSM) was introduced as a novel low-latency masking technique for hardware circuits. TSM offers area and randomness efficiency, as well as glitch-extended PINI security, but it is limited to first-order security. We address this limitation and generalize TSM to higher-order security while maintaining all of TSM's advantages. Additionally, we propose an area-latency tradeoff. We prove HO-TSM glitch-extended PINI security and successfully evaluate our circuits using formal verification tools. Furthermore, we demonstrate area- and latency-efficient implementations of the AES S-box, which do not exhibit leakage in TVLA on FPGA. Our proposed tradeoff enables a first-order secure implementation of a complete AES-128 encryption core with 92 kGE, 920 random bits per round, and 20 cycles of latency, which does not exhibit leakage in TVLA on FPGA.

1 Introduction

Cryptographic algorithms implemented on computer systems are susceptible to physical attacks that can extract sensitive information being processed by the system. An adversary could monitor side-channel information, such as power consumption [29], execution time [28], or electromagnetic emanations [16, 36], to uncover sensitive data like cryptographic secret keys. Masking [11, 20] is a popular countermeasure that splits data into multiple shares, thereby removing the correlation between the data and side-channel information to protect implementations against such attacks.

Over the past years, masking techniques for hardware have, most importantly, aimed to guarantee security in the presence of glitches [31]. To securely mask hardware implementations, several techniques have been developed, including popular methods such as Threshold Implementations (TI) [35], Consolidating Masking Schemes (CMS) [37], and Domain Oriented Masking (DOM) [22]. These methods primarily focus on reducing the area overhead and minimizing the fresh randomness required to ensure security. Another line of research in

hardware masking that has gained traction in recent years is focused solely on reducing latency. An initial effort, LMDPL [30], introduced by Leiserson *et al.*, achieved the construction of a first-order secure AES S-Box in just 2 cycles. Subsequent improvements by Sasdrich et al. [40] further refined this approach to create a first-order secure round-based AES-128 implementation operating in ten cycles. Later, Gross et al. introduced an alternative low-latency approach, GLM [21], derived from DOM, capable of securely masking vectorial Boolean functions of any algebraic degree with a latency of just one cycle, regardless of the security order. Recent works include GHPC [27] and TSM [41], both of which are low-latency approaches capable of producing first-order single-cycle masked gadgets. They offer the additional benefit of compositional security under the Probe Isolating Non-Interference (PINI) security notion [10]. A prevailing theme across all low-latency approaches is the trade-off between area and randomness to minimize latency. The single-cycle first-order GLM AES S-Box is 60.7 kGE in size and requires 2048 bits of randomness, while the GHPC_{LL} AES S-Box has an area of 64.1 kGE and also requires 2048 bits of randomness. In comparison, the single-cycle TSM AES S-Box shows improvements in both area and randomness, with a size of 14.3 kGE and a randomness requirement of 262 bits. However, the area and randomness requirements remain substantial compared to other S-Boxes with higher latency that employ techniques like DOM, TI. Notably, many low-latency approaches are limited to provide only first-order security.

Contributions. We extend TSM to achieve higher-order security (HO-TSM) and introduce an area-latency trade-off construction that leverages TSM and HO-TSM, leading to substantial reductions in area and randomness costs. Our contributions are as follows:

- We introduce a higher-order extension, HO-TSM, that builds upon the foundation of first-order TSM [41]. The fundamental principle of HO-TSM is to process one share of each input in every clock cycle while maintaining isolation of these computations through the use of registers. Notably, the latency of HO-TSM is dictated by the security order, rather than the algebraic degree of the function. For a dth-order secure implementation, the latency is precisely d cycles.
- We formally outline our construction and demonstrate that any function secured using our method achieves d^{th}-order glitch-extended PINI and SNI (Strong Non-Interference) [4] security.
- We develop a two-cycle, second-order HO-TSM AES S-Box that occupies less area and requires less randomness than the only other low-latency, second-order AES S-Box by Gross *et al.* (GLM) [22]. Our FPGA

implementation of this S-Box exhibits no signs of first- or second-order leakage in TVLA with 100 million traces.

- We present an area-latency trade-off for HO-TSM that significantly reduces both area and randomness costs, with only a single cycle increase in latency. We construct a new two-cycle, first-order PINI and SNI secure AES S-Box that shows substantial improvements in utilization costs and develop a complete round-based AES-128 implementation that executes in twenty cycles. The FPGA implementation of our full AES-128 demonstrates no first-order leakage in TVLA with 250 million traces.
- Using our trade-off construction, we develop another three-cycle, second-order PINI and SNI secure AES S-Box. Our S-Box exhibits no first- or second-order leakage in TVLA with 100 million traces.
- We use SILVER [26] to validate small examples of both HO-TSM and our trade-off construction, providing evidence for PINI and SNI security. In addition, we also verify our S-Boxes using the formal verification tools `maskVerif` [3] and PROLEAD [33].
- We provide the HDL source code[1] for the designs presented in this paper and their synthesis scripts.

Outline. Section 2 introduces notations and essential background information. In Section 3, we first illustrate our concept using a toy example by developing a straightforward second-order masked AND gate. We then provide a formal description of our approach applicable to any Boolean function, and a proof demonstrating its dth-order glitch-extended PINI and SNI security. We apply our approach to produce a two-cycle second-order secure AES S-Box. In Section 4, we introduce an area-latency trade-off for HO-TSM and establish its security under the PINI and SNI security notions. We construct both a first-order and a second-order AES S-Box. Using the first-order secure S-Box, we proceed to design a complete round-based AES-128 implementation. In Section 5, we conduct formal verification of our AES S-Boxes, and practically validate our designs using TVLA by implementing them on FPGA. In Section 6, we summarize our work and discuss possible future work.

2 Preliminaries

We introduce essential concepts and notations which form the basis for the methodologies and security proofs we present in later sections.

[1] `https://github.com/KULeuven-COSIC/TSM`

2.1 Notation

In Boolean masking, each value $x \in \mathbb{F}_2^k$ is split into $d+1$ uniform random shares x_i such that $x = x_0 + \ldots + x_d$. For $x \in \mathbb{F}_2^k$ a k-bit word, we denote the bits of the word by square brackets $(x[0], ..., x[k-1])$. We represent the sharing of x as $\bar{x} = (x_0, x_1, \cdots, x_d) \in \left(\mathbb{F}_2^k\right)^{d+1}$ such that $x_0 + x_1 \cdots + x_d = x$ with $x_i = (x_i[0], ..., x_i[k-1])$ the notation of the share-words in separate bits. For a set of share indices A $\subset \{0, \ldots, d\}$, we denote $x_A = \{x_i : x \in A\}$. We denote $x \xleftarrow{\$} X$ to represent selecting a value uniformly and randomly from the set X.

2.2 Circuit Model

For the purpose of security proofs and to explain masking methods, we represent algorithms in the shape of a directed-acyclic graph called a circuit. In this circuit, an edge represents a bit ($\mathbb{F}_2$) or a word ($\mathbb{F}_2^k$) and a node represents an operation of its fan-in such as an XOR or an AND gate. We also consider nodes without input which can output a uniformly distributed random bit or word.

2.3 Probing Model

We use the probing model, as originally proposed by Ishai *et al.* [24], to model the side-channel adversary. More specifically, the dth-order probing adversary is one who has access to the layout of the circuit he is attacking and who is able to request (before the execution of the circuit) the digital value of up to d wires (probes) in the circuit.

The above adversary is expanded following the glitch-extended robust probing model by Faust *et al.* [15] where each probe is expanded such that it returns not only the wire value but all registered values leading up to that wire. This adversarial expansion is a way to capture the effect of glitches and propagation delays in the circuit's physical implementation.

We call a circuit d^{th}-order glitch-extended probing secure if there exists a simulator which can simulate probed values in such a way that an adversary cannot distinguish the actual circuit from the simulator. The adversary has the power to choose the circuit's input secrets and the simulator is not given this information. This simulation-based security is a clever way of saying that the probed values' distribution (made by the circuit's internal randomness) is independent of its input secrets.

Composable Probing Security

We consider a simulation security game for the probing adversary, because apart from easing certain proofs, it allows for composable security. Namely, using the Strong Non-Interference (SNI) framework by Barthe *et al.* [4] and its extension to Probe-Isolated Non-Interference (PINI) by Cassiers and Standaert [10], one can show that if a small part of the circuit (gadget) is composable secure then its composition with other composable secure gadgets remains secure.

We first provide the definition of simulatability as given by in [10].

Definition 1 (Simulatability [10])**.** Let $P = \{p_1, ..., p_\ell\}$ be a set of ℓ probes of a gadget C and C_P the tuple of values of the probes for an execution of C. Let $I = \{(i_1, j_1), ..., (i_m, j_m)\} \subset \{0, ..., d-1\} \times \{0, ..., k-1\}$ be a set of input wires of C. A simulator is a randomized function $\mathcal{S} : \mathbb{F}_q^m \to \mathbb{F}_q^\ell$. The set of probes P can be simulated with the set of input wires I if there exists a simulator $\mathcal{S}$ such that for any inputs $\bar{x}$, the distributions $C_P(\bar{x})$ and $\mathcal{S}(x_{i_1}[j_1], ..., x_{i_m}[j_m])$ are equal, where the probability is over the random bits in C and $\mathcal{S}$.

In this work, we consider composability in the form of both the SNI [4] and the PINI [10] frameworks. The reason for opting for a more restrictive simulation-based model will become clear in Section 4, where we provide a low-cost area-latency trade-off. In the more restrictive NI model, we consider simulation where the simulator is not given any input shares when probing an output. We introduce both PINI and SNI.

Definition 2 (SNI)**.** A gadget is called t–SNI if for any set of t_0 (glitch-extended) probes on intermediate variables and every set of t_1 (glitch-extended) probes on output shares such that $t_0 + t_1 \leq t$, the totality of the probes can be simulated by only t_0 shares of each input.

Definition 3 (PINI)**.** Let G be a gadget over d shares and P a set of t_0 (glitch-extended) probes on wires of G (called internal probes). Let A be a set of t_1 share indices. G is t-PINI if for all P and A such that $t_0 + t_1 \leq t$, there exists a set B of at most t_0 share indices such that probes on the set of wires $P \cup y_A$ can be simulated with the wires $x_{A \cup B}$, with x_i denoting all inputs of share i and y_i denoting all outputs of share i.

In the regular PINI notion, the simulator is given the input share (the share with the same index) for every probed output share and in the regular SNI notion, the simulator is given arbitrary shares of each input instead of the same index for each input. Instead, we demand that the simulation of the gadget can be made using the same index for each input share for each placed internal probe and that the simulator is not given any shares when probing the gadget's output.

2.4 Time Sharing Masking

Time Sharing Masking (TSM) is a low-latency masking scheme introduced by Kumar S. V. *et al.* [41] designed to perform first-order masking of any (vectorial) Boolean function in hardware using a single register layer. Moreover, the technique has been proven to be secure under the PINI composable security notion. In this subsection, we provide a comprehensive description of TSM as applied to an arbitrary (vectorial) Boolean function.

Consider an arbitrary Boolean function

$$f : \mathbb{F}_2^k \;\to\; \mathbb{F}_2^\ell \quad : \quad x = \big(x[0], \ldots, x[k-1]\big) \;\mapsto\; f\big(x[0], \ldots, x[k-1]\big).$$

We denote its two-share masking by $\bar{F} : \mathbb{F}_2^{2k} \to \mathbb{F}_2^{2\ell}$, such that $\bar{x} \mapsto (F_0(\bar{x}), F_1(\bar{x}))$ and $F_0(\bar{x}) + F_1(\bar{x}) = f(x_0 + x_1)$. The sharing of the ℓ coordinate functions is given by $F_0 = (F_0[0], \ldots, F_0[\ell-1])$ and $F_1 = (F_1[0], \ldots, F_1[\ell-1])$.

We explain the TSM method constructively, beginning with a reformulation of the algebraic normal form (ANF) for one coordinate function $f[m]$ of f, where $m \in \{0, \ldots, \ell-1\}$. The ANF decomposes $f[m]$ in a way that facilitates its masking. Any single-output Boolean function $f[m]$ can be expressed in its ANF as a sum of products of input bits:

$$f[m]\big(x[0], \ldots, x[k-1]\big) \;=\; \sum_{S \in \mathcal{P}_k} \alpha_S \prod_{j \in S} x[j] \quad \text{with} \quad \alpha_S \in \mathbb{F}_2. \tag{1}$$

Here, $\mathcal{P}_k$ denotes the power set of the indices $\{0, \ldots, k-1\}$, so that $|\mathcal{P}_k| = 2^k$. The sum is over all elements $S \in \mathcal{P}_k$. For any S, α_S determines whether the product of input bits $x[j]$ for $j \in S$ is present in the ANF of $f[m]$.

We define:

$$\mathcal{M}_m \;=\; \big\{S \in \mathcal{P}_k \;\mid\; \alpha_S = 1\big\} \tag{2}$$

as the set of all indices of products present in $f[m]$.

Decomposing Product Terms in the ANF

Each product indexed by $S \in \mathcal{M}_m$ is masked using Boolean masking and decomposed into a sum of products of shares from domains 0 and 1:

$$\prod_{j \in S} x[j] \;=\; \prod_{j \in S} \big(x_0[j] + x_1[j]\big) \;=\; \sum_{I \subseteq S} \Big(\prod_{j \in I} x_0[j]\Big) \Big(\prod_{j \in S \setminus I} x_1[j]\Big). \tag{3}$$

Here, the sum is over all $I \in \mathcal{P}(S)$, the power set of S. Recall that S represents one specific combination of input bits $x[j]$. The sum is thus over all possible combinations of bits in S. The first product involves shares from domain 0 for bits in I, while the second involves shares from domain 1 for bits which are not in I, but in S.

We define two sets of functions g^0 and g^1, each operating on a single share domain, 0 and 1, respectively:

$$g^0_{\pi(I)}(x_0) \;=\; \prod_{j \in I} x_0[j], \qquad g^1_{\pi(I)}(x_1) \;=\; \prod_{j \in I} x_1[j] \tag{4}$$

where $I \subseteq \{0, \ldots, k-1\}$. For brevity, we write $g^0_{\pi(I)}(\cdot)$ to denote $g^0_{\pi(I)}$ evaluated on the share x_0. Similarly, $g^1_{\pi(I)}(\cdot)$ for the function $g^1_{\pi(I)}$ evaluated on x_1. The indexing function π maps each $I \in \mathcal{P}_k$ to a unique integer in $\{0, \ldots, 2^k - 1\}$. For the empty set, $g^0_{\pi(\emptyset)}(\cdot) = g^1_{\pi(\emptyset)}(\cdot) = 1$.

