

UNIVERSITÀ DEGLI STUDI DI MILANO
DIPARTIMENTO DI INFORMATICA “Giovanni Degli Antoni”

CORSO DI DOTTORATO IN INFORMATICA
Settore Scientifico-Disciplinare: INF/01
XXXIV Ciclo



**On the security of cryptographic circuits:
protection against probing attacks and
performance improvement of garbled
circuits**

Maria Chiara Molteni

Tutor: Prof. Stelvio Cimato

Co-Tutor: Prof. Vittorio Zaccaria

Co-Tutor: Prof. Valentina Ciriani

Coordinatore del corso: Prof. Paolo Boldi

A.A. 2020/2021

UNIVERSITÀ DEGLI STUDI DI MILANO
DIPARTIMENTO DI INFORMATICA “Giovanni Degli Antoni”

CORSO DI DOTTORATO IN INFORMATICA
Settore Scientifico-Disciplinare: INF/01
XXXIV Ciclo



**On the security of cryptographic circuits:
protection against probing attacks and
performance improvement of garbled
circuits**

Tesi di: _____
Maria Chiara Molteni

Tutor: _____
Prof. Stelvio Cimato

Co-Tutor: _____
Prof. Vittorio Zaccaria

Co-Tutor: _____
Prof. Valentina Ciriani

Coordinatore del corso: _____
Prof. Paolo Boldi

A.A. 2020/2021

ABSTRACT

Dealing with secure computation and communication in hardware devices, an attacker that threatens to security of the systems can be of two different types.

The first type of attacker is external to the exchange of secret messages and tries to steal some sensitive information. Probing a circuit is a useful technique through which an attacker can derive information correlated with the secret manipulated by a cryptographic circuit. Probing security is the branch of research that tries to devise models, tools and countermeasures against this type of attacks. We define a new methodology that allows to determine if a gadget (i.e., a portion of a circuit) is secure against probing attacks. Moreover, we reason about composability of gadgets, in such a way that also this composition is probing secure. The reasoning is extended also to the case in which glitches are considered, namely when the attacks are mounted when timing hazards are present. The proposed methodology is based on the construction of the Walsh matrix of a Boolean function that describes the operations of the circuit. This method allows reaching an exact solution, but generally needs a lot of computation's time (mainly for big gadgets). To overcome the problem, we propose to compute the Walsh matrix exploiting the theory and applications of Algebraic Decision Diagrams (ADDs).

Different is the case when the malicious part is internal: each party is interested in protecting its own sensitive information from all the others. When the parties are only two, from literature the garbled circuit protocol is a solution that allows to reach a result implying some secrets, without sharing them. The cost of this protocol depends on the number of *and* gates in the circuit that implements the Boolean function describing the protocol computations. In this context, we work to reduce the number of multiplications in two classes of particular Boolean functions, called autosymmetric and D-reducible. Moreover, in the context of the garbled circuit

protocol, we discuss some innovative solutions to further reduce the protocol's costs, as the application of the 3-valued logic. This logic is an extension of the Boolean one, resulting from the addition of a new element to the set Boolean set $\{0, 1\}$.

CONTENTS

1	Introduction	1
1.1	Outline of Contents	2
1.2	Scientific Contributions	3
2	Cryptography and secret management	7
2.1	Mathematical context	7
2.1.1	Binary Field and Boolean Space	8
2.1.2	Boolean functions	9
2.1.2.1	Fourier transform and Walsh transform	9
2.1.3	Vectorial Boolean function	11
2.1.3.1	Walsh matrix	11
2.1.4	Tensor product between matrices	12
2.1.5	String diagram	13
2.1.6	Binary Decision Diagrams	13
2.1.6.1	Manipulation of BDDs	14
2.1.6.2	Algebraic Decision Diagrams	15
2.2	Cryptographic algorithms	16
2.2.1	Advanced Encryption Standard	17
2.2.2	Circuits	19
2.3	Protecting secrets from an external attacker	21
2.3.1	Side-channel attacks	22
2.3.2	Power Analysis Attacks	22
2.3.2.1	Countermeasures	26
2.3.3	Physical Defaults	29
2.4	Protecting secrets during computations	30

2.4.1	Secure computation	30
2.4.2	Garbled circuit	32
2.4.2.1	Oblivious transfer protocol	33
2.4.2.2	Cost of gates' transfer	34
2.4.2.3	Send <i>xor</i> gates for free	34
2.4.2.4	Reduction rows in the encrypted table	35
I	Protecting secrets from an external attacker	37
3	Probing security	39
3.1	State of the art	40
3.1.1	Probing security	40
3.1.1.1	The composability problem	43
3.1.2	Robust probing security	45
3.1.2.1	The robust composability problem	50
3.1.3	Verification tools	51
3.2	A relation calculus for reasoning about t -probing security	53
3.2.1	A relation calculus for shares	54
3.2.2	Application to t -probing security	59
3.2.3	Proving general patterns of compositional security	61
3.2.4	Extending the approach to $\mathbb{F}_{2^k}^n$: the AES inversion	67
3.2.4.1	Shares encoded over $\mathbb{F}_{2^k}^n$	67
3.2.4.2	Proof of strong non-interference	68
3.2.5	Appendix	68
3.2.5.1	A: Properties of the Walsh transform	68
3.2.5.2	B: Formal definition of shares' relation matrix	71
3.2.5.3	C: Relevant theorems and proofs - Section 3.2.2	76
3.3	On the Spectral Features of Robust Probing Security	77
3.3.1	Probing security as a relation calculus	79
3.3.1.1	The vulnerability profile of a function	79
3.3.1.2	Composition of vulnerability profiles	80
3.3.1.3	Extended probes	82
3.3.2	Definition of robustness	86
3.3.3	Revisiting the probing security of CMS	88
3.3.3.1	Achieving Robust Strong non-Interference for CMS	92
3.3.4	Analysis of the robust probing security of DOM-indep	93
3.3.5	On enabling general reasoning about non-interference	96

3.3.6	Computational complexity and scalability of the proposed approach	99
3.4	On robust strong-non-interferent low-latency multiplications	100
3.4.1	Overcoming the latest constructions	101
3.4.2	A provably robust- t -SNI, 1-cycle-latency CMS-like scheme	102
3.4.2.1	Saving randomness for $t \leq 4$	105
3.4.3	Applications	106
3.5	ADD-based Spectral Analysis of Probing Security	107
3.5.1	Methodology	108
3.5.1.1	Reading and "unfolding" the circuit description	108
3.5.1.2	Computing the Walsh Spectrum and the corresponding relation matrix	110
3.5.1.3	Interference check	110
3.5.2	Experimental results	111
3.6	Conclusion and further works	114

II Protecting secrets during computations 117

4 Multiplicative complexity, autosymmetric and dimension reducible

	Boolean functions	119
4.1	State of the art	120
4.1.1	Autosymmetric functions	121
4.1.2	D-reducible functions	124
4.1.3	Mockturtle tool	125
4.2	Multiplicative Complexity of Autosymmetric Functions: Theory and Applications to Security	126
4.2.1	Multiplicative Complexity of Autosymmetric Functions	126
4.2.2	Experimental results	129
4.3	Multiplicative Complexity of Regular Functions	132
4.3.1	Multiplicative Complexity of D-reducible Functions	132
4.3.2	Multiplicative complexity of D-reducible autosymmetric functions	136
4.3.3	Experimental results	137
4.3.3.1	Autosymmetric functions	137
4.3.3.2	D-reducible functions	140
4.3.3.3	Autosymmetric and D-reducible functions	142
4.4	Conclusion and further works	143

5	Multiple-valued logic	145
5.1	State of the art	145
5.1.1	MVL as generalization of the Boolean Logic	146
5.1.2	Lindell and Yanai's 3VL approach	150
5.1.3	Cimato et al.'s 3VL approach	156
5.2	A Multiple Valued Logic Approach for the Synthesis of Garbled Circuits	157
5.2.1	Multiple Valued approaches: a comparison	158
5.2.1.1	Free <i>xor</i> gates	158
5.2.1.2	Encodings in the FTU approach	158
5.2.2	Encodings in the 012 approach	159
5.2.2.1	Encodings' equivalences	162
5.2.2.2	More convenient encodings	164
5.2.3	Free <i>xor</i> Evaluation in Multiple Valued Logic	168
5.2.4	From 3VL to Boolean logic: costs comparison	169
5.2.5	Applied case: Adder	173
5.2.5.1	Costs for garbling the Full Adder circuit	175
5.2.5.2	More convenient encodings for the Full Adder	175
5.2.6	Appendix: Cheapest transformations for all the functional en- codings	177
5.3	Conclusion and further works	177
6	Conclusion and open problems	183
	Bibliography	187

LIST OF FIGURES

2.1	String diagrams for composition and tensor product between two functions.	14
2.2	BDD for $f(x_0, x_1, x_2) = x_0 \cdot x_1 \oplus x_2$	15
2.3	ADD for $f(x_0, x_1, x_2) = x_0 \cdot x_1 + x_2$	16
2.4	Simple circuit example.	21
2.5	Three cases of possible recombinations: combinatorial (glitches), memory (transitions), routing (coupling).	29
2.6	Schemes of parties role in SC protocols. (a) Homomorphic Encryption. (b) Linear Secret Sharing. (c) Garbled Circuit.	31
2.7	Garbled <i>or</i> gate.	33
3.1	Two state of the art probing secure circuits.	42
3.2	The composition pattern of f (t -NI) and g (t -SNI) studied in Example 3.1.2: the composed function $h(a)$ is not t -NI.	44
3.3	ISW <i>and</i> circuit, with $t = 1$, in case of glitches.	46
3.4	CMS scheme when $t = 2$, and then with 3 shares.	47
3.5	CMS scheme when $t = 3$, and then with 4 shares.	48
3.6	DOM scheme when $t = 1$, and then with 2 shares.	49
3.7	The composition pattern of f (t -NI) and g (t -SNI) studied in Example 3.2.7 and derived from [52]. The composed function $h(a)$ is not t -NI as can be easily checked with our formalism.	60

3.8	The shares' relation matrix of function h in example 3.2.7 derived from [52] (we use greek letters to indicate the spectral coordinate associated with each function variable, i.e., α is the spectral coordinate associated with variable a and so on). One can see that in row $[1, 1, 0]$, column $[0, 0, 3]$ there is a potential relation between two probes and the three shares of a , meaning that the composition is not even 2-NI.	61
3.9	Part of the shares' relation matrix of SecMult function [52] (only interesting rows for t -SNI are shown). Note that α, β and ρ are the spectral coordinates associated with inputs a, b and r , while ω and π are the spectral coordinates for o and p	63
3.10	Map between Fourier transforms of probability distributions implied by a function composition $l = g \circ f$	63
3.11	Map between Fourier transforms of probability distributions implied by the second composition pattern studied in this work	65
3.12	Example of reduction operation $u^{(k,n)\triangleright i}$. The new spectral coordinate binary encoding $u^{(4,2)\triangleright 2}$ is the result of OR'ing k -bit wide blocks of the original encoding u	67
3.13	An example application of the proposed formalism to functions over $\mathbb{F}_{2^k}^n$. Blocks m_4 and m_2 in (b) are structured as in (a). Note that we have slightly modified the algorithm presented in [99] by moving two power computation blocks across duplication points. Semantically, it is always the same circuit but it is easier to see how previously introduced patterns can still be used to show that it is t -SNI.	69
3.14	Example of compositional equality derived through a string diagram. The diagram on the left corresponds to the product $(W_g \otimes W_f)$ while the one on the right corresponds to $(1 \otimes W_f)(W_g \otimes 1)$ (each factor is highlighted with a dotted box). The fact that the second can be derived simply by moving boxes without crossing wires implies (because we are in monoidal category) that the underlying formulas are equivalent, i.e., $(W_g \otimes W_f) = (1 \otimes W_f)(W_g \otimes 1)$	72
3.15	The vulnerability profile of a function corresponds to the tensor product of the regular Walsh transform of a function and of its probes f_π , multiplied by W_δ	79
3.16	The composition of two vulnerability profiles as a map in the probability space.	81
3.17	The composition pattern of f (t -NI) and g (t -SNI) derived from [50].	81

3.18	Vulnerability profile of [50] (we use greek letters to indicate the spectral coordinate associated with each function variable, i.e., α is the spectral coordinate associated with variable a and so on).	82
3.19	The vulnerability profile of a register in terms of maps over the Fourier transform of input and output distributions.	84
3.20	The vulnerability profile of a composition of functions when considering extended probes.	85
3.21	The vulnerability profile of a composition of three functions when considering extended probes.	86
3.22	(a) shows the vulnerability profile of a composition of two functions when a register is considered in the middle. Probes that come after the "circuit breaker" map to the unit of $\mathbf{Vect}_{\mathbb{R}}$ and thus do not add any information so they have been drawn with a white circle. The Fourier transform of the output distribution is isomorphic to the one produced by the diagram in (b).	86
3.23	The four-share CMS scheme considered in [92]. The scheme is decomposed in three layers, non-linear (\mathcal{N}), refresh (\mathcal{R}) and compression (\mathcal{C}). To preserve output shares from the propagation of glitches, a register (thick line) layer is inserted between compression and refresh. Orange circles correspond to regular probes that break the t -probing security.	89
3.24	The robust 3-probing secure CMS scheme found with our formalization. Highlighted in orange the probes which make the above scheme not robust 3-SNI.	90
3.25	The vulnerability profile of the robust 3-probing secure CMS scheme found with our formalization. This has been computed only for a sum of the hamming weight of the output spectral coordinates (i.e., the sum of probes) equal to 3. Red circles indicate where the vulnerability profile fails to be robust 3-SNI because for $\omega_{f_\pi} = 2$ there can be a dependency with up to $\alpha = 2$ or $\beta = 2$	91
3.26	The robust 3-SNI CMS scheme proposed in Section. Additional randoms are identified with the label q_j . Other randoms have been grayed out to avoid crowding the image.	92
3.27	Vulnerability profile of CMS scheme with $s = 4$ when using additional randoms q_j	93
3.28	The DOM scheme for $t = 1, s = 2$	94
3.29	Vulnerability profiles of DOM without (left) and with (right) output register for $t = 1$ ($s = 2$).	95

3.30	Part of the vulnerability profiles of DOM without (left) and with (right) output register for $t = 2$ ($s = 3$).	96
3.31	Map between Fourier transforms of probability distributions implied by the considered example composition pattern.	97
3.32	The considered example composition pattern, gray boxes are registers.	97
3.33	Pruning of the a vulnerability profile considering equivalences and dominance relations of the correlation matrix calculus.	98
3.34	Estimated time needed to compute the vulnerability profile for well known algorithms.	100
3.35	1-cycle latency CMS-derived gadget proposed in [89]. Green discs represent the three extended probes that make it not robust 3-SNI. The black thick line indicates the register layer. The expressions to compute the outputs are those in Eq. 3.35 except that the values in red brackets are not sampled in an additional register, i.e., only those values in the black brackets are sampled.	101
3.36	A cone of the proposed robust t -SNI CMS structure which has still 1-cycle latency. Green discs represent the possible probes used in proposition 3.4.1. The black thick lines indicate register layers.	103
3.37	The optimized construction which is valid for any $t < 5$ but fails for $t \geq 5$. Green discs represent the probes used to mount the attack.	106
3.38	The DOM-1 multiplication circuit.	108
3.39	Annotated ILANG file	109
3.40	The methodology proposed in this paper.	109
3.41	Comparison of overall (left), convolution (middle) and verification (right) times between the method proposed in [89] (il) and the proposed method (mapi)	111
3.42	Comparison of overall computation times of the proposed method (mapi) and other implementations analysed in the experimental results.	113
4.1	XAG representation of the 4-input Boolean function.	121
4.2	Synthesis process, when autosymmetry test is applied to an autosymmetric function f .	122
4.3	Karnaugh map of a Boolean function that depends on 5 Boolean variables x_1, x_2, x_3, x_4, x_5 .	123
4.4	Karnaugh map of the restriction of the Boolean function in Figure 4.3, which depends on 4 Boolean variables y_1, y_2, y_3, y_4 .	124
4.5	Karnaugh maps of a D-reducible function f and its corresponding projection f_A .	125

4.6	A XAG for an autosymmetric function f obtained adding a <i>xor</i> level implementing the reduction equations to a XAG for the restriction f_k .	127
4.7	XAG representation for the benchmark <i>rd53</i> (second output), derived exploiting the autosymmetry of the function.	129
4.8	An XAG representation for a D-reducible function f obtained combining an XAG for the affine space A with a XAG for the projection f_A . Notice that only $\dim A$ of the n input variables are actual inputs for f_A .	133
5.1	Operations in the Kleene's logic K_3	148
5.2	Operations in the Bochvar's logic B_3	148
5.3	Orders of the truth values in Belnap's logic.	149
5.4	Operations in the Kleene's logic K_3	150
5.5	Operations in the 3VL with the FTU approach.	150
5.6	Encoding steps.	152
5.7	Embedded <i>and</i> and <i>or</i>	155
5.8	Embedded <i>xor</i>	155
5.9	Operations in the 3VL with the 012 approach.	156
5.10	Synthesis of function f in a 3VL circuit.	171
5.11	Synthesis of function f in the Boolean logic, after the application of the encoding in Eq. 5.3. In each dashed circle, there is an operation among \neg_2 , \wedge_2 or \oplus_2	172
5.12	Synthesis of function f in the Mixed logic, after the application of the encoding in Eq. 5.3 only for the gate \wedge_3 , that is transformed into \wedge_2 (dashed circle).	172
5.13	Full Adder circuit: function FA_3 has as inputs C_{i-1} , A_i and B_i , and as outputs S_i and C_i	173
5.14	Half Adder circuit: function HA_3 has as inputs α and β , and as outputs σ and χ	174

LIST OF TABLES

2.1	and operation in \mathbb{F}_2	8
2.2	xor operation in \mathbb{F}_2	8
2.3	not operation in \mathbb{F}_2	8
2.4	or operation in \mathbb{F}_2	8
2.5	True table of the Boolean function $f(x_0, x_1, x_2) = x_0\bar{x}_2 \oplus x_1$	10
2.6	Application of the Fourier and Walsh transforms to the Boolean function $f(x_0, x_1, x_2, x_3) = x_0x_1x_2 \oplus x_0x_3 \oplus x_1$	11
2.7	True table of the vectorial Boolean function $f(x_0, x_1, x_2) = [x_0x_1, x_1 \oplus \bar{x}_2]$	12
2.8	Walsh matrix of the vectorial Boolean function $f(x) = [x_0x_1 \oplus x_2, x_0x_2 \oplus x_3, x_1 \oplus x_3]$ with $x = [x_0, x_1, x_2, x_3] \in \mathbb{F}_2^4$ and $f_0(x) = x_0x_1 \oplus x_2$, $f_1(x) = x_0x_2 \oplus x_3$, $f_2(x) = x_1 \oplus x_3$	12
2.9	Symbols that correspond to the gates of circuits, implementing the Boolean operations.	20
2.10	Alice's computations.	33
3.1	Algebraic composition rules for probes.	85
3.2	Meaning of some mathematical symbols employed in the text.	103
3.3	Results of the comparison between our methodology and lists of lists implementation. Values in third and fourth columns are in seconds.	112
3.4	Evaluation of different implementation choices. Values from third to sixth columns are in seconds.	113
3.5	Comparison between mapi and the state of the art tools in [7], [26] and [75]. Values from third to sixth columns are in seconds.	114

4.1	Experimental comparison of autosymmetric benchmarks, considering a XAG after the autosymmetry test and the standard XAG computed without the autosymmetry test.	130
4.2	Summary of the experimental evaluation, considering the number of <i>ands</i> in the XAGs for autosymmetric functions and non-degenerate autosymmetric functions.	131
4.3	Experimental comparison of autosymmetric benchmarks, considering an XAG after the autosymmetry test and the standard XAG computed without the autosymmetry test.	139
4.4	Summary of the experimental evaluation, considering the number of <i>ands</i> in the XAGs for autosymmetric functions and non-degenerate autosymmetric functions.	139
4.5	Experimental comparison of D-reducible benchmarks, considering XAGs computed exploiting the D-reducibility property with standard XAGs.	141
4.6	Summary of the experimental evaluation, considering the number of <i>ands</i> in the XAGs for D-reducible functions.	141
4.7	Summary of the experimental evaluation, considering the number of <i>ands</i> in the XAGs for autosymmetric and D-reducible functions. . . .	142
5.1	Two-inputs MVL operations.	146
5.2	Naive garbling for a 3VL gate.	151
5.3	FTU approach: cost of each translated operations in the Boolean logic, in case of the natural encoding.	153
5.4	FTU approach: cost of each translated operations in the Boolean logic, in case of the functional encoding.	154
5.5	FTU approach: cost of each translated operations in the Boolean logic, in case of the non-functional encoding.	155
5.6	A comparison between multiple valued approaches.	159
5.7	$Tr_{3 \rightarrow 2}$ functions, describing all functional encoding in the 012 approach.	160
5.8	012 approach: cost of translated \oplus_3 operation in the Boolean logic, in case of the example functions.	162
5.9	Boolean gates needed to compute the operation \wedge_2 for every functional encoding.	165
5.10	Boolean gates needed to compute the operation \oplus_2 for every functional encoding.	166
5.11	Boolean gates needed to compute the operation \neg_2 for every functional encoding.	167
5.12	Definition of \oplus_6 using a truth table.	169

5.13	Number of rows in the garbled tables for each gate (columns) in all the three scenarios (rows).	171
5.14	Comparison of costs.	171
5.15	Steps performed to sum the sequences $A = 0120$ and $B = 0122$.	174
5.16	Total points awarded to the Half Adder and the Full Adder, for each encoding, following the rule: 1 point to a \neg gate, 2 points to a \oplus gate and 3 points to an \wedge gate.	176
5.17	Cheapest transformations for \neg_3 .	178
5.18	Cheapest transformations for \wedge_3 , for encodings with $0_L = 0$.	179
5.19	Cheapest transformations for \wedge_3 , for encodings with $0_L = 1$.	180
5.20	Cheapest transformations for \oplus_3 .	181

CHAPTER 1

INTRODUCTION

Cryptography is the theory that studies how to protect sensitive information during computation/communication. An attacker is the party that tries to recover some knowledge about the implied secrets, without the consensus of the owner. In this work, we study the nature of the attacker in two different cases, when the cryptographic algorithms are implemented in hardware devices.

When the attacker is external to the communication, she tries to steal some sensitive information during the exchange of secret messages between other two (or more) entities. In this specific context, the main treated topic in this work is the protection of circuits against probing attacks. Probing a circuit means to read intermediate bits on the circuit's wires through metal needles placed on them, or more generally measuring the power consumption or EM emissions from a subset of nodes of a circuit. It is a useful technique through which an attacker can derive information correlated with the secret manipulated by a cryptographic circuit. Probing security is the branch of research that tries to devise models, tools, and countermeasures against this type of attacks. In this work, we define a new methodology, that allows to prove if a gadget (i.e., portion of a circuit) is secure against probing attacks. Moreover, we also reason about composability of gadgets, in such a way that the composition is probing secure. This study is extended also to the case in which glitches are considered, namely when the attacks are mounted when timing hazards are present. The proposed methodology is based on the construction of the Walsh matrix of a Boolean function that describes the operations of the circuit: it allows reaching an exact solution but needs a prohibitive computation's time for big gadgets. To overcome the problem, we discuss a new way to compute the Walsh matrix, exploiting the theory and applications of Algebraic Decision Diagrams (ADDs).

Different is the case when the malicious part is internal: each participant to the communication is interested in protecting their own sensitive information from all the others. In the two secure computation theory, a solution to this issue is the garbled circuit protocol. The cost of this protocol depends on the number of *and* gates in the circuit that implements the Boolean function describing the protocol computations. In this context, we work to reduce the number of multiplications (i.e., multiplicative complexity property) in two classes of particular Boolean functions, called autosymmetric and D-reducible. Moreover, in the context of the garbled circuit protocol, some innovative solutions are discussed, as the application of the three valued logic. This logic is an extension of the Boolean one, resulting from the addition of a new element to the set containing the classical *true* and *false*. The study of the three valued logic in the garbled circuit protocol context is possible because here the circuits are implemented in software, and it aims to further reduce the protocol's costs.

1.1 Outline of Contents

After Chapter 2, the material is split into two parts, the first one containing only one chapter, and the second one holding two chapters.

Chapter 2 In this Chapter, we introduce all the mathematical notions needed to understand the following chapters. Some other general concepts are reported, divided into those that need for the comprehension of part I and those for Part II.

I part - Protecting the secret from an external attacker

In this part of the thesis, we treat the following case: two or more entities try to communicate securely, handling sensitive messages, and an external entity (the attacker) tries to recover some information about the secrets.

Chapter 3 In this Chapter we analyze the topic of the probing security, starting from an exhaustive state of the art, and presenting also all the work done in our published works in this context.

II part - Protecting the secret during computations

In this part of the thesis, the malicious entity is part of the communication. Indeed, any participants want to keep their own sensitive information hidden from the others. In particular, in the context of the two-party secure computation, the attention is focused on the garbled circuit protocol.

Chapter 4 In this Chapter we treat the topic of the multiplicative complexity, considering in particular the autosymmetric and D-reducible Boolean functions' case of study. About this topic, we have published two papers, that are reported at the end of the Chapter.

Chapter 5 In this Chapter we analyze two different definitions of three-valued logic, and their application to the garbled circuit protocol. Our submitted work about this topic is also presented.

1.2 Scientific Contributions

I part - Protecting the secret from an external attacker

- *A relation calculus for reasoning about t -probing security* [90]. We introduce a calculus for mechanically reasoning about the shares of a variable and show that this formalism provides a lean algebraic explanation of known compositional patterns allowing for the discovery of new ones. Eventually, we show how this formalism can be applied to study the probing security of known cryptographic gadgets.
- *On the spectral features of robust probing security* [89]. We expand the spectral formalization of non-interference discussed in the previous work, investigating the case in presence of glitches. Our goal is to present new theoretical and practical tools to reason about robust- d -probing security. We show that the current understanding of extended probes lends itself to probes that participate, during gadget composition, in the creation of additional extended probes. In turn, this enables a natural extension of non-interference definitions into robust ones to build a new reasoning framework that can formally explain some semi-formal results already appeared in the past and be used to synthesize new robust- d -SNI gadgets.
- *On robust strong-non-interferent low-latency multiplications* [88]. The overarching goal of our work is to present new theoretical and practical tools to

implement robust t -probing security. In this paper, we present a low-latency multiplication gadget that is secure against probing attacks that exploit logic glitches in the circuit. The gadget is the first of its kind to present a 1-cycle input-to-output latency while belonging to the class of *probing security by optimized composition* gadgets [39]. We show that it is possible to construct robust t -SNI gadgets without compromising on latency with a moderate increase in area. We provide a theoretical proof for the robustness of the gadget and show that, for $t \leq 4$, the amount of randomness required can even be reduced without compromising on robustness.

- *ADD-based Spectral Analysis of Probing Security*. In this paper, we introduce a novel exact verification methodology for non-interference properties of cryptographic circuits. The methodology exploits the Algebraic Decision Diagram representation of the Walsh spectrum to overcome the potential slow down associated with its exact verification against non-interference constraints. Benchmarked against a standard set of use cases, the methodology speeds-up 1.88x the median verification time over the existing state-of-the-art tools for exact verification. This work has been accepted at the Design, Automation and Test in Europe Conference (DATE2022).

II part - Protecting the secret during computations

- *Multiplicative Complexity of Autosymmetric Functions: Theory and Applications to Security* [13]. In this paper we study a particular structure regularity of Boolean functions, called autosymmetry, and exploit it to decrease the number of *ands* in *xor-and* Graphs (XAGs), i.e., Boolean networks composed by *ands*, *xors*, and inverters. The interest in autosymmetric functions is motivated by the fact that a considerable amount of standard Boolean functions of practical interest presents this regularity; indeed, about 24% of the functions in the classical ESPRESSO benchmark suite have at least one autosymmetric output. The experimental results validate the proposed approach.
- *Multiplicative Complexity of XOR Based Regular Functions* [14]. In this paper we study the multiplicative complexity of Boolean functions characterized by two regularities, called autosymmetry and D-reducibility. Moreover, we exploit these regularities for decreasing the number of *and* nodes in XAGs. The experimental results validate the proposed approaches.

- *A Multiple Valued Logic Approach for the Synthesis of Garbled Circuits.* In this paper, we explore the possibility to extend the garbled circuit technique to the case of multiple valued gates, where the wires can carry some finite set of values. We first focus on 3-valued logic and define possible encodings for values in Boolean logic, allowing the evaluation of garbled circuits. Such encodings are compared, and the resulting circuits analyzed to find if some more efficiency, in terms of number of gates and overall complexity of computation, can be gained. Moreover, we prove that the 3-valued logic considered, can be easily extended to general multiple valued logic, which still guarantees the free *xor* property, exploited for the fast evaluation of the circuit. We report some experiments to evaluate the different encodings and the efficiency improvements. We submitted this work presented to IEEE Transactions on Information Forensics & Security.

CHAPTER 2

CRYPTOGRAPHY AND SECRET MANAGEMENT

Cryptography is the theory that studies how a party can protect sensitive information during some computation/communication from other parties, that can be malevolent and try to derive some knowledge about secrets. To make possible these secure computations, we employ many mathematical notions, and some of them are explained in Section 2.1.5, to make all the following material clearer. Moreover, in Section 2.2 we also present some basic notions about cryptographic algorithms.

The main concept from which this work takes place is the different kinds of malicious entities that can threaten the security of a cryptographic computation. Generally, it is possible to identify two types of them, i.e., *external entities*, also called attackers (Section 2.3), and *internal entities* (Section 2.4).

2.1 Mathematical context

A large part of cryptography is based on the mathematical theory of Binary Field \mathbb{F}_2 and functions that take inputs and outputs from this field, called Boolean functions. Many features of Boolean functions have been studied, but in this work the main topics are around the concept of Fourier and Walsh transforms (Section 2.1.2.1), and the concept of autosymmetric Boolean functions (Section 4).

\wedge	\parallel	0	1
0	\parallel	0	0
1	\parallel	0	1

Table 2.1: *and* operation in \mathbb{F}_2 .

\oplus	\parallel	0	1
0	\parallel	0	1
1	\parallel	1	0

Table 2.2: *xor* operation in \mathbb{F}_2 .

\neg	\parallel	
0	\parallel	1
1	\parallel	0

Table 2.3: *not* operation in \mathbb{F}_2 .

\vee	\parallel	0	1
0	\parallel	0	1
1	\parallel	1	1

Table 2.4: *or* operation in \mathbb{F}_2 .

2.1.1 Binary Field and Boolean Space

Definition 2.1. A *field* is a not empty set \mathbb{F} with two composition laws \diamond and \star such that \mathbb{F} with operation \diamond is a commutative group, $\mathbb{F} - \{0\}$ with operation \star is a commutative group and the distributive law holds [87].

The smallest field is the one containing only 2 elements, i.e., $\mathbb{F}_2 = \{0, 1\}$; this field is also called *binary field*, and its elements *bits* [101].

In the binary field, the two composition laws are called *and* (\wedge , also often denoted with \cdot) and *xor* (\oplus), and they are defined in Tables 2.1 and 2.2.

For the scope of this work, two other operations in binary field must be defined. The first one is the *not* operation (\neg), that gives the opposite element in the field (Table 2.3). For the sake of clarity, sometimes $\neg x$ is also written as \bar{x} , with $x \in \mathbb{F}_2$. The second operation is *or* (\vee), that is defined in Table 2.4. It is also possible to rewrite the *xor* operation depending on \wedge and \vee , through the following expression:

$$a \oplus b = (a \wedge \bar{b}) \vee (\bar{a} \wedge b).$$

There is a direct correspondence between Binary Field and Boolean Logic, substituting the elements 0 and 1 with the correspondent false (F) and true (T). The operations between the elements of the logic are the same, i.e., *not*, *and*, *or* and *xor* [91].

The following definitions are necessary for the treated arguments in Section 4.1.

Definition 2.2. The *Boolean space* $\{0, 1\}^n$ is a vector space with respect to the *xor* function and the multiplication with the scalars 0 and 1.

Definition 2.3. An *affine space* is a vector space or a translation of a vector space.

Then, if V is a vector subspace of the Boolean vector space $(\{0, 1\}^n, \oplus)$ and α is a point in $\{0, 1\}^n$, then the set $A = \{\alpha \oplus v | v \in V\}$ is an affine space over V with translation point α [15].

2.1.2 Boolean functions

A *Boolean single output function* f is a function

$$f : \mathbb{F}_2^n \rightarrow \mathbb{F}_2.$$

where \mathbb{F}_2^n is the set of all binary vectors of length n [37][123].

A possible representation of the Boolean functions is the Algebraic Normal Form (ANF), for which the representation of a Boolean function f with n inputs is:

$$f(x) = \bigoplus_{I \in \mathcal{P}(N)} a_I x_I$$

where $\mathcal{P}(N)$ is the power set of $N = \{1, \dots, n\}$, $x_I = \prod_{i \in I} x_i$ and a_I is a coefficient in \mathbb{F}_2 .

Some features of Boolean functions are relevant for the study in this work. First, the *Hamming weight* $w_H(x)$ of an element $x \in \mathbb{F}_2^n$ is the number of nonzero elements in vector x . In particular, the *Hamming weight* $w_H(f)$ of a Boolean function f is the number of elements in the set $\{x \in \mathbb{F}_2^n | f(x) \neq 0\}$. Moreover, the *Hamming distance* $d_H(f, g)$ between two Boolean functions f and g is the number of elements in the set $\{x \in \mathbb{F}_2^n | f(x) \neq g(x)\}$.

Finally, a *pseudo-Boolean function* f is a function with co-domain in \mathbb{R} :

$$f : \mathbb{F}_2^n \rightarrow \mathbb{R}.$$

As for elements in the finite field, also between Boolean functions it is possible to apply operations \neg , \cdot , \vee and \oplus , that means apply them to their outputs.

Example. Let $f : \mathbb{F}_2^3 \rightarrow \mathbb{F}_2$ be a Boolean function:

$$f(x_0, x_1, x_2) = x_0 \bar{x}_2 \oplus x_1$$

where $x = [x_0, x_1, x_2] \in \mathbb{F}_2^3$. The *true* table in Table 2.5 reports the output of function f varying the inputs. The Hamming weight of f is $w_H(f) = 4$.

2.1.2.1 Fourier transform and Walsh transform

Many features of a Boolean function can be obtained through some widely studied tools [37]. Among them, one of the most used is the Discrete Fourier Transform, i.e., the linear mapping which maps any pseudo-Boolean function f on \mathbb{F}_2^n to the function \hat{f} defined on \mathbb{F}_2^n by:

$$\hat{f}(u) = \sum_{x \in \mathbb{F}_2^n} f(x) (-1)^{x \cdot u}$$

x_2	x_1	x_0	$f(x)$
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	0
1	0	0	0
1	0	1	0
1	1	0	1
1	1	1	1

Table 2.5: True table of the Boolean function $f(x_0, x_1, x_2) = x_0\bar{x}_2 \oplus x_1$.

where $*$ denotes the inner product between two vectors in \mathbb{F}_2^n and the sum is in \mathbb{R} . The vector resulting from the evaluation of \hat{f} varying the inputs $u \in \mathbb{F}_2^n$ is called *Fourier spectrum* of f . Notice that $\hat{f}(0)$ equals to the Hamming weight of f .

An application of the Fourier transform is the following [37]: for a given Boolean function f , the knowledge of its discrete Fourier transform is equivalent to the knowledge of the weights of all the functions $f \oplus l$, where l is a linear (or affine) function.

The discrete Fourier transform can also be applied to the pseudo Boolean function $f_\chi(x) = (-1)^{f(x)}$, often called the *sign function*, instead of f itself. Then

$$\hat{f}_\chi(u) = \sum_{x \in \mathbb{F}_2^n} (-1)^{f(x) \oplus x \cdot u}$$

is called the Walsh transform of f , and vector $\hat{f}_\chi(u)$ computed varying $u \in \mathbb{F}_2^n$ is called the *Walsh spectrum* of f . Dually, f can be reconstructed from \hat{f}_χ with the inverse function:

$$f(x) = 2^{-n} \sum_{u \in \mathbb{F}_2^n} (-1)^{\hat{f}_\chi(x) \oplus x \cdot u}.$$

The Walsh transform of a Boolean function gives the knowledge of the correlation between output and inputs of the function itself [123].