Using these definitions in Eq.(4), we rewrite Eq.(3) as:

$$\prod_{j \in S} x[j] \;=\; \prod_{j \in S} \big(x_0[j] + x_1[j]\big) \;=\; \sum_{I \subseteq S} \Big(g^0_{\pi(I)}(x_0)\Big)\,\Big(g^1_{\pi(S \setminus I)}(x_1)\Big). \tag{5}$$

A concrete example of these notations is a 2-input masked AND gate with inputs $x[0]$ and $x[1]$. For $k = 2$, the power set of $\{0, 1\}$ consists of four subsets: $\emptyset, \{0\}, \{1\}, \{0, 1\}$, indexed as: $\pi(\emptyset) = 0$, $\pi(\{0\}) = 1$, $\pi(\{1\}) = 2$, $\pi(\{0, 1\}) = 3$. The corresponding functions g^0 and g^1 are summarized below:

Subset I	$\emptyset$	$\{0\}$	$\{1\}$	$\{0,1\}$
$g^0_{\pi(I)}(x_0)$	1	$x_0[0]$	$x_0[1]$	$x_0[0] \cdot x_0[1]$
$g^1_{\pi(I)}(x_1)$	1	$x_1[0]$	$x_1[1]$	$x_1[0] \cdot x_1[1]$

Applying Eq. (5) to the monomial $\prod_{j \in \{0,1\}} x[j]$, we obtain:

$$\begin{aligned}
\prod_{j \in \{0,1\}} x[j] &= \sum_{I \subseteq \{0,1\}} g^0_{\pi(I)}(x_0) \cdot g^1_{\pi(\{0,1\} \setminus I)}(x_1) \\
&= \underbrace{g^0_0(x_0)\, g^1_3(x_1)}_{I=\emptyset} \;+\; \underbrace{g^0_1(x_0)\, g^1_2(x_1)}_{I=\{0\}} \;+\; \underbrace{g^0_2(x_0)\, g^1_1(x_1)}_{I=\{1\}} \;+\; \underbrace{g^0_3(x_0)\, g^1_0(x_1)}_{I=\{0,1\}} \\
&= x_1[0] \cdot x_1[1] \;+\; x_0[0] \cdot x_1[1] \;+\; x_0[1] \cdot x_1[0] \;+\; x_0[0] \cdot x_0[1] \\
&= \big(x_0[0] + x_1[0]\big)\big(x_0[1] + x_1[1]\big).
\end{aligned}$$

Applying TSM to a Single Coordinate

This decomposition is applied to each term in the ANF of $f[m]$. Summing over all terms $S \in \mathcal{M}_m$, we have:

$$\begin{aligned} f[m](x_0 + x_1) &= \sum_{S \in \mathcal{P}_k} \alpha_S \sum_{I \subseteq S} g^0_{\pi(I)}(x_0)\, g^1_{\pi(S \setminus I)}(x_1) \\ &= \sum_{S \in \mathcal{M}_m} \sum_{I \subseteq S} g^0_{\pi(I)}(x_0)\, g^1_{\pi(S \setminus I)}(x_1) \end{aligned} \tag{6}$$

The expression in Eq.(6) can be reorganized by subsets $I \in \mathcal{P}_k$ associated with $g^0_{\pi(I)}$. This reorganization will be essential later when we extend the approach to vectorial Boolean functions. In this reorganization, for each $I \in \mathcal{P}_k$, we capture the set of g^1 functions that need to be summed and multiplied with the corresponding $g^0_{\pi(I)}$ function.

$$\sum_{S \in \mathcal{M}_m} \sum_{I \subseteq S} g^0_{\pi(I)}(\cdot)\, g^1_{\pi(S \setminus I)}(\cdot) = \sum_{I \in \mathcal{P}_k} \Big(g^0_{\pi(I)}(\cdot) \cdot \sum_{\substack{S \in \mathcal{M}_m \\ I \subseteq S}} g^1_{\pi(S \setminus I)}(\cdot) \Big) \tag{7}$$

Specifically, for each $I \in \mathcal{P}_k$, the set of g^1 functions is constructed by selecting all $S \in \mathcal{M}_m$ where $I \subseteq S$, and including the functions $g^1_{\pi(S \setminus I)}$ in this set.

Formally, this set of g^1 functions can be defined using the indices:

$$J_{m,\pi(I)} = \big\{ \pi\big(S \setminus I\big) \;\big|\; S \in \mathcal{M}_m,\ I \subseteq S \big\}. \tag{8}$$

The set $J_{m,\pi(I)}$ represents those terms in the ANF of $f[m]$ (denoted by $\mathcal{M}_m$) that contribute to the product decomposition involving $g^0_{\pi(I)}$. The final expression of $f[m]$ is given by:

$$\begin{aligned} f[m](x_0 + x_1) &= \sum_{I \in \mathcal{P}_k} \Big(g^0_{\pi(I)}(\cdot) \cdot \sum_{\substack{S \in \mathcal{M}_m \\ I \subseteq S}} g^1_{\pi(S \setminus I)}(\cdot) \Big) \\ &= \sum_{I \in \mathcal{P}_k} \Big(g^0_{\pi(I)}(\cdot) \cdot \sum_{j \in J_{m,\pi(I)}} g^1_j(\cdot) \Big). \end{aligned} \tag{9}$$

If $J_{m,\pi(I)}$ is empty, it implies that the subset I does not contribute to the ANF of $f[m]$ under the specified conditions. Consequently, the sum $\sum_{j \in J_{m,\pi(I)}} g^1_j(\cdot)$ evaluates to zero.

First-Order Security

Next, to ensure first-order security, we refresh the outputs of g^0 with fresh random bits r, where $r = \{r_0, \ldots, r_{2^k-1}\}$. Specifically, the output of each $g^0_{\pi(I)}$ is refreshed by adding the corresponding random bit $r_{\pi(I)}$, leading to

$$f[m](x_0 + x_1) = \sum_{I \in \mathcal{P}_k} \Big(\big(g^0_{\pi(I)}(\cdot) + r_{\pi(I)}\big) \cdot \sum_{j \in J_{m,\pi(I)}} g^1_j(\cdot) \Big) + \sum_{I \in \mathcal{P}_k} \big(r_{\pi(I)} \cdot \sum_{j \in J_{m,\pi(I)}} g^1_j(\cdot) \big). \tag{10}$$

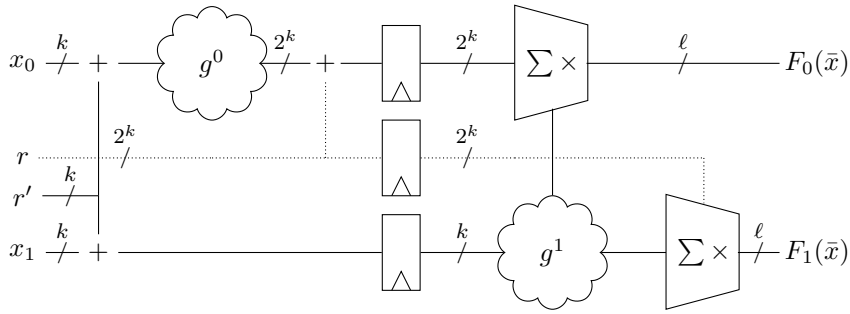


Figure 1: Application of TSM to an arbitrary (vectorial) Boolean function described by the set of functions g^0 and g^1.

We can split this expression into two phases, as depicted in Figure 1:

(i) computing and refreshing g^0 on the first share (with addition of r),

(ii) computing g^1 on the second share and recombining.

These steps allow us to safely construct the two output shares:

$$F_0[m] = \sum_{I \in \mathcal{P}_k} \Big(\big\{ g^0_{\pi(I)}(\cdot) + r_{\pi(I)} \big\} \cdot \sum_{j \in J_{m,\pi(I)}} g^1_j(\cdot) \Big), \tag{11}$$

$$F_1[m] = \sum_{I \in \mathcal{P}_k} \Big(\big\{ r_{\pi(I)} \big\} \cdot \sum_{j \in J_{m,\pi(I)}} g^1_j(\cdot) \Big).$$

where the curly brackets indicate storing a value in a register, which prohibits glitch propagation.

Lastly, before processing the gadget, the inputs x_0, x_1 themselves are refreshed with randomness $r' \in \mathbb{F}_2^k$ (i.e., $x_0 \leftarrow x_0 + r'$ and $x_1 \leftarrow x_1 + r'$) to ensure composable security.

Extending TSM to Vectorial Functions

We now generalize this approach to vectorial functions by ensuring that the masking process is applied efficiently across all coordinate functions $f[m]$, $m \in \{0, \ldots, \ell-1\}$. From Eq.(3), it is evident that all monomials up to degree k can be constructed using the functions g^0 and g^1. For each coordinate function $f[m]$, $m \in \{0, \ldots, \ell-1\}$, there is a corresponding set $J_{m,\pi(I)}$ determined by the terms that actually appear in $f[m]$, denoted by $\mathcal{M}_m$, see Eq.(8). Consequently, the same g^0 and g^1 building blocks can be efficiently re-used to extend TSM to a vectorial Boolean function ($f : \mathbb{F}_2^k \rightarrow \mathbb{F}_2^\ell$) by applying the above construction to each output coordinate $f[m]$.

To establish correctness, we demonstrate that the sum of the output shares reconstructs the original Boolean function f. The randomness $r_{\pi(I)}$, introduced during the refreshing of $g^0_{\pi(I)}$, cancels out between the two shares, leaving only the correctly reconstructed terms from the ANF of $f[m]$:

$$\begin{aligned} F_0[m] + F_1[m] &= \sum_{I \in \mathcal{P}_k} \left(\left\{ g^0_{\pi(I)}(\cdot) + r_{\pi(I)} \right\} + \left\{ r_{\pi(I)} \right\} \right) \cdot \sum_{j \in J_{m,\pi(I)}} g^1_j(\cdot) \\ &= \sum_{I \in \mathcal{P}_k} \left(g^0_{\pi(I)}(\cdot) \cdot \sum_{j \in J_{m,\pi(I)}} g^1_j(\cdot) \right) = f[m](x_0 + x_1). \end{aligned} \tag{12}$$

For vectorial functions $f : \mathbb{F}_2^k \rightarrow \mathbb{F}_2^\ell$, this reasoning extends independently to each coordinate $f[m]$, $m \in \{0, \ldots, \ell-1\}$. Consequently, the sum of the shares satisfies $F_0(\bar{x}) + F_1(\bar{x}) = f(x_0 + x_1)$.

3 Higher-Order Time Sharing Masking (HO-TSM)

We now extend TSM to achieve higher-order security while retaining the benefits of the first-order construction. We begin with a second-order secure AND gate as an illustrative example in Section 3.1. Next, we provide the construction of our higher-order secure method in Section 3.2. In Section 3.3, we prove that our

proposed solution is glitch-extended PINI and SNI secure. Finally, we apply our method to develop a two-cycle second-order masked AES S-Box in Section 3.4.

3.1 Preliminary Example

Before formally introducing HO-TSM, we illustrate its core concept with a simple example of a second-order secure AND gate. Consider two inputs, x and y, for the AND gate, producing an output $z = x \cdot y$. Assume x and y are shared as $\bar{x} = (x_0, x_1, x_2)$ and $\bar{y} = (y_0, y_1, y_2)$, respectively. The HO-TSM approach processes one share of each input per clock cycle while ensuring the isolation of computations using registers, as shown in Figure 2.

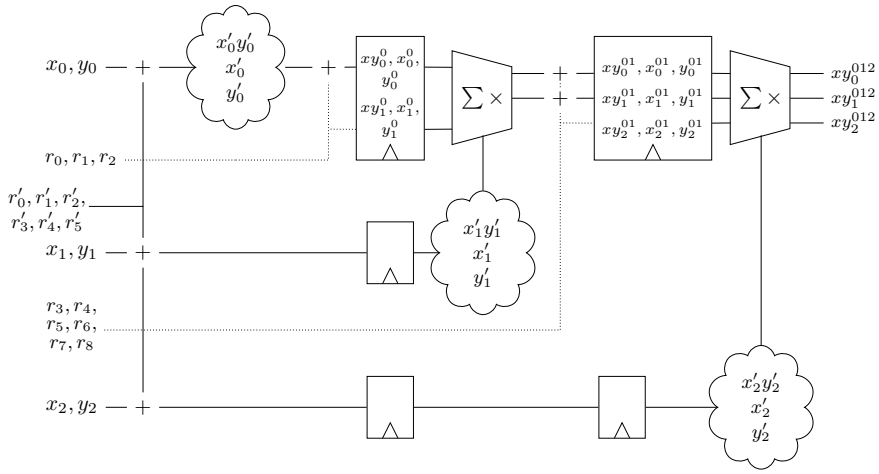


Figure 2: Application of HO-TSM$_2$: a second-order secure AND gate.