Example. Let $f : \mathbb{F}_2^4 \rightarrow \mathbb{F}_2$ be a Boolean function:

$$f(x_0, x_1, x_2, x_3) = x_0x_1x_2 \oplus x_0x_3 \oplus x_1.$$

Then, in Table 2.6 the evaluation of f varying the inputs (column $f(x)$) and its Walsh spectrum (column \hat{f}_χ) are reported.

2.1.3 Vectorial Boolean function

x_3	x_2	x_1	x_0	$f(x)$	\hat{f}_χ
0	0	0	0	0	0
0	0	0	1	0	0
0	0	1	0	1	8
0	0	1	1	1	8
0	1	0	0	0	0
0	1	0	1	0	0
0	1	1	0	1	0
0	1	1	1	0	0

x_3	x_2	x_1	x_0	$f(x)$	\hat{f}_χ
1	0	0	0	0	4
1	0	0	1	1	-4
1	0	1	0	1	4
1	0	1	1	0	-4
1	1	0	0	0	-4
1	1	0	1	1	4
1	1	1	0	1	4
1	1	1	1	1	-4

Table 2.6: Application of the Fourier and Walsh transforms to the Boolean function $f(x_0, x_1, x_2, x_3) = x_0x_1x_2 \oplus x_0x_3 \oplus x_1$.

2.1.3 Vectorial Boolean function

A vectorial Boolean function f is a function such that the inputs and outputs are elements in \mathbb{F}_2 :

$$f : \mathbb{F}_2^m \rightarrow \mathbb{F}_2^n$$

where \mathbb{F}_2^m and \mathbb{F}_2^n are the set of all binary vectors of length m and n respectively [36]. Each output of f can be read as the output of a Boolean function, i.e., $f(x) = [f_0(x), f_1(x), \dots, f_{n-1}(x)]$ for any $x \in \mathbb{F}_2^m$. It implies that also a vectorial Boolean function has its algebraic normal form representation, given by the ANF of every single f_i , for all $0 \leq i \leq n - 1$.

Example. Let $f : \mathbb{F}_2^3 \rightarrow \mathbb{F}_2^2$ be a vectorial Boolean function:

$$f(x_0, x_1, x_2) = [x_0x_1, x_1 \oplus \bar{x}_2]$$

where $x = [x_0, x_1, x_2] \in \mathbb{F}_2^3$, and $f_0(x_0, x_1, x_2) = x_0x_1$ and $f_1(x_0, x_1, x_2) = x_1 \oplus \bar{x}_2$. The true table in Table 2.7 reports the output of function f varying the inputs.

2.1.3.1 Walsh matrix

The Walsh transform explained in Section 2.1.2.1, can be also applied to the vectorial Boolean functions: the linear mapping, defined on a vectorial Boolean function f with inputs in \mathbb{F}_2^m and outputs in \mathbb{F}_2^n , is defined by:

$$W_f(u, v) = \sum_{x \in \mathbb{F}_2^m} (-1)^{f(x) * v \oplus x * u}$$

x_2	x_1	x_0	$f_0(x)$	$f_1(x)$
0	0	0	0	1
0	0	1	0	1
0	1	0	0	0
0	1	1	1	0
1	0	0	0	0
1	0	1	0	0
1	1	0	0	1
1	1	1	1	1

Table 2.7: True table of the vectorial Boolean function $f(x_0, x_1, x_2) = [x_0x_1, x_1 \oplus \bar{x}_2]$.

			0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	x_3
			0	0	0	0	1	1	1	1	0	0	0	0	1	1	1	1	x_2
			0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1	x_1
			0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	x_0
0	0	0	16	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
0	0	1	0	0	0	0	8	8	8	-8	0	0	0	0	0	0	0	0	
0	1	0	0	0	0	0	0	0	0	8	8	0	0	8	-8	0	0	0	
0	1	1	0	0	0	0	0	0	0	0	0	8	-8	8	8	0	0	0	
1	0	0	0	0	0	0	0	0	0	0	0	0	16	0	0	0	0	0	
1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	8	-8	8	8	
1	1	0	0	0	8	8	0	0	8	-8	0	0	0	0	0	0	0	0	
1	1	1	8	-8	0	0	0	0	8	8	0	0	0	0	0	0	0	0	
$f_2(x)$	$f_1(x)$	$f_0(x)$																	

Table 2.8: Walsh matrix of the vectorial Boolean function $f(x) = [x_0x_1 \oplus x_2, x_0x_2 \oplus x_3, x_1 \oplus x_3]$ with $x = [x_0, x_1, x_2, x_3] \in \mathbb{F}_2^4$ and $f_0(x) = x_0x_1 \oplus x_2$, $f_1(x) = x_0x_2 \oplus x_3$, $f_2(x) = x_1 \oplus x_3$.

where $*$ denotes the inner product between two vectors and the sum is in \mathbb{R} . The matrix resulting from the evaluation of W_f varying $u \in \mathbb{F}_2^m$ and $v \in \mathbb{F}_2^n$ is called *Walsh matrix* of f .

The Walsh matrix for a vectorial Boolean function f gives the knowledge of the correlation between the outputs and the inputs of the function.

Example. Let $f : \mathbb{F}_2^4 \rightarrow \mathbb{F}_2^3$ be the vectorial Boolean function:

$$f(x_0, x_1, x_2, x_3) = [x_0x_1 \oplus x_2, x_0x_2 \oplus x_3, x_1 \oplus x_3].$$

Then, its Walsh matrix of dimension $2^3 \times 2^4$ is reported in Table 2.8. For example, the nonzero elements in row $[f_2(x), f_1(x), f_0(x)] = [0, 1, 1]$ give information about the correlation between $f_1(x) \oplus f_0(x)$ and $x_3 \oplus x_2$, $x_3 \oplus x_2 \oplus x_0$, $x_3 \oplus x_2 \oplus x_1$, $x_3 \oplus x_2 \oplus x_1 \oplus x_0$.

2.1.4 Tensor product between matrices

For the scope of this work, we must define the operation of tensor product between matrices[72].

2.1.5 String diagram

Let A be a matrix of dimension $l \times n$ and B a matrix of dimension $m \times t$. Then the matrix $A \otimes B$ resulting from the tensor product between them has dimension $(l \cdot m) \times (n \cdot t)$ and it is such that each element is defined by:

$$A \otimes B(x, y) = A\left(\left\lfloor \frac{x}{m} \right\rfloor, \left\lfloor \frac{y}{t} \right\rfloor\right) \cdot B\left(x - \left\lfloor \frac{x}{m} \right\rfloor m, y - \left\lfloor \frac{y}{t} \right\rfloor t\right)$$

with $x \in \{0, \dots, l \cdot m - 1\}$ and $y \in \{0, \dots, n \cdot t - 1\}$. An easier way to write the resulting matrix is as follows:

$$S \otimes B = \begin{pmatrix} A(0,0) \cdot B & A(0,1) \cdot B & \cdots & A(0,m-1) \cdot B \\ A(1,0) \cdot B & A(1,1) \cdot B & \cdots & A(1,m-1) \cdot B \\ \vdots & \vdots & \ddots & \vdots \\ A(l-1,0) \cdot B & A(l-1,1) \cdot B & \cdots & A(l-1,m-1) \cdot B \end{pmatrix}$$

In literature, tensor product is also called Kronecker product.

2.1.5 String diagram

String diagram is a tool that allows to simply reason about functions and matrices, mainly when they are composed.

Let $f : U \rightarrow V$ be a linear map; through the *string diagram's representation* [104], this map can be drawn as a box labelled f with one input string labelled U and one output string labelled V .

Let $g : V \rightarrow W$ be another linear map, then the composition $f \circ g$ can be represented through horizontal juxtaposition of boxes representing the functions, pairing input and output wires with matching labels (Figure 2.1a).

Let $f : U_1 \rightarrow V_1$ and $g : U_2 \rightarrow V_2$ be two linear maps, then their tensor product is a map $f \otimes g : U_1 \otimes U_2 \rightarrow V_1 \otimes V_2$ represented graphically by vertically juxtaposition of the boxes representing the functions. Note that $f \otimes g$ has two input wires and two output wires (Figure 2.1b).

The identity map $id_V : V \rightarrow V$ is represented by a wire with no attached box, to reflect the fact that it is an identity for composition.

2.1.6 Binary Decision Diagrams

As described in the previous section, there are many ways to represent a Boolean Function. Among them, Binary Decision Diagrams (BDDs) are in general a very compact representation, that allows to compute rapidly operations between Boolean functions. BDDs are direct acyclic graphs, i.e., a representation composed of nodes (vertices) and edges (arcs), in which each edge has a direction (given by an arrow) and with a nodes' hierarchy [2, 109, 34, 76].

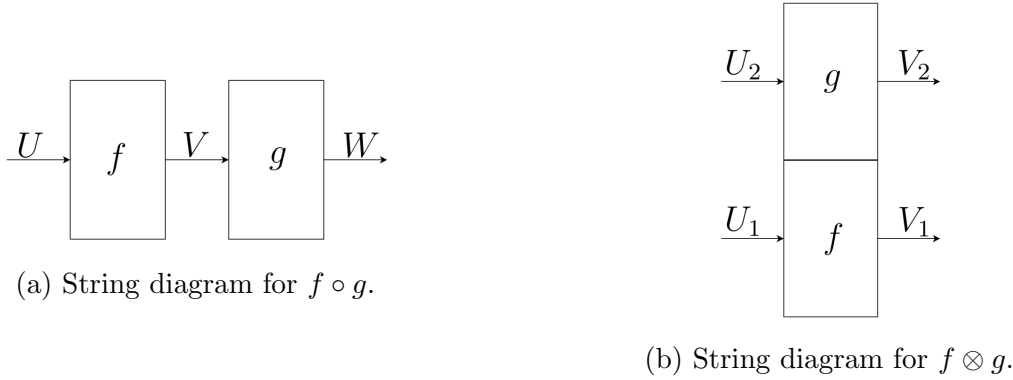


Figure 2.1: String diagrams for composition and tensor product between two functions.

Definition 2.4. A *Binary Decision Diagram* (BDD) on a set of Boolean variables $\{x_1, \dots, x_n\}$ is a rooted, connected direct acyclic graph, where each internal node N is labeled by a Boolean variable x_i and it has two outgoing edges, the 0-edge and the 1-edge, pointing to two nodes, i.e., the 0-child and the 1-child of node N , respectively. Terminal nodes (or leaves) are labeled with a constant value 0 or 1. Usually, binary decision diagrams are exploited to represent Boolean functions.

Definition 2.5. A BDD is *ordered* (OBDD) if there exists a total order $<$ over the set of variables such that if an internal node is labeled by x_i , and its 0-child and 1-child have labels x_{i_0} and x_{i_1} , respectively, then $x_i < x_{i_0}$ and $x_i < x_{i_1}$. The choice of the variable order can have a dramatic impact on the size of the BDD.

Definition 2.6. A OBDD is *reduced* if there exist no nodes whose 1-child is equal to the 0-child and there aren't two distinct nodes that are roots of isomorphic subgraphs. A reduced and ordered BDD is called *ROBDD*.

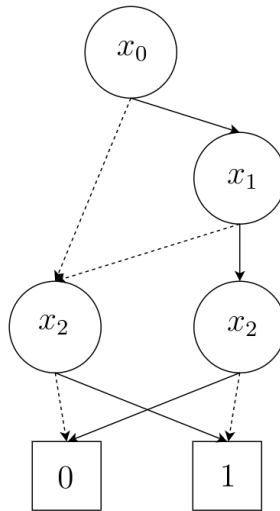
Note that, usually, the term BDD is used instead of the correct term ROBDD.

Example 2.1.1. An example of BDD is in Figure 2.2, which represents the Boolean function $f(x_0, x_1, x_2) = x_0 \cdot x_1 \oplus x_2$.

2.1.6.1 Manipulation of BDDs

Given two BDDs representing functions f and g , it is interesting to analyze how to build the BDD for $f \langle op \rangle g$, where $\langle op \rangle$ is a binary operation (*and*, *or*, *xor* for example) [109]. The basic idea is to apply the Shannon's expansion [105] as follow:

$$f \langle op \rangle g = x(f_x \langle op \rangle g_x) + \bar{x}(f_{\bar{x}} \langle op \rangle g_{\bar{x}}).$$

Figure 2.2: BDD for $f(x_0, x_1, x_2) = x_0 \cdot x_1 \oplus x_2$.

If x is the top variable of f and g , we can first cofactor the two functions with respect to x and solve two simpler problems recursively, and then create a node labelled x that points to the results of the two subproblems. The cofactors of f and g with respect to x are the two children of the top node of f , when x is the top variable of f ; otherwise, if f does not depend on x , $f_x = f_{\bar{x}} = f$. The algorithm that takes f , g and $\langle op \rangle$ as arguments and returns $f \langle op \rangle g$ is called *Apply* in the literature.

Much simpler is the procedure to compute the negation of a function, since it consists in change the values of the leaves (0 becomes 1 and vice versa).

The time occurred to compute an operation depends on the dimensions of the involved BDDs; much smaller is the dimension of BDDs, than faster are the manipulations.

Note that, if the manipulations can become hard to treat when BDDs are bigger, solve a satisfiability problem when the function is given in BDD form is always trivial, since it is sufficient to check whether the BDD is the constant 0 function and that takes constant time.

2.1.6.2 Algebraic Decision Diagrams

Definition 2.7. An *Algebraic Decision Diagram* (ADD) [5] can be described as a BDD with a generalized set of constant values. Therefore, an ADDs is the representation of a function $f : \{0, 1\}^n \rightarrow S$, where S is an arbitrary set. When S is $\{0, 1\}$ the ADD is a classical BDD.

Note that in this case, the ADD is the representation of a pseudo-Boolean function.

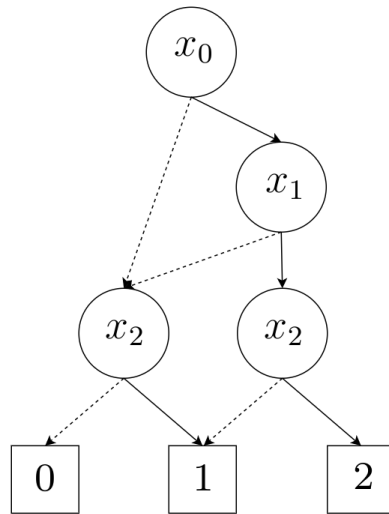


Figure 2.3: ADD for $f(x_0, x_1, x_2) = x_0 \cdot x_1 + x_2$.

ADDs can be manipulated as BDDs.

Example 2.1.2. Let $f(x_0, x_1, x_2) = x_0 \cdot x_1 + x_2$ be a Boolean function, where $+$ is the sum in \mathbb{R} . Then the corresponding ADD is in Figure 2.3, for which the results' set is $\{0, 1, 2\} \in \mathbb{R}$.

2.2 Cryptographic algorithms

The main aim of cryptography is to enable communication between two or more entities (called also parties) in a secure way, sending messages that cannot be recovered from untrusted entities [37][100]. For this purpose, cryptosystems have been developed.

The following three sets are fundamental for a cryptosystem:

- \mathcal{P} is the set of *plaintexts*, elements that have to be sent in some secure way;
- \mathcal{C} is the set of *ciphertexts*, elements that derive from plaintexts and are sent instead of plaintexts;
- \mathcal{K} is the set of *keys*, secrets used to perform transformation from plaintexts to ciphertexts.

Generally, an encryption scheme is based on an *encryption function* E_k , depending on a key $k \in \mathcal{K}$,

$$E_k : (\mathcal{P}, \mathcal{K}) \longrightarrow \mathcal{C}$$

2.2.1 Advanced Encryption Standard

and a *decryption function* D_h , depending on $h \in \mathcal{K}$, that can be equal to k ,

$$D_h : (\mathcal{C}, \mathcal{K}) \longrightarrow \mathcal{P}$$

such that $D_h(E_k(p)) = p$ for all $p \in \mathcal{P}$.

It is possible to recognize two categories of encryption functions, namely functions that constitute symmetric cryptography and those that form asymmetric cryptography.

Symmetric cryptography is based on the concept that the chosen key used for the encryption function is also the one used for the decryption function. This key must be shared among entities in a secure way. There are two types of algorithms that belong to the symmetric cryptography:

- *Stream ciphers*: starting from a (short) secret vector s , called *seed*, a very long sequence of bits, called *keystream*, is generated. The ciphertext is generated xoring bitwise the plaintext with the keystream. To decrypt the ciphertext, it has to be xored with the keystream, and the plaintext is recovered.
- *Block ciphers*: they are based on rounds. From key k , by an algorithm called *key schedule* all round keys k_1, \dots, k_r are derived. In the first round, the encryption function is applied to the plaintext with key k_1 , obtaining the partial ciphertext m_1 . At each round i , the encryption function is applied to m_{i-1} with key k_i , until in the last round r the ciphertext is generated.

Instead, *asymmetric cryptography* is based on two keys, a private key, used to encrypt the message and known only by one entity, and a public one, used to decrypt the ciphertext and recover the message.

Another type of cryptographic algorithms are *hash functions*, that generally can be used to map data of arbitrary size to data of fixed size. The most widespread applications of hash functions are digital signature, password verification, key-derivation, pseudo-random number generation and authentication.

2.2.1 Advanced Encryption Standard

The AES algorithm is one of the most popular symmetric-key block ciphers, and it is widely employed for many applications, with implementations both in software and in hardware. It was originally designed by J. Daemen and V. Rijmen under the name of Rijndael, a subset of which was selected in 2001 by the National Institute of Standard and Technology to constitute the Advanced Encryption Standard (AES) [54].

The AES algorithm operates on fixed-length data blocks of 128 bits, in encryption as well as in decryption, by using cipher keys of 128, 192 or 256 bits. Each data block is progressively transformed by a sequence of four primitive functions (AddRoundKey, SubBytes, ShiftRows and MixColumns) organized in rounds, which number depends on the key length.

The basic data unit is the byte, a vector of 8 bits ordered as in $b = \{b_7, b_6, b_5, b_4, b_3, b_2, b_1, b_0\}$. Bytes can be interpreted as elements of the finite field \mathbb{F}_2^8 , with the polynomial representation:

$$b(x) = \sum_{i=0}^7 b_i x^i$$

The *xor* operation in \mathbb{F}_2^8 , denoted by \oplus_8 , is achieved by performing the typical *xor* in \mathbb{F}_2 between the coefficients for the corresponding powers in the polynomials. For sake of clearness, the *xor* between $b, b' \in \mathbb{F}_2^8$ is computed as:

$$b(x) \oplus_8 b'(x) = (b_7 \oplus b'_7)x^7 + (b_6 \oplus b'_6)x^6 + \dots + b_0 \oplus b'_0$$

The *multiplication* in \mathbb{F}_2^8 , denoted by \cdot , is achieved by performing the polynomial multiplication modulo an irreducible polynomial (i.e., a polynomial that cannot be factored into the product of two non-constant polynomials) of degree 8. The reduction polynomial used by AES is $m(x) = x^8 + x^4 + x^3 + x + 1$. For sake of clearness, the multiplication between $b, b' \in \mathbb{F}_2^8$ is computed as:

$$b(x) \cdot b'(x) = [(b_7 x^7 + b_6 x^6 + \dots + b_0) \cdot (b'_7 x^7 + b'_6 x^6 + \dots + b'_0)] \bmod(m(x))$$

The AES-128 encryption primitive transforms a 128-bit input plaintext into a 128-bit output ciphertext by manipulating a fixed-length data block, known as *State*, arranged into a two-dimensional square matrix S of bytes s_i , where columns and rows can be viewed as 32-bit (4 bytes) words:

$$S = \begin{bmatrix} s_0 & s_4 & s_8 & s_{12} \\ s_1 & s_5 & s_9 & s_{13} \\ s_2 & s_6 & s_{10} & s_{14} \\ s_3 & s_7 & s_{11} & s_{15} \end{bmatrix}$$

The AES-128 algorithm is divided in two distinct parts:

- the *datapath*, where all the transformations on the S matrix take place. In each round but the last, the functions iteratively applied on the state are: SubBytes, ShiftRows, MixColumns and AddRoundKey. In the last round MixColumns is omitted.

2.2.2 Circuits

- the *key schedule*, which is responsible for providing a new 128-bit round key for each round, derived from the original secret key.

Right before the execution of the first round, the original encryption key is added to the State. This operation is called *Initial Key Addition* or *First Whitening*.

SubBytes This function applies a substitution table, called *S-Box*, to each of the State's 16 bytes. The S-Box itself is the result of two subsequent operations, a non-linear inversion in \mathbb{F}_2^8 followed by an affine transformation. SubBytes represents the only non-linear function in the AES cipher, due to the multiplicative inversion, which is critical to minimize the correlation between inputs and outputs (cryptographic property of “confusion”).

ShiftRows The State's rows, except for the first one, are cyclically shifted to the left by a fixed offset: one byte for the second row, two bytes for the third, three bytes for the fourth. It is designed to introduce the cryptographic property of “diffusion” on the State.

MixColumns The 32-bit columns of the State are iteratively multiplied by a matrix, namely, called s'_i the elements of the State after this transformation:

$$\begin{bmatrix} s'_i \\ s'_{i+1} \\ s'_{i+2} \\ s'_{i+3} \end{bmatrix} = \begin{pmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{pmatrix} \cdot \begin{bmatrix} s_i \\ s_{i+1} \\ s_{i+2} \\ s_{i+3} \end{bmatrix}$$

As said, the MixColumns step is not applied in the last round (in the case of AES-128, the 10-th). It provides the diffusion property to each individual column.

AddRoundKey The round key k_r for the current round is added to the State by means of a simple bitwise *xor* operation, which is equivalent to applying the addition \oplus_8 defined in the AES finite field \mathbb{F}_2^8 . Each 128-bit round key k_r is provided by the key schedule part of the algorithm.

2.2.2 Circuits

Circuits are the hardware implementations of Boolean functions [103]. A generic *circuit* is composed of some *registers*, in which the state is memorized, and a *combinatorial network* of electrical components connected by wires. The combinatorial network implements some logic operations, with n inputs and m outputs. The basic





Name	Symbol	Boolean operation
<i>not</i>		$not(x) = \neg x$
<i>and</i>		$and(x, y) = x \wedge y$
<i>or</i>		$or(x, y) = x \vee y$
<i>xor</i>		$xor(x, y) = x \oplus y$

Table 2.9: Symbols that correspond to the gates of circuits, implementing the Boolean operations.

elements that constitute the combinatorial network are called *logic gates*, and they implement the principal operations of Boolean Algebra (see Section 2.1.1 and Table 2.9).

In this work, only feedback-free circuits are treated, namely circuits without any loop.

Given some circuit's inputs, signal propagation through wires produces the gates' outputs and finally a circuit's output. For each gate in the circuit, each gate's input bit can be a circuit's input bit or an output bit of another gate in the circuit.

Example 2.2.1. *In Figure 2.4a a very simple circuit is represented. Circuit's inputs are x_0 and x_1 , gate G_0 is a not gate (with input x_0), and G_1 is an and gate (with input the output of G_0 and x_1). y , the output of G_1 , is also the output of the circuit. In Figure 2.4b there is an evaluation of that circuit, when both x_0 and x_1 are equal to 1. In this case, the computed output y has value 0.*

To conclude this overview about circuits, also some last definitions have to be given.

A *clock signal* is a particular type of signal that oscillates between a high and a low state and is utilized like a metronome to coordinate actions of digital circuits. A *clock cycle* is the execution period between two subsequent clock signals. The *clock frequency* is the number of clock signals executed in a time unit.

2.3. Protecting secrets from an external attacker



Figure 2.4: Simple circuit example.

A *synchronous circuit* is a digital circuit in which the changes in the state of memory elements are synchronized by a clock signal. An *asynchronous circuit* is a sequential digital logic circuit which is not governed by a clock signal.

2.3 Protecting secrets from an external attacker

The first kind of malicious considered is the *external entity*, then an attacker that is not involved by the computation, but that tries to recover some information about secrets from an external point of view. In the classical cryptographic situation where Alice and Bob want to communicate a secret that is encrypted, their main aim is to keep hidden any information about the secret message from another entity called Eve, that is the attacker.

There can be different types of attacks, depending on the attacker's level of capacities. Let $\{m_1, m_2, \dots, m_n\} \subseteq \mathcal{P}$ be the plaintext messages; they are encrypted by an encryption function E_k into ciphertexts $\{c_1, c_2, \dots, c_n\} \subseteq \mathcal{C}$.

- *Ciphertext-only attack*: the attacker knows \mathcal{P} and \mathcal{C} , but she sees only the ciphertexts $\{c_1, c_2, \dots, c_n\}$.
- *Known-plaintext attack*: the attacker knows some plaintexts $\{m_1, m_2, \dots, m_n\}$ and the correspondent ciphertexts $\{c_1, c_2, \dots, c_n\}$. In this case, the attacker's goal can be to use pairs of plaintext and ciphertext to recover the key k of the encryption function. When an attacker knows the key, then she can encrypt and decrypt as she likes.
- *Chosen-plaintext attack*: it is a kind of attack such that the attacker can choose messages $\{m_1, m_2, \dots, m_n\}$ and force their encryption. Sometimes there can be the presence of special messages that allow an easy reconstruction of the key.

An attack is called *algebraic attack* when it is based on the analysis of the mathematical operations computed during the encryption (or decryption). If the cryptographic algorithm is not vulnerable to any algebraic attack, then it is called *mathematically secure*. The algebraic analysis of a cryptosystem is called *cryptoanalysis*

and is performed to recover some information exploitable to learn something about sensible data processed during the encryption/decryption computations.

2.3.1 Side-channel attacks

Generally, an algebraic secure cryptosystem is considered as a system implementing some mathematical functions that output only the proper result (the ciphertext) and nothing more. But in practice, sometimes partial information about internal computations can be leaked from the device implementing the cryptographic algorithm, jeopardizing the cryptosystem's security.

Definition 2.8. *Side-channel information* are information that are recovered from the encryption device while it executes the encryption operations.

Some examples of side-channel information are: the power that is consumed by the encryption device while it computes all its operations, the electromagnetic field produced by the electronic system and the amount of time necessary to the device for the implementation. Then, an attacker can try to exploit the information recovered from these sources to mount an attack.

Definition 2.9. Attacks on cryptographic algorithms, done exploiting side-channel information, are called *side-channel attacks*.

In a circuit, a loss of sensitive information exploitable for a side-channel attack is called *leakage*. In this work, only leakage due to the *power consumption* of a circuit is treated. It is possible to distinguish three different types of power consumption in a circuit:

1. *Static leakage*: this refers to the energy needed by the circuit to maintain the current state when no switch of input bits is present.
2. *Switching of register*: the consumption taken by the circuit for updating the state in two consecutive clock cycles.
3. *Switching of combinatorial logic*: this is the consumption at combinatorial logic level, due to the switches in circuit's gates. This consumption, in most cases, spans the entire duration of the clock cycle.

2.3.2 Power Analysis Attacks

The most widespread side-channel attacks are *power consumption attacks*[83]: the amount of power used by a device is influenced by what is processing, and so power

consumption measurements contain information about a circuit's calculation. Moreover, when a device is processing some cryptographic secrets, its data-dependent power usage can expose these secrets to an attack.

A power consumption attack is based on two basic concepts:

- the attacker has access to the device that she wants to analyze, and so she can force it to do some controlled operations, or just monitoring the activity of it;
- she can measure the power consumed by the device at each operation.

The attacker can manage the plaintext in input at the cryptographic algorithm, and she knows the related ciphertext in output. However, she does not know the key used, and so this attack is a known-plaintext attack. She can also recover information knowing only the ciphertext, and in this case the attack is a ciphertext-only attack.

Definition 2.10. Measurements collected from the target device are called *power traces*, denoted by $V_i \in \mathbb{R}^T$, where $T \in \mathbb{N}$ is the number of considered time points.

When one or more traces are collected, these data about power consumption are processed with some analytic methods. Three among them, that are relatively recent and powerful, are described in the following paragraphs.

Simple Power Analysis. *Simple power analysis (SPA)* is the less intrusive side-channel attack. It involves visual examination of graphs of the current used by a device over time. Variations in power consumption occur as the device performs different operations. For example, different instructions performed by a microprocessor will have differing power consumption profiles.

Differential Power Analysis. *Differential Power Analysis (DPA)* is a statistical method for analyzing sets of measurements to identify data-dependent relations [77]. This method consist in some basic steps:

- a set of measurements are collected to be analyzed;
- some guesses about the secret key used by the algorithm are made;
- using these guessed keys, some predictions are made, i.e., some plaintexts are encrypted with the guessed keys;
- a selection function is applied, which is used to assign traces to sets;
- if a correlation between sets is noted, then the guess is correct, contrariwise if no correlation is analyzed, a new guess has to be done.

In [77], the differential power attack is computed using the Difference of Mean (DoM). Let P be the plaintext, \tilde{K} the guessed key and s the selection function. Steps of this attack are:

- a set of traces is partitioned into two subsets, M_0 and M_1 , defined as:

$$M_j = \{V_i \in \mathbb{R}^T | s(P, \tilde{K}) = j\} \text{ for } j \in \mathbb{Z}_2$$

where V_i is a trace.

- then the difference of mean is computed for each subset, i.e.:

$$DoM(M_0, M_1) = \mathbb{E}(M_0) - \mathbb{E}(M_1)$$

where \mathbb{E} is the mean of the elements in the set.

- if the choice that has assigned a set for each element is uncorrelated to the measurements, at the increasing of the traces' number, each subset's difference of mean will approach to zero;
- on the contrary, if the partition in subsets is correlated to the trace measurements, each subset's difference of mean will approach a nonzero value.

Some differential power attacks use also other statistical moments, as variance and asymmetry. Attacks using statistical moment of higher order, have more possibilities to recover some relevant information. However, these attacks are more sensible to noise, and they need more traces (higher-order attacks, versus first-order).

Correlation Power Analysis. Correlation Power Analysis (CPA) is another statistical power analysis technique, first appeared in [33], that uses Pearson's correlation coefficient as a statistical test.

Steps of this type of power analysis attack are reported in the following points.

- The power consumption relative to an execution i is stored as a power trace t_i , and N power traces t_i are acquired, each one relative to a different known data d_i ($1 \leq i \leq N$) and a fixed key \hat{k} (that is stored in the device under attack). Power traces t_i can be viewed as a vector of M power samples $\mathbf{t}_i = [t_{i,1}, t_{i,2}, \dots, t_{i,M}]$. They can be all collected in the matrix \mathbf{T} :

$$\mathbf{T} = \begin{pmatrix} \mathbf{t}_1 \\ \mathbf{t}_2 \\ \vdots \\ \mathbf{t}_N \end{pmatrix} = \begin{pmatrix} t_{1,1} & t_{1,2} & \cdots & t_{1,M} \\ t_{2,1} & t_{2,2} & \cdots & t_{2,M} \\ \vdots & \vdots & & \vdots \\ t_{N,1} & t_{N,2} & \cdots & t_{N,M} \end{pmatrix}$$

- In the first step, an intermediate value of the cryptographic algorithm that is computed as a known function $f(d, k)$, where d is a known non-constant data value and k is a small portion of the key, has to be identified. Generally, this intermediate value is called *target value*; the data d is typically either the plaintext or the ciphertext.
- In this step, the target value $v_{i,j}$ for each data d_i and every possible k_j , that are small portions of the guess key such that $v_{i,j} = f(d_i, k_j)$, are computed. In this way, matrix \mathbf{V} is computed, that contains every intermediate value for each possible key guess k_j , with $1 \leq j \leq K$:

$$\mathbf{V} = \begin{pmatrix} v_{1,1} = f(d_1, k_1) & v_{1,2} = f(d_1, k_2) & \cdots & v_{1,K} = f(d_1, k_K) \\ v_{2,1} = f(d_2, k_1) & v_{2,2} = f(d_2, k_2) & \cdots & v_{2,K} = f(d_2, k_K) \\ \vdots & \vdots & & \vdots \\ v_{N,1} = f(d_N, k_1) & v_{N,2} = f(d_N, k_2) & \cdots & v_{N,K} = f(d_N, k_K) \end{pmatrix}$$

- Now, a power model that maps the matrix \mathbf{V} of hypothetical intermediate values to a matrix \mathbf{H} of hypothetical power consumption values, is used through a function f_s that has to be chosen according to a criteria of power consumption of the device under attack:

$$\mathbf{H} = \begin{pmatrix} h_{1,1} = f_s(v_{1,1}) & h_{1,2} = f_s(v_{1,2}) & \cdots & h_{1,K} = f_s(v_{1,K}) \\ h_{2,1} = f_s(v_{2,1}) & h_{2,2} = f_s(v_{2,2}) & \cdots & h_{2,K} = f_s(v_{2,K}) \\ \vdots & \vdots & & \vdots \\ h_{N,1} = f_s(v_{N,1}) & h_{N,2} = f_s(v_{N,2}) & \cdots & h_{N,K} = f_s(v_{N,K}) \end{pmatrix}$$

- In the last step, CPA specific, Pearson's correlation is applied between columns of the power traces and columns of the matrix H . As a result, the column that presents the maximum correlation reveals j such that k_j is the real portion of \hat{k} with the highest probability.

The Pearson's correlation coefficient is a normalized version of the covariance that gives a result in the interval $[-1,1]$, namely

$$\rho_{H,T} = \frac{\text{cov}(H, T)}{\sigma_H \sigma_T} = \frac{\mathbb{E}[(H - \mu_H)(T - \mu_T)]}{\sigma_H \sigma_T}$$

where the random variable T represents the measured power consumptions for a column of \mathbf{T} and the random variable H is the random variable that represents the corresponding hypothetical power consumption; μ and σ are mean and standard deviation, respectively.

2.3.2.1 Countermeasures

The improvement of side-channel attacks has as direct consequence an improvement of methods to protect algorithms and secrets from these new threats [83]. Therefore, new countermeasures have been created. Among them, the most spread are masking and threshold implementation, that are presented after in this Section. However, some other countermeasures against side-channel attacks have been developed.

- *Clock Randomization*: the duration of each clock cycle is determined randomly; traces collected are difficult to compare, and a previous reorganization of them is necessary and not always feasible.
- *False executions*: in this case, in addition to the usual computations of the cryptographic algorithm, some other operations are executed (for example, the same algorithm with a different key), so to confuse an attacker.
- *Complemented logic*: in parallel to the usual computations, are executed the same computations but with complemented inputs and combinatorial logic. The idea behind this countermeasure is to make power consumption that comes from logical values toggling constant, so that the information leakage is not dependent on the secret data.

Masking. A countermeasure against side-channel attacks is *masking*, which tries to randomize the intermediate values of a cryptographic algorithm [70][83]. In a masking implementation, all intermediate values a are concealed by a random value m which is called *mask*. The most spread masking scheme is the *additive masking*, in which each intermediate value a is xored with a mask m generating the masked value a_m , i.e., $a_m = a \oplus m$. For some cryptographic algorithms can be used a *multiplicative masking*, where the masked value is calculated multiplying the intermediate value with a mask, i.e., $b_m = h * m$.

Masking can be used against side-channel attacks because the randomly masked intermediate values produce a no predictable power consumption. Some basic rules must be respected:

- masks are added at the beginning of the algorithm to the plaintext;
- during the execution of the algorithm, every intermediate value must stay masked: this is achieved modifying also the operations in the algorithm that has to be changed to respect the masking;
- at the end of the algorithm, the mask is removed from intermediate values, and output is recovered.

It is imperative for the security of a masked implementation that all intermediate values always remain masked.

Threshold Implementations. The second method is called Threshold Implementations, and it has been proposed for the first time in [94] and developed also in [95]. Although it has a high computational complexity, it also requires random values only at the start (differently from masking) and it stays effective in the presence of glitches (for a definition of what glitches are, see section 2.3.3).

The scheme of this method is exposed below. Given a variable $x \in \mathbb{Z}_2$, it is split into s additive shares $x_i \in \mathbb{Z}_2$ such that

$$x = \bigoplus_{i=1}^s x_i. \quad (2.1)$$

The vector of shares x_i is denoted by $\bar{x} = x_1, x_2, \dots, x_s$. Note that the knowledge of up to $s - 1$ shares does not give any additional information on the value of x .

Let f be a function with p inputs x^1, x^2, \dots, x^p and q outputs z^1, z^2, \dots, z^q . (X_1, X_2, \dots, X_p) are values that the inputs of f can take, and $(\bar{X}_1, \bar{X}_2, \dots, \bar{X}_p)$ are values that the vector of inputs' shares can take, such that $\sum_{i=1}^s X_i^j = X^j, \forall 1 \leq j \leq p$. In the same way, vector of values that the output of f can take is defined with (Z_1, Z_2, \dots, Z_q) , and $(\bar{Z}_1, \bar{Z}_2, \dots, \bar{Z}_q)$ are values that the vector of outputs' shares can take, such that $\sum_{i=1}^s Z_i^j = Z^j, \forall 1 \leq j \leq q$. Let c be a constant value. A first property that the secret sharing must respect is

$$\Pr(\bar{x}^1 = \bar{X}^1, \dots, \bar{x}^p = \bar{X}^p) = c \cdot \Pr(x^1 = \sum_{i=1}^s X_i^1, \dots, x^p = \sum_{i=1}^s X_i^p) \quad (2.2)$$

and hence

$$\sum_{i=1}^s X_i^j = \sum_{i=1}^s Y_i^j, \forall j \implies \Pr(\bar{x}^1 = \bar{X}^1, \dots, \bar{x}^p = \bar{X}^p) = (\bar{x}^1 = \bar{Y}^1, \dots, \bar{x}^p = \bar{Y}^p)$$

In words, all shares of native variables that respect equation 2.1 are equal probable.

In order to implement a vector function $(z^1, \dots, z^q) = f(x^1, \dots, x^p)$, a set of functions f_i which compute together the output(s) of f is needed: vector (f_1, \dots, f_s) is the vector of shares of f , called *realization*.

To have a protection from side-channel attacks using first statistical moment, sharing must respect three properties:

1. **Correctness**

A realization of f is correct if and only if

$$(Z_1, \dots, Z_q) = f(X^1, \dots, X^p) = \sum_{i=1}^s f_i(\bar{X}^1, \dots, \bar{X}^p)$$

for all vectors of input shares $(\bar{X}^1, \dots, \bar{X}^p)$.

2. Non-Completeness

Every function is independent of at least one share of the input variable x and consequently, independent of at least one share of each component. If the reduced vector $(x_1^j, \dots, x_{i-1}^j, x_{i+1}^j, \dots, x_s^j)$ is denoted by \bar{x}_i^j , without loss of generality z_i are required to be independent of $x_i^j, \forall j$, i.e.:

$$\begin{aligned} z_1 &= f_1(\bar{x}_1^1, \bar{x}_1^2, \dots, \bar{x}_1^p) \\ z_2 &= f_2(\bar{x}_2^1, \bar{x}_2^2, \dots, \bar{x}_2^p) \\ &\dots \\ z_s &= f_s(\bar{x}_s^1, \bar{x}_s^2, \dots, \bar{x}_s^p) \end{aligned}$$

3. Uniformity

A realization is uniform if for all distributions of the inputs (x_1, x_2, \dots, x_p) and for all input share distributions satisfying property 2.2, the conditional probability

$$\Pr(\bar{z}_1 = \bar{Z}^1, \dots, \bar{z}_q = \bar{Z}^q | z^1 = \sum_{i=1}^s Z_i^1, \dots, z^q = \sum_{i=1}^s Z_i^q)$$

is constant.

In a realization satisfying these three properties, each of the output shares z_i^j is statistically independent of the input variables x^j and the output variables z^j . The same holds for all intermediate results and physical quantities (power consumption, electromagnetic radiation, ...). Then, this realization is protected from first order side-channel attack, also in presence of glitches.

Property 1 and Property 2 impose a lower bound on the number of shares s , as is shown in the following theorem, presented in article [95].

Theorem 2.3.1. *The minimum number of shares required to implement a product of d variables with a realization satisfying Property 1 and Property 2 is given by*

$$s = 1 + d$$

Proof. For the proof of this Theorem, see [95], Theorem 3. □

Observation 1. Starting from theorem 2.3.1, it follows that at least three shares are needed to implement a nonlinear function.

2.3.3 Physical Defaults

In this Section, we discuss the concept of *physical defaults*, which can be exploited for (side-channel) attacks. The following presentation is based on explanations in [60].

Physical defaults happening in a circuit can be attributed to one of the following types (Figure 2.5).

- *Combinatorial recombinations (glitches)*. In this case, they are caused by the mixing (recombination) of the inputs during the operations implemented by gates. Note that recombination actually happens in most cases for standard circuits [23]. Generally, to mitigate the effect of glitches, registers are added in strategic positions in the circuit (Section 3.1), in such a way that partial results are saved and glitches don't propagate anymore. In this thesis, the discussion is focused on this type of physical defaults and their threats in circuits.
- *Memory recombinations (transitions)*. Memory recombinations can mix the content of the memory elements in consecutive cycles. This would happen if the same memory gate is used to store both the inputs and output of a cycle.
- *Routing recombinations (couplings)*. In this case, the leakage can happen if there is a mix of the shares manipulated by adjacent wires.

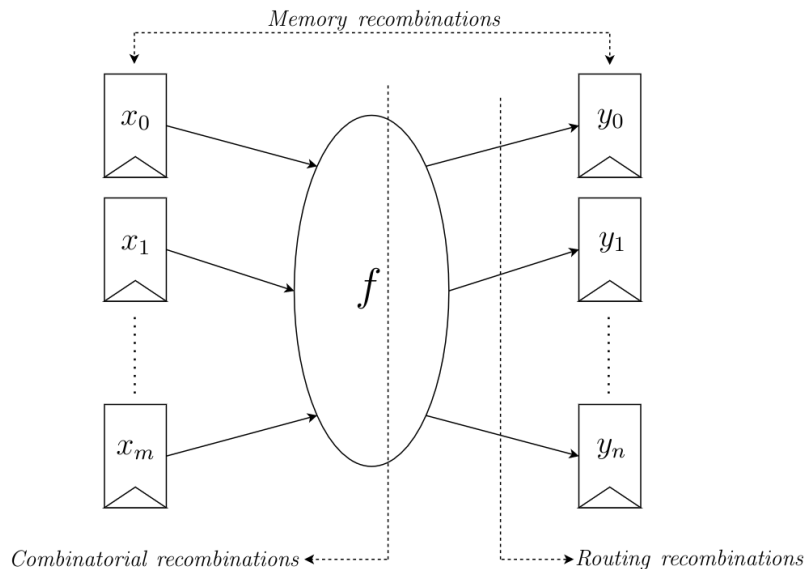


Figure 2.5: Three cases of possible recombinations: combinatorial (glitches), memory (transitions), routing (coupling).

2.4 Protecting secrets during computations

Now the malicious considered is an *internal entity*, then an attacker that is involved by the computation. Then there are two or more entities that want to produce an output from their own secrets, keeping them unshared with the others. In this case, the exploited procedures are called *Secure Computation Protocols* [79][4].

Example 2.4.1. A classic example, usually proposed to explain the concept of protection of secret from internal malicious, is the *Millionaires' Example*. Suppose that three millionaires want to know the sum of all their money, without sharing their capitals. Then a protocol that they can use is the following: the first millionaire chooses a key k and add it to his capital c_1 , i.e., he computes $c_1 + k$ and sends it to the second millionaire. The second adds his capital c_2 to what he has received, i.e., he computes $(c_1 + k) + c_2$ and sends it to the third millionaire. The last adds his capital c_3 , i.e., he computes $(c_1 + k + c_2) + c_3$ and sends it to the first. Now this millionaire (the only one that knows the key) can subtract k to what he has received, recovering the total amount of their money. Finally, he shares the result with all the others. Note that, if the participants are only two, this protocol is not useful anymore, because, when the second millionaire sends $(c_1 + k) + c_2$ to the first, this can recover c_2 (knowing c_1 and k).

2.4.1 Secure computation

Secure Computation (SC) is a term describing all those technologies that allow the computation of a function without revealing the inputs, which remain secret [4]. The wide collection of secure computation protocols includes also the well known Multi-Party Computation Protocols, which is a secure computation technology programmable, i.e., it is Turing-complete.

Definition 2.11. A *Secure Multi-Party Computation (MPC) Protocol* enables a set of parties to interact and compute a joint function of their private inputs, without revealing nothing about the secrets and sharing only the output of the function. This computation have to be performed without a third-part meddling (for example, authority parties) [79].

If the participants are only two, then it is a *two-party computation protocol*.

The participants could be dishonest and become internal malicious. In this context, usually secure computation protocols are divided into *passive*, i.e., those that are secure against adversaries who follow the protocol but aim to learn all they can

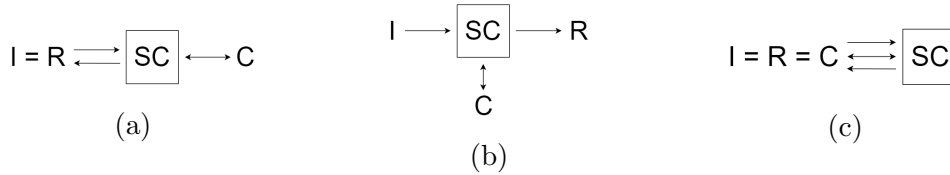


Figure 2.6: Schemes of parties role in SC protocols. (a) Homomorphic Encryption. (b) Linear Secret Sharing. (c) Garbled Circuit.

from observing the execution, or *active*, i.e., those that are secure also against parties trying to learn information about the inputs cheating and do not following the protocol.

In [4], an exhaustive classification of SC protocols is explained, which is based on a definition of participating parties and their role (Bogdanov model [28]): *input parties* (I) are those that provide their data for the computation, *computing parties* (C) are those that effectively execute the computation, and *result parties* (R) are those that reveal the outputs of the computation. The inputs keys remain secret during all the protocol, and only outputs are shared at the end. Three of the major SC paradigms are *Homomorphic Encryption (HE)*, *Linear Secret Sharing (LSS)* and *Garbled Circuit (GC)*; explicative schemes are in Figure 2.6.

Homomorphic Encryption. This protocol is generally used when a single computing party, the server, offers its resources to the other two parties, that coincide in the client (Figure 2.6a). HE is a two-party protocol in which the client has the keys and outsources some computation to the server, by providing it with the encrypted inputs and requesting a result from it.

Figuratively, HE can be associated to an opaque locked glove box: the client inserts its inputs and locks the box. Anyone with the box can use the gloves to manipulate the items, but only the box’s owner has the key to open it and remove the contents.

Linear Secret Sharing. This protocol is used when more parties collaborate (Figure 2.6b). In this SC paradigm, the input parties can divide secrets into shares such that, however chosen a subset of them, it does not reveal information about the secret. Each computing party receives a share, and they collaboratively execute interactive protocols to compute operations.

Figuratively, this protocol can be associated to a set of interconnected gloves boxes with input hatches. Any party distributes its input between the boxes through the hatches, and the box operators can exchange pieces and manipulate the introduced

values using the gloves. The result is obtained only when all boxes are opened, and their results combined.

Garbled Circuit. This protocol is mostly implemented when two parties want to compute a function over their inputs, and then they are input, result and computing parties together (Figure 2.6c). Garbled circuit is an integrated digital circuit in which the wire values are encoded as random strings and each gate's truth table is encrypted. Ideally, each garbled circuit can be exploited for only one execution, because the materials used in the construction are fragile and dissolve after use. For more details, see Section 2.4.2.

2.4.2 Garbled circuit

The aim of two-party secure computation protocols is performing collaborative computation between two parties who do not want to disclose the input values they own. In the general formulation, two parties called Alice and Bob want to compute a function on their respective inputs, which must remain private. The first protocol with this aim was presented in the seminal work by Yao et al. [122] and is usually referred to as the Garbled Circuit (GC) construction of Yao, where the function to be computed securely is represented as a simple Boolean circuit c_f .

In this protocol, Alice is called the garbler and Bob the evaluator [43]. Alice produces the garbled circuit c_g from c_f , and together the parties evaluate it. The protocol is based on three steps: garbling, evaluation and sharing of the output.

Garbling. The garbling is obtained by producing a garbled table for each gate contained in the circuit. Considering a gate g in circuit c_f , with input wires W_i and W_j , Alice randomly chooses two secret keys, w_i^0 and w_i^1 corresponding to the garbling of input values 0 and 1, and does the same for the other input wire, selecting the keys w_j^0 and w_j^1 . Through this procedure, Alice produce a garbled table for each gate in the circuit, and sends them to Bob.

Example 2.4.2. *In order to better understand the Yao's construction, consider a circuit composed of a single gate with two input wires, w_a and w_b , and one output wire w_z (Figure 2.7). Alice chooses the secret keys w_a^0 and w_a^1 for her input, w_b^0 and w_b^1 for Bob's input and w_z^0 and w_z^1 for the output. She produces an encrypted table and transforms it in a garbled table permuting its rows. Finally, Alice sends it to Bob (Table 2.10).*

2.4.2 Garbled circuit

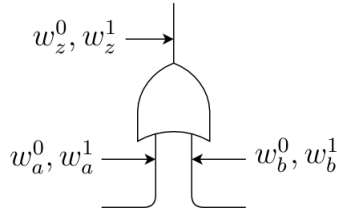


Figure 2.7: Garbled *or* gate.

truth table		encrypted table	garbled table
0	0	$E_{w_a^0}(E_{w_b^0}(w_z^0))$	$E_{w_a^1}(E_{w_b^0}(w_z^1))$
0	1	$E_{w_a^0}(E_{w_b^1}(w_z^1))$	$E_{w_a^1}(E_{w_b^1}(w_z^1))$
1	0	$E_{w_a^1}(E_{w_b^0}(w_z^1))$	$E_{w_a^0}(E_{w_b^0}(w_z^0))$
1	1	$E_{w_a^1}(E_{w_b^1}(w_z^1))$	$E_{w_a^0}(E_{w_b^1}(w_z^1))$

Table 2.10: Alice's computations.

Evaluation. Alice encrypts her secret input x_a and sends it to Bob. After that, she also encrypts Bob's secret input x_b without knowing its value, (this procedure is possible thanks to an oblivious protocol). Bob obtains the garbled output simply by evaluating the garbled circuit gate by gate, using the garbled tables previously sent by Alice. Note that Bob can evaluate the garbled circuit on the two inputs ignoring the value of x_a , while Alice does not learn anything about x_b (apart from what the parties can deduce from the result and their current input).

Example 2.4.3. In the situation described in Example 2.4.2, consider for example that $x_a = 1$ and $x_b = 0$. Then Alice sends to Bob w_a^1 and w_b^0 , and Bob computes $E_{w_a^1}(E_{w_b^0}(w_z^1))$ in the garbled table.

Sharing of the output. Bob sends to Alice the result of his evaluation on c_g , and Alice decrypts it (only she knows the keys). Finally, she shares the result of c_f with Bob.

2.4.2.1 Oblivious transfer protocol

The protocol of oblivious transfer (OT) is an important concept needed for the execution of the Yao's GC Protocol, that allows to send a single value without either the sender learning which exact value was received or the receiver finding out any other value than the one he actually intended to receive [106]. In particular, the Yao's GC makes use of a 1-out-of-2 Oblivious Transfer (1-2 OT), since it works on a set of only two values, which provides security against semi-honest adversaries.

Formally, assume that S (the sending party) holds a pair of strings (s_0, s_1) one of which is to be sent to R (the receiving). R selects $i \in \{0, 1\}$, depending on whether she wants to learn s_0 or s_1 . She then generates a pair of asymmetric cryptographic keys (k^{priv}, k^{pub}) , and another value \tilde{k} that looks like a public key to S , but to which R has no private key. Then, R chooses the working public key to be k_i^{pub} and \tilde{k} as k_{i-1}^{pub} , and advertises them to S as keys for s_0 and s_1 , respectively. S encrypts s_0 with the received k_0^{pub} and s_1 with k_1^{pub} and transmits the resulting c_0 and c_1 to R , who

will then decrypt her desired value c_i with the corresponding k_i^{priv} , which results in the correct s_i . R will not be able to decrypt the c_{i-1} because she has not generated a corresponding private key for \tilde{k} , and S will not know which value R has actually seen.

2.4.2.2 Cost of gates' transfer

In Yao's GC implementations, each gate has a cost, that refers to computation and communication required for creation, transfer, and evaluation of its garbled table [78]. Generally, it is considered proportional to the number of rows in the garbled table that has to be sent for each gate, since they must be transferred one-by-one from the garbler to the evaluator. Then, in Boolean logic circuits, gates with 2 inputs and 1 output cost 4 rows of the garbled table (e.g. the one in Example 2.4.2).

2.4.2.3 Send *xor* gates for free

In [78] Kolesnikov and Schneider present an optimization, which allows the evaluation of *xors* for free, avoiding any interaction between the two parties for such gates. In other words, there is no need to compute and send the garbled tables for the *xor* gates.

Let G be a *xor* gate that has two input wires W_a , W_b and output wire W_c . Their idea is to garble the wire values as follows. Randomly choose w_a^0 and w_b^0 and $R \in \{0, 1\}$. Set $w_c^0 = w_a^0 \oplus w_b^0$, and $\forall i \in \{a, b, c\} : w_i^1 = w_i^0 \oplus R$. It is easy to see that the garbled gate output is simply obtained by xoring garbled gate inputs:

$$\begin{aligned} w_c^0 &= w_a^0 \oplus w_b^0 = (w_a^0 \oplus R) \oplus (w_b^0 \oplus R) = w_a^1 \oplus w_b^1 \\ w_c^1 &= w_c^0 \oplus R = w_a^0 \oplus (w_b^0 \oplus R) = w_a^0 \oplus w_b^1 = (w_a^0 \oplus R) \oplus w_b^0 = w_a^1 \oplus w_b^0 \end{aligned}$$

Further, the garblings w_i^j do not reveal the wire values they correspond to.

This reasoning shows how the *xor* gate in a Boolean logic can be evaluated without communication between the parties. The optimization then requires that there is only a global random R known only to Alice the garbler, such that the garbled value corresponding to 1 for a wire is determined by xoring the garbled 0 value with the quantity R . In this way, computing the output for a garbled *xor* gate amounts to compute the value resulting by the *xor* of the two inputs. Security of this solution is proved under different assumptions in [78, 97].

From AIG to XAG. Before the Kolesnikov and Schneider's paper, the most widespread circuit construction was the **And-Inverter graph (AIG)**, i.e., a directed, acyclic graph, consisting of two-input nodes representing logical conjunction,

terminal nodes labelled with variable names, and edges optionally containing markers indicating logical negation. This was the most employed and studied circuits' form mainly because the area occupied by the *xor* gates is bigger than that of the *and* gates.

When sending *xors* in a garbled circuit protocol was declared without any cost, then the attention is gradually shifted to the **Xor-And graph (XAG)**, which is a representation of the Boolean functions on a basis of gates that are *xor*, *and* and *not*, trying to minimize the number of *ands* because they are those costing in the transfers during the protocol (see Chapter 5 for more details and examples on XAGs).

2.4.2.4 Reduction rows in the encrypted table

Trying to reduce the cost needed to send a garbled table, in [97] is proposed a trick that allows to decrease the size of the tables of the non-*xor* gates. This optimization provides a 25% reduction in the sizes of the tables needed to represent two-inputs gates.

The trick is that, instead of defining the two garbled values of the output wires randomly, Alice the garbler defines one of them as a function of garbled values of the two input wires. In other words, she chooses an input pair (b_1, b_2) with $b_i \in \{0, 1\}$, and defines the garbled output value of $c_g(b_1, b_2)$ to be a function of the garbled values of b_1 and b_2 . The gate table therefore needs not store an entry for the input combination (b_1, b_2) . In the evaluation phase, if Bob the evaluator has the garbled values of the pair (b_1, b_2) he can compute the corresponding garbled output directly, without consulting the garbled table.

Some years later, in [124] the authors present a method for garbling *and* gates that requires only two rows in the garbled table. It consists in the employment of the **half-gates** procedure, involving *and* gates for which one party knows one of the inputs in clear. In the paper is described how to construct half-gates when the garbler knows one of the inputs, and the evaluator knows one of the inputs, using one ciphertext each party, in a way that is compatible with free-*xor*. Then, an *and* gate can be written as a combination of *xors* and two half-gates of opposite types. Hence, the resulting *and* gate uses only two ciphertexts in combination with free-*xor*, and then its garbled table has only two rows.

Part I

Protecting secrets from an external attacker

CHAPTER 3

PROBING SECURITY

In Section 2.3.2.1, we discuss about side-channel attacks, with a particular focus on power consumption attacks, i.e., those in which the attacker is able to collect power traces and use them to recover information about secrets.

Following the same idea of attacks on hardware computing cryptographic operations, in this chapter we present *probing attacks*, starting from the state of the art, and concluding with some interesting and innovative developments.

In a probing attack to an hardware device, the attacker is able to place some probes (for example, metal needles) on wires of a circuit and read with them some punctual information flowing in those wires. Usually, what the attacker can read with a probe is the value carried along the wires (bit) or the power consumed in that specific point of the circuit.

Many countermeasures against these attacks have been developed in the last years, trying to protect circuits also from probing attacks that exploit glitches occurring in the circuit.

The goal of this Chapter is to discuss about probing analysis, and introduce a new methodology to identify if a circuit is not probing secure, with an exact final claim. It is finalized both with and without glitches. We also develop a tool implementing it, based on BDDs. In Section 3.1 we propose an analysis of the state of the art about this topic, both with and without glitches. In Sections 3.2, 3.3, 3.4 and 3.5 we discuss some further innovative studies about this argument, that have been presented in four papers of which the author of this thesis is co-author.

3.1 State of the art

This Section is divided into two parts: in the first part there is a discussion about the classical probing security, i.e., the one without considering the effects that glitches create inside a circuit. In the second part, the robust probing security is presented, i.e., the one in which the attacker is able to exploit also the information recovered thanks to glitches.

3.1.1 Probing security

All the amount of works about probing security takes origin to paper [73], in which the concept of probing security is developed, taking inspiration by notes in [3], written some years earlier. In paper [73], following definition is given:

Definition 3.1. A *probing attack* is an attack computed placing a metal needle on a wire of interest and reading off the value carried along that wire during the device's operations.

The *order of a probing attack* t is the number of probes that the attacker can place in the circuit. If with those probes she can mount an attack, then it is called a t -probing attack. Otherwise, the gadget is considered *t-probing secure*. Ideally, the attacker could recover all k bits of the key placing k probes in proper positions on circuit's wires. However, each probe has a specific cost and dimension, then in a circuit it is possible to place only a limited number of them, depending on its physical features and the employed tools' power. Consequently, the order of a probing attack on a specific circuit has an upper bound, and the ideal case previously described is substantially impracticable.

Generally, to face to attack of higher order (i.e., in presence of a more powerful attacker), more complex hardware constructions are studied. Many efforts are done to find secure solution to protect the *and* operation in circuits. The reason of this interest resides in the fact that the circuit's nonlinear part generally is the one leaking more information about secrets.

Ishai-Sahai-Wagner multiplication. In [73], the authors propose a construction that provide a generic defence against probing attacks (denoted here as ISW scheme). This construction is finalized to protect the *and* gate in circuits.

Let a , b two input bits of an *and* gate, and c the output bit, such that $c = a \cdot b$. To protect information about a , b from a probing attack in which the attacker can place at most t probes, they give the following procedure.

3.1.1 Probing security

First step. The inputs are split into $t + 1$ shares (see Section 2.3.2.1), i.e., sets $\{a_0, \dots, a_t\}$ and $\{b_0, \dots, b_t\}$ are defined, such that $a = \bigoplus_i a_i$ and $b = \bigoplus_i b_i$.

Second step. For each $1 \leq i < j \leq t$, a random bit $z_{i,j}$ is generated. Then, $z_{j,i}$ are computed, such that $z_{j,i} = (z_{i,j} \oplus a_i \cdot b_j) \oplus a_j \cdot b_i$. Note that individually each $z_{i,j}$ is distributed uniformly, but any pair $z_{i,j}$ and $z_{j,i}$ depend on a_i, a_j, b_i and b_j .

Third step. The output bits $\{c_0, \dots, c_t\}$ (such that $c = \bigoplus_i c_i$) are computed as

$$c_i = a_i \cdot b_i \oplus \bigoplus_{j \neq i} z_{i,j}.$$

This construction is provably secure, and to verify that, in [73] the author develop a formal model of the adversary and propose definitions of security against probing attacks. This scheme doubles the number of shares to achieve t -probing security. It is proved in [99] that the scheme is actually t -probing secure with the optimal number of $t + 1$ shares only.

Example 3.1.1. Consider for example the ISW scheme that give a protection against first order probing attacks, i.e., with 2 shares. The secure construction is (Figure 3.1a):

$$\begin{aligned} c_0 &= a_0 \cdot b_0 \oplus z_{0,1} \\ c_1 &= a_1 \cdot b_1 \oplus ((a_0 \cdot b_1 \oplus z_{0,1}) \oplus a_1 \cdot b_0) \end{aligned}$$

Same scheme, but secure against second order probing attacks, is the following construction:

$$\begin{aligned} c_0 &= a_0 \cdot b_0 \oplus z_{0,1} \oplus z_{0,2} \\ c_1 &= a_1 \cdot b_1 \oplus ((a_0 \cdot b_1 \oplus z_{0,1}) \oplus a_1 \cdot b_0) \oplus z_{1,2} \\ c_2 &= a_2 \cdot b_2 \oplus ((a_0 \cdot b_2 \oplus z_{0,2}) \oplus a_2 \cdot b_0) \oplus ((a_1 \cdot b_2 \oplus z_{1,2}) \oplus a_2 \cdot b_1) \end{aligned}$$

Trichina multiplication. In [115] the author presented the Trichina *and* gate, that allows to implement a masked *and* gate ($c = ab$) securely in the probing security context, only at the first order. The construction requires two shares for each secret, i.e., $a = a_0 \oplus a_1$ and $b = b_0 \oplus b_1$, and a random z , and its equation is:

$$c_0 = (((z \oplus a_0 \cdot b_1) \oplus a_1 \cdot b_0) \oplus a_1 \cdot b_1) \oplus a_0 \cdot b_0$$

Note that the security of this scheme relies strictly on the order of the operations to avoid unmasking certain bits, on the assumption that the sharing of a are independent from those of b (Figure 3.1b).

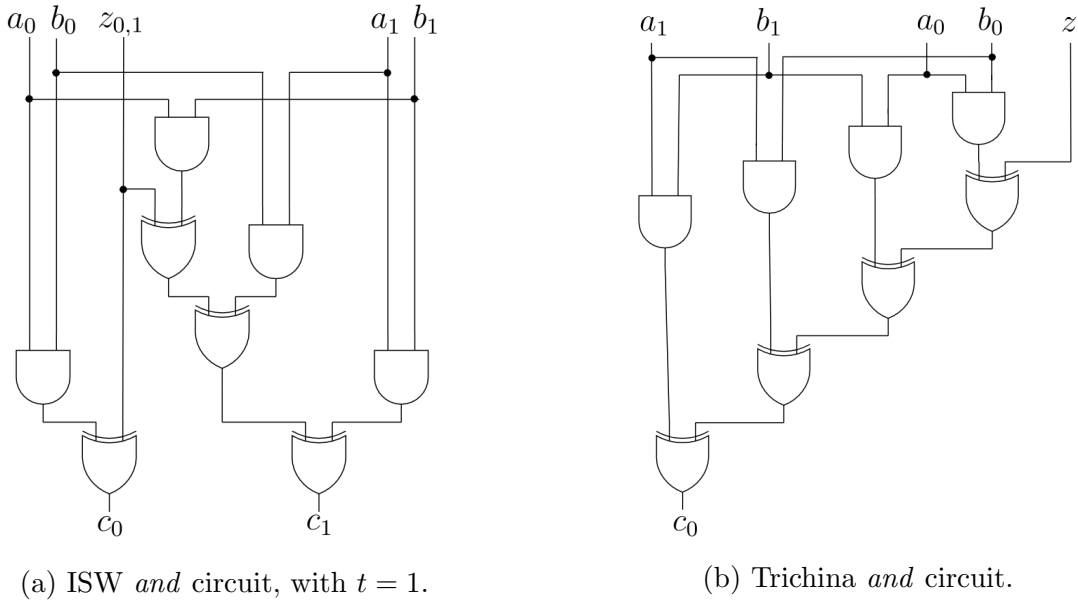


Figure 3.1: Two state of the art probing secure circuits.

Threshold Implementation multiplication. In Section 2.3.2.1, we present the Threshold Implementation solution against glitches [95]. Here, we explain how to apply the scheme to protect an *and* gate. At first, when a single *and* gate is considered, it has been shown that there exists no 3-sharing satisfying both uniformity and first-order non-completeness [25]. Therefore, the output shares of the 3-shares *and* gate must be refreshed. Then, considering an *and* and a *xor* gate instead of a single *and* gate, i.e., considering the function $d = c \oplus a \cdot b$ (Toffoli gate), it is possible to reach all TI properties with the following construction:

$$\begin{aligned}
 d_0 &= c_1 \oplus a_1 \cdot b_1 \oplus a_0 \cdot b_1 \oplus a_1 \cdot b_0 \\
 d_1 &= c_2 \oplus a_2 \cdot b_2 \oplus a_2 \cdot b_1 \oplus a_1 \cdot b_2 \\
 d_2 &= c_0 \oplus a_0 \cdot b_0 \oplus a_0 \cdot b_2 \oplus a_2 \cdot b_0
 \end{aligned} \tag{3.1}$$

Note that this construction results 2 probing secure only if assuming that the gadget is implemented in a single cycle. In this case, the adversary can only probe the input shares a_i, b_i, c_i and output shares d_i , but not the intermediate values.

Observation 2. Note that in all these constructions the number of shares s in which the secrets are split increases in relation to the order of the probing attack t . In particular, in all of them $s = t + 1$. This is the smarter trade-off that is possible to reach, because:

- if the shares' number s is less or equal to t , then with s probes placed on the wires that connect the inputs to the netlist the attacker can recover all the shares and the secret itself;

- construction of the circuit becomes trickier with the increase of the shares' number. Indeed, in this process the correctness property must be respected any time.

3.1.1.1 The composability problem

Since the appearance of this first solution against probing attacks, it becomes clear that, while proving the security for small gadgets (portion of circuit) requires a small effort, reasoning about their composition is not so trivial. In fact, one of the main problems studied in literature is the *composability* of security properties, i.e., determining if the functional composition of two t -probing secure gadgets is still t -probing secure. Over the years, it has been observed that composability depends on the amount of used refreshing [50] and on some properties of the circuits' implementation, that are called non interference and strong non interference [9, 8]. These properties ensure that the probabilistic distribution of the probed values does not depend on all of the secret's shares.

Definition 3.2. A function is t -non interfering (shortly t -NI) if, when given o outputs and i internal probes (i.e., probes placed on wires that are internal to the circuit), with $o + i \leq t$, it implies a dependency with maximum $o + i$ input shares.

Definition 3.3. A function is t -strong non interfering (shortly t -SNI) if, when given o outputs and i internal probes, with $o + i \leq t$, it implies a dependency with maximum i input shares.

These properties give information about the composability of gadgets. Generally, to prove that a composition of gadgets is probing secure becomes a long and complex task, and sometimes requires much expertise in type theoretical or formal validation area [9].

Example 3.1.2. *This Example is often shown in literature [50], and is used to explain how one could identify a violation when some (declared secure) gadgets are composed.*

Figure 3.2 shows the structure of a function $h(a)$ which is a composition of two functions f and g . Function f refreshes its input $a = (a_0, a_1, a_2)$ with two random bits $r^f = (r_0^f, r_1^f)$:

$$o^f(a, r^f) = [a_0 \oplus r_0^f \oplus r_1^f, a_1 \oplus r_0^f, a_2 \oplus r_1^f]$$

while g is the ISW multiplication [73] which consumes 3 random bits $r^g = (r_0^g, r_1^g, r_2^g)$

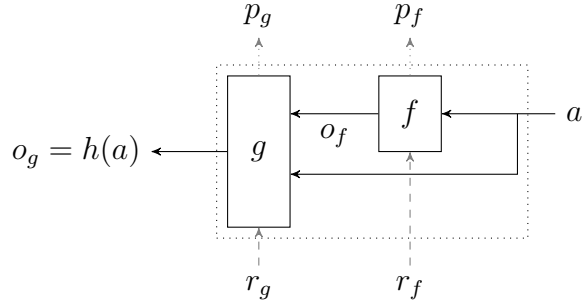


Figure 3.2: The composition pattern of f (t -NI) and g (t -SNI) studied in Example 3.1.2: the composed function $h(a)$ is not t -NI.

for the secret computation:

$$\begin{aligned} o^g(o^f, a, r^g) = & [o_0^f \cdot a_0 \oplus r_0^g \oplus r_1^g, \\ & o_1^f \cdot a_1 \oplus ((o_0^f \cdot a_1 \oplus r_0^g) \oplus o_1^f \cdot a_0) \oplus r_2^g, \\ & o_2^f \cdot a_2 \oplus ((o_0^f \cdot a_2 \oplus r_1^g) \oplus o_2^f \cdot a_0) \oplus ((o_1^f \cdot a_2 \oplus r_2^g) \oplus o_2^f \cdot a_1)] \end{aligned}$$

From Definition 3.2, it's clear that function f is a 2-NI function: however choosing two elements among the outputs and the internal probes, at most two shares of a are recovered. Moreover, function g is 2-SNI, because the attacker can recover at most as shares as the number of internal probes that places.

Now, considering the composition function h , an attacker can place an internal probe $p_0 = a_0 \oplus r_0^f$ on f and an internal probe $p_1 = o_1^f \cdot a_2$ on g . In this way, since o^f is the output of f :

$$\begin{aligned} p_0 &= a_0 \oplus r_0^f \\ p_1 &= (a_1 \oplus r_0^f) \cdot a_2 \end{aligned}$$

From probe p_1 , the attacker recovers information about both $a_1 \oplus r_0^f$ and a_2 , because the multiplication is correlated with both the factors. Then XORing $a_0 \oplus r_0^f$ and $a_1 \oplus r_0^f$, a_0 and a_1 are unmasked and finally she has all the three shares of a . This proves that the composition of a t -NI gadget with a t -SNI one can be not probing secure.

The *and* construction proposed by Ishai et Al. is t -SNI [9]. Also Trichina *and* is declared t -SNI, but only in the case in which the order of operations doesn't change.

Probe-isolating non-interference. To overcome the composability problem, recently the new security notion of Isolating Non-Interference (PINI) has been proposed [41]. The main difference between the (Strong) Non-Interference (NI/SNI) definitions

3.1.2 Robust probing security

and PINI is that the former rely on the number of probes in a target implementation, while the latter rather relies on their position (i.e., the shares' indices). The whole circuit can then be cut into d circuit shares that are not interconnected, except for non-linear gadgets. If we neglect those gadgets, the circuit is t -probing secure (for $t < d$): the adversary can only probe t of the circuit shares, hence it has no information about at least $d - t \geq 1$ circuit shares, which contains at least one share of each input. Non-linear PINI gadgets then behave in the probing model as if they had no connection between circuit shares, which allows implementing non-linear functions while keeping the previous circuit sharing intuition. More formally:

Definition 3.4. Let G be a gadget over d shares, $x_{i,j}$ and $y_{i,j}$ its inputs and outputs (respectively), such that all the inputs and outputs in the circuit share i are denoted as $x_{i,*}$ and $y_{i,*}$. Let P be a set of t_1 probes on wires of G (internal probes), and A a set of t_2 share indices. G is *t -Probe-Isolating Non-Interfering (t -PINI)* iff for all P and A such that $t_1 + t_2 \leq t$, there exists a set B of at most t_1 share indices such that probes on the set of wires $P \cup y_{A,B}^G, *$ can be simulated with the wires $x_{A \cup B}^G, *$.

The authors in [41] prove that a t -PINI gadget is t -probing secure and composition of t -PINI gadgets is t -PINI.

3.1.2 Robust probing security

Recently, significant efforts have been devoted to the design of masking gadgets for hardware implementations. In this context, one important additional issue is that physical defaults such as glitches can easily contradict the independence assumption required for secure masking (Section 2.3.3). Since this break of the independence assumption directly leads to devastating attacks, the literature then focused on the design of gadgets with better resistance to glitches. Since then, various papers proposed innovative ways to implement higher-order masking in hardware, also interesting in the probing security area [98, 47, 67, 10].