To ensure composable security, the input shares (x_0, x_1, x_2) and (y_0, y_1, y_2) are initially refreshed with six random bits $\{r'_0, r'_1, r'_2, r'_3, r'_4, r'_5\}$, as follows:

$\boldsymbol{x' = x'_0 + x'_1 + x'_2}$	$\boldsymbol{y' = y'_0 + y'_1 + y'_2}$
$x'_0 = x_0 + r'_0 + r'_1$	$y'_0 = y_0 + r'_3 + r'_4$
$x'_1 = x_1 + r'_0 + r'_2$	$y'_1 = y_1 + r'_3 + r'_5$
$x'_2 = x_2 + r'_1 + r'_2$	$y'_2 = y_2 + r'_4 + r'_5$

In the first phase, all cross-product combinations of $(x'_0, y'_0, x'_0 y'_0)$ are calculated, refreshed using random bits $\{r_0, r_1, r_2\}$, and stored in registers:

$\boldsymbol{xy^0 = x'_0 \cdot y'_0}$	$\boldsymbol{x^0 = x'_0}$	$\boldsymbol{y^0 = y'_0}$
$xy_0^0 = x'_0 y'_0 + r_0$	$x_0^0 = x'_0 + r_1$	$y_0^0 = y'_0 + r_2$
$xy_1^0 = r_0$	$x_1^0 = r_1$	$y_1^0 = r_2$

In the second phase, all cross-product combinations of $(x'_1, y'_1, x'_1 y'_1)$ are computed and combined with the register contents from Phase 1. This phase also increases the number of intermediate shares from two to three:

$\boldsymbol{xy^{01} = (x'_0 + x'_1) \cdot (y'_0 + y'_1)}$	$\boldsymbol{x^{01} = (x'_0 + x'_1)}$	$\boldsymbol{y^{01} = (y'_0 + y'_1)}$
$xy_0^{01} = xy_0^0 + x_0^0 \cdot y'_1 + y_0^0 \cdot x'_1 + r_3$	$x_0^{01} = x_0^0 + r_5$	$y_0^{01} = y_0^0 + r_7$
$xy_1^{01} = xy_1^0 + x_1^0 \cdot y'_1 + y_1^0 \cdot x'_1 + x'_1 y'_1 + r_4$	$x_1^{01} = x_1^0 + x'_1 + r_6$	$y_1^{01} = y_1^0 + y'_1 + r_8$
$xy_2^{01} = r_3 + r_4$	$x_2^{01} = r_5 + r_6$	$y_2^{01} = r_7 + r_8$

In the third phase, cross-product combinations of $(x'_2, y'_2, x'_2 y'_2)$ are computed and combined with the intermediate shares from Phase 2 to produce the final outputs:

$\boldsymbol{xy^{012} = (x'_0 + x'_1 + x'_2) \cdot (y'_0 + y'_1 + y'_2)}$
$xy_0^{012} = xy_0^{01} + x_0^{01} \cdot y'_2 + y_0^{01} \cdot x'_2$
$xy_1^{012} = xy_1^{01} + x_1^{01} \cdot y'_2 + y_1^{01} \cdot x'_2$
$xy_2^{012} = xy_2^{01} + x_2^{01} \cdot y'_2 + y_2^{01} \cdot x'_2 + x'_2 y'_2$

This computation requires 15 fresh random bits and 21 registers to store the intermediate shares, with a latency of two clock cycles. While the area and randomness costs of HO-TSM may be significant for a single AND gate, which serves merely as an illustrative example, the method is intended for use with complex non-linear functions, where its efficiency and advantages are most pronounced.

3.2 Construction

In this section, we provide a formal description of HO-TSM. The d-th order HO-TSM, denoted by HO-TSM$_d$, extends the principles of TSM to higher-order security. To provide a comprehensive understanding, we first present a general overview of HO-TSM. We then explain how to construct HO-TSM$_d$ step by step using a recursive approach.

HO-TSM$_d$ operates on $(d + 1)$ shares and requires d register stages (or equivalently, $(d + 1)$ phases). In each phase, the algorithm processes one share,

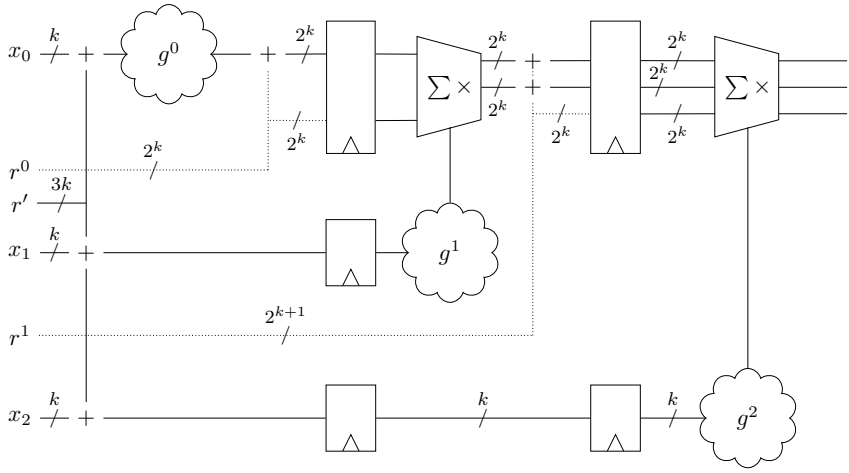


Figure 3: Application of HO-TSM$_2$ to an arbitrary (vectorial) Boolean function described by the functions g^0, g^1, and g^2.

computes all its monomial combinations denoted by g^i(functions operating on individual shares), and combines the result with the output from the previous phase stored in the register. This combination is then refreshed to generate an additional output share, except in the final phase, where the combination output serves as the final output of the gadget . The d-th register stage maintains a $(d+1)$-sharing of cross-products using shares $\{0, \ldots, d\}$.

For example, Figure 3 illustrates HO-TSM$_2$. In the first phase, share x_0 is processed to compute its monomials, which are refreshed to produce two output shares. In the subsequent phase, x_1 is processed and combined with the refreshed outputs from the first phase, resulting in three shares. Finally, in the third phase, share x_2 is processed and combined with the outputs of the previous phase to produce the final output shares of HO-TSM$_2$.

An intuitive way to understand the HO-TSM method is to recognize that it is constructed recursively from lower-order HO-TSM instances. As illustrated in Figure 4, HO-TSM$_d$ is built from HO-TSM$_{d-1}$, which securely operates on the first d shares $(x_0, \ldots, x_{d-1})$. A new set of functions, g^d, is then defined to operate exclusively on the additional share x_d. Finally, the outputs of HO-TSM$_{d-1}$ are combined with the outputs of g^d, merging the partial results to produce a secure $(d+1)$-share masking of the target function.

Consider an arbitrary Boolean function

$$f : \mathbb{F}_2^k \;\to\; \mathbb{F}_2^\ell, \quad x = \big(x[0], \ldots, x[k-1]\big) \;\mapsto\; f\big(x[0], \ldots, x[k-1]\big). \tag{13}$$

We denote its $(d+1)$-share masking by $\bar{F} : \mathbb{F}_2^{k(d+1)} \to \mathbb{F}_2^{\ell(d+1)}, \bar{x} \mapsto \big(F_0(\bar{x}), \ldots, F_d(\bar{x})\big)$, such that $F_0(\bar{x}) + F_1(\bar{x}) + \cdots + F_d(\bar{x}) = f\big(x_0 + x_1 + \cdots + x_d\big)$.

The description of HO-TSM builds on the concepts introduced in TSM, including the decomposition of terms, refreshing of shares, and the reformulation of the algebraic normal form (ANF) for masked computation. It is recommended to review Section 2.4 beforehand, as many of these concepts are directly reused and extended in this section.

Following the same strategy as in Section 2.4, we first focus on a single coordinate function $f[m]$, $m \in \{0, \ldots, \ell - 1\}$, and subsequently generalize to the full vectorial case. Extending to all ℓ outputs involves applying the same procedure efficiently to each coordinate function.

Decomposing Product Terms

In analogy to how TSM uses g^0 and g^1, we define two sets of functions:

(i) A set h operating on the first d share domains (0 to $d-1$),

(ii) A set g^d operating on the last (i.e., d-th) share domain.

The functions $h_{\pi(I)}$ compute cross-products across the first d shares, while $g^d_{\pi(I)}$ focuses on the monomials involving the d-th share exclusively. Concretely:

$$h_{\pi(I)}(x_0, \ldots, x_{d-1}) = \prod_{i \in I} \Big(x_0[i] + x_1[i] + \cdots + x_{d-1}[i] \Big), \quad g^d_{\pi(I)}(x_d) = \prod_{i \in I} x_d[i]. \tag{14}$$

Here, $I \subseteq \{0, \ldots, k-1\}$, and π is an indexing function over the power set of $\{0, \ldots, k-1\}$. To simplify notation, we use $h_{\pi(I)}(\cdot)$ to represent the function $h_{\pi(I)}$ applied to the first d shares. Similarly, $g^d_{\pi(I)}(\cdot)$ refers to $g^d_{\pi(I)}$ evaluated using the final share x_d. More details on the notations used can be found in Section 2.4.

As in the two-share case for first-order TSM, each shared term in $f[m]$ (of the form $\prod_{j \in J} x[j]$) can be decomposed into a sum of products that splits between the d-th share and the sum of the previous d shares. Formally, for $J \subseteq \{0, \ldots, k-1\}$,

$$\begin{aligned} \prod_{j \in J} x[j] &= \prod_{j \in J} \left((x_0[j] + \cdots + x_{d-1}[j]) + x_d[j] \right) \\ &= \sum_{I \subseteq J} h_{\pi(I)}(x_0, \ldots, x_{d-1}) \cdot g^d_{\pi(J \setminus I)}(x_d). \end{aligned} \tag{15}$$

To illustrate, consider the second-order masked AND gate introduced in Section 3.1, representing $x[0] \cdot x[1]$ with $k = 2$. The corresponding h-functions and g^2-functions for all subsets $I \in \mathcal{P}_2$ are summarized below:

Subset I	$\emptyset$ $(\pi(I) = 0)$	$\{0\}$ $(\pi(I) = 1)$	$\{1\}$ $(\pi(I) = 2)$	$\{0,1\}$ $(\pi(I) = 3)$
$h_{\pi(I)}(x_0, x_1)$	1	$(x_0[0] + x_1[0])$	$(x_0[1] + x_1[1])$	$(x_0[0] + x_1[0])(x_0[1] + x_1[1])$
$g^2_{\pi(I)}(x_2)$	1	$x_2[0]$	$x_2[1]$	$x_2[0] \cdot x_2[1]$

By applying Eq. (15), we decompose the term $\prod_{j\in\{0,1\}} x[j]$, where each $x[j]$ is $\big(x_0[j] + x_1[j] + x_2[j]\big)$:

$$\begin{aligned}\prod_{j\in\{0,1\}} \big(x[j]\big) &= \sum_{I\subseteq\{0,1\}} h_{\pi(I)}(x_0, x_1) \cdot g^2_{\pi(\{0,1\}\setminus I)}(x_2) \\ &= \underbrace{h_0(\cdot)\, g_3^2(\cdot)}_{I=\emptyset} + \underbrace{h_1(\cdot)\, g_2^2(\cdot)}_{I=\{0\}} + \underbrace{h_2(\cdot)\, g_1^2(\cdot)}_{I=\{1\}} + \underbrace{h_3(\cdot)\, g_0^2(\cdot)}_{I=\{0,1\}} \\ &= \big(x_0[0] + x_1[0] + x_2[0]\big)\big(x_0[1] + x_1[1] + x_2[1]\big).\end{aligned}$$

Decomposing a Single Coordinate

Analogous to Eq. (9) in the first-order TSM case, the decomposition in Eq. (15) is applied to each term in the algebraic normal form of $f[m]$. By utilizing the same sets $J_{m,\pi(I)}$ as defined in Eq. (8), we express:

$$f[m]\Big((x_0 + \cdots + x_{d-1}) + x_d\Big) \;=\; \sum_{I\in\mathcal{P}_k} \Big(h_{\pi(I)}(x_0, \ldots, x_{d-1}) \cdot \sum_{j\in J_{m,\pi(I)}} g_j^d(x_d)\Big). \tag{16}$$

The sets $J_{m,\pi(I)}$ from TSM can be directly reused in HO-TSM because they are constructed solely based on the structure of the ANF of $f[m]$, which remains unchanged regardless of the number of shares.

Recursive Construction of HO-TSM$_d$

Having decomposed $f[m]$ into the functions h and g^d, we now outline the recursive steps required to construct HO-TSM$_d$. These steps detail how the outputs of h and g^d are securely computed and combined to achieve the $(d+1)$-share masking of $f[m]$. As illustrated in Figure 4, a HO-TSM$_{d-1}$ block securely

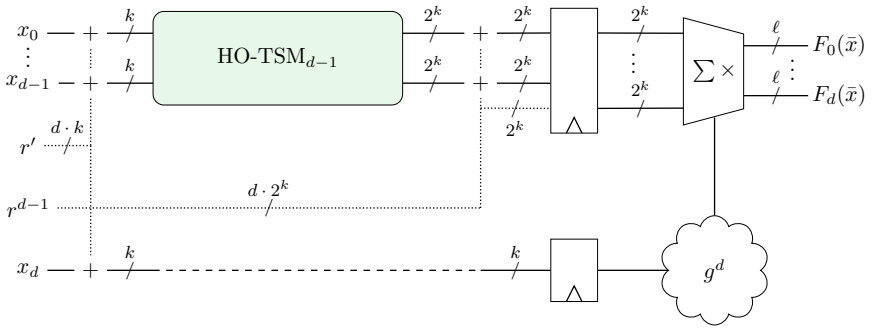


Figure 4: Recursive method of HO-TSM$_d$.

implements each of the $h_{\pi(I)}$ functions, shared across d shares, while the g^d cloud computes all the functions $g^d_{\pi(I)}$. The recursive process proceeds as follows:

1. The input shares $\big(x_0, \ldots, x_d\big)$ are first refreshed with randomness $r' = \big(r'_0, \ldots, r'_d\big)$ to ensure composable security. Each $r'_i \in \mathbb{F}_2^k$ for $i \in \{0, \ldots, d-1\}$ is an independent random value, while $r'_d = \sum_{i=0}^{d-1} r'_i$. The refreshed shares are $\big(x'_0, \ldots, x'_d\big)$, with $x'_i = x_i + r'_i$ for $i \in \{0, \ldots, d\}$.[2]

2. The first d refreshed input shares, $(x'_0, \ldots, x'_{d-1})$, are securely processed by the HO-TSM$_{d-1}$ block. This block computes the outputs of all $h_{\pi(I)}$ functions (as defined in Eq. (14)), producing intermediate d-shared outputs:

$$\overline{h_{\pi(I)}}(x'_0, \ldots, x'_{d-1}) = \big(h_{\pi(I)}(\cdot)|_0, h_{\pi(I)}(\cdot)|_1, \ldots, h_{\pi(I)}(\cdot)|_{d-1}\big),$$

   where $h_{\pi(I)}(\cdot)\big|_i$ represents the i-th share of the function $h_{\pi(I)}$, evaluated over the first d shares $(x'_0, \ldots, x'_{d-1})$.

3. Each of the d shares $h_{\pi(I)}(\cdot)\big|_i$ is refreshed using a dedicated random word r_i^{d-1}, where each r_i^{d-1} consists of 2^k bits (one random bit per subset $\pi(I)$). Altogether, $d \cdot 2^k$ fresh random bits are required in this step. Specifically, $h_{\pi(I)}(\cdot)\big|_i$ is refreshed as $\big(h_{\pi(I)}(\cdot)\big|_i + r_i^{d-1}[\pi(I)]\big)$. A corresponding new share $r_d^{d-1} = \sum_{i=0}^{d-1} r_i^{d-1}$ is also generated in this step.

4. The refreshed outputs

$$\Big\{ h_{\pi(I)}(\cdot)\big|_i + r_i^{d-1}[\pi(I)] \Big\} \quad \text{for } i = 0, \ldots, d-1$$

[2]Due to the recursive structure, the input shares may be re-shared multiple times. Although one could attempt to reduce this overhead, it is not the main source of randomness usage and the current approach allows for a clean recursion and straightforward security arguments.

along with the newly generated share r_d^{d-1}, are stored in a register. Consequently, the register now contains $(d+1)$ shares for each $h_{\pi(I)}$ function. This process ensures that $h_{\pi(I)}(\cdot)$, which was previously shared across d shares, is now securely represented in a $(d+1)$-share domain.

5. In the last phase, the remaining refreshed input share x'_d is processed to compute the outputs of the $g^d_{\pi(I)}$ functions. These results are then combined with the refreshed outputs of $h_{\pi(I)}$ functions, which were previously computed by the HO-TSM$_{d-1}$ block, to produce the final $(d+1)$-shared outputs $\big(F_0[m], \ldots, F_d[m]\big)$. The output equations are as follows:

$$F_i[m] = \sum_{I \in \mathcal{P}_k} \left(\left\{ h_{\pi(I)}(\cdot)\big|_i + r_i^{d-1}[\pi(I)] \right\} \cdot \sum_{j \in J_{m,\pi(I)}} g_j^d(x'_d) \right) \qquad \text{for } i < d,$$

$$F_d[m] = \sum_{I \in \mathcal{P}_k} \left(\left\{ r_d^{d-1}[\pi(I)] \right\} \cdot \sum_{j \in J_{m,\pi(I)}} g_j^d(x'_d) \right) \qquad \text{for } i = d. \tag{17}$$

The recursion proceeds iteratively, reducing the order at each step until it reaches the base case of first-order masking, HO-TSM$_1$ (i.e., TSM), which is securely implemented using the original first-order method of [41].

Extending HO-TSM to Vectorial Functions

Extending HO-TSM to vectorial Boolean functions ($f : \mathbb{F}_2^k \to \mathbb{F}_2^\ell$) involves applying the decomposition described for a single coordinate function $f[m]$, $m \in \{0, \ldots, \ell-1\}$, to each coordinate. The key observation is that the sets $J_{m,\pi(I)}$ (as defined in Eq. (8)), which determine which g^d-functions are summed together for each coordinate, need to be updated to reflect the ANF of each $f[m]$. Importantly, the sets $J_{m,\pi(I)}$ constructed for TSM can be directly reused in HO-TSM, as they depend solely on the structure of the ANF of $f[m]$. Furthermore, the refreshed and registered shares of $h_{\pi(I)}$, along with the functions $g^d_{\pi(I)}$, which decompose the terms across the $(d+1)$-share domain, are universal and can be reused across all coordinates of f. This ensures an efficient extension of HO-TSM to vectorial functions by leveraging the same decomposition principles, Eq. (16), and building blocks, Eq. (14), for all coordinates.

We provide the algorithmic costs of number of registers, register stages, and random bits in Table 1. Comparing HO-TSM with other methods is difficult as it easily becomes dependent on the actual function that is implemented. Nevertheless, we compare to the single cycle case of the GLM method where we

Method	# Registers	# Register Layers	# Random Bits
HO-TSM$_d$	$2^k \frac{d(d+3)}{2} + k\frac{d(d+1)}{2}$	d	$2^k \frac{d(d+1)}{2} + k\frac{d(d+1)}{2}$
GLM [21]	$(d+1)^k k$	1	$(d+1)^k k$

Table 1: Comparison of algorithmic costs of HO-TSM and GLM for order d when masking a function $\mathbb{F}_2^k \to \mathbb{F}_2^k$ function of algebraic degree $k-1$.

clearly see that we trade-off the number of registers and randomness for more register layers. A better view of the cost of HO-TSM is given in Section 3.4 where it is applied to create a second-order masked AES S-box.

3.3 Correctness and Security

We begin with the correctness and security proof for HO-TSM$_d$ and then provide SILVER results for some small examples.

Theorem 1. *Any circuit secured by HO-TSM$_d$ is correct.*

Proof. We establish the correctness of HO-TSM$_d$ by induction on d, the order of the masking.

- **Base Case** ($d = 1$): When $d = 1$, HO-TSM$_1$ is simply the first-order two-share masking (TSM). As shown in Section 2.4 and established in [41], we have, for each output coordinate m: $F_0[m] + F_1[m] = f[m](x_0 + x_1)$, and therefore $F_0(\bar{x}) + F_1(\bar{x}) = f(x_0 + x_1)$. Hence, correctness holds for $d = 1$.
- **Inductive Hypothesis:** Assume that HO-TSM$_{d-1}$, which produces d-shared outputs, satisfies correctness. That is, for some $(d-1) \geq 1$, the outputs $F_0[m], \ldots, F_{d-1}[m]$ satisfy $\sum_{i=0}^{d-1} F_i[m] = f[m](x_0 + x_1 + \cdots + x_{d-1})$, meaning each coordinate function $f[m]$ is correctly reconstructed from the sum of all d-output shares in the HO-TSM$_{d-1}$ design.
- **Inductive Step** (d): To complete the proof, we verify that HO-TSM$_d$ correctly produces $(d+1)$-shared outputs satisfying the required property. By construction:

    1. The input shares $(x_0, \ldots, x_d)$, refreshed with randomness $r' = (r'_0, \ldots, r'_d)$ for composable security, do not affect correctness as the refreshing process preserves $\sum_{i=0}^{d} x_i$. This is because $\sum_{i=0}^{d} r'_i = \sum_{i=0}^{d-1} r'_i + r'_d = 0$.

2. HO-TSM$_{d-1}$ is applied to the first d refreshed shares $(x'_0, \ldots, x'_{d-1})$, yielding intermediate outputs:

$$\overline{h_{\pi(I)}(\cdot)} = \left(h_{\pi(I)}(\cdot)\big|_0, h_{\pi(I)}(\cdot)\big|_1, \ldots, h_{\pi(I)}(\cdot)\big|_{d-1}\right),$$

whose sum of shares equals $\sum_{i=0}^{d-1} h_{\pi(I)}(\cdot)\big|_i = h_{\pi(I)}(x'_0, \ldots, x'_{d-1})$. By the inductive hypothesis, HO-TSM$_{d-1}$ is correct.

3. Each share $h_{\pi(I)}(\cdot)\big|_i$ is refreshed with randomness $r_i^{d-1}[\pi(I)]$, producing $\left\{h_{\pi(I)}(\cdot)\big|_i + r_i^{d-1}[\pi(I)]\right\}$. Adding the refreshed d-shares and the extra random share $r_d^{d-1} = \sum_{i=0}^{d-1} r_i^{d-1}$, the randomness cancels out, recovering: $\sum_{i=0}^{d-1}\left(h_{\pi(I)}(\cdot)\big|_i + r_i^{d-1}[\pi(I)]\right) + r_d^{d-1}[\pi(I)] = h_{\pi(I)}(x'_0, \ldots, x'_{d-1})$.

4. The functions $g^d_{\pi(I)}$, which operate exclusively on the last refreshed share x'_d, are then computed. These results are combined with the refreshed shares of $h_{\pi(I)}(\cdot)$, producing the final $(d+1)$-shared outputs $(F_0[m], \ldots, F_d[m])$. Summing the outputs gives:

$$\begin{aligned}\sum_{i=0}^{d} F_i[m] &= \sum_{I \in \mathcal{P}_k} \left(h_{\pi(I)}(x'_0, \ldots, x'_{d-1}) \cdot \sum_{j \in J_{m,\pi(I)}} g_j^d(x'_d)\right) \\ &= f[m](x'_0 + \cdots + x'_d) = f[m](x_0 + \cdots + x_d).\end{aligned}$$

This correctness argument applies independently to all coordinates $f[m]$, $m \in \{0, \ldots, \ell - 1\}$, as the sets $J_{m,\pi(I)}$ ensure proper reconstruction for each $f[m]$ based on its ANF. The final $(d+1)$-shared outputs for the vectorial function f are obtained by concatenating the outputs for all coordinates, ensuring that $F_0(\bar{x}) + F_1(\bar{x}) + \cdots + F_d(\bar{x}) = f(x_0 + x_1 + \cdots + x_d)$, where $F_i(\bar{x}) = \left(F_i[0], \ldots, F_i[\ell - 1]\right)$ for $i \in \{0, \ldots, d\}$.

□

Theorem 2. *Any circuit secured by HO-TSM$_d$ is d^{th}-order glitch-extended PINI and SNI.*

Proof. We prove the theorem by induction on d.

- **Base Case** ($d = 1$): From [41], it is established that the method is secure for $d = 1$. We observe in the proof that the simulation can be performed from scratch without giving any input shares when probing an output.
- **Induction Hypothesis:** Assume that any circuit secured by HO-TSM$_{d-1}$ is $(d-1)^{\text{th}}$-order glitch-extended PINI and SNI.

- **Induction Step** (d): We aim to show that if the theorem holds for HO-TSM$_{d-1}$, then it also holds for HO-TSM$_d$. Please refer to Figure 4 for reference. To demonstrate d^{th}-order PINI and SNI security, we use a simulation-based argument. First, we categorize all possible probe locations as follows:

  1. Probing the HO-TSM$_{d-1}$ block.
  2. Probing the ith share in the upper register.
  3. Probing the lower register.
  4. Probing the ith output share.

  Next, we construct the simulator $\mathcal{S}$. By the induction hypothesis, the HO-TSM$_{d-1}$ block is $(d-1)^{\text{th}}$-order PINI and SNI secure, and we can utilize its simulator $\mathcal{S}'$. Depending on the probe positions, we provide the following shares to $\mathcal{S}$:

  – Probes in Group 1: We provide $\mathcal{S}$ the same input which would be given to $\mathcal{S}'$.
  – Probes in Group 2: We provide the simulator with nothing.
  – Probes in Group 3: We provide the simulator with x_d.
  – Probes in Group 4: We provide the simulator with nothing.

  With the above input shares, the simulator $\mathcal{S}$ can perfectly simulate the probes as follows:

  – For any probe in Group 1: By the induction hypothesis, $\mathcal{S}'$ perfectly simulates this probe. When d probes are placed in Group 1 (or 2), since the share x_d is excluded, the simulator can be given $(x_0, \ldots, x_{d-1})$ to perfectly simulate the entire HO-TSM$_{d-1}$ block.
  – For any probe in Group 2: A probe in this group views a single output of the HO-TSM$_{d-1}$ block. The simulator $\mathcal{S}'$ can perfectly simulate this output due to its $(d-1)^{\text{th}}$-order PINI and SNI security. The simulator $\mathcal{S}$ also creates the randomness r_i^{d-1}. If $i = d$, $\mathcal{S}$ creates all random shares r_i^{d-1} without needing to simulate any output of the HO-TSM$_{d-1}$ block.
  – For any probe in Group 3: The simulator has access to x_d and creates the randomness r'_d.
  – For any probe in Group 4: A probe in this group views one share of the upper register. Due to the randomness r_i^{d-1}, each value in the upper register can be simulated as uniform randomness unless a probe has already been placed in that position in Group 2 or when a probe was

placed on r_d^{d-1}. In such cases, the simulation follows the logic of Group 2. Additionally, a probe in this group also views the lower register value. If a probe was placed in Group 3, the simulator has x_d and can simulate g^d. Furthermore, if d outputs of the HO-TSM$_{d-1}$ block need to be simulated, r'_d is also observed. However, since the adversary has only d probes, no probe could have been placed in Group 4 simultaneously. Otherwise, $r'_d = \sum_{i=0}^{d-1} r'_i$ remains unobserved, and the lower register can be simulated as uniform random, with g^d simulated appropriately.

The above demonstrates that the construction achieves d^{th}-order PINI and SNI security. Since both the base case and the induction step have been established, the theorem holds for all $d \geq 1$.

□

Formal Verification. SILVER [26] is a formal verification tool that evaluates the security and composability of masked gadgets against several security notions such as probing security, Non-Interference (NI), Strong Non-Interference (SNI), and Probe-Isolating Non-Interference (PINI). We applied our HO-TSM method to create two small examples for different orders of security: a HO-TSM$_2$ AND gate, introduced in Section 3.1, and a HO-TSM$_3$ AND gate. Our two examples successfully pass all second-order and third-order tests, respectively, under all security notions (probing, NI, SNI, PINI) in both the standard and robust probing models. This confirms that our two gadgets are indeed (2-PINI,2-SNI) and (3-PINI,3-SNI) secure.

3.4 Application: Second-Order Masked AES S-Box

An unprotected AES S-Box takes eight input bits and produces eight output bits, described as $S : \mathbb{F}_2^8 \to \mathbb{F}_2^8$. The coordinate functions, when presented in their algebraic normal form, include all terms up to the seventh degree, in total 254 terms. To mask the AES S-Box using HO-TSM$_2$, we begin by computing the sharings for all 254 terms by processing each share individually in three phases, see Figure 3. These sharings are then commonly reused across all eight coordinate functions.

In total, for the first stage, we require 254 random bits (r^0) and 508 (254×2) registers. In the second stage, we require 508 (254×2) random bits (r^1) and 762 (254×3) registers. Additionally, 24 random bits (r') are required for the initial input refresh. The total utilization cost for the second-order secure AES S-Box, including the cost for combinatorial logic, is summarized in Table 2.

Table 2: Utilization results of low-latency second-order masked AES S-Boxes.