Definition 3.5. A probe is called *extended* if it is placed in circuit leaking information due to glitches. By contrast, the probe considered when no physical defaults are considered are called *classical*.

Note that, if the probe is extended, (i.e., if the attacker can exploit also glitches), the attacker can recover (in the best case for her) all the inputs concurring to the computation of the value flowing in the wire on which the probe is placed.

Example 3.1.3. Consider for example the following function:

$$c = a \cdot b \oplus r.$$

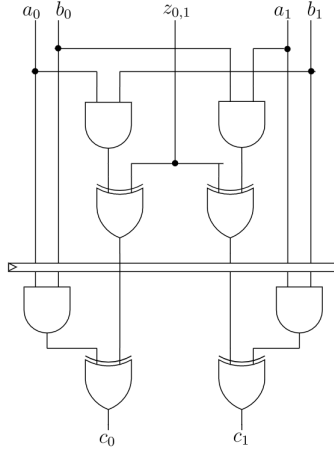


Figure 3.3: ISW *and* circuit, with $t = 1$, in case of glitches.

If an attacker places a classical probe on the output, then she can know only the combination $a \cdot b \oplus c$ of the inputs. Instead, if the probe is extended, she can recover (in the best case for her) all the values in the set $\{a, b, r\}$.

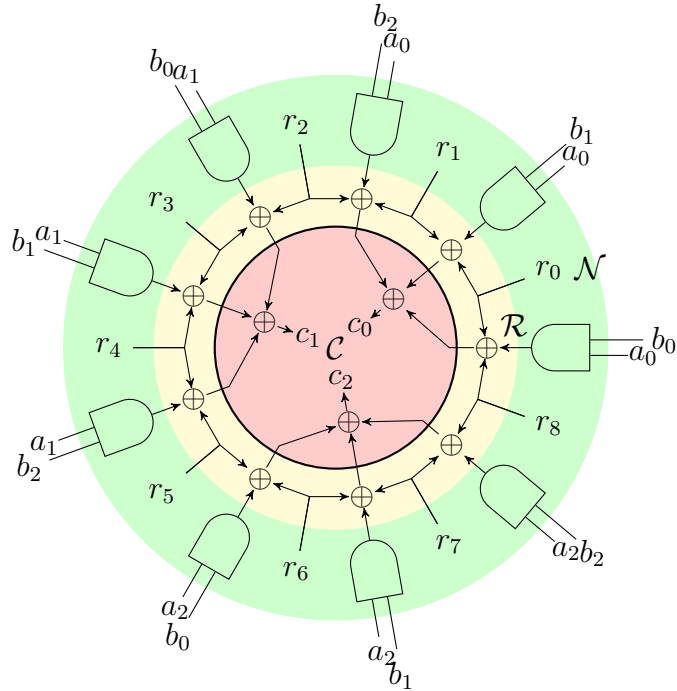
If an extended probe is placed on an output wire, it is called *external* extended probe, while if it is placed on an internal wire, it is called *internal* extended probe.

Observation 3. Usually, to prevent leakages due to glitches, besides randoms addition, a useful and wide technique is to add some register in a smart way, to protect intermediate values. Note that this patch causes an increase of the latency of the circuit. Then, the study of a new construction to preserve security in presence of glitches implies finding a trade-off between the randomness usage (increase of area) and the added registers (increase of latency).

Ishai-Sahai-Wagner multiplication. With a smart reordering of the intermediate values of the scheme, and saving refreshed results in registers, the ISW *and* is robust t probing secure [9]. Explicitly, for $t = 1$ (square brackets indicate registers):

$$\begin{aligned} c_0 &= a_0 \cdot b_0 \oplus [a_0 \cdot b_1 \oplus z_{0,1}] \\ c_1 &= a_1 \cdot b_1 \oplus [a_1 \cdot b_0 \oplus z_{0,1}] \end{aligned}$$

This construction is depicted in Figure 3.3. Note that for example, without these registers, an external extended probe placed on the output wire $c_0 = a_0 \cdot b_0 \oplus (a_0 \cdot b_1 \oplus z_{0,1})$ would give information about two shares of a , rendering all the scheme not robust t -probing secure.


 Figure 3.4: CMS scheme when $t = 2$, and then with 3 shares.

Consolidating Masking Scheme for multiplication. This scheme has been presented in [98], with the aim to define a secure construction for the *and* gate also in presence of glitches. Unfortunately, it has been proved secure only since the second order, i.e., it is not t probing secure for $t \geq 3$ [92].

The Consolidating Masking Scheme (shortly CMS) descends from a reorganization of the three more studied probing secure schemes, namely ISW, Trichina and TI (Section 2.3.2.1), trying to exploit better features from all, i.e., probing security from ISW and Trichina, glitches resistance from TI. The result is the general construction

$$c_i = \bigoplus_{j=0 \dots t} (a_i \cdot b_j \oplus r_{i \cdot (t+1) + j} \oplus r_{i \cdot (t+1) + j + 1}) \quad (3.2)$$

for $i = 0 \dots t$. This scheme involves 2^{t+1} randoms, and in Equation 3.2 is $r_{t \cdot (t+1) + t + 1} = r_0$. In particular, case when $t = 2$ is reported in Figure 3.4.

The construction is decomposed into three layers. The *non-linear layer* \mathcal{N} (ring green in Figure 3.4), is the part in which the products $a_i \cdot b_j$ are computed. Note that this layer maps $2 \cdot (t + 1)$ inputs in $(t + 1)^2$ outputs. In the *refresh layer* \mathcal{R} (ring yellow in Figure 3.4) the 2^{t+1} randoms are xored to the products computed in \mathcal{N} ; main features of this ring descend from ISW scheme. The outputs of this layer are in number $(t + 1)^2$. In the last *compression layer* \mathcal{C} (ring red in Figure 3.4), outputs of the construction are computed, xoring all the outputs of \mathcal{R} $(t + 1)$ -by- $(t + 1)$.

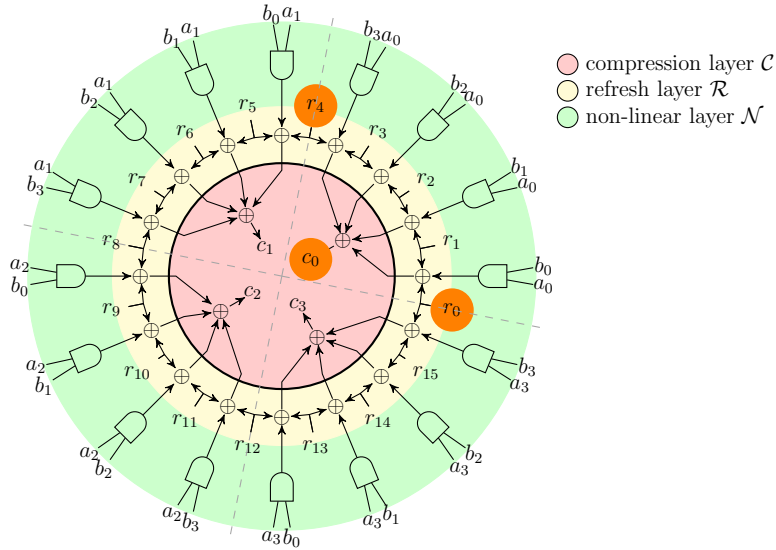


Figure 3.5: CMS scheme when $t = 3$, and then with 4 shares.

To satisfy the non-completeness property and avoid glitches causing leakage of more than the intended number of shares, it is crucial to isolate the \mathcal{R} and \mathcal{C} layers using registers (black ring in Figure 3.4).

All the c_i output shares are computed in a pseudo-isolating way, and each part of the construction outputting them is called *cone*. Every cone shares only two randoms with the adjacent ones, one on the left side and one on the right.

In [98], it is proved that the CMS scheme until $t = 2$ is robust probing secure, while in [92] the non-robust probing security of it is proved for $t \geq 3$. Note that this vulnerability exists even if one does not consider extended probes. Indeed, when $t = 3$ (Figure 3.5), an attacker can place for example a classical probe on the output c_0 , recovering the quantity

$$a_0 \cdot \left(\bigoplus_{j=0 \dots t} b_j \right) \oplus r_0 \oplus r_4$$

and two internal probes knowing values of r_0 and r_4 , with the aim to unmask the *xor* among all the shares of b .

Domain-oriented masking for multiplication. The domain-oriented masking (DOM) [67] is a generic masking scheme that leads to hardware designs which can be synthesized for arbitrary protection orders. DOM involves a relatively few randomness amount ($\frac{t+1}{2}$ randoms to achieve t probing security), without being vulnerable to glitches.

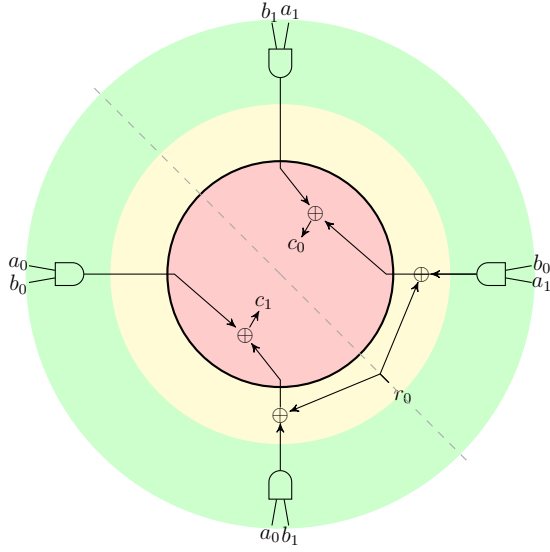


Figure 3.6: DOM scheme when $t = 1$, and then with 2 shares.

The core idea is that each share of a variable is associated with one share domain, i.e., shares of x are such that x_0 is associated to the 0–domain, x_1 to the 1–domain and so on. Then, in this construction, each multiplication between two shares from different domains (*cross domain* product) is refreshed and saved into registers. In opposition, multiplications between elements from the same domain are called *inner domain* products.

In [67] two DOM schemes are presented: the DOM-indep and the DOM-dep (the last has been declared not secure in presence of glitches [92]). For what concerns the DOM-indep scheme, its general form is described by the following equations:

$$\begin{aligned}
 c_0 &= \mathbf{a_0 b_0} + [a_0 b_1 + r_0] + [a_0 b_2 + r_1] + [a_0 b_3 + r_3] \dots \\
 c_1 &= [a_1 b_0 + r_0] + \mathbf{a_1 b_1} + [a_1 b_2 + r_2] + [a_1 b_3 + r_4] \dots \\
 c_2 &= [a_2 b_0 + r_1] + [a_2 b_1 + r_2] + \mathbf{a_2 b_2} + [a_2 b_3 + r_5] \dots \\
 c_3 &= [a_3 b_0 + r_3] + [a_3 b_1 + r_4] + [a_3 b_2 + r_5] + \mathbf{a_3 b_3} \dots \\
 &\dots
 \end{aligned}$$

where in bold there are the inner domains terms, that don't need to be stored in registers. In Figure 3.6 there is the representation of this scheme when $t = 1$. Note that, in relation to Figures 3.4 and 3.5, the construction is divided into three parts: nonlinear, refreshing and compression.

This scheme, is proved robust t -probing secure and also t -SNI [67, 92].

3.1.2.1 The robust composability problem

In literature, the problem to define secure and composable gadgets has been studied also when glitches are considered. In [59], it is exposed for the first time this issue.

Let G be a gadget declared t -SNI. Clearly, as long as one assumes that no information is leaked about the internal values, G remains t -SNI, but this model is unrealistic. Indeed, as just seen before, a concrete hardware implementation may leak about intermediate values via glitches. So, despite G is SNI in a classical model (namely, without considering glitches), in this context it is not SNI because the intermediate values can be leaked due to physical defaults, preventing any successful simulation.

On the other side, if the gadget G is declared glitch-resistant, this is not a sufficient condition for the composability, and then the composition of more of such gadgets can however leak information about the secrets.

This shows that these two properties alone are not enough to reason about probing security in hardware (i.e., in presence of glitches).

A first solution can be to add some *refresh gadgets*, namely gadgets with the function to mask the shares of the secrets [98]. But the more recent solution to this issue is given in [59], in which the robust probing model is defined.

Definition 3.6. A function is t -robust non interfering (shortly robust t -NI) if, when given o external output probes and i internal extended probes, with $o + i \leq t$, it implies a dependency with maximum $o + i$ input shares.

Definition 3.7. A function is t -robust strong non interfering (shortly robust t -SNI) if, when given o external extended probes and i internal extended probes, with $o + i \leq t$, it implies a dependency with maximum i input shares.

In [59], a notable result about robust composability is presented: if registers are inserted after t -SNI gadgets, a designer can deal with glitch robustness and composability separately.

Proposition 3.1.1. *If a gadget G storing its outputs in registers is both robust t -NI and t -SNI (without glitches), then it is also robust t -SNI.*

Proof. For the proof, see [59]. □

Ishai-Sahai-Wagner multiplication. In [59], the ISW scheme is declared robust t -SNI, with a little finessing, i.e., reordering the intermediate values, saving them after refreshing part in registers, and placing also other registers on the outputs, rendering the implementation of two cycles. Explicitly, for $t = 1$:

$$c_0 = [a_0 \cdot b_0 \oplus [a_0 \cdot b_1 \oplus z_{0,1}]]$$

$$c_1 = [a_1 \cdot b_1 \oplus [a_1 \cdot b_0 \oplus z_{0,1}]]$$

This means to add external registers to the construction in Figure 3.3. Note that now an extended probe placed on the output before been saved in register is an internal probe, preserving robust t -SNI definition. This is a clear example of the effects presented in Proposition 3.1.1.

Domain-oriented masking multiplication. DOM-indep scheme is not robust t -SNI [92], but, since the DOM-indep is proved to be t -SNI and robust t -NI, another time thanks to Proposition 3.1.1 it is possible to achieve the robust strong non-interference storing the outputs in registers.

Probe-isolating non-interference. Also in the context of the probe-isolating non-interference property, some considerations in the presence of glitches can be done.

Indeed, in [39] the authors prove that a compositional strategy that is correct without glitches remains valid with glitches. Then, they use this extended framework and present some masked gadgets that enable trivial composition with glitches at arbitrary orders. For example, they describe the Hardware Private Circuits 1 (HPC1), an efficient glitch-robust PINI multiplication gadget, proving its security at all orders and for any field \mathbb{F}_n .

3.1.3 Verification tools

In the literature, many efforts have been done to define tools verifying if a gadget is secure (probing secure, t -NI, t -SNI, ...). Generally, these verification tools are divided into software and hardware: in the first set are counted the automated methods to build or verify masked implementations in a "classical" probing situation, then when glitches are not considered. Instead, in the second set, there are those that declare if an implementation is secure also in case of glitches.

Software-based tools. The first work done in this direction is in [93], where the authors consider the use of automated methods to build or verify masked implementations. They propose and implement a masking compiler that track which variables are masked by random values and iteratively modifies an unprotected program until all secrets are masked. This strategy suffices to ensure security against first-order power attacks and works well on many examples. This method is mostly efficient

and scalable, but often overly conservative, and sometimes also secure programs are rejected.

In [11] a SMT-based method for analyzing the security of masked implementations against first-order power attacks is suggested: this method directly proves the statistical independence between secrets and leakage. The approach is limited to first order masking, but was extended to higher orders in [58]. Unfortunately, this method incurs an exponential blow-up in the security order, and also the improved version in [58] remains limited from this side.

Some years later, in [125] the authors propose a tool called SCINFER, in which they implement some abstraction-refinement techniques, providing significant improvement in terms of precision and scalability. Indeed, that tool alternates between fast and moderately precise approaches and computationally expensive but precise approaches.

Among all the methods appeared in literature to verify if a gadget is non interfering, the most relevant in the last years is certainly the MASKVERIF tool, presented for the first time at the EUROCRYPT conference [10]. In this work, the authors establish a tight connection between the security of masked implementations and probabilistic non interference, for which they suggest efficient verification methods. Specifically, they show how a relational program logic previously used for mechanizing proofs of provable security can be specialized into an efficient procedure for proving probabilistic non interference and develop techniques that overcome the combinatorial explosion of observation sets for high orders. The main idea of their algorithm is to carefully select sets of t or more intermediate variables and to repeatedly apply optimistic sampling on the tuple of expressions that represent the results of these intermediate variables until they do not depend on the secret. The MASKVERIF tool achieves practicality at reasonably high orders. For instance, MASKVERIF is used to automatically and formally verify the probing security of gadgets also at higher order (e.g., ISW scheme at fifth order). A slightly modified version also can verify the security of higher-order implementations in the transition-based model. After the introduction of the strong non interference definition, they adapt MASKVERIF to check the t -SNI property. The adaptation achieves similar coverage as the original tool, i.e., it achieves practicality at reasonably high-orders.

More recently, in [51] a tool called CHECKMASKS has been presented. CHECKMASKS achieves similar functionalities as MASKVERIF, but exploits a more extensive set of transformations for operating on tuples of expressions. This is useful to achieve better verification times on selected examples.

Hardware-based tools. In a context in which also glitches are considered, in [26] the authors propose a formal technique for proving security of implementations in the threshold probing model with glitches. Their method is based on Xiao-Massey lemma, which provides a necessary and sufficient condition for a Boolean function to be statistically independent of a subset of its variables. Informally, the lemma states that a Boolean function f is statistically independent of a set of variables X if and only if the Fourier coefficients of every non-empty subset of X is null. To overcome the computational expensiveness of the computation of Fourier coefficients, they use instead an approximation method. By encoding their approximation in logical form, they can instantiate their approach using SAT-based solvers. Their tool is able to verify implementations of S-Boxes of AES, Keccak and FIDES. However, the cost of the verification is significant.

Later, also MASKVERIF tool has been improved [6] with a unified framework to efficiently and formally verify both software and hardware implementations. In this process, the authors introduce a simple but expressive intermediate language: their representation requires that each instruction is instrumented with leakage expressions that may depend on the expressions that arise in the instruction and on previous computation.

In the context of the hardware-based tools, another important branch has been recently developed, i.e., that based on the exploitation of BDDs as formal verification (see Section 2.1.6). In particular, in [75] the authors propose SILVER tool, based on a symbolic analysis of probability distributions and statistical independence of joint distributions. With Silver, they are able to formally analyze and verify masked circuits in the d -probing model, even in the presence of glitches as physical defaults. Recently, a new BDDs-based tool has been presented in [74], which allows creating secure and efficient masked cryptographic circuits originating from an unprotected design.

3.2 A relation calculus for reasoning about t -probing security

In this work [90], we investigate an alternative formalization which, we argue, is simpler to reason with. In fact, it allowed us to prove some new properties of probing security (see Theorem 3.2.12 and 3.2.19). Our approach is based on the spectral theory of Boolean functions, and represents, to some extent, an extension of the work presented in [26]. However, while the latter only addressed single output Boolean functions, we are the first to formalize the multiple dependencies between outputs

and inputs of a vector function, with the added benefit of being supported by many matrix-based toolboxes.

3.2.1 A relation calculus for shares

Let us consider a hardware implementation of a generic function:

$$f(x_1, \dots, x_n) : \mathbb{F}_2^n \rightarrow \mathbb{F}_2^m$$

where the values x_i are *sensitive* (i.e., they have been computed using a secret). A side-channel attack consists of measuring the power consumption of internal nodes of the circuit (through *probes*) and by searching through a set of guesses of the secret for the one that maximizes the correlation.

To design a mitigation against a side-channel attack, designers split each sensitive value x_i into d values $\alpha_i = \{\alpha_{i,j}\}_{j \in 1 \dots d}$ such that $\sum_j \alpha_{i,j} = x_i$; these d values are called *shares*. In principle, this is done by using $d - 1$ auxiliary random values (aka *masks*) and, unless one obtains all d shares $\alpha_{i,j}$, the correlation of each share with the sensitive value x_i is negligible [73]. The implementation of f must be changed to provide the result as a set of shares much like the original sensitive values. The computation of each output f_i is thus split into a set of d vector functions $\omega_i = \{\omega_{i,j}\}_{j \in 1 \dots d}$ such that

$$f_i(x_1, \dots, x_n) = \sum_j \omega_{i,j}(A_1, \dots, A_n), A_i \subseteq \{\alpha_{i,1}, \dots, \alpha_{i,d}\}$$

where each $\omega_{i,j}$ is called an *output share* of f_i and it must be impossible to reconstruct f_i unless one obtains all d output shares.

In the probing-security attack model, aside from regular output shares ω_i , attackers can observe (through probes) a group of the internal values of the circuit as additional outputs

$$\Pi = \{\pi_1, \dots, \pi_{|\Pi|}\}$$

where each π_i is a function of the input shares. A mitigation against a probing attack ensures that none of the π_i are correlated with the original sensitive values. To design such countermeasures, besides the shares of the original sensitive values, designers use an additional group of inputs $P = \{\rho_1 \dots \rho_{|P|}\}$ which are uniformly random. These values are used to "refresh" the internally computed values of the function to make each π and ω not correlated with the sensitive values.

It is clear that correlation between each ω and π with any α and ρ is critical to determine whether the circuit is probing secure. A possible way to encode this

3.2.1 A relation calculus for shares

information is to have a multi-dimensional matrix* called the *shares' relation matrix*:

Definition 3.2.1 (Shares' relation matrix). Given a boolean function $f : \mathbb{F}_2^{|A|} \rightarrow \mathbb{F}_2^{|\Omega|}$, where A is the set of the function's input shares α_k , Ω is the set of output shares ω_k , we define the *shares' relation matrix* of f as a multidimensional matrix F where each element:

$$F_{i_\pi i_{\omega_p} \dots i_{\omega_1}}^{j_\rho j_{\alpha_{|A|}} \dots j_{\alpha_1}} \in \{0, 1\}, \quad (3.3)$$

is indexed by:

- $j_{\alpha_k} \in \{0, \dots, d\}, k \in \{1, \dots, |A|\}$
- $j_\rho \in \{0, \dots, |P|\}$
- $i_{\omega_p} \in \{0, \dots, d\}, p \in \{1, \dots, |\Omega|\}$
- $i_\pi \in \{0, \dots, |\Pi|\}$

and it is equal to 1 only if there exist a non-zero correlation between j_{α_k} shares of α_k and j_ρ randoms with i_{ω_p} output shares of ω_p and i_π probes, for all k, p .

A formal definition of such type of matrices is presented in Appendix 3.2.5.2 where, in particular, we consider multiple random P_l and probe Π_z groups instead of a single P and Π as above; however, for the rest of this Section, it is only necessary to get an intuitive understanding of it which we will develop in the following paragraphs. A practical way to compute a shares' relation matrix for a function f is deriving it from the Walsh matrix of f . Indeed, correlation matrices are useful to determine whether a set of output shares is *vulnerable*, i.e., correlated with one or more sensitive variables [123, 119, 38]. In particular, for a circuit f , any combination of outputs (encoded with the spectral coordinate ϕ) is correlated with a set of inputs (encoded with the spectral coordinate ψ) if $W_f(\phi, \psi) \neq 0$. It is possible to see a correlation matrix as an incidence matrix which encodes a *dependency relation* between the inputs and outputs of f . Such relation matrices (which are typically built over a Boolean semiring $K = \{(0, 1), \vee, \wedge\}$) are the fundamental building block for the *calculus of relations*[†], an algorithmic device that allows the substitution of computation for a sometimes

*Perhaps the most appropriate name for this type of object would be *tensor* but this name also implies some additional properties that are not used in this work.

[†]A relation matrix element $R_{i,j} \in K$ represents the absence (0) or presence (1) of a relationship iRj between entities encoded through row index i and column index j and the logic composition of relations can be encoded into a linear algebra expression and analyzed with conventional tools. In particular, logical disjunction is represented as matrix sum ($(R + S)_{i,j} = R_{i,j} \vee S_{i,j}$), logical conjunction as the Hadamard product ($(R \circ S)_{i,j} = R_{i,j} \wedge S_{i,j}$), and "Relative product" as conventional matrix multiplication ($(RS)_{i,j} = \exists k R_{i,k} \wedge S_{k,j}$).

3.2.1 A relation calculus for shares

Note that we have labeled columns (rows) with the corresponding combination of inputs (outputs) in binary form. For example, element $\widetilde{W}_f([011], [0011])$ (which is 1) represents an existing dependency between $f_0 \oplus f_1$ and $a_0 \oplus a_1$; note that in this specific case $W_f = \widetilde{W}_f = 1$ but in general $\widetilde{W}_f(i, j)$ is 1 whenever $W_f(i, j)$ is different from zero.

From the original correlation matrix \widetilde{W}_f (Eq. 3.5), it is thus possible to derive the corresponding *shares' relation matrix* F which accounts only for the amount of shares of the output and probes whose combination is correlated with a specified number of shares of the input and of randoms:

$$\begin{array}{cccccccc}
 & 0 & 0 & 0 & 1 & 1 & 1 & 2 & 2 & 2 & \rho \\
 & 0 & 1 & 2 & 0 & 1 & 2 & 0 & 1 & 2 & \alpha \\
 \pi & \omega & & & & & & & & & \\
 0 & 0 & 1 & & & & & & & & \\
 0 & 1 & & & & & & & & & 1 \\
 0 & 2 & & 1 & & & & & & & \\
 1 & 0 & & & & & & & & & 1 \\
 1 & 1 & & & & 1 & & & & & 1 \\
 1 & 2 & & & & & & & & & 1
 \end{array} \tag{3.6}$$

Note that the coordinates of this new relation matrix are computed by the Hamming weights of the spectral coordinates of \widetilde{W}_f split by their type (r the randoms, a the inputs, o the outputs, p the probes). This allows us to index in an alternative way any element $F(i, j)$:

$$F_{i_p i_o}^{j_r j_a}$$

where, as shown in the appendix, i_p, i_o, j_r, j_a are exactly the multi-radix representation of i, j and carry additional information, i.e., the distinction between the related input-random (output-probe) composition.

This work concerns itself with deriving security properties associated with the composition of functions. In the following we consider the function $h(x) = g(f(x))$ as an horizontal composition of g with f while the vector function

$$v(x_1, x_2) = [f(x_1); g(x_2)]$$

as the vertical composition of f and g . We will show that the shares' relation matrix of a function distributes over vertical composition while, concerning horizontal composition, we can assert a weaker rule (see below) which will be still valid for inferring probing security. With regard to the proofs of following theorems, the reader is invited to refer to Appendix 3.2.5.2.

Note that the definition of the shares' relation matrix is different from the Probe Distribution Table (PDT) introduced in [40] because the latter does not account for

the potential compression of information that is obtained by encoding the hamming weights of the spectral coordinates. With respect to [40] we show that it is possible to work with such minimal objects without resorting to encoding explicitly all possible input/output relationships. Note also that the goal of our work is more related to explaining how the composition of primitive gadgets works rather than in determining inner properties for such primitives through their correlation matrices. However, we have provided in other work [89] some deduction on the complexity required for deriving from scratch the above correlation matrices.

Theorem 3.2.3 (Identity). *Given $id : \mathbb{F}_2^n \rightarrow \mathbb{F}_2^n$ the identity function, its shares' relation matrix is \mathbf{I}_{n+1} , where \mathbf{I} is the identity matrix (see Appendix for the proof).*

The horizontal compositionality of the shares' relation matrices is determined by a weaker rule with respect to the conventional correlation matrix (see Theorem 3.2.32); in particular, as long as we look at the constituent parts of an horizontal composition of shares' relation matrices, their product will be always *conservatively more* than the original shares' relation matrix, as stated in the following theorem:

Theorem 3.2.4 (Shares' relation matrices pseudo-horizontal composition). *Given two functions f and g , and F, G, FG the shares' relation matrices of f, g and $g \circ f$ respectively, the following dominance holds:*

$$(FG)_{I_\pi I_\omega}^{J_\rho J_\alpha} \preceq F_{I_\pi I_\omega}^{KL} G_{KL}^{J_\rho J_\alpha}$$

(see Appendix for the proof).

Practically speaking, if the product of two shares' relation matrices does not imply a dependency between variables, this will be absent from the whole shares' relation matrix as well. Vertical composition, however, still holds as the following theorems show:

Theorem 3.2.5 (Shares' relation matrices pseudo-distributivity over tensor product). *Given two functions f and g , and $F, G, F|G$ the shares' relation matrices of f, g and the vertical juxtaposition of g above f respectively, the following holds:*

$$(F|G)_{(I_{\pi_f}|I_{\pi_g})(I_{\omega_f}|I_{\omega_g})}^{(J_{\rho_f}|J_{\rho_g})(J_{\alpha_f}|J_{\alpha_g})} = F_{I_{\pi_f} I_{\omega_f}}^{J_{\rho_f} J_{\alpha_f}} \otimes G_{I_{\pi_g} I_{\omega_g}}^{J_{\rho_g} J_{\alpha_g}}$$

where \otimes is the Kronecker (or tensor) product, see Appendix for the proof.

Corollary 3.2.6 (Tensor product with identity). *From the previous theorems, it follows that the following equalities hold:*

$$\begin{aligned} (\mathbf{I}_{n+1}|G)_{(I_{\pi_{id}}|I_{\pi_g})(I_{\omega_{id}}|I_{\omega_g})}^{(J_{\rho_{id}}|J_{\rho_g})(J_{\alpha_{id}}|J_{\alpha_g})} &= \delta(J_{\rho_{id}} J_{\alpha_{id}}, I_{\pi_{id}} J_{\omega_{id}}) \cdot G_{I_{\pi_g} I_{\omega_g}}^{J_{\rho_g} J_{\alpha_g}} \\ (G|\mathbf{I}_{n+1})_{(I_{\pi_g}|I_{\pi_{id}})(I_{\omega_g}|I_{\omega_{id}})}^{(J_{\rho_g}|J_{\rho_{id}})(J_{\alpha_g}|J_{\alpha_{id}})} &= G_{I_{\pi_g} I_{\omega_g}}^{J_{\rho_g} J_{\alpha_g}} \cdot \delta(J_{\rho_{id}} J_{\alpha_{id}}, I_{\pi_{id}} J_{\omega_{id}}) \end{aligned} \quad (3.7)$$

where δ is the Kronecker's delta.

3.2.2 Application to t -probing security

In this section we revisit and enhance known theorems about t -probing security by showing how they naturally descend from the relation calculus of shares based on shares' relation matrices. We recall that t -probing security centers around the concept of t -non-interfering function. A function f is t -NI if, when given a total of s outputs and internal probes, $s \leq t$ implies a dependency with maximum s input shares. A function f is t -SNI if $s \leq t$ implies a dependency with maximum i input shares, where i is the number of internal probes.

Much has been said about the composition rules of such functions and, unfortunately, their proofs are complex, long or require much expertise in type theoretical or formal validation area [9]; we will show that the relation calculus of shares allows to revisit and extend these proofs with conventional linear algebra tools, broadening the potential audience.

To talk about t -probing security, we've found useful to follow this general pattern: *i*) we explicitly include random refresh values as inputs* and *ii*) we include in the signature of the function also the probes considered. This creates a natural subdivision of the shares' relation matrix for the considered function. Before introducing some general results that can be derived with our formalism, however, we introduce an additional example that shows how one could identify a violation of compositionality in an existing gadget with our formalism.

Example 3.2.7. (Extended from [89]). In this example, we revisit through our formalism a case discovered in [52] that proves that, in general, the composition of t -NI and t -SNI functions is not t -NI.

Figure 3.7 shows the structure of a function $h(a)$ which is a composition of two functions f and g ; the assumptions are that f is t -NI and g is t -SNI. The secrets are split into three shares. f refreshes its input a with two random bits r_f :

$$o_f(a_0, a_1, a_2, r_0, r_1) = [a_0 \oplus r_0 \oplus r_1, a_1 \oplus r_0, a_2 \oplus r_1]$$

and it is assumed to have been probed at location $p_f = a_0 \oplus r_0$. On the other hand, $g(a, b, r_g)$ is the ISW multiplication [73] which consumes 3 random bits r_g for the secret computation. Also in this case, it is assumed a single probe $p_g = a_2 \wedge b_1$. We will show that our method provides a sufficient precision to individuate the vulnerability spotted in [52]. To fit into our formalism however, we must consider the underlying correlation matrices that include explicitly *i*) the random values both f and g consume to refresh the data and *ii*) the probes that are present. The string diagram in Figure

*After all, these are values generated independently by a separated random number generator so it makes sense to include them in the signature of the function itself.

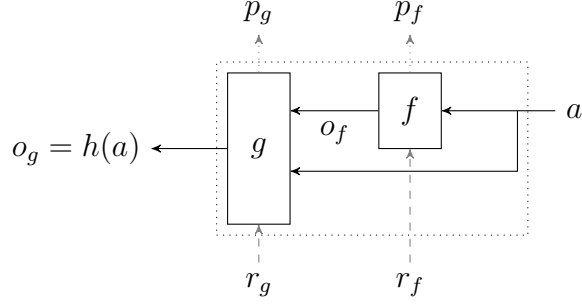


Figure 3.7: The composition pattern of f (t -NI) and g (t -SNI) studied in Example 3.2.7 and derived from [52]. The composed function $h(a)$ is not t -NI as can be easily checked with our formalism.

3.11 describes the composition pattern of correlation matrices as a mapping from the space of the Fourier transform of the input distribution $\mathbb{A} \otimes \mathbb{R}_f \otimes \mathbb{R}_g$ (i.e., the actual inputs plus the random values) to the one of the output distribution $\mathbb{O}_g \otimes \mathbb{P}_g \otimes \mathbb{P}_f$ (i.e., the actual output of g and the probes in both f and g). Still considering the string diagram of Figure 3.11, one can derive one of the equivalent expressions of the correlation matrix of h as

$$W_h = (\mathbf{I}_2 \otimes W_g)(W_q \otimes \mathbf{I}_{2^3} \otimes \mathbf{I}_{2^3})(\mathbf{I}_{2^3} \otimes W_f \otimes \mathbf{I}_{2^3})(\mathbf{I}_{2^3} \otimes \mathbf{I}_{2^2} \otimes W_s)$$

where W_s is the correlation matrix of the duplication function $s = (x) \mapsto (x, x)$ and W_q is the correlation matrix of function $q = (x, y) \mapsto (y, x)$. We are interested in computing the potential dependencies between any combination of output/probes and inputs that are not masked by random values. Thus, computing the shares' relation matrices from all the previous correlation matrices, by Theorems 3.2.4 and 3.2.5, the following holds:

$$H \preceq (\mathbf{I}_2 \otimes G)(Q \otimes \mathbf{I}_4 \otimes \mathbf{I}_4)(\mathbf{I}_4 \otimes F \otimes \mathbf{I}_4)(\mathbf{I}_4 \otimes \mathbf{I}_3 \otimes S) \quad (3.8)$$

where H , F , G , S and Q are the shares' relation matrices computed for functions h , f , g , s and q respectively.

The value of the right-hand side of Eq. 3.8 is shown in Figure 3.8. First, we are interested only in the first 4 columns, as these are the ones that represent relationships between the outputs and the shares of a not masked by any random value. We note that there is a potential dependency in row $[1, 1, 0]$, column $[0, 0, 3]$, exactly the one found in [52], which says that one needs only two probe values to get three shares; h is thus not even 2-NI, showing that t -NI and t -SNI do not compose into a t -NI function. This example shows that the proposed calculus of shares has sufficient precision to

3.2.3 Proving general patterns of compositional security

$$\begin{array}{cccccccccccc}
 & & & & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \dots \rho_g \\
 & & & & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 2 & 2 & \dots \rho_f \\
 & & & & 0 & 1 & 2 & 3 & 0 & 1 & 2 & 3 & 0 & 1 & \dots \alpha \\
 \pi_f & \pi_g & \omega_g & & & & & & & & & & & & \\
 0 & 0 & 0 & 1 & & & & & & & & & & & \\
 0 & 0 & 1 & & & & & & & & & & & & \\
 0 & 0 & 2 & & & & & & & & & & & & \\
 0 & 0 & 3 & 1 & & & & 1 & & & & & & & \\
 0 & 1 & 0 & 1 & 1 & & & & 1 & 1 & 1 & & & 1 & 1 \\
 0 & 1 & 1 & & & & & & & & & & & & \\
 0 & 1 & 2 & & & & & & & & & & & & \\
 0 & 1 & 3 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\
 1 & 0 & 0 & & & & & & & 1 & & & & & \\
 1 & 0 & 1 & & & & & & & & & & & & \\
 1 & 0 & 2 & & & & & & & & & & & & \\
 1 & 0 & 3 & & & & & & & & 1 & 1 & & & \\
 1 & 1 & 0 & & 1 & 1 & 1 & 1 & 1 & 1 & & & & & 1 \\
 1 & 1 & 1 & & & & & & & & & & & & \\
 1 & 1 & 2 & & & & & & & & & & & & \\
 1 & 1 & 3 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1
 \end{array} \tag{3.9}$$

Figure 3.8: The shares’ relation matrix of function h in example 3.2.7 derived from [52] (we use greek letters to indicate the spectral coordinate associated with each function variable, i.e., α is the spectral coordinate associated with variable a and so on). One can see that in row $[1, 1, 0]$, column $[0, 0, 3]$ there is a potential relation between two probes and the three shares of a , meaning that the composition is not even 2-NI.

discover these cases. On one hand, these could be false positives because of the dominance relation in Eq. (3.8); on the other hand, however, this formalism rules out any false negative. We will show that the stronger concept of t -SNI naturally emerges, in our relation calculus, as a fundamental property to ensure compositionality.