Design	Method	Area (kGE)	Random bits	Cycles	Library
This work	**HO-TSM**	**33.7**[a] / **48.6**[b]	**786**	**2**	**NanGate45**
[21]	GLM	57.1	4446	2	UMC90

[a] `compile_ultra -no_autoungroup -no_boundary_optimization`
[b] `compile -exact_map`

The synthesis results are gathered using the NanGate 45nm Open Cell Library [34]. We use Synopsys DC Compiler v2021.06 for Synthesis and provide results for two different sets of options. The first set of options (`compile_ultra -no_autoungroup -no_boundary_optimization`) aims for maximum logic optimization while making sure that no logic is moved across the register layer by separating the registers from logic by placing them in distinct modules. The second set of options (`compile -exact_map`) aims for a direct mapping of our design to logic. The area numbers were gathered by synthesizing our designs with a target frequency of 100 MHz and do not include the area cost of generating the required randomness. Our two-cycle second-order AES S-Box has both lower area as well as randomness cost when compared to the GLM design proposed by Gross *et al.* [21].

4 An Area-Latency Trade-Off

In this section, we present a trade-off for HO-TSM which notably decreases both the area and randomness costs while increasing the latency by only a single cycle. In Section 4.1, we introduce our trade-off construction with a toy example for ease of understanding. Section 4.2 details the construction process for arbitrary functions. In Section 4.3, we provide proof that our trade-off construction achieves d^{th}-order PINI and SNI security. Sections 4.4 and 4.5 demonstrate the practical application of our method by constructing first- and second-order secure AES S-Boxes, showing improvements in reduced randomness and area. Finally, in Section 4.6, we construct a first-order round-based AES-128 and discuss the associated implementation costs.

4.1 Preliminary Example

We provide a toy example to illustrate the area-latency trade-off construction. While this approach is better suited for functions with a larger number of

inputs—such as the AES S-Box presented later in this section, which better highlights the efficiency gains—a smaller function simplifies the explanation and aids understanding. Consider a Boolean function fn that takes four input bits and produces one output:

$$fn(a,b,c,d) = a + d + ab + ac + bc + cd + abc + acd + abcd.$$

First, we partition the inputs into two subsets: $P = \{a, b\}$ and $Q = \{c, d\}$. A function gn is applied to each subset to generate the three outputs: $gn(P) = \{a, b, ab\}$ and $gn(Q) = \{c, d, cd\}$. To securely compute these outputs, we apply HO-TSM$_d$ to the two instances of gn, thereby producing the shared outputs P' and Q', as shown in Eq. (18):

$$\begin{aligned} P'_i &= \text{HO-TSM}_d(gn(P)) = \{(a)_i, (b)_i, (ab)_i\}, \quad i \in \{0, 1, \ldots, d\}, \\ Q'_j &= \text{HO-TSM}_d(gn(Q)) = \{(c)_j, (d)_j, (cd)_j\}, \quad j \in \{0, 1, \ldots, d\}. \end{aligned} \tag{18}$$

Next, we generalize the Domain-Oriented Masking (DOM) multiplier introduced by Gross *et al.*[22] to combine the two HO-TSM$_d$ blocks and compute the shared output of fn. Specifically, we define $(d+1)^2$ domains and perform share-wise multiplication for the terms in fn, distributing the results to their respective domains. For $d = 1$, the masked terms of fn are distributed across the domains as follows:

$$\begin{aligned} fn_{\{0,0\}} &= (a)_0 \cdot (c)_0 + (b)_0 \cdot (c)_0 + (ab)_0 \cdot (c)_0 + (a)_0 \cdot (cd)_0 + (ab)_0 \cdot (cd)_0 \\ &\quad + (a)_0 + (d)_0 + (ab)_0 + (cd)_0, \\ fn_{\{0,1\}} &= (a)_0 \cdot (c)_1 + (b)_0 \cdot (c)_1 + (ab)_0 \cdot (c)_1 + (a)_0 \cdot (cd)_1 + (ab)_0 \cdot (cd)_1, \\ \hline fn_{\{1,0\}} &= (a)_1 \cdot (c)_0 + (b)_1 \cdot (c)_0 + (ab)_1 \cdot (c)_0 + (a)_1 \cdot (cd)_0 + (ab)_1 \cdot (cd)_0, \\ fn_{\{1,1\}} &= (a)_1 \cdot (c)_1 + (b)_1 \cdot (c)_1 + (ab)_1 \cdot (c)_1 + (a)_1 \cdot (cd)_1 + (ab)_1 \cdot (cd)_1 \\ &\quad + (a)_1 + (d)_1 + (ab)_1 + (cd)_1. \end{aligned} \tag{19}$$

After computing the domains, we refresh them using $d(d+1)/2$ random bits, ensuring that each off-diagonal domain $fn_{\{i,j\}}$ is symmetrically refreshed with the same random bit. The domains are then registered and compressed to produce the $(d+1)$ output shares $\{fn_0, fn_1, \ldots, fn_d\}$. For $d = 1$, the final output shares $\{fn_0, fn_1\}$ are computed as:

$$fn_0 = \Big\{fn_{\{0,0\}}\Big\} + \Big\{fn_{\{0,1\}} + r\Big\},$$
$$fn_1 = \Big\{fn_{\{1,1\}}\Big\} + \Big\{fn_{\{1,0\}} + r\Big\},$$
where $r \xleftarrow{\$} \mathbb{F}_2$.

Formal Verification. In Section 4.3, we show that our DOM-based multiplication is sufficient to ensure composable security, due to the PINI and SNI composability properties of HO-TSM. To further validate the construction, we implemented the function fn for $d = 1$ and $d = 2$ and tested both implementations using SILVER. The two circuits passed all first- and second-order probing, NI, SNI, and PINI tests, respectively, in the standard and robust probing models, confirming the security of the construction.

4.2 Construction

The area and randomness requirements for HO-TSM$_d$ scale with the number of inputs, as detailed in Table 1. To improve efficiency, a trade-off is made by splitting the input space of a function into two (preferably equal-sized) parts and applying a HO-TSM$_d$ block to each. This approach provides a square root improvement in the area and randomness overhead for each block. However, an additional phase is necessary to combine the outputs of the two HO-TSM$_d$ blocks, which results in a one-cycle latency gain.

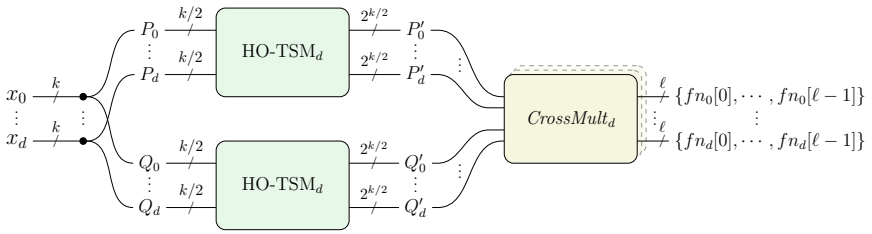


Figure 5: An area-latency trade-off construction for fn.

For an arbitrary Boolean function $fn : \mathbb{F}_2^k \to \mathbb{F}_2^\ell$, the construction applied to fn to provide d^{th}-order security proceeds as follows:

1. Let $gn : \mathbb{F}_2^n \to \mathbb{F}_2^{2^n}$ be a function that enumerates all monomials of its n-bit input. Concretely, for each subset $I \subseteq \{0, \ldots, n-1\}$, the corresponding output coordinate $gn[\pi(I)]$ is the product $\prod_{i \in I} x[i]$. Thus, gn outputs

every possible product of the input variables, including each input on its own.

2. Referring to Figure 5, we take a shared input $x = (x[0], x[1], \ldots, x[k-1])$, such that $\bar{x} = (x_0, x_1, \ldots, x_d)$, and partition it into two disjoint subsets: : $P = \{x[0], \ldots, x[\lfloor k/2 \rfloor - 1]\}$ and $Q = \{x[\lfloor k/2 \rfloor], \ldots, x[k-1]\}$. Each subset is processed by a secure implementation of gn using a HO-TSM$_d$ block. In other words, one HO-TSM$_d$ block takes the shares of P and produces shared outputs $P' = (P'_0, P'_1, \cdots, P'_d)$, while another HO-TSM$_d$ instance takes as input Q and produces $Q' = (Q'_0, Q'_1, \cdots, Q'_d)$.

   An indexing function π_P maps each $I \subseteq \{0, \ldots, \lfloor k/2 \rfloor - 1\}$ to a unique integer in $\{0, \ldots, 2^{k/2} - 1\}$. Consequently, $P'[\pi_P(I)]$ represents the $(d+1)$-shared representation of $\prod_{i \in I} x[i]$. Similarly, an indexing function π_Q is defined for subsets $J \subseteq \{\lfloor k/2 \rfloor, \ldots, k-1\}$, with $Q'[\pi_Q(J)]$ representing shares of $\prod_{j \in J} x[j]$.

3. We introduce a masked multiplier *CrossMult*$_d$(Algorithm 1), which then takes as inputs P' and Q' and produces the shared outputs of a coordinate function $fn[m] = (fn_0[m], fn_1[m], \ldots, fn_d[m])$, where $m \in \{0, 1, \ldots, \ell - 1\}$. *CrossMult*$_d$ involves $(d+1)^2$ domains, denoted as $fn_{\{i,j\}}[m]$, where each domain operates on a pair of share domains, P'_i and Q'_j. The domains are then refreshed, registered, and compressed to generate $(d+1)$ output shares of $fn[m]$.

4. To compute all ℓ coordinate functions of fn, *CrossMult*$_d$ is instantiated in parallel ℓ times, producing the output shares $(fn_0[m], fn_1[m], \ldots, fn_d[m])_{m=0}^{\ell-1}$.

4.3 Correctness and Security

We first establish the correctness of the area-latency trade-off construction, followed by its security proof.

Theorem 3. *Any circuit secured by the area-latency trade-off construction is correct.*

Proof. The correctness of HO-TSM$_d$ has already been proven in Theorem 1. Therefore, we assume that the outputs P' and Q' from the HO-TSM$_d$ blocks are correct and proceed to prove the correctness of *CrossMult*$_d$.

We begin by initializing every domain $fn_{\{i,j\}}[m]$ to zero. Next, the set of indices S for each term in $fn[m]$ must lie in one of two categories. In the first category, the indices of the term appear across both subsets $P = \{0, \ldots, \lfloor k/2 \rfloor - 1\}$ and

Algorithm 1 *CrossMult*$_d$

Input: $P' = (P'_0, P'_1, \cdots, P'_d), Q' = (Q'_0, Q'_1, \cdots, Q'_d)$
Output: $fn[m] = (fn_0[m], fn_1[m], \cdots, fn_d[m])$

for $i = 0$ **to** d **do** ▷ Initialize relevant domains for fn[m]
  for $j = 0$ **to** d **do**
    $fn_{\{i,j\}}[m] \leftarrow 0$
  end for
end for

for each $S \in \mathcal{M}_m$ **do** ▷ $\mathcal{M}_m$ is the set of all terms in $fn[m]$, see Eq.(2).
  $J_1 \leftarrow S \cap \{0, \ldots, \lfloor k/2 \rfloor - 1\}$
  $J_2 \leftarrow S \cap \{\lfloor k/2 \rfloor, \ldots, k-1\}$
  if $J_1 \neq \emptyset$ **and** $J_2 \neq \emptyset$ **then**
    $\bar{p} \leftarrow P'[\pi_P(J_1)]$ ▷ $\bar{p} = (p_0, p_1, \ldots, p_d)$
    $\bar{q} \leftarrow Q'[\pi_Q(J_2)]$ ▷ $\bar{q} = (q_0, q_1, \ldots, q_d)$
    for $i = 0$ **to** d **do**
      for $j = 0$ **to** d **do**
        $fn_{\{i,j\}}[m] \leftarrow fn_{\{i,j\}}[m] + p_i \cdot q_j$ ▷ Update corresponding domains
      end for
    end for
  else if $J_1 = \emptyset$ **then**
    $\bar{q} \leftarrow Q'[\pi_Q(J_2)]$ ▷ $\bar{q} = (q_0, q_1, \ldots, q_d)$
    for $j = 0$ **to** d **do**
      $fn_{\{j,j\}}[m] \leftarrow fn_{\{j,j\}}[m] + q_j$
    end for
  else if $J_2 = \emptyset$ **then**
    $\bar{p} \leftarrow P'[\pi_P(J_1)]$ ▷ $\bar{p} = (p_0, p_1, \ldots, p_d)$
    for $i = 0$ **to** d **do**
      $fn_{\{i,i\}}[m] \leftarrow fn_{\{i,i\}}[m] + p_i$
    end for
  end if
end for

for $i = 0$ **to** d **do** ▷ Refresh and store domains in registers
  for $j = i + 1$ **to** d **do**
    $r''_{\{i,j\}} \xleftarrow{\$} \mathbb{F}_2$
    $v_{\{i,j\}}[m] \leftarrow \left\{ fn_{\{i,j\}}[m] + r''_{\{i,j\}} \right\}$ ▷ $\left\{ \cdot \right\}$ denotes register storage
    $v_{\{j,i\}}[m] \leftarrow \left\{ fn_{\{j,i\}}[m] + r''_{\{i,j\}} \right\}$
  end for
end for

for $i = 0$ **to** d **do** ▷ Compression and final computation of $fn_i[m]$
  $fn_i[m] \leftarrow \left\{ fn_{\{i,i\}}[m] \right\} + \sum_{j=0, j\neq i}^{d} v_{\{i,j\}}[m]$
end for

$Q = \{\lfloor k/2 \rfloor, \ldots, k-1\}$. In that case, there exist partial terms $\bar{p} = P'[\pi_P(J_1)]$ and $\bar{q} = Q'[\pi_Q(J_2)]$ such that $\prod_{i \in S} x[i] = p \times q$. Expressing $\bar{p} = (p_0, \ldots, p_d)$ and $\bar{q} = (q_0, \ldots, q_d)$ in their shared form, the algorithm adds $p_i \cdot q_j$ into the domain $fn_{\{i,j\}}[m]$ for all i, j, including the diagonal when $i = j$.

In the second category, the term lies entirely within one subset (either P' or Q'), so the partial term is simply $\bar{p}$ or $\bar{q}$, and the algorithm places p_i or q_j only in the diagonal domain $\{i, i\}$ or $\{j, j\}$.