3.2.3 Proving general patterns of compositional security

The shares’ relation matrix can be a reasonable way for exploring t -probing security, but there is more. In fact, it is possible to demonstrate that in order to rule out dependencies similar to Example 3.2.7, both f and g must be t -SNI. In this section, we will revisit some known composition patterns (e.g., Theorem 3.2.19 and Corollaries 3.2.16 and 3.2.24 appeared in [8]) and introduce a new one not known in literature (Theorem 3.2.12).

Here, we restate what it means for a function f to be t -NI/ t -SNI in terms of the

shares' relation matrix F :

Definition 3.2.8. f is t -SNI iff, for any set of probes that could be introduced in it, the following predicate is true for any element (i, j) of its shares' relation matrix:

$$|i_\pi| + |i_\omega| \leq t \wedge (\exists a. j_{\alpha_a} > |i_\pi|) \implies \neg F_{i_\pi | \Pi | \dots | i_{\pi_1} i_\omega | \Omega | \dots | i_{\omega_1}}^{0 \dots 0 j_{\alpha_1 | A_1} \dots j_{\alpha_1}}$$

Definition 3.2.9. f is t -NI iff, for any set of probes that could be introduced in it, the following predicate is true for any element (i, j) of its shares' relation matrix:

$$|i_\pi| + |i_\omega| \leq t \wedge (\exists a. j_{\alpha_a} > |i_\pi| + |i_\omega|) \implies \neg F_{i_\pi | \Pi | \dots | i_{\pi_1} i_\omega | \Omega | \dots | i_{\omega_1}}^{0 \dots 0 j_{\alpha_1 | A_1} \dots j_{\alpha_1}}$$

where it is evident that t -NI corresponds to a weaker version of t -SNI.

Example 3.2.10. The Coron's linear-space variant [52] of the ISW multiplication [73] is t -SNI [9] and this can be easily seen through the shares' relation matrix. Let us consider its form for $t = 1$; in this case we have two shares for two inputs a and b , one random value r , two output shares o and six possible internal probes p :

$$\text{SecMult}(a_0, a_1, b_0, b_1, r) = [o_0, o_1, p_0, p_1, p_2, p_3, p_4, p_5]$$

where

$$\begin{bmatrix} o_0 \\ o_1 \\ p_0 \\ p_1 \\ p_2 \\ p_3 \\ p_4 \\ p_5 \end{bmatrix} = \begin{bmatrix} a_0 b_0 + r \\ a_1 b_1 + ((a_0 b_1 + r) + a_1 b_0) \\ a_0 b_0 \\ a_1 b_1 \\ a_0 b_1 \\ a_1 b_0 \\ a_0 b_1 + r \\ (a_0 b_1 + r) + a_1 b_0 \end{bmatrix} \quad (3.10)$$

Part of the corresponding shares' relation matrix is shown in Figure 3.9; it can be seen that for $\pi + \omega \leq 1$, $\rho = 0$ and $\alpha, \beta > 1$ (white areas) we have a null dependency, i.e., the function is 1-SNI.

The simplest composition pattern for which we can derive general rules is $l = g \circ f$. The corresponding map between the Fourier transforms of distributions is shown in Figure 3.10. The question we address is if l (with the associated shares' relation matrix L) is t -SNI/ t -NI according to definition 3.2.8 and 3.2.9, by making assumptions on the probing security of the underlying functions f and g (whose shares' relation matrices are called F and G respectively). Note that, to fit within our formalism, we need to explicitly route the refresh values for g and probed value of f with a function q that

just swaps those values. Note that, since matrix Q is the shares' relation matrix of $q : (x, y) \mapsto (y, x)$ function, it can be shown that the following holds:

$$Q_{i_{\rho_g}, i_{\pi_f}}^{j_{\pi_f}, j_{\rho_g}} = \delta(i_{\pi_f}, j_{\pi_f}) \cdot \delta(i_{\rho_g}, j_{\rho_g})$$

Besides, by Theorem 3.2.4, we know that L is dominated by the product:

$$ABC = (\mathbf{I}_{n_{\pi_f}} \otimes G) \cdot (Q \otimes \mathbf{I}_{n_{\omega_f}}) \cdot (\mathbf{I}_{n_{\rho_g}} \otimes F)$$

where n_{π_f} (n_{ω_f} , n_{ρ_g}) is the number of probes in f (output's shares of f , randoms needed to refresh g) plus 1 (see Theorem 3.2.3).

The following lemma can be proved

Lemma 3.2.11. *The product ABC is such that*

$$(ABC)_{i_{\pi_f} i_{\pi_g} i_{\omega_g}}^{0,0,j_\alpha} = \sum_r G_{i_{\pi_g} i_{\omega_g}}^{0,r} F_{i_{\pi_f},r}^{0,j_\alpha}$$

For a proof see Appendix.

We are now able to derive formally if and when L is t -SNI/ t -NI.

Theorem 3.2.12. *If f is t -SNI and g is t -NI, then $l(x) = g(f(x))$ is t -SNI. Formally, the following three axioms:*

Axiom 3.2.13. $r + |i_{\pi_f}| \leq t \wedge v > |i_{\pi_f}| \implies \neg F_{i_{\pi_f},r}^{0,v}$

Axiom 3.2.14. $|i_{\omega_g}| + |i_{\pi_g}| \leq t \wedge r > |i_{\pi_g}| + |i_{\omega_g}| \implies \neg G_{i_{\pi_g},i_{\omega_g}}^{0,r}$

Axiom 3.2.15. $(|i_{\pi_g}| + |i_{\pi_f}| + |i_{\omega_g}| \leq t) \wedge (j_\alpha > |i_{\pi_g}| + |i_{\pi_f}|)$

entail $(ABC)_{i_{\pi_f} i_{\pi_g} i_{\omega_g}}^{0,0,j_\alpha} = 0$

Proof. Exploiting above Axioms and Lemma 3.2.11 we can derive that:

$$\begin{aligned} (ABC)_{i_{\pi_f} i_{\pi_g} i_{\omega_g}}^{0,0,j_\alpha} &\stackrel{\text{Lem. 3.2.11}}{=} \sum_r G_{i_{\pi_g} i_{\omega_g}}^{0,r} F_{i_{\pi_f},r}^{0,j_\alpha} \\ &\stackrel{\text{A3.2.13,A3.2.14}}{\preceq} \sum_{t - i_{\pi_f} < r \leq i_{\pi_g} + i_{\omega_g}} G_{i_{\pi_g},i_{\omega_g}}^{0,r} \stackrel{\text{A3.2.15}}{\preceq} 0 \end{aligned}$$

□

Corollary 3.2.16. *If f and g are t -SNI functions then also $l(x) = g(f(x))$ is t -SNI.*

Proof. Assuming g is t -SNI, then it is also t -NI and the thesis follows from theorem 3.2.12. □

3.2.3 Proving general patterns of compositional security

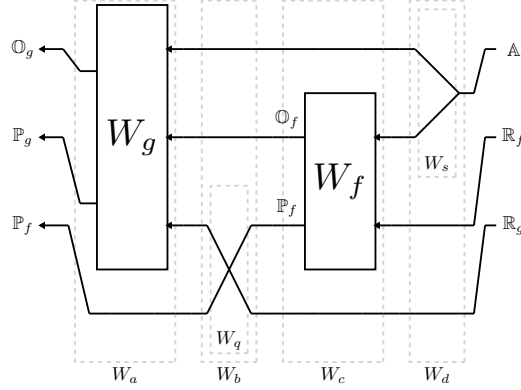


Figure 3.11: Map between Fourier transforms of probability distributions implied by the second composition pattern studied in this work

We already saw an example of another composition pattern studied in the literature, whose circuit diagram is shown in Figure 3.7. The diagram associated to its correlation matrices is the one shown in Figure 3.11. With our formalism, it is possible to identify some general rules to determine if such a composed function is t -NI/ t -SNI (according to definitions 3.2.8 and 3.2.9) by making assumptions on the probing security of the underlying functions f and g . Note that, to reconcile with our model of function, we explicitly split the whole function l into a composition $a \circ b \circ c \circ d$. In particular, d contains the duplication function s that sends a copy of the shared input to both f and g , while b contains q as in the pattern that we previously studied. The shares relation matrix S associated to $s : x \mapsto (x, x)$ function is characterized by the following lemma:

Lemma 3.2.17. *For any $i_{\alpha_1}, i_{\alpha_2}, j_{\alpha}$ indices, the following holds:*

$$|i_{\alpha_1}| + |i_{\alpha_2}| < |j_{\alpha}| \implies S_{i_{\alpha_1}, i_{\alpha_2}}^{j_{\alpha}} = 0$$

For a proof see Appendix.

From the point of view of the shares' relation matrix involved, we know that whole function is dominated by the product (see Theorem 3.2.4):

$$ABCD = (\mathbf{I}_{n_{\pi_f}} \otimes G) \cdot (Q \otimes \mathbf{I}_{n_{\omega_f}} \otimes \mathbf{I}_{n_{\alpha_1}}) \cdot (\mathbf{I}_{n_{\rho_g}} \otimes F \otimes \mathbf{I}_{n_{\alpha_1}}) \cdot (\mathbf{I}_{n_{\rho_g}} \otimes \mathbf{I}_{n_{\rho_f}} \otimes S)$$

where n_{α_1} (n_{ρ_f}) is the number of shares of the first g 's input (randoms needed to refresh f) plus 1.

Lemma 3.2.18. *The complete relation matrix $ABCD$ computed in Figure 3.11 is such that*

$$(ABCD)_{i_{\pi_f} i_{\pi_g} i_{\omega_g}}^{0,0,j_\alpha} = \sum_{v,z} \left(\sum_r G_{i_{\pi_g} i_{\omega_g}}^{0,r,z} F_{i_{\pi_f},r}^{0,v} \right) S_{v,z}^{j_\alpha}$$

For a proof see Appendix.

We are now able to derive formally when L is t -SNI/ t -NI.

Theorem 3.2.19. *If f is t -SNI function and g is t -NI, then $l(x) = g(f(x), x)$ is t -NI. Formally, the following three axioms:*

Axiom 3.2.20. $r + |i_{\pi_f}| \leq t \wedge v > |i_{\pi_f}| \implies \neg F_{i_{\pi_f},r}^{0,v}$

Axiom 3.2.21. $|i_{\omega_g}| + |i_{\pi_g}| \leq t \wedge (r > |i_{\pi_g}| + |i_{\omega_g}| \vee z > |i_{\pi_g}| + |i_{\omega_g}|) \implies \neg G_{i_{\pi_g},i_{\omega_g}}^{0,r,z}$

Axiom 3.2.22. $(|i_{\pi_g}| + |i_{\pi_f}| + |i_{\omega_g}| \leq t) \wedge (|j_\alpha| > |i_{\pi_g}| + |i_{\pi_f}| + |i_{\omega_g}|)$

entail $(ABCD)_{i_{\pi_f} i_{\pi_g} i_{\omega_g}}^{0,0,j_\alpha} = 0$

Proof. Exploiting above axioms and Lemmas 3.2.17 and 3.2.18:

$$(ABCD)_{i_{\pi_f} i_{\pi_g} i_{\omega_g}}^{0,0,j_\alpha} \stackrel{\text{Lem. 3.2.18}}{=} \sum_{v,z} \left(\sum_r G_{i_{\pi_g} i_{\omega_g}}^{0,r,z} F_{i_{\pi_f},r}^{0,v} \right) S_{v,z}^{j_\alpha} \quad (3.12)$$

$$\stackrel{\text{A3.2.20}}{\preceq} \sum_{v \leq i_{\pi_f}, z} \sum_{r > t - i_{\pi_f}} G_{i_{\pi_g} i_{\omega_g}}^{0,r,z} S_{v,z}^{j_\alpha} \quad (3.13)$$

$$\stackrel{\text{A3.2.21, A3.2.22}}{\preceq} \sum_{v \leq i_{\pi_f}, z \leq i_{\pi_g} + i_{\omega_g}} S_{v,z}^{j_\alpha} \stackrel{\text{Lem. 3.2.17, A3.2.22}}{\preceq} 0 \quad (3.14)$$

□

Remark 3.2.23. Note that the case handled in Theorem 3.2.19 concerns f t -SNI and g t -NI; vice versa, Example 3.2.7 concerns the inverted case f t -NI and g t -SNI.

Corollary 3.2.24. *If f and g are t -SNI functions then also $l(x) = g(f(x), x)$ is t -SNI. Formally, the following three axioms:*

Axiom 3.2.25. $r + |i_{\pi_f}| \leq t \wedge v > |i_{\pi_f}| \implies \neg F_{i_{\pi_f},r}^{0,v}$

Axiom 3.2.26. $|i_{\omega_g}| + |i_{\pi_g}| \leq t \wedge (r > |i_{\pi_g}| \vee z > |i_{\pi_g}|) \implies \neg G_{i_{\pi_g},i_{\omega_g}}^{0,r,z}$

Axiom 3.2.27. $(|i_{\pi_g}| + |i_{\pi_f}| + |i_{\omega_g}| \leq t) \wedge (|j_\alpha| > |i_{\pi_g}| + |i_{\pi_f}|)$

entail $(ABCD)_{i_{\pi_f} i_{\pi_g} i_{\omega_g}}^{0,0,j_\alpha} = 0$

Proof. The initial part of the proof is the same of Theorem 3.2.19 up to Equation (3.13); then the different axioms apply:

$$\sum_{v \leq i_{\pi_f}, z} \sum_{r > t - i_{\pi_f}} G_{i_{\pi_g} i_{\omega_g}}^{0,r,z} S_{v,z}^{j_\alpha} \stackrel{\text{A3.2.26, A3.2.27}}{\preceq} \sum_{v \leq i_{\pi_f}, z \leq i_{\pi_g}} S_{v,z}^{j_\alpha} \stackrel{\text{Lem. 3.2.17, A3.2.27}}{\preceq} 0$$

□

3.2.4 Extending the approach to $\mathbb{F}_{2^k}^n$: the AES inversion

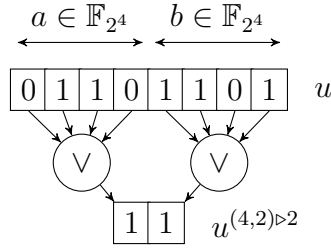


Figure 3.12: Example of reduction operation $u^{(k,n)\triangleright i}$. The new spectral coordinate binary encoding $u^{(4,2)\triangleright 2}$ is the result of OR'ing k -bit wide blocks of the original encoding u

3.2.4 Extending the approach to $\mathbb{F}_{2^k}^n$: the AES inversion

A function that has been widely studied in the probing security framework is the inversion function in AES algorithm; finding a gadget that implements it in a probing secure way, also when it is composed with previous and following gadgets, has been a well-known research's cornerstone [8]. In this section we show how the same result can be obtained with our formalism; before proceeding, we need to extend some of the previous results to the case where shares encode values over k bits, i.e., they belong to $\mathbb{F}_{2^k}^n$.

3.2.4.1 Shares encoded over $\mathbb{F}_{2^k}^n$

Let us thus consider a function $f : \mathbb{F}_{2^k}^n \rightarrow \mathbb{F}_{2^k}^m$; we can extend Eq. 3.4 as follows:

$$\widetilde{W}_f(i, j) = \exists u, v. W_f(u, v) \neq 0 \wedge (u^{(k,n)\triangleright n} = i) \wedge (v^{(k,m)\triangleright m} = j) \quad (3.15)$$

where $u^{(k,n)\triangleright n}$ is a reduction operation over the binary encoding of the spectral coordinate u (see Figure 3.12). It can be shown that the shares' relation matrix for the relation matrices computed as in Eq. 3.15 still complies with Definitions 3.2.8 and 3.2.9 and Theorems 3.2.4 and 3.2.5. In this setting, affine functions have a nice representation that will be useful to extend the application of previous theorems:

Definition 3.2.28. A function $f : \mathbb{F}_{2^k}^n \rightarrow \mathbb{F}_{2^k}^n$ is a (multi-share) *affine function* if:

$$\forall x \in \mathbb{F}_{2^k}^n, \forall i \in \{0, \dots, n-1\} \exists g. f(x)_i = g(x_i)$$

where g is an affine function, x_i is the i -th share of x and $f(x)_i$ is the i -th share of $f(x)$ (see [8]). For conciseness, we will refer to f as an affine function as well.

The relation matrix of an affine function (as well as its shares' relation matrix) is an identity, as the following lemma shows.

Lemma 3.2.29. *Let $f : \mathbb{F}_{2^k}^n \rightarrow \mathbb{F}_{2^k}^n$ be an affine function; then $\widetilde{W}_f = I_{2^n}$.*

Proof. The affine function f can be seen as the parallel application of n functions g_i such that $f(x)_i = g_i(x_i)$ with $0 \leq i \leq n - 1$. This implies that

$$W_f = \bigotimes_{i=0}^{n-1} W_{g_i}$$

Since each g_i is an affine (and balanced) function, then $\widetilde{W}_{g_i} = I_2$ and

$$\widetilde{W}_f = \bigotimes_{i=0}^{n-1} I_2 = I_{2^n}$$

□

3.2.4.2 Proof of strong non-interference

Let us consider the AES inversion in \mathbb{F}_{2^8} shown in Figure 3.13b and presented originally in [99]. First of all, we note that there is a recurring pattern, i.e., the circuit in Figure 3.13a. The block is composed of a mask refresh **Refresh** (t -SNI), the ISW multiplication **SecMult** (t -SNI), and \cdot^x , an affine power function parameterized over the exponent x (which is a multiple of two). It is possible to demonstrate that m_x is t -SNI following the same line of reasoning of Theorem 3.2.24 because, by Lemma 3.2.29, the relation matrix of the power function can be interpreted as an identity, thus the same case as the one shown in Figure 3.11 applies. Considering the overall algorithm in Figure 3.13b, we observe that this is t -SNI if $b \circ m_2$ is t -SNI (by Theorem 3.2.24). By Corollary 3.2.16, $b \circ m_2$ is t -SNI if b is t -SNI and the latter is true by Theorem 3.2.24 and by Lemma 3.2.29.

3.2.5 Appendix

Related to this work, there are the three appendices added to this article.

3.2.5.1 A: Properties of the Walsh transform

This section recaps the important properties of the Walsh transform of a vectorial Boolean function and introduces the concept of tensor product for the resulting matrices [38].

Definition 3.2.30 (Walsh transform of a vectorial function). Given a vectorial Boolean function $f : \mathbb{F}_2^n \rightarrow \mathbb{F}_2^m$, we define its Walsh transform as a $2^m \times 2^n$ matrix \widehat{f} whose elements are:

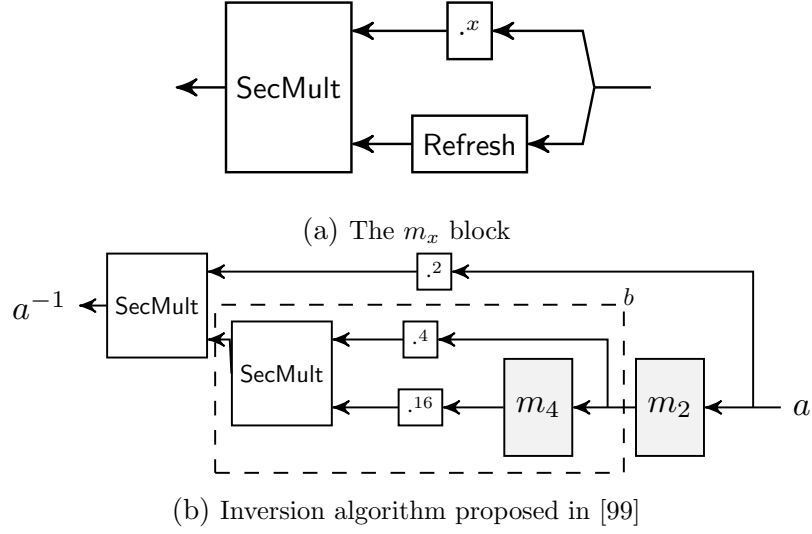


Figure 3.13: An example application of the proposed formalism to functions over \mathbb{F}_2^n . Blocks m_4 and m_2 in (b) are structured as in (a). Note that we have slightly modified the algorithm presented in [99] by moving two power computation blocks across duplication points. Semantically, it is always the same circuit but it is easier to see how previously introduced patterns can still be used to show that it is t -SNI.

$$\hat{f}_{\omega, \alpha} = \sum_{x \in \mathbb{F}_2^n} (-1)^{\omega^\top f(x) \oplus \alpha^\top x} \quad (3.16)$$

$\omega \in \mathbb{F}_2^m, \alpha \in \mathbb{F}_2^n$ being the binary encoding of the row and column indices, called *spectral coordinates* (or sometimes *masks*).

These matrices encode the correlation information between input variables' XOR-combinations and the corresponding output ones. For this reason they sometimes appear in the literature, scaled by a coefficient 2^{-n} , as *correlation matrices* [53]:

$$W_f = 2^{-n} \hat{f}$$

For correlation matrices, the following theorem holds:

Theorem 3.2.31 (Correlation matrix as a map of probability distributions). *Given a function $f : \mathbb{F}_2^n \rightarrow \mathbb{F}_2^m$ and a probability distribution $p_X : \mathbb{F}_2^n \rightarrow \mathbb{R}$ for its input variable, the following relation holds:*

$$W_f F_{p_X} = F_{p_Y}$$

where p_Y is the distribution of the output values while F_g is the Fourier transform of

any pseudo-Boolean function $g : \mathbb{F}_2^n \rightarrow \mathbb{R}$ and defined as the following:

$$F_g(\gamma) = \sum_{x \in \mathbb{F}_2^n} g(x)(-1)^{\gamma^\top x}$$

For a proof see [53, 55].

Interpreting the Fourier transform of a probability distribution of a variable in \mathbb{F}_2^n as a vector in a subset* \mathbb{P}_n of \mathbb{R}^{2^n} , we find that the correlation matrix W_f of a function $f : \mathbb{F}_2^n \rightarrow \mathbb{F}_2^m$ is just a linear map $\mathbb{P}_n \rightarrow \mathbb{P}_m$. These maps are endowed with composition:

Theorem 3.2.32 (Composition of correlation matrices). *Given two functions $f : \mathbb{F}_2^n \rightarrow \mathbb{F}_2^m$ and $g : \mathbb{F}_2^m \rightarrow \mathbb{F}_2^q$, the following holds:*

$$W_{g \circ f} = W_g W_f$$

Moreover, if f is a bijection, $W_{f^{-1}} = W_f^{-1}$. For a proof see [38, 96].

Given two independent variables $x_f \in \mathbb{F}_2^{n_f}$ and $x_g \in \mathbb{F}_2^{n_g}$, one can form the probability distribution of the vector $[x_f, x_g]$ with the product of distributions. From the point of view of its Fourier transform, this is a mapping $\mathbb{P}_{n_f} \times \mathbb{P}_{n_g} \rightarrow \mathbb{P}_{n_f+n_g}$. The following theorem holds:

Theorem 3.2.33 (Tensor product of correlation matrices). *Given two functions $f : \mathbb{F}_2^{n_f} \rightarrow \mathbb{F}_2^{m_f}$ and $g : \mathbb{F}_2^{n_g} \rightarrow \mathbb{F}_2^{m_g}$, the correlation matrix of the function $h([x_f, x_g]) = [f(x_f), g(x_g)]$ is $W_h = W_g \otimes W_f$ where the symbol \otimes is the Kronecker product (or tensor product) of matrices (proof in the appendix). It is customary to say that W_h is a mapping from the space $\mathbb{P}_{n_f} \otimes \mathbb{P}_{n_g}$ to the space $\mathbb{P}_{m_f} \otimes \mathbb{P}_{m_g}$.*

Theorem 3.2.33 is informally proven in [53] and it is applied in several works, as in [24]; taking this into account, in this appendix we try to give to it a formal proof. For this scope, we define the quotient and remainder operators as follows, to remind ourselves of the structure of the indices:

$$i_{\uparrow p} = \left\lfloor \frac{i}{p} \right\rfloor, i_{\downarrow p} = i - pi_{\uparrow p}$$

When $p = 2^n$ and i is a number that can be encoded over $k > n$ bit, $i_{\uparrow p}$ corresponds to the value encoded by the upper $k - n$ bits, while $i_{\downarrow p}$ corresponds to the value encoded by the lower n bits.

*Note that these are not sub-spaces as the set \mathbb{P}_n is not closed under addition.

Definition 3.2.34 (Kronecker product of matrices). The tensor product of two matrices. X (of $n \times m$ elements) and Y (of $p \times q$ elements) can be defined as:

$$(X \otimes Y)_{i,j} = X_{i_{\uparrow p}, j_{\uparrow q}} Y_{i_{\downarrow p}, j_{\downarrow q}}$$

Theorem 3.2.33. Note that ω (α) can be treated as a decimal number or as the corresponding (vector) binary encoding; moreover, the encoding of ω (α) can in turn be decomposed into two parts $[\omega_g, \omega_f]$ ($[\alpha_g, \alpha_f]$) of m_g (n_g) and m_f (n_f) bits respectively. We start by rewriting the definition of \hat{h} :

$$\begin{aligned} \hat{h}(\omega, \alpha) &= \frac{1}{2^{n_f+n_g}} \sum_{x \in \mathbb{F}_2^{n_f+n_g}} (-1)^{\omega^\top h(x) + \alpha^\top x} \\ &= \frac{1}{2^{n_f+n_g}} \sum_{[x_f, x_g] \in \mathbb{F}_2^{n_f+n_g}} (-1)^{\omega_f^\top f(x_f) + \omega_g^\top g(x_g) + \alpha_f^\top x_f + \alpha_g^\top x_g} \\ &= \hat{g}(\omega_g, \alpha_g) \hat{f}(\omega_f, \alpha_f) \\ &= \hat{g}\left(\left\lfloor \frac{\omega}{2^{m_g}} \right\rfloor, \left\lfloor \frac{\alpha}{2^{n_g}} \right\rfloor\right) \hat{f}\left(\omega - 2^{m_g} \left\lfloor \frac{\omega}{2^{m_g}} \right\rfloor, \alpha - 2^{n_g} \left\lfloor \frac{\alpha}{2^{n_g}} \right\rfloor\right) \\ &= \hat{g}(\omega_{\uparrow 2^{m_g}}, \alpha_{\uparrow 2^{n_g}}) \hat{f}(\omega_{\downarrow 2^{m_g}}, \alpha_{\downarrow 2^{n_g}}) \end{aligned}$$

and conclude (using Definition 3.2.34) that the last equation represents the generic element in position (ω, α) of the Kronecker product $\hat{g} \otimes \hat{f}$. \square \square

Reasonings on the effect of composing correlation matrices can be intuitively allowed through diagrams. Each correlation matrix is drawn as a box (except for identities which are drawn as simple wires), composition is the horizontal juxtaposition while tensor product is the vertical one (see Figure 3.14 for an example). We note that there is a remarkable correspondence between a diagram and the underlying circuit diagram to the point that we could talk about "the" diagram of the circuit. Moreover, there always exist two mappings $B_{a,b} : \mathbb{P}_a \otimes \mathbb{P}_b \rightarrow \mathbb{P}_b \otimes \mathbb{P}_a$ and $B_{b,a}$ such that $B_{a,b} B_{b,a} = I$. $B_{a,b}$ is exactly the Walsh transform of a function that permutes variables a and b (this is typically drawn with crossing wires, such as block Q in Figure 3.11).

3.2.5.2 B: Formal definition of shares' relation matrix

To formally define the shares' relation matrix, let us introduce, with a slight abuse of notation, the mixed-radix representation $mr_\rho(n)$ of a number n over the vector of parts $\rho = [\rho_N, \dots, \rho_1]$ as a vector $b = [b_{N+1}, \dots, b_1]$ such that:

$$n = \sum_{i=1}^{N+1} b_i \prod_{j=1}^{i-1} (\rho_j + 1) \text{ where } 0 \leq b_i < \rho_i$$

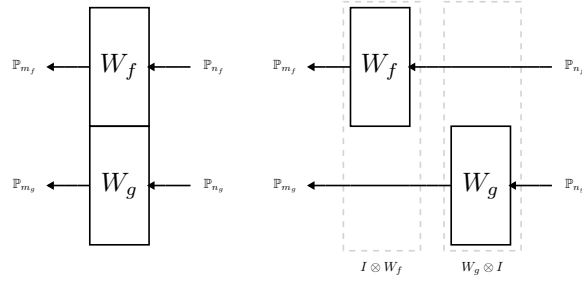


Figure 3.14: Example of compositional equality derived through a string diagram. The diagram on the left corresponds to the product $(W_g \otimes W_f)$ while the one on the right corresponds to $(1 \otimes W_f)(W_g \otimes 1)$ (each factor is highlighted with a dotted box). The fact that the second can be derived simply by moving boxes without crossing wires implies (because we are in monoidal category) that the underlying formulas are equivalent, i.e., $(W_g \otimes W_f) = (1 \otimes W_f)(W_g \otimes 1)$

The shares' relation matrix notation $F_{i_p i_o}^{j_r j_a}$ (see Example 3.2.2) refers to an element (i, j) of F where $([i_p, i_o], [j_r, j_a])$ are just the *mixed-radix representation* of (i, j) over the vector of parts $([f_p, f_o], [f_r, f_a])$ where f_a is the number of shares of the input of function f , f_r is the number of refresh values, f_o is the number of shares for function f 's outputs and f_p is the number of probes:

$$i = i_p \cdot (f_o + 1) + i_o \quad (3.17)$$

$$j = j_r \cdot (f_a + 1) + j_a \quad (3.18)$$

Example 3.2.35. For example, the fifth row with index $i = 4$ of the matrix in Equation (3.6) has a mixed radix representation $[i_p, i_o] = [1, 1]$ over the vector of parts $[f_p, f_o] = [1, 2]$ because:

$$i = 1 \cdot (2 + 1) + 1$$

Same reasoning goes for column index 3 which corresponds to the representation $[1, 0]$ over the vector of parts $[f_r, f_a] = [2, 2]$, i.e.:

$$j = 1 \cdot (2 + 1) + 0$$

Thus we have that $F_{1,1}^{1,0}$ is the element $(4, 3)$ of matrix F and its indexes carries the fact that it corresponds the correlation of both probes and outputs $([i_p, i_o] = [1, 1])$ with one of the random values $([j_r, j_a] = [1, 0])$.

To produce precise proofs of the theorems introduced in this appendix we need to slightly modify the notation above to show the actual vector of parts over which the

multi-radix representation is computed. For the sake of generality, we will consider a generic vector of parts $\underline{\Omega}$ (\underline{A}) for the matrix rows (columns). Given a function f , we thus talk about a shares' relation matrix in the following form:

$$H_{\underline{\Omega}, \underline{A}}[\widetilde{W}_f]$$

With the understanding that each element (i, j) of $H_{\underline{\Omega}, \underline{A}}[\widetilde{W}_f]$ is such that

$$F_{mr_{\underline{\Omega}}(i)}^{mr_{\underline{A}}(j)} = H_{\underline{\Omega}, \underline{A}}[\widetilde{W}_f]_{i,j} \quad (3.19)$$

We will also use $|i_\xi|$ to indicate $\sum_k i_{\xi_k}$, i.e., the sum of the mixed-radix components of index i associated with the vector of parts Ξ . With this notation, $|i_\pi|$ practically means the number of probes associated with a specific value of index i .

Example 3.2.36. Considering Example 3.2.35, we have the following notational equivalence

$$F_{1,1}^{1,0} = H_{[1,2],[2,2]}[\widetilde{W}_f]_{4,3}$$

The shares' relation matrix can be seen as the encoding of a predicate over the original relation matrix; this fact will be used to prove the remaining theorems in this appendix and corresponds to an equivalent definition of the matrix itself, as the following theorem shows.

Alternative Definition 1 (Shares' relation matrix). The shares' relation matrix computed from a relation matrix $\widetilde{Q} \in K^{2^\Omega \times 2^A}$ is a matrix $H_{\underline{\Omega}, \underline{A}}[\widetilde{Q}] \in K^{\pi_\Omega \times \pi_A}$ where

$$\pi_\Omega = \prod_i (\omega_i + 1), \pi_A = \prod_i (\alpha_i + 1) \quad (3.20)$$

and such that each element $H[\widetilde{Q}]_{i,j}$ is 1 iff:

$$\exists \widetilde{Q}_{r,s}. \widetilde{Q}_{r,s} \wedge wt_{\underline{\Omega}}(r) = mr_{\underline{\Omega}}(i) \wedge wt_{\underline{A}}(s) = mr_{\underline{A}}(j) \quad (3.21)$$

where $wt_V : \mathbb{N} \rightarrow \mathbb{N}^v$ is the hamming weight of each of the v binary parts according to the vector of parts V .

Remark 3.2.37 (Compact definition). We will sometimes use the notation $r \sim_{\underline{\Omega}} i$ to indicate the predicate $wt_{\underline{\Omega}}(r) = mr_{\underline{\Omega}}(i)$ (read r is a valid encoding for i). By construction the following predicate is thus true:

$$H_{\underline{\Omega}, \underline{A}}[\widetilde{Q}]_{i,j} \iff \exists r, s. \widetilde{Q}_{r,s} \wedge (r \sim_{\underline{\Omega}} i) \wedge (s \sim_{\underline{A}} j) \quad (3.22)$$

In the following paragraphs we provide a few relevant theorems and proofs valid for the shares' relation matrix.

Theorem 3.2.3. With the new notation, Theorem 3.2.3 can be rewritten as follows:

Given a vector of parts composed of a single part $\underline{\Pi} = [\pi]$ we have that $H_{\underline{\Pi}, \underline{\Pi}}[I_{2^\pi}] = I_{\pi+1}$.

To prove it, we elaborate the predicate in Eq. (3.22):

$$\begin{aligned} H_{\underline{\Pi}, \underline{\Pi}}[I_{2^\pi}]_{i,j} &\iff \exists r, s. I_{r,s} \wedge (r \sim_{\underline{\Pi}} i) \wedge (s \sim_{\underline{\Pi}} j) \\ \vdash H_{\underline{\Pi}, \underline{\Pi}}[I_{2^\pi}]_{i,j} &\iff \exists r, s. (r == s) \wedge (r \sim_{\underline{\Pi}} i) \wedge (s \sim_{\underline{\Pi}} j) \\ \vdash H_{\underline{\Pi}, \underline{\Pi}}[I_{2^\pi}]_{i,j} &\iff (i == j) \end{aligned}$$

and note that the implied condition means exactly that it must be an identity matrix. \square

Theorem 3.2.4. With the new notation, Theorem 3.2.4 can be rewritten as follows:

Given two correlation matrices $X \in K^{2^\Omega \times 2^Z}$ and $Y \in K^{2^Z \times 2^A}$, the following dominance holds between the shares' relation matrices:

$$H_{\underline{\Omega}, \underline{A}}[\widetilde{XY}] \preceq H_{\underline{\Omega}, \underline{Z}}[\widetilde{X}] H_{\underline{Z}, \underline{A}}[\widetilde{Y}]$$

for any choice of compatible* parts $\underline{\Omega}$, \underline{Z} and \underline{A} .