After these updates, each domain $\{i, j\}$ is combined with a random bit $r''_{\{i,j\}}$, which is also added to $fn_{\{j,i\}}[m]$. Because each $r''_{\{i,j\}}$ appears exactly twice symmetrically, these bits cancel out when all domains are summed together. Finally, every output $fn_i[m]$ is formed by gathering $fn_{\{i,i\}}[m]$ and the off-diagonal domains relevant to i, and then we sum over i. Concretely,

$$fn[m] \;=\; \sum_{i=0}^{d} fn_i[m] \;=\; \sum_{i=0}^{d} \sum_{j=0}^{d} fn_{\{i,j\}}[m].$$

In this double sum, each random bit $r''_{\{i,j\}}$ is added twice and thus vanishes, while every product $p_i \cdot q_j$ or pure partial term p_i or q_j appears exactly once. This ensures that $fn[m]$ is reconstructed without any missing or extra terms, establishing the correctness of $\mathit{CrossMult}_{\mathrm{d}}$. For vectorial Boolean functions $fn : \mathbb{F}_2^k \to \mathbb{F}_2^\ell$, $\mathit{CrossMult}_d$ processes each output coordinate $fn[m]$ independently and in parallel. By ensuring the correctness of each coordinate separately, the correctness of the entire vectorial function is guaranteed. □

Theorem 4. *The area-latency trade-off for order d is d^{th}-order PINI and SNI.*

Proof. The security of the area-latency trade-off construction relies on the security of HO-TSM$_d$ (Theorem 2) and that of $\mathit{CrossMult}_d$. We can categorize all possible probe locations into three groups as follows:

1. Probing any of the HO-TSM$_d$ blocks.
2. Probing the first stage of $\mathit{CrossMult}_d$. (Lines 1 to 35 in Algorithm 1)
3. Probing the second stage of $\mathit{CrossMult}_d$. (Lines 36 to 38 in Algorithm 1)

For each of these locations, we provide the following shares to the simulator:

- For any probe in Group 1, the simulator is given the share domain following the simulation for HO-TSM$_d$.
- For any probe in Groups 2 or 3, the simulator is given nothing.

We then demonstrate the simulation of the probed values:

- For any probe in Group 1: A probe in this group is simulated following the simulation procedure for HO-TSM$_d$.
- For any probe in Group 2: Since the first stage of *CrossMult*$_d$ is higher-order non-complete and operates on one share of each input, namely P'_i and Q'_j, the probe observes a single share domain of an output share from each HO-TSM$_d$ block. Recall that a probe on the output of an HO-TSM$_d$ block can be simulated without giving any input shares, owing to its d-th order PINI and SNI security. Consequently, as the output of each HO-TSM$_d$ block can be simulated without providing any information to the simulator, the simulation is perfect.
- For any probe in Group 3: The security is derived from the security of *CrossMult*$_d$. Specifically, according to the proof in Faust *et al.* [15, Proposition 3], a glitch-extended probe in the second stage of *CrossMult*$_d$ can be simulated with one share per input. Since this corresponds to probing a single output share from each HO-TSM$_d$ block, this can be simulated perfectly without giving the simulator any shares.

□

4.4 First-Order Secure AES S-Box

As our first case study, we apply our construction to build a first-order secure AES S-Box. The input bits $\{a, b, c, d, e, f, g, h\}$ are divided into two equal sets: $P = \{a, b, c, d\}$ and $Q = \{e, f, g, h\}$. In the initial step, we apply HO-TSM$_1$, the first-order case of TSM presented in [41], to both sets P and Q. The HO-TSM$_1$ blocks produce all degree-1, -2, -3, and -4 product terms as outputs.

The shared outputs of the first and second HO-TSM$_1$ blocks are:

$$\begin{aligned} P'_i = \{&(a)_i, (b)_i, (c)_i, (d)_i, (ab)_i, (ac)_i, (ad)_i, (bc)_i, (bd)_i, (cd)_i, \\ &(abc)_i, (abd)_i, (acd)_i, (bcd)_i, (abcd)_i\}, \quad i \in \{0, 1\} \\ Q'_j = \{&(e)_j, (f)_j, (g)_j, (h)_j, (ef)_j, (eg)_j, (eh)_j, (fg)_j, (fh)_j, (gh)_j, \\ &(efg)_j, (efh)_j, (egh)_j, (fgh)_j, (efgh)_j\}, \quad j \in \{0, 1\} \end{aligned} \tag{20}$$

The eight coordinate functions for the output of the AES S-Box in algebraic normal form contain 254 terms, including terms up to the seventh degree. A term is classified as an inner-module term if it includes inputs exclusively from

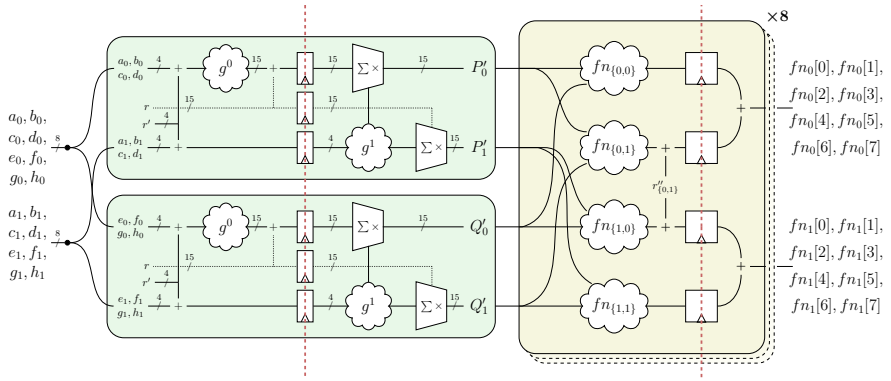


Figure 6: The area-latency trade-off for a first-order secure AES S-Box. Red vertical dashed lines indicate register stages.

either the set P or the set Q. For example, (abd) is an inner-module term. A cross-module term includes inputs from both sets P and Q. For example, $(abcfg)$ is a cross-module term. We define the first-order $CrossMult_1$ block, shown in Figure 6, using four domains: $fn_{\{0,0\}}[l], fn_{\{0,1\}}[l], fn_{\{1,0\}}[l], fn_{\{1,1\}}[l]$ for the eight coordinate functions, where $l \in \{0, 1, \ldots, 7\}$. The domain $fn_{\{i,j\}}[l]$ takes P'_i and Q'_j as inputs. Each cross-module term in the coordinate function $fn[l]$, such as $abcfg$, is calculated and distributed across all four domains. Specifically, $(abc)_0 \cdot (fg)_0$, $(abc)_0 \cdot (fg)_1$, $(abc)_1 \cdot (fg)_0$, and $(abc)_1 \cdot (fg)_1$ are assigned to domains $fn_{\{0,0\}}[l], fn_{\{0,1\}}[l], fn_{\{1,0\}}[l], fn_{\{1,1\}}[l]$ respectively. Each inner-module term in the coordinate function $fn[l]$ is distributed to domains $fn_{\{0,0\}}[l]$ and $fn_{\{1,1\}}[l]$. For example, $(abd)_0$ is assigned to $fn_{\{0,0\}}[l]$ and $(abd)_1$ to $fn_{\{1,1\}}[l]$. The four domains are subsequently refreshed using one random bit per coordinate function $fn[l]$ and stored in four registers: $v_{\{0,0\}}[l], v_{\{0,1\}}[l], v_{\{1,0\}}[l], v_{\{1,1\}}[l]$. The outputs of the S-Box are calculated straightforwardly by summing the contents of the registers: $fn_0[l] = v_{\{0,0\}}[l] + v_{\{0,1\}}[l]$ and $fn_1[l] = v_{\{1,0\}}[l] + v_{\{1,1\}}[l]$.

Implementation Costs. The area-latency trade-off for the first-order secure AES S-Box was synthesized using the same compiler options discussed in Section 3.4, targeting a clock frequency of 100 MHz. It occupies an area of 4657 GE with `compile` option and 4020 GE with `compile_ultra` option. Each HO-TSM$_1$ block occupies approximately 500 GE, aligning with the area reported for the PRINCE S-Box in [41], which also operates on four inputs. As for $CrossMult_d$, since the domains of all coordinate functions $\{fn_{\{i,j\}}[0], fn_{\{i,j\}}[1], fn_{\{i,j\}}[2], fn_{\{i,j\}}[3], fn_{\{i,j\}}[4], fn_{\{i,j\}}[5], fn_{\{i,j\}}[6], fn_{\{i,j\}}[7]\}$ operate on the same set

Table 3: Relevant First-Order Masked AES S-Box Implementations.

Design	Area (GE)	Rand (bits)	Latency (Cycles)	Library
LMDPL[40]	3480	36	1	GF28
TSM[41]	14352[a] / 20523[b]	262	1	NanGate45
GLM[21]	60730	2048	1	UMC90
$GHPC_{LL}$[27]	64111	2048	1	UMC180
TSM	**4020**[a] / **4657**[b]	**46**	**2**	**NanGate45**
LMDPL[30]	2825	36	2	-
GLM[21]	6740	416	2	UMC90
LLTI[1]	25780	0	2	NanGate45
TI[1]	58410	0	2	NanGate45
GHPC[27]	77145	8	2	UMC180
TI (B-P)[17]	2913	20	3	NanGate45
TI (Canright)[17]	2835	32	3	NanGate45
HPC (B-P)[8]	2780	46	4	NanGate45
HPC (Canright)[8]	1950	36	4	NanGate45

a `compile_ultra -no_autoungroup -no_boundary_optimization`
b `compile -exact_map`

of inputs P'_i and Q'_j, we group their combinatorial logic to further optimize for area and avoid any logic duplication. The S-box requires 46 random bits and has a latency of two cycles. Eight random bits are used for the initial refresh (r' as shown in Figure 6), fifteen random bits are used for refreshing the outputs of g^0 for each HO-TSM_1 block (r), and eight random bits are used for the eight $CrossMult_1$ blocks (r'').

Table 3 summarizes the area, latency and randomness requirements of various relevant works. Our AES S-Box demonstrates significant improvements over existing techniques. It is both smaller in area and demands considerably less randomness compared to GLM [21], while maintaining the same latency of two cycles. When compared to other PINI implementations such as HPC and GHPC, our AES S-Box achieves a latency reduction by half compared to the HPC S-Box and occupies significantly less area than the GHPC S-Box. Despite these advantages, our S-Box is slightly larger than the LMDPL S-Box [40] and it requires more randomness and an additional clock cycle. However, it is important to note that LMDPL employs dual-rail logic, which inherently requires a pre-charge phase between consecutive evaluations.

4.5 Second-Order Secure AES S-Box

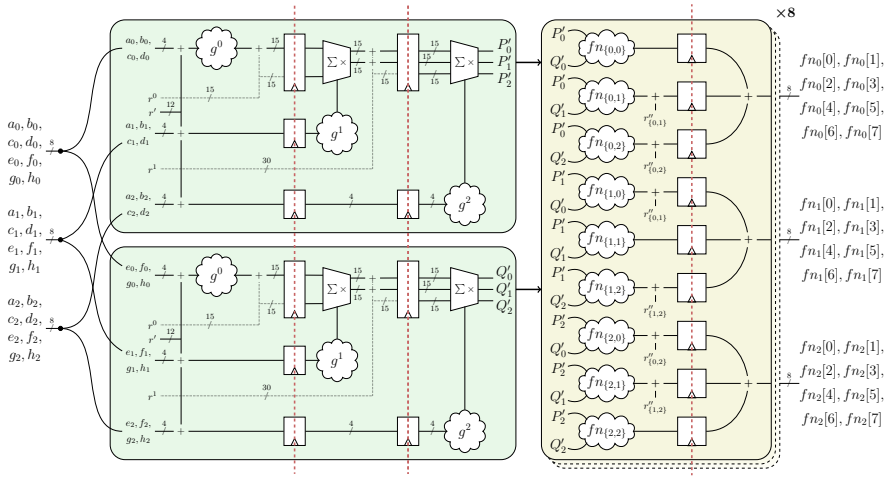


Figure 7: The area-latency trade-off for the second-order secure AES S-Box. Red vertical dashed lines indicate register stages.

In our second case study, we apply the area-latency trade-off for the second-order secure AES S-Box. The structure of the S-Box, as depicted in Figure 7, closely resembles that of the first-order S-Box described in Section 4.4. To avoid redundancy, we will focus more on the differences between the two. We implement the two HO-TSM$_2$ blocks using the second-order case of the HO-TSM methodology introduced in Section 3. These blocks accept inputs $P = \{a, b, c, d\}$ and $Q = \{e, f, g, h\}$ and generate all degree-1, -2, -3, and -4 product terms as outputs P'_i and Q'_j (where $i, j \in \{0, 1, 2\}$). These outputs are similar to Eq.(20), with each term represented using three shares instead of two. We define the second-order $CrossMult_2$ block, illustrated in Figure 7, using nine domains: $fn_{\{i,j\}}[l]$, where $i, j \in \{0, 1, 2\}$ for the eight coordinate functions. The cross-module terms are spread across all nine domains, while the inner-module terms are assigned to $fn_{\{0,0\}}[l]$, $fn_{\{1,1\}}[l]$, and $fn_{\{2,2\}}[l]$. Each of the nine domains is then refreshed using three random bits per coordinate function $fn[l]$ and stored in nine registers.

Implementation Costs. The area-latency trade-off of the second-order secure S-Box occupies an area of 9469 GE with the `compile` option and 8401 GE with the `compile_ultra` option. It requires 138 random bits in total. Twenty-four random bits are used for the initial refresh (r'), as shown in Figure 7. Fifteen random bits are used for refreshing the outputs of g^0 in the first stage, and

Table 4: Relevant Second-Order Masked AES S-Box Implementations.

Design	Area (GE)	Rand (bits)	Latency (Cycles)	Library
HO-TSM (Section 3.4)	**33700**[a] / **48600**[b]	**786**	**2**	**NanGate45**
GLM[21]	57110	4446	2	UMC90
HO-TSM (Section 4.5)	**8401**[a] / **9469**[b]	**138**	**3**	**NanGate45**
HPC (B-P)[8]	6790	138	4	NanGate45
HPC (Canright)[8]	4560	96	4	NanGate45
TI[13]	3796	162	5	NanGate45
DOM[18]	4300	78	5	UMC65
TI[12]	7849	126	6	NanGate45
DOM[22]	5300	54	8	UMC180

a `compile_ultra -no_autoungroup -no_boundary_optimization`
b `compile -exact_map`

thirty random bits are used for refreshing the outputs of the second stage (r^1) for each HO-TSM$_2$ block. Additionally, twenty-four random bits are used for the eight $CrossMult_2$ blocks (r''). The latency of this S-Box is three cycles, two register stages inside HO-TSM$_2$ and one inside $CrossMult_2$. In Table 4, we enumerate all relevant second-order masked AES S-Boxes. Our proposed S-Box demonstrates good performance compared to other designs, particularly when evaluated based on the product of area and latency. Additionally, our S-Box maintains a randomness requirement that is comparable to other designs. It is noteworthy that our construction improves both area efficiency and randomness requirements when compared to our two-cycle second-order secure S-Box described in Section 3.4, with the trade-off of one additional clock cycle.