To prove it, note that

$$H_{\underline{\Omega}, \underline{A}}[\widetilde{XY}] \preceq H_{\underline{\Omega}, \underline{Z}}[\widetilde{X}] H_{\underline{Z}, \underline{A}}[\widetilde{Y}]$$

represents the following implication (assuming $\sum_{z_i \in \underline{Z}} z_i = \underline{Z}$):

$$H_{\underline{\Omega}, \underline{A}}[\widetilde{XY}]_{i,j} \implies \exists \zeta. H_{\underline{\Omega}, \underline{Z}}[\widetilde{X}]_{i,\zeta} \wedge H_{\underline{Z}, \underline{A}}[\widetilde{Y}]_{\zeta,j}$$

which follows directly from the definition in Eq (3.22) (note that once we have t , ζ exists since we assume \underline{Z} is compatible):

$$\begin{aligned} H_{\underline{\Omega}, \underline{A}}[\widetilde{XY}]_{i,j} &\iff \exists r, s. (XY_{r,s} \neq 0) \wedge (r \sim_{\underline{\Omega}} i) \wedge (s \sim_{\underline{A}} j) \\ \vdash H_{\underline{\Omega}, \underline{A}}[\widetilde{XY}]_{i,j} &\implies \exists r, s, t. (r \sim_{\underline{\Omega}} i) \wedge (s \sim_{\underline{A}} j) \wedge (X_{r,t} \neq 0) \wedge (Y_{t,s} \neq 0) \\ \vdash H_{\underline{\Omega}, \underline{A}}[\widetilde{XY}]_{i,j} &\implies \exists r, s, t, \zeta. (r \sim_{\underline{\Omega}} i) \wedge (s \sim_{\underline{A}} j) \wedge (\zeta \sim_{\underline{Z}} t) \wedge \widetilde{X}_{r,t} \wedge \widetilde{Y}_{t,s} \\ \vdash H_{\underline{\Omega}, \underline{A}}[\widetilde{XY}]_{i,j} &\implies \exists \zeta. H_{\underline{\Omega}, \underline{Z}}[\widetilde{X}]_{i,\zeta} \wedge H_{\underline{Z}, \underline{A}}[\widetilde{Y}]_{\zeta,j} \end{aligned}$$

where, in the second step, we applied the following axiom:

$$XY_{r,s} \neq 0 \implies \exists t. X_{r,t} \neq 0 \wedge Y_{t,s} \neq 0$$

Note that the converse (\Leftarrow) does not hold since matrix multiplication between Walsh matrices is done over rational numbers which might have different signs so it may cancel out. \square

*By *compatible*, we mean that the set of parts sums up to the specified size, e.g., $\sum_{\omega_i \in \underline{\Omega}} \omega_i = \underline{\Omega}$.

3.2.5 Appendix

Theorem 3.2.5. With the new notation, Theorem 3.2.5 can be rewritten as follows:

Given two correlation matrices $X \in K^{2^\Phi \times 2^\Psi}$ and $Y \in K^{2^\Omega \times 2^A}$, the following holds:

$$H_{\underline{\Phi}, \underline{\Psi}}^{\underline{\Omega}, \underline{A}}[\widetilde{X \otimes Y}] = H_{\underline{\Phi}, \underline{\Psi}}[\widetilde{X}] \otimes H_{\underline{\Omega}, \underline{A}}[\widetilde{Y}]$$

where $\frac{\underline{\Omega}}{\underline{\Phi}} = \underline{\Phi} \parallel \underline{\Omega} = \{\phi_{N_\Phi}, \dots, \phi_1, \omega_{N_\Omega}, \dots, \omega_1\}$, i.e., the concatenation of parts $\underline{\Phi}$ and $\underline{\Omega}$

Before proceeding with the proof, we need to introduce the following lemma:

Lemma 3.2.38 (Equivalences over concatenations of vector of parts). *When dealing with a concatenation of parts $\frac{\underline{\Omega}}{\underline{\Phi}}$, both the extended Hamming weight and the multiradix representation comply with the following equivalences:*

$$\begin{aligned} wt_{\frac{\underline{\Omega}}{\underline{\Phi}}}(r) &= wt_{\underline{\Phi}}(r_{\uparrow\pi_{\underline{\Omega}}}) \parallel wt_{\underline{\Omega}}(r_{\downarrow\pi_{\underline{\Omega}}}) \\ mr_{\frac{\underline{\Omega}}{\underline{\Phi}}}(i) &= mr_{\underline{\Phi}}(i_{\uparrow\pi_{\underline{\Omega}}}) \parallel mr_{\underline{\Omega}}(i_{\downarrow\pi_{\underline{\Omega}}}) \end{aligned}$$

Where \parallel is the vector concatenation while $\pi_{\underline{\Omega}} = \prod_i (\omega_i + 1)$. This means that we can split $r \sim_{\frac{\underline{\Omega}}{\underline{\Phi}}} i$ in the conjunction of two sub conditions:

$$r \sim_{\frac{\underline{\Omega}}{\underline{\Phi}}} i \iff (r_{\uparrow\pi_{\underline{\Omega}}} \sim_{\underline{\Phi}} i_{\uparrow\pi_{\underline{\Omega}}}) \wedge (r_{\downarrow\pi_{\underline{\Omega}}} \sim_{\underline{\Omega}} i_{\downarrow\pi_{\underline{\Omega}}})$$

Theorem 3.2.5 is easily proved by expanding $H_{\underline{\Phi}, \underline{\Psi}}^{\underline{\Omega}, \underline{A}}[\widetilde{X \otimes Y}]$ through the definition in Eq (3.22), and apply successively Lemma 3.2.38 and Definition 3.2.34:

$$\begin{aligned} H_{\underline{\Phi}, \underline{\Psi}}^{\underline{\Omega}, \underline{A}}[\widetilde{X \otimes Y}]_{i,j} &= \exists r, s. (X \otimes Y)_{r,s} \neq 0 \wedge (r \sim_{\frac{\underline{\Omega}}{\underline{\Phi}}} i) \wedge (s \sim_{\frac{\underline{A}}{\underline{\Psi}}} j) \\ &= \exists r, s. X_{r_{\uparrow\pi_{\underline{\Omega}}}, s_{\uparrow\pi_{\underline{A}}}} \neq 0 \wedge Y_{r_{\downarrow\pi_{\underline{\Omega}}}, s_{\downarrow\pi_{\underline{A}}}} \neq 0 \wedge (r \sim_{\frac{\underline{\Omega}}{\underline{\Phi}}} i) \wedge (s \sim_{\frac{\underline{A}}{\underline{\Psi}}} j) \\ &= \exists r, s. X_{r_{\uparrow\pi_{\underline{\Omega}}}, s_{\uparrow\pi_{\underline{A}}}} \neq 0 \wedge Y_{r_{\downarrow\pi_{\underline{\Omega}}}, s_{\downarrow\pi_{\underline{A}}}} \neq 0 \wedge (r_{\uparrow\pi_{\underline{\Omega}}} \sim_{\underline{\Phi}} i_{\uparrow\pi_{\underline{\Omega}}}) \\ &\quad \wedge (r_{\downarrow\pi_{\underline{\Omega}}} \sim_{\underline{\Omega}} i_{\downarrow\pi_{\underline{\Omega}}}) \wedge (s_{\uparrow\pi_{\underline{A}}} \sim_{\underline{\Psi}} j_{\uparrow\pi_{\underline{A}}}) \wedge (s_{\downarrow\pi_{\underline{A}}} \sim_{\underline{A}} j_{\downarrow\pi_{\underline{A}}}) \\ &= H_{\underline{\Phi}, \underline{\Psi}}[\widetilde{X}]_{i_{\uparrow\pi_{\underline{\Omega}}}, j_{\uparrow\pi_{\underline{A}}}} \wedge H_{\underline{\Omega}, \underline{A}}[\widetilde{Y}]_{i_{\downarrow\pi_{\underline{\Omega}}}, j_{\downarrow\pi_{\underline{A}}}} \\ &= (H_{\underline{\Phi}, \underline{\Psi}}[\widetilde{X}] \otimes H_{\underline{\Omega}, \underline{A}}[\widetilde{Y}])_{i,j} \end{aligned}$$

□

Corollary 3.2.6. With the new notation, Corollary 3.2.6 can be rewritten as follows:

$$\begin{aligned} H_{\frac{\underline{\Omega}}{[x]}, \frac{\underline{A}}{[x]}}^{\underline{\Omega}, \underline{A}}[I_{2^x} \otimes \widetilde{W}_f]_{i,j} &= \delta(i_x, j_x) \cdot F_{i_{\omega_{|\underline{\Omega}|}} \dots i_{\omega_1}}^{j_{\alpha_{|\underline{A}|}} \dots j_{\alpha_1}} \\ H_{\frac{[x]}{\underline{\Omega}}, \frac{[x]}{\underline{A}}}^{\underline{\Omega}, \underline{A}}[\widetilde{W}_f \otimes I_{2^x}]_{i,j} &= F_{i_{\omega_{|\underline{\Omega}|}} \dots i_{\omega_1}}^{j_{\alpha_{|\underline{A}|}} \dots j_{\alpha_1}} \cdot \delta(i_x, j_x) \end{aligned}$$

For the first equality:

$$\begin{aligned} H_{\frac{\Omega}{[x]}, \frac{A}{[x]}}[I_{2^x} \otimes \widetilde{W}_f]_{i,j} &\iff [I_{x+1} \otimes H_{\Omega, A}(\widetilde{W}_f)]_{i,j} \\ &\iff \delta(i_x, j_x) \cdot F_{i_{\omega_1} \dots i_{\omega_1}}^{j_{\alpha_1} \dots j_{\alpha_1}} \end{aligned}$$

The second equality can be deduced in the same way. \square

3.2.5.3 C: Relevant theorems and proofs - Section 3.2.2

Lemma 3.2.11. The complete relation matrix ABC computed in Figure 3.10 is such that

$$\begin{aligned} (ABC)_{i_{\pi_f} i_{\pi_g} i_{\omega_g}}^{0,0,j_\alpha} &= \sum_{p,q,r} AB_{i_{\pi_f} i_{\pi_g} i_{\omega_g}}^{p,q,r} C_{p,q,r}^{0,0,j_\alpha} \\ &= \sum_{p,q,r} \sum_{l,m,n} A_{i_{\pi_f} i_{\pi_g} i_{\omega_g}}^{l,m,n} B_{l,m,n}^{p,q,r} C_{p,q,r}^{0,0,j_\alpha} \stackrel{\text{Cor. 3.2.6}}{=} \sum_{p,q,r} A_{i_{\pi_f} i_{\pi_g} i_{\omega_g}}^{q,p,r} C_{p,q,r}^{0,0,j_\alpha} \\ &\stackrel{\text{Cor. 3.2.6}}{=} \sum_{p,r} G_{i_{\pi_g} i_{\omega_g}}^{p,r} C_{p,i_{\pi_f},r}^{0,0,j_\alpha} \stackrel{\text{Cor. 3.2.6}}{=} \sum_r G_{i_{\pi_g} i_{\omega_g}}^{0,r} F_{i_{\pi_f},r}^{0,j_\alpha} \end{aligned}$$

\square

Lemma 3.2.17. S is the shares' relation matrix computed from the relation matrix \widetilde{W}_s of the duplication function $s = x \mapsto (x, x)$. It can be shown that elements of this relation matrix are such that:

$$\widetilde{W}_s(l, m) \iff l_{\uparrow 2^n} \oplus l_{\downarrow 2^n} = m$$

To prove Lemma 3.2.17, we proceed with a reduction ad absurdum, i.e., we show that

$$\exists i_{\alpha_1}, i_{\alpha_2}, j_\alpha \cdot (i_{\alpha_1} + i_{\alpha_2} < j_\alpha) \wedge S_{i_{\alpha_1}, i_{\alpha_2}}^{j_\alpha} = 1$$

is a contradiction. To derive it we expand in it the definition of shares' relation matrix S :

$$\begin{aligned} &\exists i_{\alpha_1}, i_{\alpha_2}, j_\alpha \cdot (i_{\alpha_1} + i_{\alpha_2} < j_\alpha) \wedge S_{i_{\alpha_1}, i_{\alpha_2}}^{j_\alpha} = 1 \\ \vdash &\exists l, m. \widetilde{W}_s(l, m) \wedge wt_{\alpha_1}(l) = \binom{i_{\alpha_1}}{i_{\alpha_2}} \wedge wt_{\alpha_2}(m) = j_\alpha \wedge (i_{\alpha_1} + i_{\alpha_2} < j_\alpha) \\ \vdash &\exists l, m. (l_{\uparrow 2^n} \oplus l_{\downarrow 2^n} = m) \wedge wt_{\alpha_1}(l_{\uparrow 2^n}) = i_{\alpha_1} \wedge wt_{\alpha_2}(l_{\downarrow 2^n}) = i_{\alpha_2} \wedge \\ &\quad \wedge wt_{\alpha_2}(m) = j_\alpha \wedge (i_{\alpha_1} + i_{\alpha_2} < j_\alpha) \\ \vdash &\exists l. wt_{\alpha_1}(l_{\uparrow 2^n}) = i_{\alpha_1} \wedge wt_{\alpha_2}(l_{\downarrow 2^n}) = i_{\alpha_2} \wedge wt_{\alpha_2}(l_{\uparrow 2^n} \oplus l_{\downarrow 2^n}) = j_\alpha \wedge \\ &\quad \wedge (i_{\alpha_1} + i_{\alpha_2} < j_\alpha) \\ \vdash &wt_{\alpha_1}(l_{\uparrow 2^n}) + wt_{\alpha_2}(l_{\downarrow 2^n}) < wt_{\alpha_2}(l_{\uparrow 2^n} \oplus l_{\downarrow 2^n}) \end{aligned}$$

3.3. On the Spectral Features of Robust Probing Security

the latter judgment is absurd because for all binary vectors a, b holds that $wt(a) \oplus wt(b) \geq wt(a \oplus b)$. \square

Lemma 3.2.18. The complete relation matrix $ABCD$ computed in Figure 3.11 is such that

$$\begin{aligned} (ABCD)_{i_{\pi_f} i_{\pi_g} i_{\omega_g}}^{0,0,j_\alpha} &= \sum_{t,u,v,z} ABC_{i_{\pi_f} i_{\pi_g} i_{\omega_g}}^{t,u,v,z} D_{t,u,v,z}^{0,0,j_\alpha} \\ &\stackrel{\text{Lem. 3.2.39}}{=} \sum_{t,u,v,z} \sum_r G_{i_{\pi_g} i_{\omega_g}}^{t,r,z} F_{i_{\pi_f},r}^{u,v} D_{t,u,v,z}^{0,0,j_\alpha} \stackrel{\text{Cor. 3.2.6}}{=} \sum_{v,z} \left(\sum_r G_{i_{\pi_g} i_{\omega_g}}^{0,r,z} F_{i_{\pi_f},r}^{0,v} \right) S_{v,z}^{j_\alpha} \end{aligned}$$

where we used the following:

Lemma 3.2.39.

$$\begin{aligned} ABC_{i_{\pi_f} i_{\pi_g} i_{\omega_g}}^{t,u,v,z} &= \sum_{p,q,r,s} AB_{i_{\pi_f} i_{\pi_g} i_{\omega_g}}^{p,q,r,s} C_{p,q,r,s}^{t,u,v,z} \\ &= \sum_{p,q,r,s} \sum_{l,m,n,o} A_{i_{\pi_f} i_{\pi_g} i_{\omega_g}}^{l,m,n,o} B_{l,m,n,o}^{p,q,r,s} C_{p,q,r,s}^{t,u,v,z} \stackrel{\text{Cor. 3.2.6}}{=} \sum_{p,q,r,s} A_{i_{\pi_f} i_{\pi_g} i_{\omega_g}}^{q,p,r,s} C_{p,q,r,s}^{t,u,v,z} \\ &\stackrel{\text{Cor. 3.2.6}}{=} \sum_{p,r,s} G_{i_{\pi_g} i_{\omega_g}}^{p,r,s} C_{p,i_{\pi_f},r,s}^{t,u,v,z} \stackrel{\text{Cor. 3.2.6}}{=} \sum_r G_{i_{\pi_g} i_{\omega_g}}^{t,r,z} F_{i_{\pi_f},r}^{u,v} \end{aligned}$$

\square

3.3 On the Spectral Features of Robust Probing Security

This work revisits t -probing security fundamentals by providing a spectral formalization of non-interference that encompasses recently introduced advancements such as *robust t -probing security* [85, 92], and we published it in Transactions of Conference on Cryptographic Hardware and Embedded Systems (CHES) [89]. The overarching goal is to give an alternative yet comprehensive view of the problem which might be more amenable to proof mechanization, in the same vein as [26, 92]. We thus take a detour from conventional information theoretical considerations (see, for example, [85]) for a more algebraic approach which exploits the characterization of the spectrum of vector Boolean functions and its connections with correlation immunity [119, 123]. Our approach aims to be more foundational than other approaches based on spectral characterization which are based on approximations and do not encompass composability [26]. In this sense, we derive formal conditions for t -probing security in the presence of glitches by further categorizing probes (e.g., *pure vs composed*) to enable compositional reasoning of vulnerability profiles. More importantly, we

have found that, to conciliate with composability, the nature of an extended probe must afford an additional distinction, i.e., output vs internal, where output probes participate, during composition, in the creation of additional extended probes while internal do not. We thus discovered a new definition of robust non-interference which complies with existing observations in literature but has, from our point of view, a more intuitive meaning.

To corroborate the usefulness of our approach, we show that the underlying tensor calculus is useful to reason formally about both conventional and robust t -probing security by giving new meaning to some results already appeared in the past [60]. On the other side, we show that it can enable the exploration of the design space of known gadgets by deriving an improved *consolidated masking scheme* [98] which is robust 3-probing secure and robust 3-SNI without the need of an additional register at the output (compared to [60]). While this is done only for $t = 3$, we can derive sufficient conditions for making a generalized CMS scheme into a robust t -SNI one. We also give some deductions around the DOM multiplication scheme [67] that can be made with our framework.

Before starting this research endeavor, we felt that there was a lack of mathematical definitions of robust strong non-interference. This concern was raised before in the community. Recently, in a paper published on TCHES [85], the authors recognized that despite the existence of the concept of robust SNI, it remained unclear how to automate the verification of composability of hardware gadgets, as it was unclear how to define a single mathematical equation. They acknowledge that there is still room for more automated ways to reason about robust non-interference. To understand how our approach fills this gap, we would like to highlight how our work can benefit the community from both the *research* and *development* standpoints.

The research standpoint. As it is known, one of the main goals of any research endeavor is to build inference rules to derive general solutions to common problem patterns. This is distinguished from solving those problems with an instance-by-instance approach or a tool. To show how our approach can be used for deriving such general rules, we refer the reader to Appendix 3.3.5, where we present some general conclusions about the robust probing security of a common pattern found in cryptography, for any number of shares.

The development standpoint. Existing tools such as `maskVerif` [7] can be helpful in verifying if a fixed configuration instance of a gadget is probing secure or strong non-interferent; we call these *instance-by-instance* tools. Our approach can be used also on an instance-by-instance basis. More importantly, notwithstanding the efficiency of `maskVerif`, its developers argue that "more precise approaches remain important, when verification with more efficient methods fail"[7]. Given that our

3.3.1 Probing security as a relation calculus

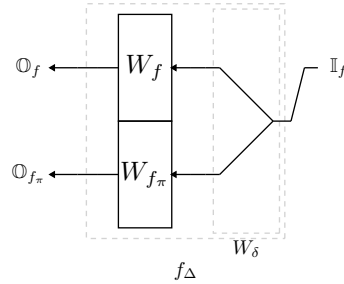


Figure 3.15: The vulnerability profile of a function corresponds to the tensor product of the regular Walsh transform of a function and of its probes f_π , multiplied by W_δ .

approach is not based on a syntactic model but on the exact theory of Boolean functions, it is probably the first to fit this purpose as previous works have only provided approximations [26] or partial solutions [85]. We note that our approach provides the added benefit of a linear algebra based approach which is supported by many mathematical toolboxes. However, given the exponential size of correlation matrices, some analysis of computational complexity is in order. We refer the reader to Appendix 3.3.6 for an estimate of the time needed for computing the vulnerability profile for several known gadgets.

Note that the construction that we propose in this work as revision of the CMS scheme 3 strong non interfering (Section 3.3.3.1) is now declared not 3-SNI. Indeed, though the theoretical concept of the vulnerability profile is well structured and correct, for this paper we wrongly implemented it, and when considering more than one probe we faced a loss of some information. We fixed the problems in it and give a new better solution in Section 3.4.

3.3.1 Probing security as a relation calculus

The methodology that we propose is heavily based on the Walsh transform of a vectorial Boolean function. For all the conventional concepts around it, we refer the reader to Sections 2.1.2.1, 2.1.4 and 2.1.5. To understand the following discussion, we also recommend to read Sections 3.2.1 and 3.2.5.1, for an introduction about the relation calculus that is at the base of our probing security analysis.

3.3.1.1 The vulnerability profile of a function

Figure 3.15 shows a typical wiring diagram of the mapping between the Fourier transform of its input and output distributions. In particular, it is related to the Boolean function f and its potential probes f_π . The tensor product

$$f_{\Delta} = (W_{f_{\pi}} \otimes W_f)W_{\delta} \quad (3.23)$$

encodes all the vulnerability data associated to f . In practice, each row of this matrix corresponds to a convolution of a combination of rows in W_f and in $W_{f_{\pi}}$ and we know that, if there is some input variable combination for which this convolution is not zero, we have a dependency between a combination of outputs (either outputs of f or its probes) and a subset of input variables [123, 119]. We call this data *the vulnerability profile of f* . It is a special case of *fan*, a notion that will be useful to deal with glitches and extended probes as well:

Definition 3.3.1 (fan of a family of matrices). The *fan* of a family of matrices $M = \{M_i\}_{i=1\dots n}$ is a matrix:

$$\Delta M = \left(\bigotimes_i M_i\right)W_{\delta}^{n-1}$$

where W_{δ} is the correlation matrix associated with the duplication function.

3.3.1.2 Composition of vulnerability profiles

It is possible to derive the vulnerability profile of a composition of two functions by studying the composition of two fans:

$$k_{\Delta} \bullet h_{\Delta} = \Delta\{W_{h_{\pi}}, W_{k_{\pi}h}, W_{kh}\}$$

which is the fan of the composition of the original functions:

$$k_{\Delta} \bullet h_{\Delta} = (k \bullet h)_{\Delta}$$

and it is possible to show that it is associative. Figure 3.16 shows the string diagram associated to it where we exploited tensor product equivalences to create a compact yet equivalent representation.

This way of modeling vulnerability allows to reason around t -probing security and t -non-interference in a composable way. Recall that a function f is t -non interferent (t -NI) if, when given a total of s outputs and internal probes, $s \leq t$ implies a dependency with maximum s input shares. A function f is strongly t -non interferent (t -SNI) if $s \leq t$ implies a dependency with maximum i input shares, where i is the number of internal probes, among those placed [9].

Let us for example reconsider a case discovered in [50] that proves that, in general, the composition of t -NI and t -SNI functions is not t -NI. Figure 3.17 shows the

3.3.1 Probing security as a relation calculus

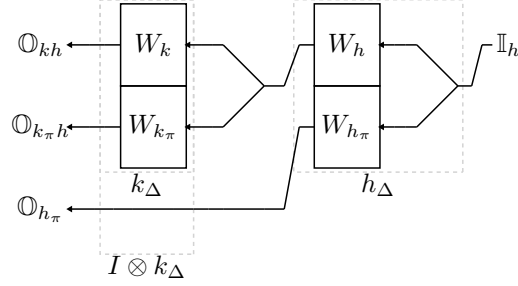


Figure 3.16: The composition of two vulnerability profiles as a map in the probability space.

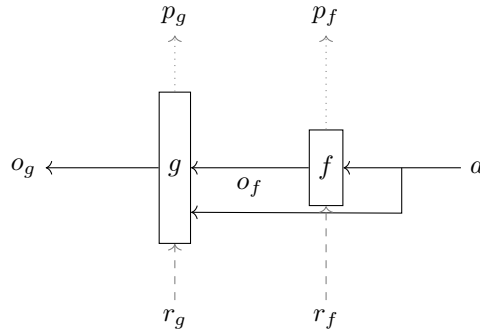


Figure 3.17: The composition pattern of f (t -NI) and g (t -SNI) derived from [50].

structure of a function h which is a composition of two functions f and g . The assumptions are that f is t -NI and g is t -SNI. In particular, f refreshes its input a with two random bits r_f :

$$o_f(a_0, a_1, a_2, r_0, r_1) = [a_0 \oplus r_0 \oplus r_1, a_1 \oplus r_0, a_2 \oplus r_1]$$

and it is assumed to have been probed at location $p_f = a_0 \oplus r_0$. On the other hand, $g(a, b, r_g)$ is the ISW multiplication [73] which consumes 3 random bits r_g for the secret computation. Also in this case, it is assumed a single probe $p_g = a_2 \wedge b_1$.

The string diagram in Figure 3.16 can describe the vulnerability profile of the circuit by considering $h(a, r_f, r_g) = [f(a, r_f), (a, r_g)]$ and $k(a, r_f, r_g, o_f) = g(a, o_f, r_g)$ where the space of the input distributions is $\mathbb{I}_h = \mathbb{A} \otimes \mathbb{R}_f \otimes \mathbb{R}_g$ while for output distributions we have $\mathbb{O}_{kh} = \mathbb{O}_g, \mathbb{O}_{k\pi h} = \mathbb{P}_g, \mathbb{O}_{h\pi} = \mathbb{P}_f$.

Figure 3.18 shows the compact representation of the vulnerability profile. First of all, we are interested only in the first 4 columns, as these are the ones that represent relationships between the outputs and the shares of a not masked by any random value. We note that there is a potential dependency in row $[1, 1, 0]$, column $[0, 0, 3]$, exactly the one found in [50], which says that one needs only two probe values to get three shares. h is thus not even 2-NI, showing that t -NI and t -SNI do not compose

$$\begin{array}{cccccccccccc}
& & & & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \dots \rho_g \\
& & & & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 2 & 2 & \dots \rho_f \\
& & & & 0 & 1 & 2 & 3 & 0 & 1 & 2 & 3 & 0 & 1 & \dots \alpha \\
\pi_f & \pi_g & \omega_g & & & & & & & & & & & & \\
0 & 0 & 0 & 1 & & & & & & & & & & & \\
0 & 0 & 1 & & & & & & & & & & & & \\
0 & 0 & 2 & & & & & & & & & & & & \\
0 & 0 & 3 & 1 & & & & 1 & & & & & & & \\
0 & 1 & 0 & 1 & 1 & & & 1 & 1 & 1 & & & 1 & 1 & \\
0 & 1 & 1 & & & & & & & & & & & & \\
0 & 1 & 2 & & & & & & & & & & & & \\
0 & 1 & 3 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & \\
1 & 0 & 0 & & & & & & 1 & & & & & & \\
1 & 0 & 1 & & & & & & & & & & & & \\
1 & 0 & 2 & & & & & & & & & & & & \\
1 & 0 & 3 & & & & & & & 1 & 1 & & & & \\
1 & 1 & 0 & & 1 & 1 & 1 & 1 & 1 & 1 & & & & 1 & \\
1 & 1 & 1 & & & & & & & & & & & & \\
1 & 1 & 2 & & & & & & & & & & & & \\
1 & 1 & 3 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 &
\end{array} \tag{3.24}$$

Figure 3.18: Vulnerability profile of [50] (we use greek letters to indicate the spectral coordinate associated with each function variable, i.e., α is the spectral coordinate associated with variable a and so on).

into a t -NI function. It is possible to show through the compact representation of vulnerability profiles that, for this composition pattern, if f is t -SNI and g is t -NI (t -SNI) then the composition is t -NI (t -SNI).

3.3.1.3 Extended probes

Extended probes change the attack model in the sense that they allow the attacker to observe all the inputs of a gadget by probing its output wires. We will show that the fan linear algebra introduced above is still suitable for computing probing security profiles with a little more sophistication. In part this is because one has to model in a composable way the information flow from inputs to outputs. Before going into the details let us classify the probes used in this model (we will drop the term *extended* as it is implicit in this discussion and we will introduce some symbol-coding to identify probes):

- a *pure* probe (notation symbol \circ) w_π over a wire computing the combinatorial function $w(x)$, modeled as a Boolean function that has a *stable* non-zero

3.3.1 Probing security as a relation calculus

correlation with all the inputs of w (and their combinations). A stable correlation means that *any* transient effect is observable through that probe*. We will exploit the spectral characteristics of the *and* operator (which has a non-zero correlation with all of its operands and their combination) to model such probes:

$$w_\pi(x) = \bigwedge_{x_i \in \text{support}(w)} x_i$$

- a *composed* probe (notation symbol \circledast) w_κ over a wire computing $w(x) = (w^a \bullet w^b)(x)$ is a probe that can be factored into a pure probe over the intermediate values of w :

$$w_\kappa(x) = (w_\pi^a \bullet w^b)(x)$$

where $w^b(x)$ is different from the identity.

Probes might be orthogonally classified in *output* probes and *internal* ones. This orthogonal characterization is relevant when talking about the composition of blocks:

- potential *output* probes (notation symbol \uparrow) are grouped in sets whose size corresponds to the actual outputs of the function. Among them we will distinguish one set of pure probes and zero or more sets of composed probes. We will use the symbol ω to indicate the overall number of sets ($\omega \geq 1$). Output probes are important during composition of functions because they will produce new probes (either pure or composed).
- *Internal* probes might be pure or composed. Compared to output ones, these will not produce new probes when composing functions but will participate in the computation of the probing profile of the result.

In this context, we define an extended fan that encompasses the probes of the function

$$f_\nabla = \Delta \left\{ \underbrace{W_{f_\pi}^{\uparrow\circ}, W_{f_\kappa^1}^{\uparrow\infty}, \dots, W_{f_\kappa^{\omega-1}}^{\uparrow\infty}}_{\text{output}}, \underbrace{W_{f_i^1}^{\circ|\infty}, \dots, W_{f_i^{\omega}}^{\circ|\infty}, W_f^{\uparrow}}_{\text{internal}} \right\}$$

Provided that one has all the matrices involved, f_∇ describes the overall security profile of the function. An important observation (which will be useful later) is that if one considers a register r there is a single set of pure output probes that one can build, i.e., constant ones.

$$r_\nabla = \Delta \{ W_c^{\uparrow\circ}, I^{\uparrow} \} \tag{3.25}$$

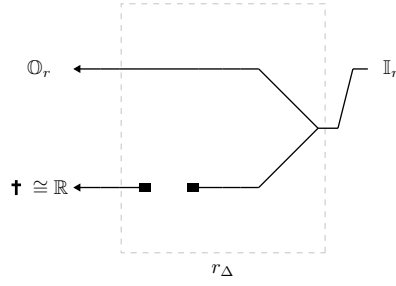


Figure 3.19: The vulnerability profile of a register in terms of maps over the Fourier transform of input and output distributions.

This will have an important implication because, when computing the composition of blocks, the zero matrix will become a circuit breaker, essentially forcing all successive functions to map to it as well. Let us consider the composition of two fans:

$$g_{\nabla} \bullet f_{\nabla} = h_{\nabla}$$

For it to be associative, the new fan h_{∇} will be such that:

- its *internal* probes (\circ or ω) will be all those internal to f plus those produced by composing:
 - internal probes in g (\circ or ω) with f 's outputs and
 - internal pure probes in g (\circ) with output probes in f ($\uparrow\omega$ or $\uparrow\circ$).
- its *output* probes ($\uparrow\omega$ or $\uparrow\circ$) will be generated by combining
 - pure output probes in g ($\uparrow\circ$) with f 's outputs and its output probes ($\uparrow\omega$ or $\uparrow\circ$).
 - composed output probes in g ($\uparrow\omega$) with f 's outputs.

Table (3.1) shows all the composition rules.

Example 3.3.2. Assume that both f and g have only a set of pure (output) probes:

$$f_{\nabla} = \Delta\{W_{f_{\pi}}^{\uparrow\circ}, W_f^{\uparrow}\}$$

$$g_{\nabla} = \Delta\{W_{g_{\pi}}^{\uparrow\circ}, W_g^{\uparrow}\}$$

then, $g_{\nabla} \bullet f_{\nabla}$ will be

$$g_{\nabla} \bullet f_{\nabla} = \Delta\{W_{g_{\pi}f}^{\uparrow\omega}, W_{g_{\pi}f_{\pi}}^{\uparrow\circ}, W_{f_{\pi}}^{\circ}, W_{gf}^{\uparrow}\}$$

3.3.1 Probing security as a relation calculus

g	f	$g \bullet f$
\uparrow	\uparrow	\uparrow
$\uparrow \omega$	\uparrow	$\uparrow \omega$
$\uparrow \circ$	\uparrow	$\uparrow \omega$
$\uparrow \circ$	$\uparrow \omega$	$\uparrow \omega$
$\uparrow \circ$	$\uparrow \circ$	$\uparrow \circ$
\circ	\uparrow	ω
\circ	$\uparrow \omega$	ω
\circ	$\uparrow \circ$	\circ
-	\circ, ω	\circ, ω

Table 3.1: Algebraic composition rules for probes.

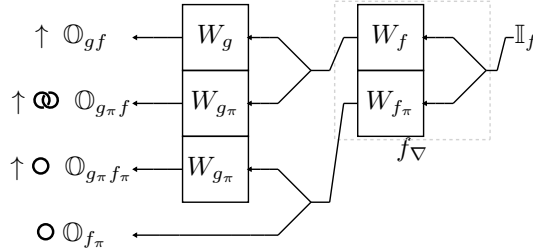


Figure 3.20: The vulnerability profile of a composition of functions when considering extended probes.

Diagrammatically, one could picture the above vulnerability profile as in Figure 3.20. If we compare this with the non-extended case (Figure 3.16), we see an additional pure output probe whose correlation matrix is

$$W_{g_\pi f_\pi}^{\uparrow \circ}$$

This probe practically connects the outputs of the resulting vulnerability profile to the inputs of f (with maximum correlation).

Example 3.3.3. Let us now consider the case where, between g and f , we put a register r . The pure composition of these three blocks is shown in Figure 3.21. However, if we consider the vulnerability profile of the register (Figure 3.19), we get a more explicative diagram in Figure 3.22 which faithfully translates into correlation matrices and corresponding Fourier transform of the probability distributions. In practice, the probes are isomorphic to the one that would be produced by

*This definition works only for combinatorial functions. For a register, instead, any pure probe on its output will have zero correlation with its inputs.

$$g_{\nabla} \bullet r_{\nabla} \bullet f_{\nabla} = \Delta\{W_{g_{\pi}f}^{\uparrow\omega}, W_{f_{\pi}}^{\circ}, W_{gf}^{\uparrow}\}$$

i.e, to the composition of vulnerability with probes acting as regular probes, not extended ones.

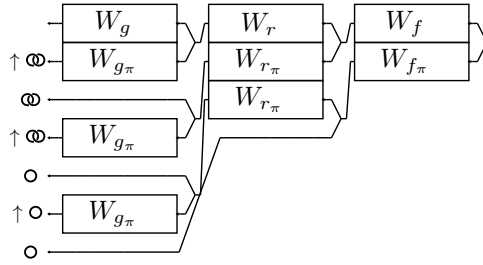


Figure 3.21: The vulnerability profile of a composition of three functions when considering extended probes.

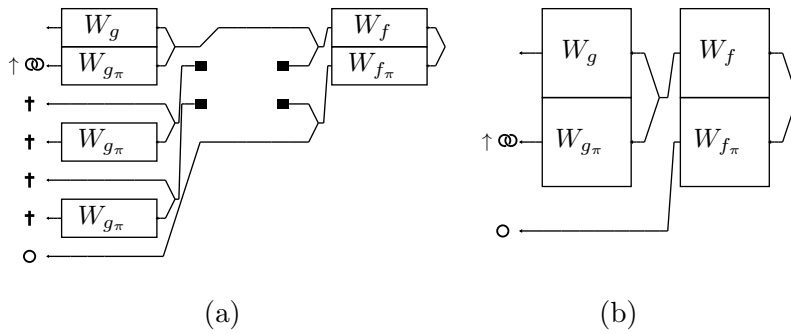


Figure 3.22: (a) shows the vulnerability profile of a composition of two functions when a register is considered in the middle. Probes that come after the "circuit breaker" map to the unit of $\mathbf{Vect}_{\mathbb{R}}$ and thus do not add any information so they have been drawn with a white circle. The Fourier transform of the output distribution is isomorphic to the one produced by the diagram in (b).