4.6 Round-Based First-Order Secure AES-128

In order to evaluate the performance of the area-latency trade-off of the first-order secure AES S-Box in comparison to the state-of-the-art, we implemented a low-latency, round-based AES-128 design with parallel instances of our AES S-Box. Our architecture includes both a masked round function and a masked key schedule. We instantiate sixteen parallel instances of our S-Box for SubBytes in the round function and four instances for SubWord in the key expansion. Since our S-Box is glitch-extended PINI and SNI-secure, we trivially compose our S-Boxes with other linear elements such as MixColumns and ShiftRows and

avoid introducing any register layers outside the S-Box to store the intermediate state, which would increase latency.

Table 5: Utilization results of relevant first-order secure round-based AES-128 implementations.

Design	Method	Area (kGE)	Rand (bits/round)	Latency (Cycles)	Library
This Work	**TSM**	**91.8**[a] / **106.1**[b]	**920**	**20**	**NanGate45**
[40]	LMDPL	157.5	720	10	GF28
[2]	TI	115.7	0	42	NanGate45
[25]	HPC	44.5	1360	55	NanGate45

a `compile_ultra -no_autoungroup -no_boundary_optimization`
b `compile -exact_map`

Table 5 presents some relevant first-order, round-based AES-128 designs with low latency. Our area results were obtained with the target frequency set to 100 MHz. Notably, our design occupies a smaller area compared to both the TI [2] and LMDPL [40] designs. The TI S-Box (5.84kGE) is slightly larger than ours, which explains the difference in area between the designs. The LMDPL S-Box operates on dual-rail logic, requiring pre-charging before each evaluation. To avoid delays caused by pre-charging, the authors use a duality approach by duplicating the round function and state/key registers, creating two instances of the evaluation circuit. This method allows alternating evaluations: while one instance pre-charges, the other evaluates. Consequently, they achieve a latency of 10 cycles for the entire AES operation. Despite the smaller size of their LMDPL S-Box, duplicating the circuitry leads to a larger overall area compared to our design. Our design requires more randomness per round than the LMDPL design, whereas the TI design requires no online randomness at all. With a latency of twenty cycles, our design falls between the LMDPL and TI designs. The HPC design is the most compact among all designs, although it has higher latency compared to the others.

5 Security Evaluation

In this section, we conduct an experimental evaluation of the security of our designs, using both formal and practical verification methods. Specifically, in Section 5.1, we demonstrate the security of our AES S-Boxes using the formal verification tools `maskVerif` [3] and PROLEAD [33]. In Section 5.2, we

practically verify and present side-channel analysis results of our designs by implementing them on FPGA.

5.1 Formal Verification

Since SILVER is limited to verifying only small circuits and cannot handle larger designs, we use two other formal verification tools, `maskVerif` and PROLEAD, to verify the two-cycle, first-order AES S-Box (Section 4.4) and the three-cycle, second-order AES S-Box (Section 4.5).

`maskVerif` [3] is a language-based formal verification tool capable of verifying masked circuits under the t-probing, t-NI, and t-SNI security notions for hardware designs, with scalability for arbitrary orders. To verify our S-Boxes, we represented them in Static Single Assignment (SSA) form. For the SNI tests, defining more than one output at a time caused the tool to hang, so we designated each of the eight coordinate functions as the output, one at a time, and conducted eight separate verifications. The two S-Boxes successfully passed all (1-probing, 1-NI, 1-SNI) and (2-probing, 2-NI, 2-SNI) tests, respectively.

PROLEAD [33] is a simulation-based leakage detection tool that evaluates the statistical independence of intermediate probes in a masked circuit under the robust probing model. We generated netlists for the two S-Boxes using the `compile_ultra` command and performed fixed versus random G-tests in `compact` mode with default parameters. For the first-order S-Box, no leakage was detected across 200 million simulations, as the $-\log_{10}(p)$ value did not exceed the default threshold of 5.0 for any probing set, confirming its first-order probing security. For the second-order S-Box, we encounter a substantial number of probing sets (5.98×10^5), resulting in a higher probability of false positives, as noted by the authors of both [33] and [42]. We again conducted 200 million simulations in the multivariate setting(which took eight days), and the $-\log_{10}(p)$ values for almost all probing sets did not exceed the default threshold of 5.0, except for very few probing sets that have $5.0 < -\log_{10}(p) < 6.0$. However, we do not observe any continuous increase in $-\log_{10}(p)$ with an increasing number of simulations, which is ultimately the deciding factor for labeling a design as insecure [32]. Hence, we affirm the second-order probing security of our S-Box.

5.2 Practical Evaluation

For our larger designs, we revert to practical side-channel analysis by implementing three distinct protected designs on FPGA. These are:

- **Design A**: Two-cycle second-order secure AES S-Box (Section 3.4)
- **Design B**: Three-cycle second-order secure AES S-Box (Section 4.5)
- **Design C**: First-order secure round-based AES-128 (Section 4.6)

We now describe the measurement platforms, the measurement procedure, and finally present the leakage assessment results for each design.

Experimental Setup. We use two different FPGA boards, each with distinct advantages, in conjunction with Xilinx ISE v14.7.

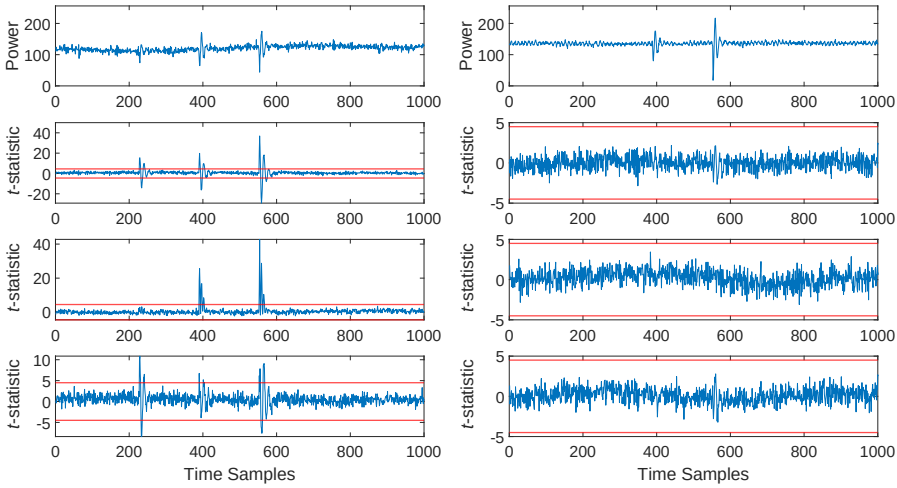
1. SAKURA-X [39]: This board features a Xilinx Kintex-7 FPGA, which provides sufficient resources for implementing larger designs. Consequently, we deploy our larger designs (A and C) on this board. To mitigate potential IO leakage, these designs receive pre-shared inputs from a secondary SPARTAN-6 control FPGA.
2. SAKURA-G [23]: Equipped with two Xilinx SPARTAN-6 FPGAs, this board generally achieves a better signal-to-noise ratio (SNR) for smaller designs, making it particularly suitable for Design B. Similar to SAKURA-X, one SPARTAN-6 FPGA handles the pre-shared inputs and outputs, while the other executes the protected design.

For power measurement, we capture the on-board amplified power via the J3 measurement point on SAKURA-G and via J19 with an additional 30 dB pre-amplifier on SAKURA-X. To prevent overlap in power traces between clock cycles, both boards operate at a conservative clock frequency of 3 MHz. This frequency is externally generated and synchronized with our Tektronix DPO7254C oscilloscope, enabling precise synchronous sampling at 500 MS/s.

Randomness Generation. For the two second-order secure AES S-Boxes (Designs A and B), we employ an unprotected AES-128 core in Output Feedback (OFB) mode on the control FPGA (SPARTAN-6) to generate the needed randomness. Each time the S-Box operates, the target FPGA fetches sufficient random bits from the control side.

For the first-order secure round-based AES-128 (Design C), we instantiate twelve unrolled Trivium [9, 6] cores on the Kintex-7 FPGA, each generating 64 bits of fresh randomness in parallel with the main AES core. This setup ensures that fresh random bits are available every clock cycle.

TVLA. We apply the non-specific Test Vector Leakage Assessment (TVLA) approach as outlined by Bilgin *et al.* [5] and originally introduced in [19, 14]. For efficiency, we adopt the fast computation method proposed by Reparaz *et al.* [38]. Fixed versus random tests are conducted by supplying either a fixed or random input in a randomly interleaved sequence. An absolute t-value threshold of ± 4.5 is used to identify potential leakage.

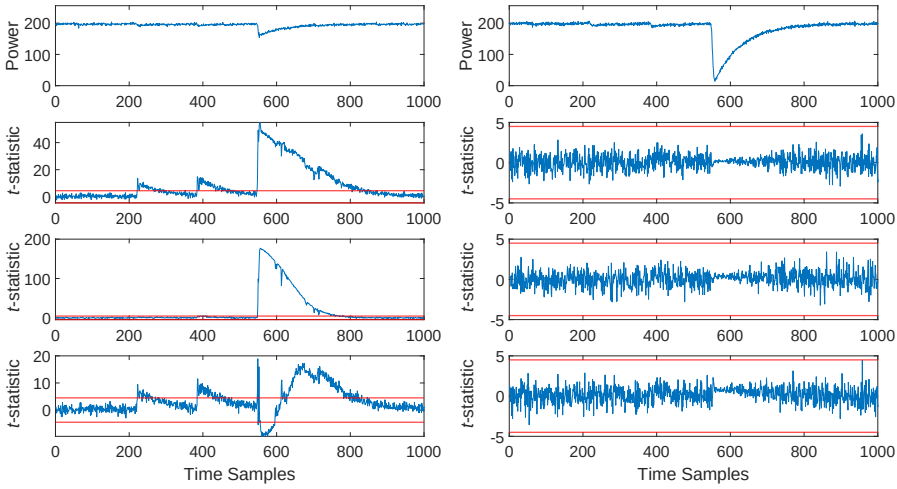


(a) PRNGs set to *OFF* with 100k traces.

(b) PRNGs set to *ON* with 100 million traces.

Figure 8: Univariate TVLA results for Design A: 2-cycle 2nd-order AES S-Box(Section 3.4) on SAKURA-X.(From top to bottom: power trace, 1st-, 2nd-, and 3rd-order t-statistics)

Validation with Disabled PRNGs. As the first step, we validate our setup by reporting leakage results with the PRNGs disabled. In this configuration, our designs operate without online randomness, and the randomness for the initial input sharing is set to zero. Figures 8a, 9a, and 10a present (from top to bottom) the power trace, and the first-, second-, and third-order univariate t-test statistics for Designs A, B, and C, respectively, based on 100 k traces. Red lines indicate the ± 4.5 threshold commonly used to identify leakage. The top row in each figure shows the sample power trace (raw oscilloscope ADC output) for the three designs. For Designs A and B, the power measurement in the top row corresponds to one AES S-Box computation. For Design C, the measurement covers the complete ten rounds of the protected round-based



(a) PRNGs set to *OFF* with 100k traces.

(b) PRNGs set to *ON* with 100 million traces.

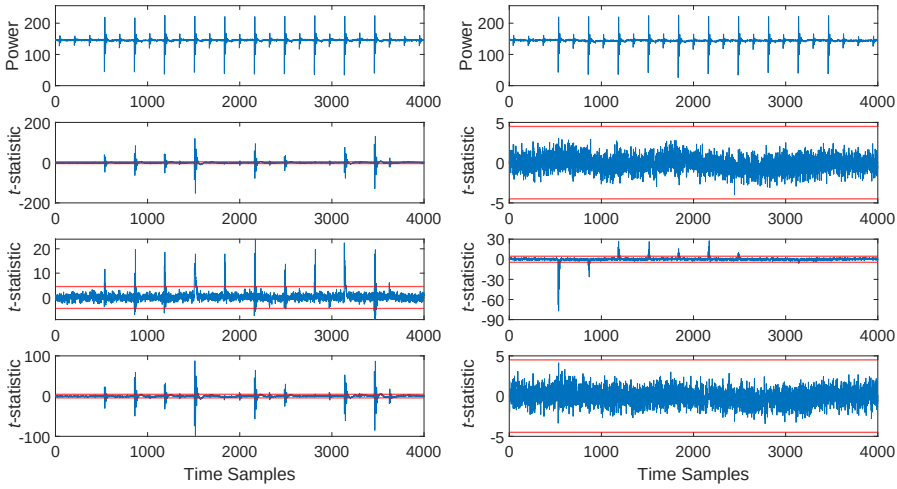
Figure 9: Univariate TVLA results for Design B: 3-cycle 2nd-order AES S-Box (Section 4.5) on SAKURA-G.(From top to bottom: power trace, 1st-, 2nd-, and 3rd-order t-statistics)

AES-128 core. As expected, we observe clear peaks in the first-order t-test, confirming the validity of our setup and measurement methodology.

Results with Enabled PRNGs. Figures 8b, 9b, and 10b show the leakage assessment results with PRNGs enabled.

- Designs A and B: We analyze 100 million traces. We observe no first- or second-order leakage, as the t-test values do not exceed the threshold of ± 4.5 at any time sample.
- Design C: We analyze 250 million traces. It is no surprise that we observe no first-order leakage and significant peaks for second-order leakage, both results are as expected. We expect all three designs to exhibit third-order leakage, but with the given number of traces they do not. One may speculate that the number of traces is not sufficient.

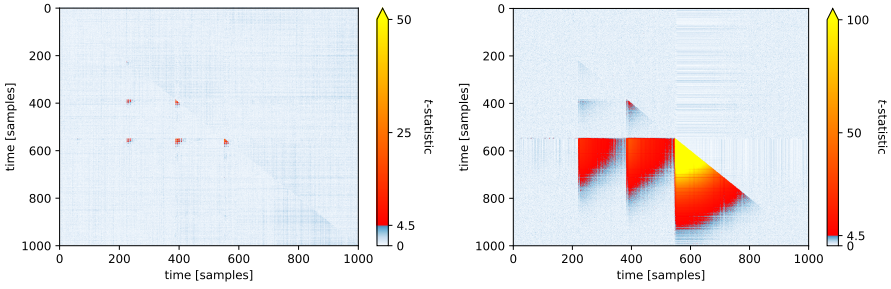
Bivariate Analysis. We also conduct bivariate analysis using the tool by Cassiers *et al.* [7] for our two second-order secure AES S-Boxes (Designs A and



(a) PRNGs set to *OFF* with 100k traces.

(b) PRNGs set to *ON* with 250 million traces.

Figure 10: Univariate TVLA results for Design C: 10 rounds of first-order secure round-based AES-128 (Section 4.6).(From top to bottom: power trace, 1st-, 2nd-, and 3rd-order t-statistics)



(a) Design A: Two-cycle 2nd-order AES S-Box(Section 3.4) on SAKURA-X.

(b) Design B: Three-cycle 2nd-order AES S-Box(Section 4.5) on SAKURA-G.

Figure 11: Bivariate analysis of our two 2nd-order AES S-Box designs. Bottom left triangle: PRNGs set to *OFF* with 100k traces. Top right triangle: PRNGs set to *ON* with 100 million traces.

B) using the same number of sample points as in the univariate tests, without downsampling. We verify the functionality of our setup by initially conducting tests with PRNGs disabled. The lower left triangles of Figures 11a and 11b

display the second-order t-test results based on 100k traces, revealing expected leakage. Interestingly, the time samples for the leaks (around 200, 400, 600) align closely with the univariate t-test results shown in Figures 8a and 9a, confirming the robustness of our setup. Finally, for both Designs A and B, we observe no leakage in the top right triangles when the PRNGs are enabled.

6 Conclusion

We introduce HO-TSM as a higher-order extension of TSM. The fundamental concept behind HO-TSM remains akin to TSM, ensuring security by separating share processing through the use of register layers. Our method is versatile, as it applies to any arbitrary (vectorial) Boolean function. We also establish the composable security of HO-TSM under the PINI and SNI security notions, which simplifies circuit construction by alleviating concerns about composition.

Furthermore, we introduce a trade-off between area and latency for HO-TSM that significantly improves both area and randomness efficiency. These enhancements are demonstrated by our low-latency first-order AES S-Box and the full AES-128 implementation. Our three-cycle second-order AES S-Box balances low-latency with moderate area and randomness requirements, offering a middle ground between designs that prioritize either latency reduction or minimizing area and randomness.

While our focus was on the AES S-Box as a case study, an intriguing direction for future research would be to implement various other S-Boxes and assess their performance relative to existing techniques.

Acknowledgments

This work was supported in part by the Flemish Government through the Cybersecurity Research Program with grant number VOEWICS02, by the European Commission through the Horizon 2020 research and innovation program under grant agreement Belfort ERC Advanced Grant 101020005 695305, and through the Horizon Europe research and innovation program under grant agreement HORIZON-CL3-2021-CS-01-02 101070008 ORSHIN.

References

[1] Arribas, V., Zhang, Z., and Nikova, S. LLTI: low-latency threshold implementations. *IEEE Trans. Inf. Forensics Secur. 16* (2021), 5108–5123.

[2] 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 - 21st International Conference, CARDIS 2022, Birmingham, UK, November 7-9, 2022, Revised Selected Papers* (2022), I. Buhan and T. Schneider, Eds., vol. 13820 of *Lecture Notes in Computer Science*, Springer, pp. 103–122.

[3] Barthe, G., Belaïd, S., Cassiers, G., Fouque, P., Grégoire, B., and Standaert, F. maskverif: Automated verification of higher-order masking in presence of physical defaults. In *Computer Security - ESORICS 2019 - 24th European Symposium on Research in Computer Security, Luxembourg, September 23-27, 2019, Proceedings, Part I* (2019), K. Sako, S. A. Schneider, and P. Y. A. Ryan, Eds., vol. 11735 of *Lecture Notes in Computer Science*, Springer, pp. 300–318.

[4] 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 *CCS* (2016), ACM, pp. 116–129.

[5] Bilgin, B., Gierlichs, B., Nikova, S., Nikov, V., and Rijmen, V. Higher-order threshold implementations. In *Advances in Cryptology - ASIACRYPT 2014 - 20th International Conference on the Theory and Application of Cryptology and Information Security, Kaoshiung, Taiwan, R.O.C., December 7-11, 2014, Proceedings, Part II* (2014), P. Sarkar and T. Iwata, Eds., vol. 8874 of *Lecture Notes in Computer Science*, Springer, pp. 326–343.

[6] Cannière, C. D. Trivium: A stream cipher construction inspired by block cipher design principles. In *Information Security, 9th International Conference, ISC 2006, Samos Island, Greece, August 30 - September 2, 2006, Proceedings* (2006), S. K. Katsikas, J. López, M. Backes, S. Gritzalis, and B. Preneel, Eds., vol. 4176 of *Lecture Notes in Computer Science*, Springer, pp. 171–186.

[7] Cassiers, G., and Bronchain, O. Scalib: A side-channel analysis library. *Journal of Open Source Software 8*, 86 (2023), 5196.

[8] Cassiers, G., Gigerl, B., Mangard, S., Momin, C., and Nagpal, R. Compress: Reducing area and latency of masked pipelined circuits. *IACR Cryptol. ePrint Arch.* (2023), 1600.

[9] Cassiers, G., Masure, L., Momin, C., Moos, T., Moradi, A., and Standaert, F.-X. Randomness generation for secure hardware masking - unrolled trivium to the rescue. Cryptology ePrint Archive, Paper 2023/1134, 2023. `https://eprint.iacr.org/2023/1134`.

[10] 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.

[11] Chari, S., Jutla, C. S., Rao, J. R., and Rohatgi, P. Towards sound approaches to counteract power-analysis attacks. In *Advances in Cryptology - CRYPTO '99, 19th Annual International Cryptology Conference, Santa Barbara, California, USA, August 15-19, 1999, Proceedings* (1999), M. J. Wiener, Ed., vol. 1666 of *Lecture Notes in Computer Science*, Springer, pp. 398–412.

[12] Cnudde, T. D., Bilgin, B., Reparaz, O., Nikov, V., and Nikova, S. Higher-order threshold implementation of the AES s-box. In *Smart Card Research and Advanced Applications - 14th International Conference, CARDIS 2015, Bochum, Germany, November 4-6, 2015. Revised Selected Papers* (2015), N. Homma and M. Medwed, Eds., vol. 9514 of *Lecture Notes in Computer Science*, Springer, pp. 259–272.

[13] 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.

[14] Cooper, J., De Mulder, E., Goodwill, G., Jaffe, J., Kenworthy, G., and Rohatgi, P. Test vector leakage assessment (TVLA) methodology in practice. `http://icmc-2013.org/wp/wp-content/uploads/2013/09/goodwillkenworthtestvector.pdf`, 2013.

[15] 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.

[16] Gandolfi, K., Mourtel, C., and Olivier, F. Electromagnetic analysis: Concrete results. In *Cryptographic Hardware and Embedded Systems - CHES 2001, Third International Workshop, Paris, France, May 14-16, 2001, Proceedings* (2001), Ç. K. Koç, D. Naccache, and C. Paar, Eds., vol. 2162 of *Lecture Notes in Computer Science*, Springer, pp. 251–261.

[17] Ghoshal, A., and Cnudde, T. D. Several masked implementations of the boyar-peralta AES s-box. In *Progress in Cryptology - INDOCRYPT 2017 - 18th International Conference on Cryptology in India, Chennai, India, December 10-13, 2017, Proceedings* (2017), A. Patra and N. P. Smart, Eds., vol. 10698 of *Lecture Notes in Computer Science*, Springer, pp. 384–402.

[18] Gigerl, B., Klug, F., Mangard, S., Mendel, F., and Primas, R. Smooth passage with the guards: Second-order hardware masking of the AES with low randomness and low latency. *IACR Trans. Cryptogr. Hardw. Embed. Syst. 2024*, 1 (2024), 309–335.

[19] Goodwill, G., Jun, B., Jaffe, J., and Rohatgi, P. A testing methodology for side channel resistance validation. `http://csrc.nist.gov/newsevents/non-invasive-attack-testing-workshop/08Goodwill.pdf`, 2011.

[20] Goubin, L., and Patarin, J. DES and differential power analysis (the "duplication" method). In *Cryptographic Hardware and Embedded Systems, First*

International Workshop, CHES'99, Worcester, MA, USA, August 12-13, 1999, Proceedings (1999), Ç. K. Koç and C. Paar, Eds., vol. 1717 of *Lecture Notes in Computer Science*, Springer, pp. 158–172.

[21] Gross, H., Iusupov, R., and Bloem, R. Generic low-latency masking in hardware. *IACR Trans. Cryptogr. Hardw. Embed. Syst. 2018*, 2 (2018), 1–21.

[22] 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 2016 Vienna, Austria, October, 2016* (2016), B. Bilgin, S. Nikova, and V. Rijmen, Eds., ACM, p. 3.

[23] Guntur, H., Ishii, J., and Satoh, A. Side-channel attack user reference architecture board sakura-g. In *2014 IEEE 3rd global conference on consumer electronics (GCCE)* (2014), IEEE, pp. 271–274.

[24] Ishai, Y., Sahai, A., and Wagner, D. A. Private circuits: Securing hardware against probing attacks. In *Advances in Cryptology - CRYPTO 2003, 23rd Annual International Cryptology Conference, Santa Barbara, California, USA, August 17-21, 2003, Proceedings* (2003), D. Boneh, Ed., vol. 2729 of *Lecture Notes in Computer Science*, Springer, pp. 463–481.

[25] 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.

[26] 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.

[27] Knichel, D., Sasdrich, P., and Moradi, A. Generic hardware private circuits towards automated generation of composable secure gadgets. *IACR Trans. Cryptogr. Hardw. Embed. Syst. 2022*, 1 (2022), 323–344.

[28] 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.

[29] Kocher, P. C., Jaffe, J., and Jun, B. Differential power analysis. In *Advances in Cryptology - CRYPTO '99, 19th Annual International Cryptology Conference, Santa Barbara, California, USA, August 15-19, 1999, Proceedings* (1999), M. J. Wiener, Ed., vol. 1666 of *Lecture Notes in Computer Science*, Springer, pp. 388–397.

[30] 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.

[31] Mangard, S., Popp, T., and Gammel, B. M. Side-channel leakage of masked CMOS gates. In *Topics in Cryptology - CT-RSA 2005, The Cryptographers' Track at the RSA Conference 2005, San Francisco, CA, USA, February 14-18, 2005, Proceedings* (2005), A. Menezes, Ed., vol. 3376 of *Lecture Notes in Computer Science*, Springer, pp. 351–365.

[32] Müller, N. Prolead results. `https://github.com/ChairImpSec/PROLEAD/wiki/Results`. Accessed: October 12, 2024.

[33] Müller, N., and Moradi, A. PROLEAD A probing-based hardware leakage detection tool. *IACR Trans. Cryptogr. Hardw. Embed. Syst. 2022*, 4 (2022), 311–348.

[34] NANGATE. The nangate 45nm open cell library. `https://www.nangate.com`.

[35] Nikova, S., Rechberger, C., and Rijmen, V. Threshold implementations against side-channel attacks and glitches. In *Information and Communications Security, 8th International Conference, ICICS 2006, Raleigh, NC, USA, December 4-7, 2006, Proceedings* (2006), P. Ning, S. Qing, and N. Li, Eds., vol. 4307 of *Lecture Notes in Computer Science*, Springer, pp. 529–545.

[36] 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.

[37] Reparaz, O., Bilgin, B., Nikova, S., Gierlichs, B., and Verbauwhede, I. Consolidating masking schemes. In *Advances in Cryptology - CRYPTO 2015* (2015), R. Gennaro and M. Robshaw, Eds., vol. 9215 of *Lecture Notes in Computer Science*, Springer, pp. 764–783.

[38] Reparaz, O., Gierlichs, B., and Verbauwhede, I. Fast leakage assessment. In *Cryptographic Hardware and Embedded Systems - CHES 2017 - 19th International Conference, Taipei, Taiwan, September 25-28, 2017, Proceedings* (2017), W. Fischer and N. Homma, Eds., vol. 10529 of *Lecture Notes in Computer Science*, Springer, pp. 387–399.

[39] SAKURA. Side-channel Attack User Reference Architecture. `https://satoh.cs.uec.ac.jp/SAKURA/hardware/SAKURA-X.html`.

[40] 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.

[41] V., D. K. S., Dhooghe, S., Balasch, J., Gierlichs, B., and Verbauwhede, I. Time sharing - a novel approach to low-latency masking. *IACR Transactions on Cryptographic Hardware and Embedded Systems 2024*, 3 (Jul. 2024), 249–272.

[42] Zhao, C., Zhao, H., Liu, J., Yang, B., Zhu, W., Yin, S., Zhu, M., Wei, S., and Liu, L. Breaking ground: A new area record for low-latency first-order masked sha-3: Advancing from the 4x area era to the 3x area era, Sep. 2024. Artifact available at `https://artifacts.iacr.org/tches/2024/a25`.

Statement on the use of Generative AI

I did use the generative AI tool ChatGPT solely for writing assistance. It was not used to generate or contribute to research content, code, figures, or any technical material. Its use was limited to improving clarity, refining phrasing, and enhancing the structure and flow of the text.
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

Dilip Kumar Shanmugasundaram Veeraraghavan received the Bachelor of Technology (B.Tech.) and Master of Technology (M.Tech.) degrees in Computer Science and Engineering from the Indian Institute of Technology Kharagpur in 2018 and 2019, respectively. Since 2019, he has been pursuing a Ph.D. in the Computer Security and Industrial Cryptography (COSIC) research group at KU Leuven, Belgium, under the supervision of Prof. Ingrid Verbauwhede and Dr. Benedikt Gierlichs. His research focuses on countermeasures against side-channel attacks, with a particular interest in efficient masking techniques for securing hardware implementations.

FACULTY OF ENGINEERING SCIENCE
DEPARTMENT OF ELECTRICAL ENGINEERING
Kasteelpark Arenberg 10, bus 2452
B-3001 Leuven
dshanmug@esat.kuleuven.be
https://www.esat.kuleuven.be/cosic/