3.3.2 Definition of robustness

Given a vulnerability profile f_{∇} , we propose the following robustness definitions.

Definition 3.3.4 (robust- t -probing-secure vulnerability profile). A vulnerability profile f_{∇} is t -probing secure when given a total of t outputs (either conventional or output probes) and internal probes (either composed or pure), there is no dependency with all the shares of a secret.

Definition 3.3.5 (robust t -NI vulnerability profile). A vulnerability profile f_{∇} is robust t -NI when given a total of s outputs (either conventional or output probes) and internal probes (either composed or pure), $s \leq t$ implies a dependency with maximum s input shares.

Definition 3.3.6 (robust t -SNI vulnerability profile). A vulnerability profile f_{∇} is robust t -SNI when given a total of s outputs (either conventional or output probes) and internal probes (either composed or pure), $s \leq t$ implies a dependency with maximum i input shares, where i is the number of internal probes.

Example 3.3.7. To show how the above definition of robust t -probing security matches with the existing understanding, let us rederive the considerations exposed in [60, 85] concerning the compositionality of a second-order secure multiplier when considering glitches. The example considers inputs (x_0, x_1, x_2) and (y_0, y_1, y_2) and consists of two stages separated by a register r . The first stage (let us call it f) contains 9 products $x_i y_j$ some of them (cross-domain products) are remasked:

$$\begin{aligned} f_{0,0} &= x_0 y_0 & f_{0,1} &= x_0 y_1 \oplus r_1 & f_{0,2} &= x_0 y_2 \oplus r_2 \\ f_{1,0} &= x_1 y_0 \oplus r_1 & f_{1,1} &= x_1 y_1 & f_{1,2} &= x_1 y_2 \oplus r_3 \\ f_{2,0} &= x_2 y_0 \oplus r_2 & f_{2,1} &= x_2 y_1 \oplus r_3 & f_{2,2} &= x_2 y_2 \end{aligned} \quad (3.26)$$

The second stage (let us call it g) compresses the triplets:

$$\begin{aligned} g_0 &= f_{0,0} \oplus f_{0,1} \oplus f_{0,2} \\ g_1 &= f_{1,0} \oplus f_{1,1} \oplus f_{1,2} \\ g_2 &= f_{2,0} \oplus f_{2,1} \oplus f_{2,2} \end{aligned} \quad (3.27)$$

The question is whether outputs $g_0 \dots g_2$ should be saved into a register s to preserve composability in the sense of robust t -probing security. We thus compute the two vulnerability profiles:

$$s_{\Delta} \bullet g_{\Delta} \bullet r_{\Delta} \bullet f_{\Delta} = \Delta \{W_{f_{\pi}}^{\circ}, W_{g_{\pi}f}^{\infty}, W_{gf}^{\uparrow}\} \quad (3.28)$$

and

$$g_{\Delta} \bullet r_{\Delta} \bullet f_{\Delta} = \Delta \{W_{f_{\pi}}^{\circ}, W_{g_{\pi}f}^{\uparrow\infty}, W_{gf}^{\uparrow}\} \quad (3.29)$$

We note that, when the register s is not present, probe $g_{\pi}f$ is an output probe ($\uparrow\infty$) and will participate in creating new probes in the following compositions. Instead, when the register s is present (Eq. 3.28), the *outputs* are only the conventional outputs of $g \bullet f$ while $g_{\pi}f$ is just an internal composed probe (∞).

Considering again Eq. 3.29, if we take just one output in $g_\pi \bullet f$ and no internal probes, one would get a dependency with $f_{0,0}$ which in turn depends* on one share of x and y . This shows that the case without output register is not robust t -SNI, because there should not be any dependency over input shares. Note that this observation has already been done in the past [60]. However, we argue that ours is one the first attempts to formalize this point mathematically.

Before closing we note that, in a general case such as Eq. 3.29, one has always $W_{gf} \preceq W_{g_\pi f}$ because extended probes over g are always more powerful of g itself. in this case, robust t -probing security is thus determined by W_{f_π} and $W_{g_\pi f}$ alone:

$$g_\Delta \bullet r_\Delta \bullet f_\Delta = \Delta\{W_{f_\pi}^\circ, W_{g_\pi f}^{\circ\uparrow}\} \quad (3.30)$$

we will exploit this consideration in the following sections.

3.3.3 Revisiting the probing security of CMS

The acronym CMS stems from the title of the proposing article [98] and identifies an evolution of the ISW scheme [73] meant to provide, at the same time, t -probing security and protection against glitches by borrowing ideas from the TI scheme [95].

A CMS scheme with $s = 4$ shares is organised as in Figure 3.23. Every output share c_i is computed in a *logic cone* which involves s pairs $(a_i, b_h), h \in \{0 \dots s - 1\}$. *Adjacent* cones share only a random bit while internal bits within a cone preserve uniformity, as is usual in a TI scheme. The computation is typically decomposed in three layers: non-linear (\mathcal{N}), refresh (\mathcal{R}) and compression (\mathcal{C}), the latter two separated by a register to mitigate the propagation of glitches to the outputs.

While the original proposal identified a scheme that was t -probing secure up to $t = 2$, a simple generalization of the scheme to $t = 3$ has shown that, as it is, it cannot be made probing secure anymore [92]. Figure 3.23 shows the scheme for $t = 3, s = 4$ and a triplet of probes that reveals the four shares of b . Note that this vulnerability exists even if one does not consider extended probes.

Considering robust probing security, we can say something more. First of all, note that we are in the case covered by Eq. 3.30 where:

$$f = \mathcal{R} \bullet \mathcal{N}, g = \mathcal{C}$$

We thus know that the only probes that determine robust t -probing security are:

- the pure probes at the output of the refresh layer, i.e., f_π ;

*Recall that $a \wedge b$ is correlated with both a, b and $a \oplus b$, as its correlation matrix shows.

3.3.3 Revisiting the probing security of CMS

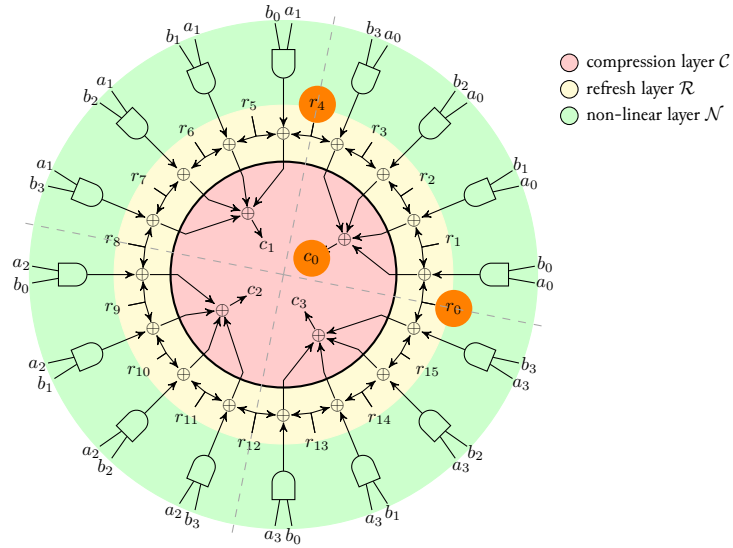


Figure 3.23: The four-share CMS scheme considered in [92]. The scheme is decomposed in three layers, non-linear (\mathcal{N}), refresh (\mathcal{R}) and compression (\mathcal{C}). To preserve output shares from the propagation of glitches, a register (thick line) layer is inserted between compression and refresh. Orange circles correspond to regular probes that break the t -probing security.

- the composed probes at the output of the compression layer $g_\pi \bullet f$.

Composed probes are just four and the number of shares that they cover is important to determine probing security. To show this, let us assign them a label:

$$S_c = \{c_0, c_1, c_2, c_3\}$$

and show, in a table, which pairs (a_i, b_h) are covered by which extended probe:

	b_0	b_1	b_2	b_3
a_0	c_0	c_0	c_0	c_0
a_1	c_1	c_1	c_1	c_1
a_2	c_2	c_2	c_2	c_2
a_3	c_3	c_3	c_3	c_3

Note that, given one of these output probes, e.g., c_0 , one needs to recover only the two random bits that separate it from adjacent cones c_1 and c_3 . These two bits can be derived only by using just two pure probes of f_π .

It has been observed that *non-completeness* might be useful in this case to reach robust t -probing security for $t = 3$ (see [92]). To find it, we note that a combination*

*With the symbol $\mathcal{P}(S_c)$ we denote the power set of S_c

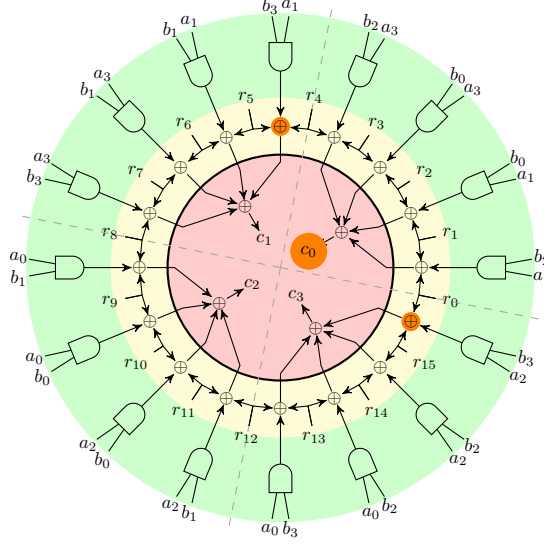


Figure 3.24: The robust 3-probing secure CMS scheme found with our formalization. Highlighted in orange the probes which make the above scheme not robust 3-SNI.

of output probes $\Pi \in \mathcal{P}(S_c)$ is such that it needs R_Π additional randoms depending on adjacency. For example, $\Pi = \{c_0\}$ would need $R_\Pi = 2$ pure probes in f_π , while $\Pi = \{c_0, c_2\}$ would need $R_\Pi = 4$ because they form two partitions in terms of adjacency. In fact, given T_Π as the number of such partitions we have $R_\Pi = 2T_\Pi$.

The organization of the input shares can be seen as a surjective mapping from the set of pairs of input shares to the set of output cones:

$$\lambda : S_a \times S_b \rightarrow S_c$$

Define $\delta_a^\lambda(\Pi)$ as the maximum number of shares of a that a specific probe configuration Π covers ($\delta_b^\lambda(\Pi)$ is analogously defined). To find a configuration that is robust 3-probing secure we can state the following problem:

Find a mapping λ such that, if the total number of probes is less or equal to three, the number of shares that one can get is always less or equal to three, i.e.:

$$\begin{aligned} \forall \Pi \in \mathcal{P}(S_c). \quad |\Pi| + R_\Pi \leq 3 &\implies \delta_a^\lambda(\Pi) \leq 3 \quad \wedge \\ |\Pi| + R_\Pi \leq 3 &\implies \delta_b^\lambda(\Pi) \leq 3 \end{aligned}$$

We formalized the problem as an *satisfiability modulo theory* one and solved it through Microsoft's z3 SMT solver. The solver provided the following solution λ (also depicted in Figure 3.24):

3.3.3 Revisiting the probing security of CMS

$$\begin{array}{cccccccccccccccccccc}
0 & \dots & \rho \\
0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 & 2 & 2 & 2 & 2 & 2 & 3 & 3 & 3 & 3 & 3 & 4 & 4 & 4 & 4 & \dots & \beta \\
0 & 1 & 2 & 3 & 4 & 0 & 1 & 2 & 3 & 4 & 0 & 1 & 2 & 3 & 4 & 0 & 1 & 2 & 3 & 4 & 0 & 1 & 2 & 3 & 4 & \dots & \alpha \\
\omega_{f_\pi} & \omega_{g_\pi f} & \\
\dots & \dots & \\
0 & 3 & \\
\dots & \dots & \\
1 & 2 & \\
\dots & \dots & \\
2 & 1 & 1 & 1 & 1 & \textcircled{1} & 1 & 1 & 1 & 1 & \textcircled{1} & 1 & 1 & 1 & 1 & \textcircled{1} & 1 & 1 & 1 & 1 & \textcircled{1} & & & & & & & \\
\dots & \dots & \\
3 & 0 & 1 & \\
\dots & \dots &
\end{array} \tag{3.31}$$

Figure 3.25: The vulnerability profile of the robust 3-probing secure CMS scheme found with our formalization. This has been computed only for a sum of the hamming weight of the output spectral coordinates (i.e., the sum of probes) equal to 3. Red circles indicate where the vulnerability profile fails to be robust 3-SNI because for $\omega_{f_\pi} = 2$ there can be a dependency with up to $\alpha = 2$ or $\beta = 2$.

	b_0	b_1	b_2	b_3
a_0	c_2	c_2	c_3	c_3
a_1	c_0	c_1	c_0	c_1
a_2	c_2	c_2	c_3	c_3
a_3	c_0	c_1	c_0	c_1

We verified this solution by computing the vulnerability profile as in Eq. 3.30. We computed the underlying correlation matrices W_{f_π} and $W_{g_\pi f}$ by using a sparse representation while the complete fan is computed by convolving the rows of the above two matrices. The used sparse representation of the correlation matrix is a modified version of a List of Lists representation (LIL): each stored list refers to a specific row of the correlation matrix, and the elements of every list are the column coordinates of the nonzero element in the correlation matrix row. We do not need to store the value of nonzero elements, because the presence of this nonzero elements is the only thing that matters. To give an idea of the space required for correlation matrices for $t = 3$, W_{f_π} is a $2^{16} \times 2^{28}$ correlation matrix with less than 2^{24} elements different from 0. However, one needs to store data associated with only 16 rows because the remaining part can be computed with convolution (if needed) by definition of vector Walsh transform. In this use case, the overall operations involved in computing the convolutions for determining strong-non interference for $t = 3$ are about 1.35×10^6 , for $t = 4$ about 4.92×10^7 and for $t = 5$ about 2.63×10^9 .

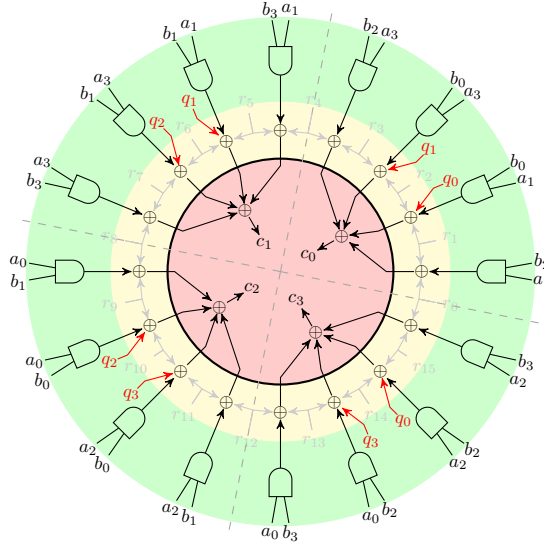


Figure 3.26: The robust 3-SNI CMS scheme proposed in Section. Additional random are identified with the label q_j . Other randoms have been grayed out to avoid crowding the image.

The compact representation of the resulting vulnerability profile can be seen in Figure 3.25. In this matrix, $\omega_{f_\pi}, \omega_{g_\pi f}$ are the *compact spectral index* of pure internal probes f_π and output composed probes $g_\pi f$ respectively. Similarly, α, β, ρ are the compact spectral index of the shares of a, b and the refresh random bits r . Note that, for the spectral indexes $\alpha = 4$ or $\beta = 4$ and $\rho = 0$ the correlation between the extended probes f_π and $g_\pi f$ is 0; this solution is thus robust 3-probing secure.

One could check whether for $s > 4$ there can be suitable solutions to the above problem. However, we have found that for $s \geq 6$ the underlying formulas are not satisfiable anymore, leaving us conjecture that the above scheme hits an upper bound for $s = 5$.

3.3.3.1 Achieving Robust Strong non-Interference for CMS

Figure 3.25 shows that the solution is not robust 3-SNI (we have marked in red the correlation matrix positions that violate t -SNI properties). This is because, for two internal probes f_π one can get up to three shares of either a, b or both*.

Indeed, as shown in Figure 3.24, if an output probe is placed on c_0 , and two internal probes are placed in the refresh layer of adjacent cones, e.g., after the operation $r_{15} \oplus (a_2 \cdot b_3) \oplus r_0$ and $r_4 \oplus (a_1 \cdot b_3) \oplus r_5$, one can recover information about three shares of a (a_1, a_2, a_3) and three shares of b (b_0, b_2, b_3) with only two internal pure

*Note that we have shown columns up to all combinations of a and b not covered by random values.

3.3.4 Analysis of the robust probing security of DOM-indep

$$\begin{array}{r}
 0000000000000000000000000000 \dots \rho \\
 0000011111222223333344444 \dots \beta \\
 0123401234012340123401234 \dots \alpha \\
 \omega_{f_\pi} \quad \omega_{g_\pi f} \\
 \dots \quad \dots \\
 0 \quad 3 \\
 \dots \quad \dots \\
 1 \quad 2 \\
 \dots \quad \dots \\
 2 \quad 1 \\
 \dots \quad \dots \\
 3 \quad 0 \quad 1111 \quad 1111 \quad 1111 \quad 1111 \\
 \dots \quad \dots
 \end{array} \tag{3.32}$$

Figure 3.27: Vulnerability profile of CMS scheme with $s = 4$ when using additional randoms q_j .

probes. The rationale is that, whenever we try to attack the input shares of a cone, we need to remove the two protecting randoms through two internal probes in adjacent cones. Since any adjacent cone will work with different shares, these will be attacked as well.

One countermeasure would be to increase the number of randoms that protect adjacent cones. For example, by adding one random for each pair of adjacent cones we would have that if an output probe is placed on one output c_i , no matter how the two internal probes are placed in the scheme, the output is always protected by two random and one would need obviously two other internal probes. Note that, even by adding output probes from adjacent cones, these will still be protected by four random so one is forced to use internal probes. In Figure 3.26, we show a proposal for such a scheme for $s = 4$ where additional random q_j are applied pairwise to cones. Computing the vulnerability profile for such scheme yields Figure 3.27. As can be seen, two internal probes now do not imply a correlation with any share.

3.3.4 Analysis of the robust probing security of DOM-indep

As another example of application of our framework, we analyze the robust t -probing security of another multiplication gadget referred to Domain Oriented Masking with independent shares (DOM-indep). Domain Oriented Masking is an alternative shared multiplication scheme which aims to be t -probing secure by using $t(t+1)/2$ random bits [67]. It is the basis above more sophisticated schemes have been built (such as DOM-dep or *DOM with dependent shares* [66]). The generic structure of DOM-indep is as follows:

$$\begin{aligned}
c_0 &= \mathbf{a_0 b_0} + (a_0 b_1 + r_0) + (a_0 b_2 + r_1) + (a_0 b_3 + r_3) \dots \\
c_1 &= (a_1 b_0 + r_0) + \mathbf{a_1 b_1} + (a_1 b_2 + r_2) + (a_1 b_3 + r_4) \dots \\
c_2 &= (a_2 b_0 + r_1) + (a_2 b_1 + r_2) + \mathbf{a_2 b_2} + (a_2 b_3 + r_5) \dots \\
c_3 &= (a_3 b_0 + r_3) + (a_3 b_1 + r_4) + (a_3 b_2 + r_5) + \mathbf{a_3 b_3} \dots \\
&\dots
\end{aligned}$$

Bold multiplication terms are called inner-domain terms and do not require to be masked with randoms while, for the remaining cross-domain terms, the same random is reused to mask terms with mirrored indices. Parentheses indicate that terms are saved into registers before being *compressed* into the output share.

The current understanding of the DOM scheme is that, at least in the implementation that considers dependent shares (DOM-dep), it is not robust t -SNI [92]. We will show here that also DOM-indep is not robust t -SNI. To do so, we will study how inner pure probes f_π and output composed probes $g_\pi f$ behave.

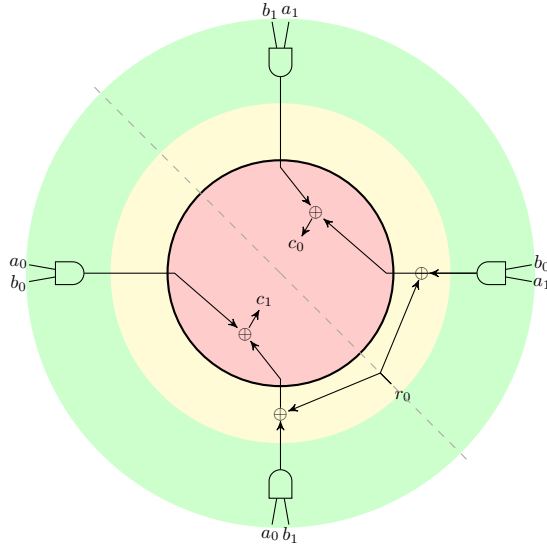


Figure 3.28: The DOM scheme for $t = 1, s = 2$

The reasoning is simple: consider Figure 3.28. For it to be robust 1-SNI, taking an output composed probe and no internal pure probes should not provide any information on input shares. However, an extended output probe on c_1 , for example, allows to observe one of its inputs, i.e., $a_0 b_0$ which is not covered by any random. Given that $a_0 b_0$ correlates with either share a_0 or b_0 we would have one input share observable with zero internal probes, which goes against robust 1-SNI premises.

The vulnerability profile is shown in Figure 3.29 (left) where inner probes are accounted with ω_i while outputs and output probes are accounted with ω_o . We can see that it is not robust 1-SNI, as for $(\omega_i = 0, \omega_o = 1)$ there is a dependency with at least one share of α and β ; however the gadget is still robust 1-probing secure.

3.3.4 Analysis of the robust probing security of DOM-indep

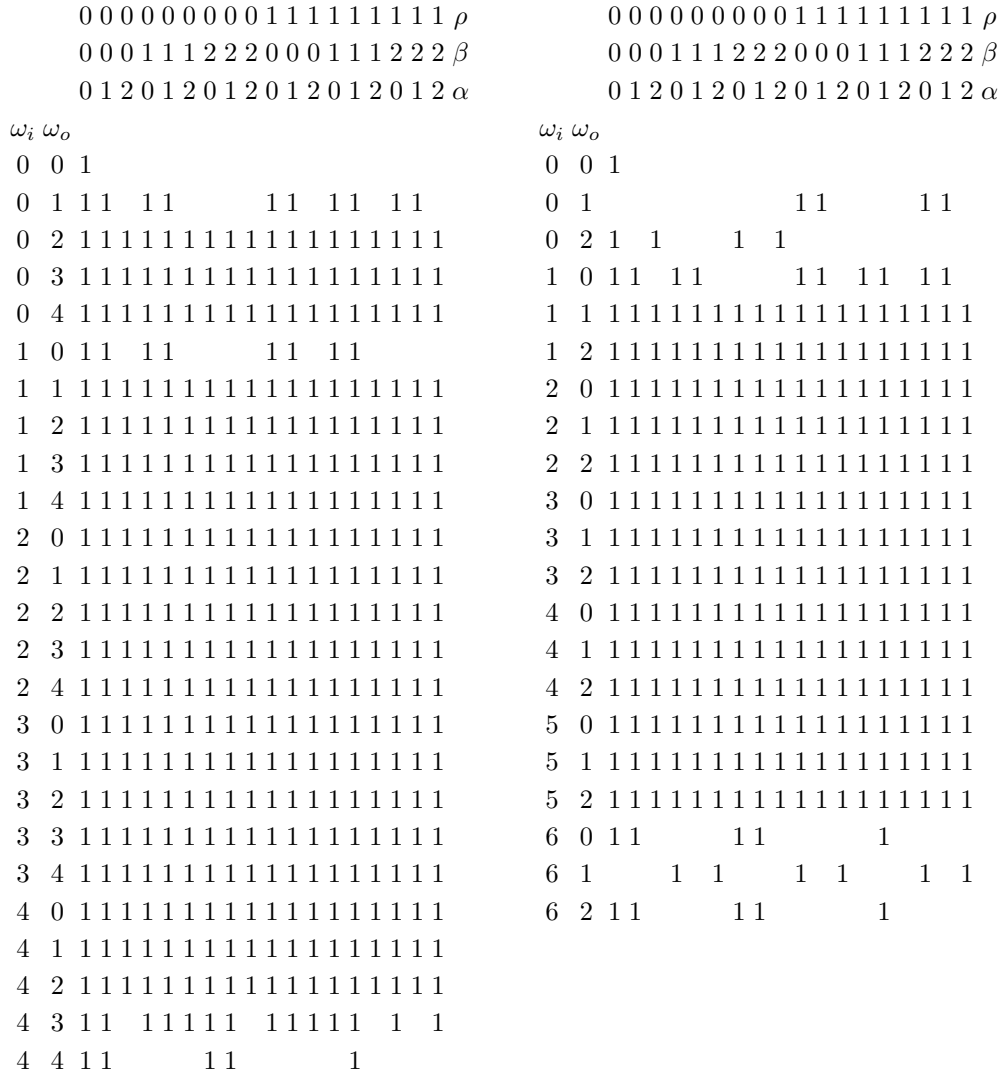


Figure 3.29: Vulnerability profiles of DOM without (left) and with (right) output register for $t = 1$ ($s = 2$).

Adding an output register we obtain Figure 3.29 (right) where we can see that it is actually robust 1-SNI. Note that the one with the register has more inner probes because the original non-registered outputs have become internal. Figure 3.30 shows (part of) the vulnerability profiles for $t = 2$ which confirm that adding a register at the outputs makes the gadget robust 2-SNI.

We verified that the same happens for $t = 3$. We note that, in this case, the gadget DOM-indep uses 6 randoms, while the robust-3-SNI variant we propose for CMS uses 20 (without output register). This suggests that there exists a trade-off between registers and randomness when dealing with robust non interference. The

00000000000000000001111 ρ	00000000000000000001111 ρ
00001111222233330000 β	00001111222233330000 β
01230123012301230123 α	01230123012301230123 α
$\omega_i \omega_o$	$\omega_i \omega_o$
0 0 1	0 0 1
0 1 11 11 11	0 1
0 2 111 111 111 1111	0 2
0 3 11111111111111111111	0 3 1 1 1 1
0 4 11111111111111111111	1 0 11 11 1111
0 5 11111111111111111111	1 1 11 11 1111
0 6 1111111111111111 111	1 2 11111111111111111111
1 0 11 11 11 1111	1 3 11111111111111111111
1 1 111 111 111 1111	2 0 111 111 111 1111
1 2 11111111111111111111	2 1 11111111111111111111
1 3 11111111111111111111	2 2 11111111111111111111
1 4 11111111111111111111	2 3 11111111111111111111
1 5 11111111111111111111	3 0 11111111111111111111
1 6 11111111111111111111	3 1 11111111111111111111
2 0 111 111 111 111	3 2 11111111111111111111
2 1 11111111111111111111	3 3 11111111111111111111
2 2 11111111111111111111	4 0 11111111111111111111
2 3 11111111111111111111	4 1 11111111111111111111
2 4 11111111111111111111	4 2 11111111111111111111
2 5 11111111111111111111	4 3 11111111111111111111
.....

Figure 3.30: Part of the vulnerability profiles of DOM without (left) and with (right) output register for $t = 2$ ($s = 3$).

ratio of random usage between DOM and our CMS construction is:

$$\frac{2 \left(\frac{s^2}{2} + \left(\frac{s}{2} + 1 \right) s \right)}{(s-1) s}$$

which, asymptotically, provides a $2\times$ size factor. We conjecture that this is the cost one has to pay for sparing registers when building a robust t -SNI gadget.

3.3.5 On enabling general reasoning about non-interference

The aim of this Section is to show how the proposed formalization can enable the analysis and derivation of new inference rules around (robust) non-interference. For this purpose, consider the circuit represented in Figure 3.17. This circuit is one

3.3.5 On enabling general reasoning about non-interference

of building blocks used for computing the inverse in $GF(2^8)$ [50]. It is known that this type of circuit is t -SNI when both g and f are t -SNI [9]. Figure 3.31 shows the corresponding correlation matrices diagram when not considering robustness against glitches (namely probes are non-extended probes).

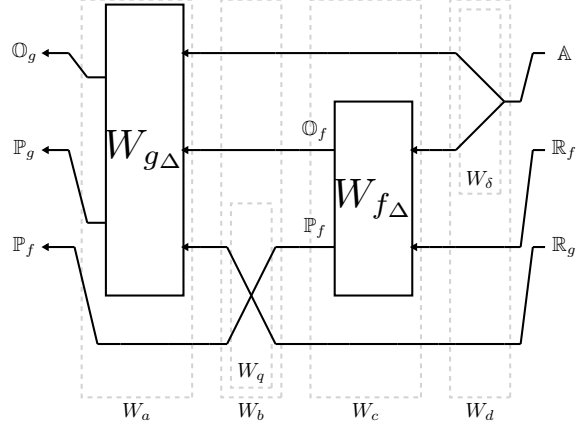
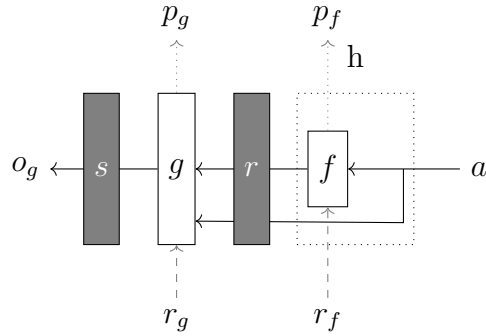


Figure 3.31: Map between Fourier transforms of probability distributions implied by the considered example composition pattern.

Figure 3.32: The considered example composition pattern, gray boxes are registers.



Consider now the same circuit with a register between f and g (Figure 3.32). This can be seen as the composition of two blocks g and h , where h is the block that duplicates the input and propagates one of the paths through f . We already know that, for robust non-interference, this composition has the vulnerability profile shown in Eq. (3.28) and rewritten here to consider h :

$$s_{\Delta} \bullet g_{\Delta} \bullet r_{\Delta} \bullet h_{\Delta} = \Delta\{W_{h\pi}^{\circ}, W_{g\pi h}^{\infty}, W_{gh}^{\uparrow}\}$$

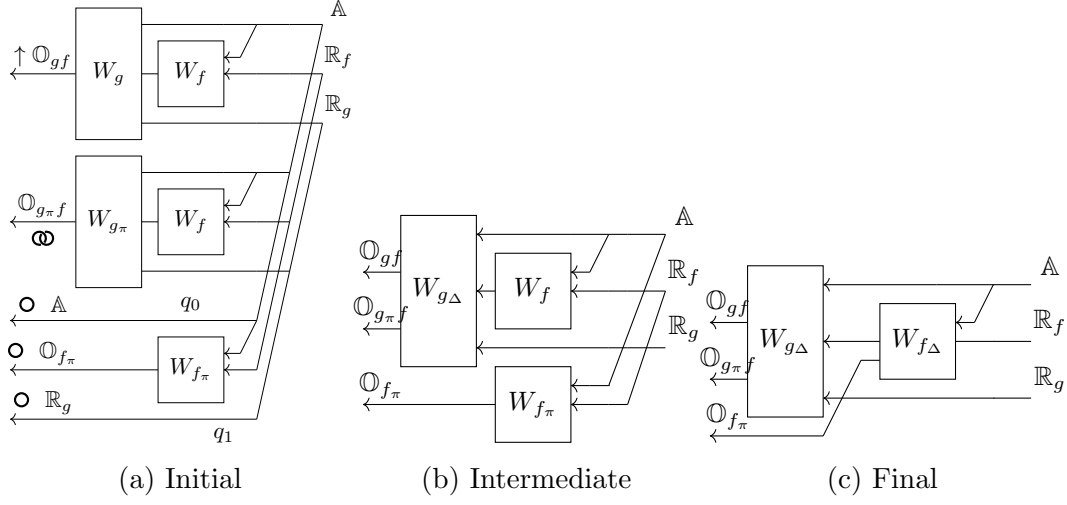


Figure 3.33: Pruning of the a vulnerability profile considering equivalences and dominance relations of the correlation matrix calculus.

where

$$W_h = (I \otimes W_f \otimes I)(I \otimes I \otimes W_\delta) \quad (3.33)$$

$$W_{h_\pi} = (I \otimes W_{f_\pi} \otimes I)(I \otimes I \otimes W_\delta) \quad (3.34)$$

while I is the identity, and W_δ is the correlation matrix of the duplication function $\delta = x \mapsto (x, x)$.

Graphically, the above equations correspond to Figure 3.33a. From there, one can reason by pruning redundant paths. For example, the distribution of \mathbb{A} reaches the output through the extended probe $g_\pi f$, so path q_0 can be pruned off without loss of generality (same thing for path q_1) obtaining 3.33b. Considering the commutativity across duplication points we can move f_π and then substitute the definition of fan (Eq. 3.23) for both f and g , one can obtain 3.33c, which is exactly the vulnerability profile of the non-robust case shown in Figure 3.31 where probes now are extended probes. This means that the reasoning about the non-robust case can be directly applied also to this case, i.e., if f_Δ is t -SNI and g_Δ is t -SNI (t -NI) then the composition in Figure 3.32 is robust t -SNI (robust t -NI). Note that the derivation of the same general conclusions through a classic approach would have possibly required a much more involved demonstration.

3.3.6 Computational complexity and scalability of the proposed approach

The vulnerability profile of a function f is computed starting from its correlation matrices W_f and W_{f_π} (i.e. W_{f_Δ}). The complete computation of these matrices could become quickly impracticable due to the large number of their elements, which, in turn, is exponentially related to the number of inputs, outputs and probes analysed.

To reduce the time and space computational complexity of this operation, we store only the rows that refer to single outputs and probes, and compute on-demand the remaining rows by using convolution. Besides, we exploit the fact that correlation matrices are sparse (as explained in Section 3.3.3), to speed up the convolution itself. In the following, we show some estimates of computation time when such sparsity is considered.

Let us consider the nonlinear function χ of Keccak, implemented using the DOM multiplication gadget to make it probing secure at the t -th order [68]. We recall that the internal state of Keccak is divided into groups of five bits, called *rows*, and function χ is applied row by row. Given x_0, x_1, x_2, x_3, x_4 (the row bits), χ is defined as:

$$y_i = \chi(x_i, x_{i+1}, x_{i+2}) = x_i + (\bar{x}_{i+1}x_{i+2})$$

where indices are computed modulus 5. To make it probing secure at the t -th order, each element x_i is split into $t + 1$ shares $x_i^0, x_i^1, \dots, x_i^t$, and a share y_i^j of the output y_i is computed as follows [68]:

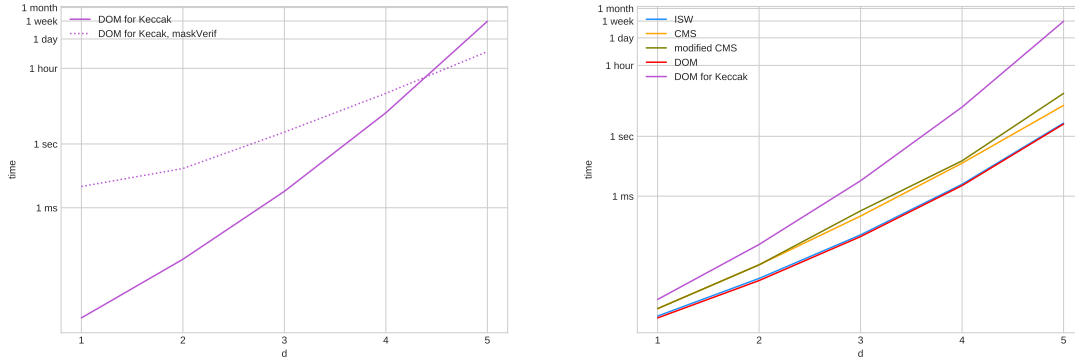
$$y_i^j = \left(x_i^j + (\bar{x}_{i+1}^j x_{i+2}^j + \sum_{h>j} (x_{i+1}^j x_{i+2}^h + r_{j+\frac{h(h-1)}{2}})) + \sum_{h<j} (x_{i+1}^j x_{i+2}^h + r_{h+\frac{j(j-1)}{2}})) \right)$$

Note that parentheses indicate that terms are saved into registers. Thus, there is a register before the compression layer in DOM (reg_1), one that stores multiplication results (reg_2) and one that stores the final output's share y_i^j (reg_3).

Define as χ_t the function that computes all the shares of y_i . In the corresponding circuit, we assume an extended probe on each wire that ends into a register: we thus have $n_1 = (t + 1)t$ probes placed before reg_1 , $n_2 = t + 1$ before reg_2 and $n_3 = t + 1$ before reg_3 .

In the correlation matrix of χ_{t_Δ} each row referring to a probe before reg_1 has only $a_1 = 8$ nonzero elements, while for a probe before reg_2 or reg_3 the nonzero elements are respectively $a_2 = 2^{t+1}$ and $a_3 = 2^{t+2}$. Each remaining row is computed by convolution of p single-probe rows, and it has, in average, $av(p)$ elements different from 0:

$$av(p) = \frac{\sum_{h=0}^p \left(\sum_{k=0}^{p-h} n_1^k a_1^k \cdot n_2^{p-k-h} a_2^{p-k-h} \right) n_3^h a_3^h}{(n_1 + n_2 + n_3)^p}$$



(a) Scalability computed for χ of Keccak with DOM, and comparison with time needed to apply maskVerif tool [7] to the same algorithm. (b) Scalability computed for known algorithms.

Figure 3.34: Estimated time needed to compute the vulnerability profile for well known algorithms.

where $2 \leq p \leq n_1 + n_2 + n_3$. Figure 3.34(a) reports the estimated time needed to compute the correlation matrix of χ_{t_Δ} (solid line), in comparison with the time needed to execute maskVerif [7] to show that χ with DOM algorithm is robust t -NI (dotted line). Figure 3.34(b) reports the estimated time for other known gadgets to compute their correlation matrices. In both cases, the value of t is varying between 1 and 5 and we assume to work with a 8 processors, 4GHz machine with a 1 integer operation per clock cycle throughput*.

3.4 On robust strong-non-interferent low-latency multiplications

We published this work in IET (Institution of Engineering and Technology) Information Security journal [88]. The overarching goal of our work is to present new theoretical and practical tools to implement robust t -probing security. In this Section we present a low-latency multiplication gadget that is secure against probing attacks that exploit logic glitches in the circuit. The gadget is the first of its kind to present a 1-cycle input-to-output latency while belonging to the class of *probing security by optimized composition* gadgets [39]. In particular, we show that it is possible to construct robust t -SNI gadgets without compromising on latency with a moderate

*Luckily, the computation of multiple rows of the correlation matrix can be done in parallel.

3.4.1 Overcoming the latest constructions

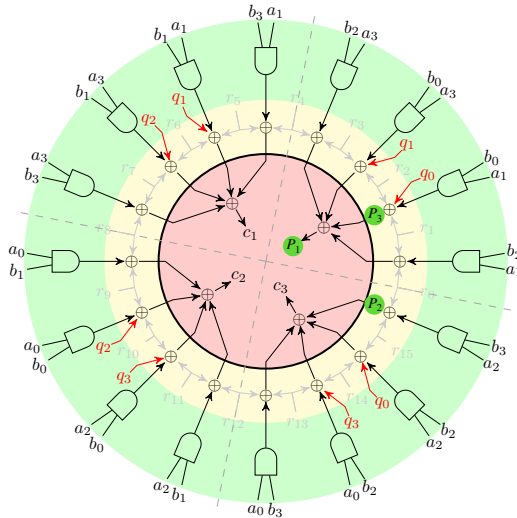


Figure 3.35: 1-cycle latency CMS-derived gadget proposed in [89]. Green discs represent the three extended probes that make it not robust 3-SNI. The black thick line indicates the register layer. The expressions to compute the outputs are those in Eq. 3.35 except that the values in red brackets are not sampled in an additional register, i.e., only those values in the black brackets are sampled.

increase in area. We provide a theoretical proof for the robustness of the gadget and show that, for $t \leq 4$, the amount of randomness required can even be reduced without compromising on robustness.

3.4.1 Overcoming the latest constructions

In this work we address the problem of robust- t -probing security in the context of optimized composability. Recent efforts put into improving CMS masking without increasing the latency have been proposed [89]. Figure 3.26 shows a solution for the case for $t = 3, s = 4$ as proposed by the authors of [89]. Note that the authors elaborate this scheme starting from the first CMS proposed in [98], changing the order of products $a_i b_j$ and introducing additional random bits q_i to protect the shares; however, as we now show, this gadget is not robust 3-SNI. In fact, consider the three probes marked in green P_1, P_2 and P_3 as in Figure 3.35: probes P_2 and P_3 are the only internal probes so all three probes should convey information about up to two shares. P_1 allows to get $(a_1 b_2 + r_0 + r_1, a_1 b_0 + r_1 + r_2 + q_0, a_3 b_0 + r_2 + r_3 + q_1, a_3 b_2 + r_3 + r_4)$, while the two internal probes P_2 and P_3 allow to get $(a_2 b_3, r_0, r_{15})$ and $(a_1 b_0, r_1, r_2, q_0)$ respectively. In principle, the information on the secrets derived from P_1 (e.g., $a_1 b_2$) is covered by at least two random bits (e.g., $a_1 b_2$ is covered with r_0 and r_1). However, it is possible to unmask $a_1 b_2$ from P_1 adding r_0 and r_1 recovered from P_2 and P_3

respectively. Then, three shares of b are exposed (b_2 from P_1 , b_3 from P_2 and b_0 from P_3) with only two internal probes.

3.4.2 A provably robust- t -SNI, 1-cycle-latency CMS-like scheme

The problem with the scheme in Figure 3.35 is that internal extended probes give access to each random used in the refresh layer (yellow section). To overcome this leak, one can sum and save into a register these pairs of random bits to avoid that a single probe (such as for example P_3) has access to both intermediate products and individual refresh random bits. Note that, from the point of view of the input-to-output latency, the gadget is still one cycle as this sum could be pre-computed before receiving the shares a and b . For the above gadget we would have the following expressions:

$$\begin{aligned}
c_0 &= [a_1b_2 + [r_0 + r_1]] + [a_1b_0 + [r_1 + r_2] + q_0] + \\
&\quad + [a_3b_0 + [r_2 + r_3] + q_1] + [a_3b_2 + [r_3 + r_4]] \\
c_1 &= [a_1b_3 + [r_4 + r_5]] + [a_1b_1 + [r_5 + r_6] + q_1] + \\
&\quad + [a_3b_1 + [r_6 + r_7] + q_2] + [a_3b_3 + [r_7 + r_8]] \\
c_2 &= [a_0b_1 + [r_8 + r_9]] + [a_0b_0 + [r_9 + r_{10}] + q_2] + \\
&\quad + [a_2b_0 + [r_{10} + r_{11}] + q_3] + [a_2b_1 + [r_{11} + r_{12}]] \\
c_3 &= [a_0b_3 + [r_{12} + r_{13}]] + [a_0b_2 + [r_{13} + r_{14}] + q_3] + \\
&\quad + [a_2b_2 + [r_{14} + r_{15}] + q_0] + [a_2b_3 + [r_{15} + r_0]]
\end{aligned} \tag{3.35}$$

where square brackets indicate registered values (see Table 3.2), with additional red color when they refer to the registered sum of refresh random bits. One can verify with MASKVERIF [6] that the above gadget is in fact robust 3-SNI. Note that Eq. 3.35 describes the scheme in Fig. 3.35 with some added registers (red brackets). This strategy is not entirely new as it has been used, to the best of our knowledge, only recently [39] in the field of *trivial composability*. However, we will show that also optimized composability might benefit from such strategy, as it is possible to generalize this idea to derive a sufficient condition for a gadget being 1-cycle robust t -SNI, whose general cone structure is shown in Figure 3.36.

Note that the shares a_i and b_j are organized as in the original CMS scheme [98], and the random bits are summed up and registered before using them in the refresh layer.

Proposition 3.4.1. *Given $2s^2$ independent random bits $(q_{ij})_{0 \leq i, j \leq t}$ and $(r_{ij})_{0 \leq i, j \leq t}$ the following and-gadget is robust t -SNI:*

$$c_i := \sum_{0 \leq j \leq t} [a_i \cdot b_j + [r_{i,j} + r_{i,j+1} + q_{i,j} + q_{i+1,j}]] \tag{3.36}$$

3.4.2 A provably robust- t -SNI, 1-cycle-latency CMS-like scheme

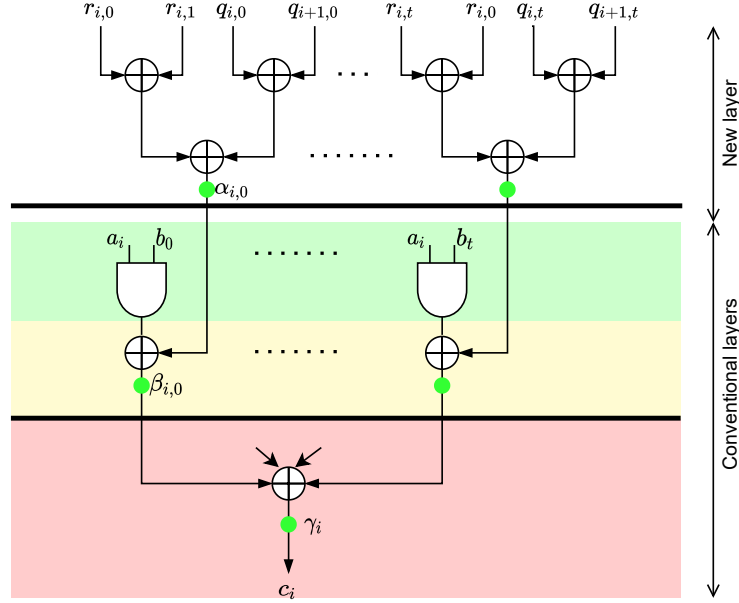


Figure 3.36: A cone of the proposed robust t -SNI CMS structure which has still 1-cycle latency. Green discs represent the possible probes used in proposition 3.4.1. The black thick lines indicate register layers.

where $r_{is} := r_{i0}$, $q_{si} := q_{0i}$ for $0 \leq i \leq t$ and $s := t + 1$.

Proof. For the meaning of mathematical symbols see Table 3.2. Setting $o_{i,j} := a_i \cdot b_j + s_{i,j}$ with $s_{i,j} := r_{i,j} + r_{i,j+1} + q_{i,j} + q_{i+1,j}$ for $0 \leq i, j \leq t$, the extended output probes are $\gamma_i := \{o_{i,j} \mid 0 \leq j \leq t\}$ for $0 \leq i \leq t$, and the maximal extended inner probes are $\alpha_{i,j} := \{r_{i,j}, r_{i,j+1}, q_{i,j}, q_{i+1,j}\}$ and $\beta_{i,j} := \{a_i \cdot b_j, s_{i,j}\}$ for $0 \leq i, j \leq t$.

An attacker gets to pick at most t extended probes, let's say a set Γ of output probes of type γ_j , a set A of inner probes of type $\alpha_{i,j}$ and a set B of inner probes of type $\beta_{i,j}$, s.t. $|\Gamma| + |A| + |B| \leq t$.

Setting $I := \{i \mid \alpha_{i,j} \in A \text{ or } \beta_{i,j} \in B\}$ and $J := \{j \mid \alpha_{i,j} \in A \text{ or } \beta_{i,j} \in B\}$, we

Table 3.2: Meaning of some mathematical symbols employed in the text.

Symbol	Meaning
$[\cdot]$	Value saved into a register
$:=$	Mathematical definition
$ \cdot $	Set cardinality
$\langle v_i \mid i \in I \rangle$	Vector space generated by the vectors $(v_i)_{i \in I}$
$x = y \bmod V$	x equals y modulo the subspace V , i.e.: $x = y + v$ for some $v \in V$

claim that the attacker can simulate all those probes knowing just the inputs a_i for $i \in I$ and b_j for $j \in J$, where clearly $|I| \leq |A| + |B|$ and $|J| \leq |A| + |B|$ ($|A| + |B|$ is the number of the chosen inner probes).

All standard (i.e., non-extended) probes are linear combinations of the linearly independent values $a_i \cdot b_j$, $r_{i,j}$ and $q_{i,j}$ for $0 \leq i, j \leq t$, i.e., elements of the vector space $\langle a_i \cdot b_j, r_{i,j}, q_{i,j} \mid 0 \leq i, j \leq t \rangle$. Applying to the probes the modulo operation w.r.t. the vector subspace $\langle a_i \cdot b_j, r_{i,j} \mid 0 \leq i, j \leq t \rangle$, the probes have values $q_{i,j}$ resp. $q_{i,j} + q_{i+1,j}$. For each j , the values $q_{i,j} + q_{i+1,j}$ span a t -dimensional subspace of the $(t+1)$ -dimensional space generated by the $q_{i,j}$ with $0 \leq i \leq t$, so $\sum_{0 \leq i \leq t} (q_{i,j} + q_{i+1,j}) = 0$ is the only non-trivial linear dependency of the values $q_{i,j} + q_{i+1,j}$ for fixed j . Then for any j , with $R := \langle a_i \cdot b_j, r_{i,j} \mid 0 \leq i, j \leq t \rangle$

$$\sum_{i \in I} s_{i,j} = 0 \pmod R \implies I = \emptyset \text{ or } I = \{0, \dots, t\}. \quad (3.37)$$

Analogously, applying to the probes the modulo operation w.r.t. the vector space $Q := \langle a_i \cdot b_j, q_{i,j} \mid 0 \leq i, j \leq t \rangle$, for fixed j the only non-trivial linear dependency of the values $r_{i,j} + r_{i,j+1}$ is $\sum_{0 \leq j \leq t} (r_{i,j} + r_{i,j+1}) = 0$. Then for any i ,

$$\sum_{j \in J} s_{i,j} = 0 \pmod Q \implies J = \emptyset \text{ or } J = \{0, \dots, t\}. \quad (3.38)$$

Implications (3.37) and (3.38) can be read as: If there are no inner probes of type $\alpha_{i,j}$, a summand $o_{i,j}$ can be cancelled out $\pmod{\langle a_i \cdot b_j \mid 0 \leq i, j \leq t \rangle}$ only by a probe $\beta_{i,j}$ or by another t summands $o_{i',j}$ or $\beta_{i',j}$ in (3.37) resp. t summands $o_{i,j'}$ or $\beta_{i,j'}$ in (3.38).

As the image of the uniform distribution under a linear map is the uniform distribution on its image, a sum σ of standard probes with $\sigma \in \langle \Gamma, A, B \rangle$ has uniform distribution and is independent of all inputs a_i and b_j , unless $\sigma \in \langle a_i \cdot b_j \mid 0 \leq i, j \leq t \rangle$. Hence we have to show that $\langle \Gamma, A, B \rangle \cap \langle a_i \cdot b_j \mid 0 \leq i, j \leq t \rangle \subseteq \langle a_i \cdot b_j \mid i \in I, j \in J \rangle$.

Write $\sigma \in \langle \Gamma, A, B \rangle$ as a sum of standard probes. If σ involves a summand containing the term $a_i \cdot b_j$, this term stems either from the inner probe $\beta_{i,j} \in B$ – implying $i \in I$ and $j \in J$ – or from the summand $o_{i,j} \in \gamma_i \in \Gamma$. As $o_{i,j} = s_{i,j} \pmod{\langle a_i \cdot b_j \mid 0 \leq i, j \leq t \rangle}$, assuming $\beta_{i,j} \notin B$ implies by (3.37) that σ involves either (a) for each $0 \leq i' \leq t$ standard probes as summands that contain the term $s_{i',j}$ or (b) a summand $q_{i',j}$ for some $0 \leq i' \leq t$. The latter case (b) implies that $\alpha_{i',j}$ or $\alpha_{i'-1,j}$ is probed, and hence $j \in J$. The former case (a) requires at least t more probes, as no extended probe involves terms $s_{i,j}$ for more than one i . Both cases lead to a contradiction.

Analogously, given the implication of (3.38), σ involves either (a) for each $0 \leq j' \leq t$ standard probes as summands that contain the terms $s_{i,j'}$ or (b) a summand

$r_{i,j'}$ for some $0 \leq j' \leq t$. The latter case (b) implies that $\alpha_{i,j'}$ or $\alpha_{i,j'-1}$ is probed, and hence $i \in I$. For the former case (a), by just probing γ_i , an attacker can get all the terms $s_{i,j'}$. But in the last paragraph we showed that for each term $s_{i,j'}$ contained in a summand of σ necessarily $j' \in J$, implying $J = \{j \mid 0 \leq j \leq t\}$. As for each inner probe at most one element is added to J , this contradicts that the attacker can choose at most t probes. \square

The placement of the products $a_i \cdot b_j$ in the output cones c_i as well as the presence of randomness in Eq. 3.36 is essential to guarantee that the proposed construction is robust t -SNI. Indeed, a different placement can break (robust) strong non-interference for s big enough. In fact, assume that an attacker chooses n extended output probes $\gamma_1, \dots, \gamma_n$ placed on adjacent cones, and $4(n-1)$ inner probes $\alpha_{1,i}, \alpha_{i,1}, \alpha_{n,i}, \alpha_{i,n}$ for $1 \leq i \leq n$. The probes $\gamma_1, \dots, \gamma_n$ give access to all values $o_{i,j}$ for $1 \leq i, j \leq n$, whose sum is

$$\begin{aligned} & \sum_{1 \leq i, j \leq n} a_i \cdot b_j + \sum_{1 \leq i, j \leq n} (r_{i,j} + r_{i,j+1}) + \sum_{1 \leq i, j \leq n} (q_{i,j} + q_{i+1,j}) \\ &= \sum_{1 \leq i, j \leq n} a_i \cdot b_j + \sum_{1 \leq i \leq n} (r_{i,1} + r_{i,n+1}) + \sum_{1 \leq j \leq n} (q_{1,j} + q_{n+1,j}). \end{aligned}$$

The inner probes allow to derive $r_{i,1} \in \alpha_{i,1}$, $r_{i,n+1} \in \alpha_{i,n}$, $q_{1,i} \in \alpha_{1,i}$ and $q_{n+1,i} \in \alpha_{n,i}$, effectively exposing the first summand $\sum_{1 \leq i, j \leq n} a_i \cdot b_j$ of the equation above; thus $4(n-1) + n$ probes allow to derive n^2 different products $a_i \cdot b_j$. The arrangement of the $a_i \cdot b_j$ in Eq. 3.36 is such that even knowing these n^2 products does not break strong non-interference as the attacker only obtains n different shares a_i and b_j ($1 \leq i, j \leq n$). But already for $s = 12$ and $n = 3$, a different placement of the products $a_i \cdot b_j$ can expose more than $4(n-1)$ shares of either secret, making it not robust strong-interferent.

3.4.2.1 Saving randomness for $t \leq 4$

For $t \leq 4$, the scheme presented in Proposition 3.4.1 can be simplified by removing the random bits $r_{i,j}$ without compromising security. This decreases the number of involved random bits from $2 \cdot s^2$ to s^2 (see Figure 3.37 for this construction). In particular, as one can verify with MASKVERIF, robust t -probing security can be ensured with just the $q_{i,j}$:

$$c_i = \sum_{0 \leq j \leq t} [a_i b_j + [q_{i,j} + q_{i+1,j}]] \quad (3.39)$$

for $0 \leq i, j \leq t, t \leq 4$. However, for $t \geq 5$ this particular scheme breaks because, with a specific choice of three external probes on adjacent c_i and two internal probes, an attacker can recover three shares of a . For example, if the attacker places five probes

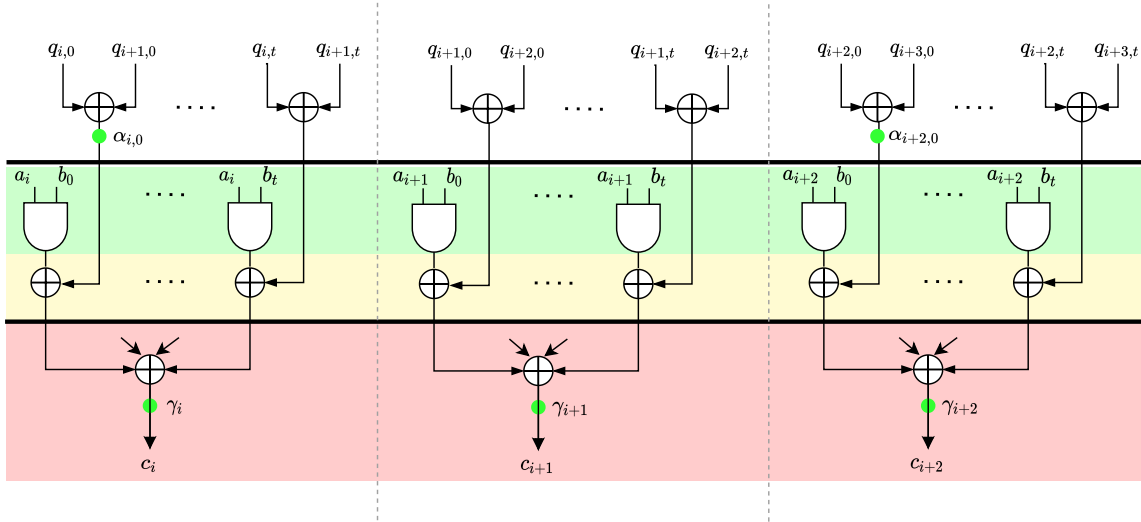


Figure 3.37: The optimized construction which is valid for any $t < 5$ but fails for $t \geq 5$. Green discs represent the probes used to mount the attack.

(see Figure 3.37's green dots) on $\gamma_i, \gamma_{i+1}, \gamma_{i+2}, \alpha_{i,0}$ and $\alpha_{i+2,0}$ then she is able to derive 3 shares of a , with only 2 internal probes. This attack is possible for any $t \geq 5$.

For $t \leq 2$ one can additionally remove the random bits $q_{i,i}$, deriving for $t = 1$ the following scheme with only 2 random bits instead of $s^2 = 4$:

$$\begin{aligned} c_0 &= [a_0 b_0 + q_{1,0}] + [a_0 b_1 + q_{0,1}] \\ c_1 &= [a_1 b_0 + q_{1,0}] + [a_1 b_1 + q_{0,1}] \end{aligned} \quad (3.40)$$

Similarly, for $t = 2$ one obtains the following construction with only 6 random bits instead of $s^2 = 9$:

$$\begin{aligned} c_0 &= [a_0 b_0 + q_{1,0}] + [a_0 b_1 + q_{0,1}] + [a_0 b_2 + [q_{0,2} + q_{1,2}]] \\ c_1 &= [a_1 b_0 + [q_{1,0} + q_{2,0}]] + [a_1 b_1 + q_{2,1}] + [a_1 b_2 + q_{1,2}] \\ c_2 &= [a_2 b_0 + q_{2,0}] + [a_2 b_1 + [q_{2,1} + q_{0,1}]] + [a_2 b_2 + q_{0,2}] \end{aligned} \quad (3.41)$$

Both schemes are robust t -SNI (for $t = 1$ and $t = 2$ respectively), as one can verify with MASKVERIF.

3.4.3 Applications

Our proposed structure allows to obtain an input-share-to-output-share latency of one cycle while still being robust t -SNI, at the expense of increased randomness. A t -SNI gadget could be made robust t -SNI with reasonable latency by replacing all t -SNI *ands* with our proposed gadget, all t -NI *ands* with DOM *ands*, and all t -SNI

refresh gadgets with the robust t -SNI refresh gadgets from [39]. Indeed, compared to the DOM [67] and the HPC2 [39] gadgets, which both need $s(s-1)/2$ random bits, our gadgets require 2x randomness for $s = 2, 3$, about 2.5x for $s = 4, 5$ and more than 4x for $s > 5$. However, our solution requires only 1-cycle latency instead of at least 2 cycles of latency that characterizes the current DOM and HPC2; it is thus clearly a matter of trade-off of latency and randomness. Another application could be to lower the latency of an HPC2-based construction by "kickstarting" the S-boxes: after 1, 2, 3 resp. 4 cycles one can obtain with HPC2 gadgets values of algebraic degrees 1, 2, 3 resp. 5 in the input bits due to their asymmetric latency of 1 resp. 2 in their inputs. Replacing just all HPC2 gadgets in the first layer with our gadget can save one cycle latency, as the achievable algebraic degrees are then 2, 3, 5 resp. 8. This can be done for example for the optimized PRESENT S-box of Fig. 6(b) of [39] to regain the better latency of the DOM-based construction. If additionally all S-box inputs that are added to the PRESENT S-Box outputs are refreshed before with a robust mask refresh, the resulting circuit becomes robust probing secure for, we believe, a moderate increase in area.

3.5 ADD-based Spectral Analysis of Probing Security

This work has been accepted to Design, Automation and Test in Europe (DATE2022) Conference. In this paper, we address the problem of tooling needed for the verification of non-interference properties. To contextualize our work, note that existing *heuristic* tools such as `maskVerif` [7] can be helpful in verifying if a fixed configuration instance of a gadget is t -probing secure or d strong-non-interferent (t -SNI). Notwithstanding the efficiency of `maskVerif`, its developers argue that *more precise approaches remain important when verification with more efficient methods fail* [7]. Therefore, the importance of studying exact techniques is quite evident. A few other approaches have been proposed in the past to address this verification problem through some kind of approximation [26, 85], while existing exact approaches either suffer from size and exponential time complexity [89] or have not been tested on $d > 3$ [75].

In this paper, we introduce an Algebraic Decision Diagram (ADD)-based [5] methodology for the exact validation of circuits against required strong non-interference properties. To our knowledge, the methodology is faster than the state of arts exact methods proposed in the literature and builds on decades of work on BDD/ADD libraries. Benchmarked against a standard set of use cases (taken from the `maskVerif`

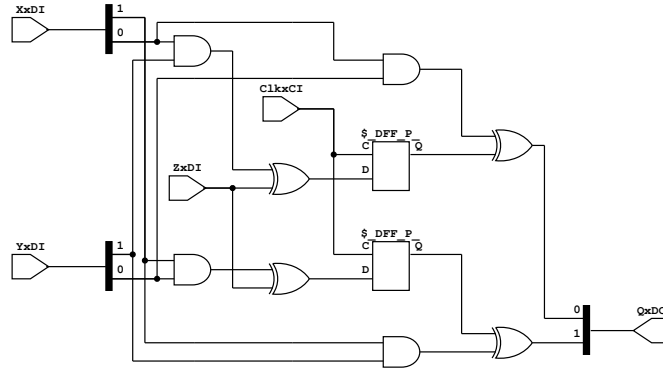


Figure 3.38: The DOM-1 multiplication circuit.

[7] repository), we show that the proposed exact tool can compete with heuristic methods as well.

3.5.1 Methodology

The overarching goal of this paper is to present a new methodology that facilitates the analysis of a circuit description, to provide a proof that it is strongly d -non interferent (d -SNI), i.e., given s outputs as long as $s \leq d$ implies a dependency with maximum i input shares, where i is the number of internal probes [8].

From a high level point of view (see Figure 3.40), the first step of the proposed methodology is a reading phase of a gate-level circuit description, which is annotated with sensitive variables, sensitive outputs, and their corresponding shares. The description is then "unfolded" to produce all the intermediate probes that can be derived. Moreover, an overall Walsh transform is computed for any combination of either outputs/probes (1). Then, the derived Walsh transform is compressed into a compact representation exploiting Algebraic Decision Diagrams (2). To perform the interference check, the latter is then multiplied by a *relation vector*, which has non-null values only in the regions where the Walsh transform must be zero (3, see also Figure 3.18). If the resulting value is not zero then it means that the function is not t -SNI; otherwise, we pass to the next output/probe combination. The following paragraphs show a detailed description of the above steps.

3.5.1.1 Reading and "unfolding" the circuit description

The tool reads-in a standard intermediate language (ILANG) format as produced by YOSYS tool [118]. Figure 3.39 shows part an example annotation for the Domain Oriented Masking *and* [67] protected at the first order (whose circuit is shown in

3.5.1 Methodology

```
# Generated by Yosys 0.7 ...
...
module \dom_and
  ## public \ClkxCI \RstxBI
  ## input \XxDI
  ## input \YxDI
  ## output \QxD0
  ## random \ZxDI
  ...
  wire width 2 input 3 \XxDI
  wire width 2 input 4 \YxDI
  wire width 1 input 5 \ZxDI
  wire width 2 output 6 \QxD0
  # (other wires and cells)
  ...

```

Figure 3.39: Annotated ILANG file

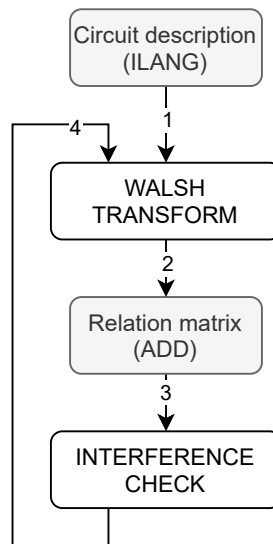


Figure 3.40: The methodology proposed in this paper.

Figure 3.38).

The description is extended with a `Maskverif` compliant set of annotations for identifying sensitive inputs (e.g., `XxDI`), outputs (e.g., `QxD0`) and additional random bit (`ZxDI`). Being an implementation protected at the second order, each sensitive value (e.g., `XxDI`) is encoded in two shares.

”Unfolding” the circuit means deriving the expression of all the possible intermediate nodes in the circuit. This is of course a potentially exponential operation whose time increases with the levels of the circuit. Practically, at least for the considered benchmarks, see Section 3.5.2, the complexity is still manageable. This part produces a C++ file which builds the BDDs for all the outputs/probes that have been found

(by exploiting the C++ bindings of the CUDD library*). The idea is that the corresponding manager will be able to build an internal representation exploiting common subexpressions, especially when these correspond to actual factors and co-factors of another function already parsed.

3.5.1.2 Computing the Walsh Spectrum and the corresponding relation matrix

The Walsh transform of each base output/probe is computed through the Fujita Walsh transform [61]. The algorithm works on the BDD representation of the function and returns an ADD whose variables are the bits associated with the spectral coordinates. In principle, one could use this transform to work on any combination of output/probes. However, we have found the performance of the algorithm suboptimal with respect to a simpler computation which exploits the known fact that the row of the correlation matrix associated with multiple base output/probes is proportional to the convolution of the source rows when these are represented in suitable associative container data types. While state of the art solutions are based on list of lists [89], in this paper we propose to adopt hash-based containers (in C++ parlance these are called *unordered maps*). Operations on such containers are $O(1)$ on average and allow fast insertion/update times of the result of the convolution. The data is then converted back in an ADD for further processing.

3.5.1.3 Interference check

The machinery associated with BDD/ADDs allows to quickly prove predicates over the data itself. In particular, one can express and solve existentially quantified predicates, over the convolution W computed above, and have the ADD manager work out the result. The interference check can be defined as a suitable predicate of this form:

$$\exists \alpha. T(\alpha, \rho) \wedge W(\alpha, \rho) \wedge (\rho = 0)$$

where α and ρ are the spectral coordinates of sensitive values and refresh values, $T(\alpha, \rho)$ is a predicate matrix which is equal to 1 only where the convolution W is expected to be 0 (essentially the white areas in Figure 3.18). If the predicate evaluates to true, then it means that the function is not t -SNI. If the predicate is false then it means that, for this particular combination of output/probes, no vulnerability has been found. However, the search must continue for combinations of up to d among

*Colorado University Decision Diagrams [110]

3.5.2 Experimental results

outputs and probes for determining whether the function is t -SNI. To speed up the search it has already been noted [7] that it is useful to start from combinations of the maximum size and evaluate simpler combinations if those are not found vulnerable; this is because there is a low probability that multiple output/probes mask out single output/probe vulnerabilities.

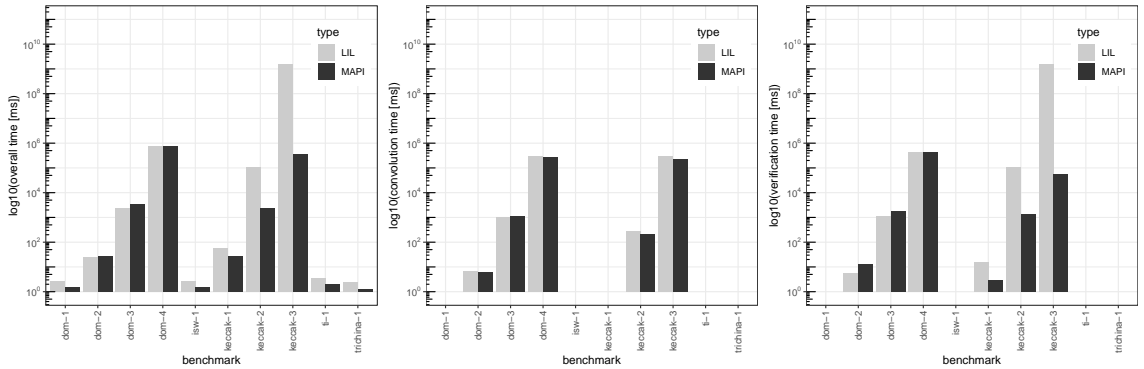


Figure 3.41: Comparison of overall (left), convolution (middle) and verification (right) times between the method proposed in [89] (**lil**) and the proposed method (**mapi**)

3.5.2 Experimental results

This experimental result section has a threefold goal: *i*) to compare the performance of the proposed methodology with the state of arts exact method in [89], *ii*) to compare alternative implementations of the proposed methodology with varying degree adoption of BDD/ADD, and *iii*) to show a comparison with other current state of the art approaches.

The experiments are based on the benchmarks from the **maskVerif** repository [7]. These benchmarks are a set of primitive cryptographic gadgets implemented to prevent probing attacks. In particular, for the first level of security, we test the Threshold Implementation algorithm (ti-1 in Tables 3.3 and 3.4) [95], Trichina (trichina-1)[116] and ISW multiplication (isw-1) [73]; DOM (dom-*) [67] is tested from the first to the fourth level, while the implementation of probing-protected Keccak algorithm (keccak-*) [68] from the first to the third. We run our experiments on a single core Intel Celeron N3150 at 1.601GHz. First, we compare the performance of our methodology with the implementation proposed in [89], where the authors exploited a lists of lists (**lil**) data structure to store the Walsh spectrum and compute both the convolution and the verification over such lists. Table 3.3, shows a comparison between **lil** and our method (*maps improved* or **mapi**). The first column refers to the tested security level, while the names of gadgets are listed in the second column. Third and

Table 3.3: Results of the comparison between our methodology and lists of lists implementation. Values in third and fourth columns are in seconds.

sec. lev.	gadget	lil	mapi	speed-up
1	ti-1	0.00367	0.00194	1.89
	trichina-1	0.00248	0.00129	1.93
	isw-1	0.00276	0.00157	1.76
	dom-1	0.00272	0.00145	1.87
	Keccak-1	0.05506	0.02633	2.09
2	dom-2	0.02478	0.02731	0.91
	Keccak-2	106.60330	2.39039	44.6
3	dom-3	2.38042	3.29725	0.72
	Keccak-3	1482378.91197	351.71293	4214.74
4	dom-4	756.00070	740.17401	1.02
median				1.88

fourth columns report the time taken for the implementation with **lil** and with **mapi** respectively. The last column shows the speed-up of **mapi**, computed as the ratio between the two previous columns. The overall execution time and the breakout of the convolution and verification operations is presented in Figure 3.41. Note that the y axis is logarithmic so the breakout is not meant to be additive as one can intuitively think. Numerically, we can note:

- On convolution, the methods are comparable with a slight advantage for **mapi**, given perhaps the faster average access time.
- On verification, the use of ADDs by **mapi** allows a significant speedup which benefits the overall execution time.
- On the overall median, **mapi** can provide a speedup of 1.88x with respect.
- Whenever the speedup is lesser than one (in only two over ten cases) the difference is less than 30%.
- For Keccak, which is a benchmark of greater complexity with respect the other ones, **mapi** shows a speedup of at least 3 orders of magnitude.

One could think that applying ADDs also for convolution would imply a better performance. To answer this question we evaluate our methodology with two variants, one in which both computation and verification is done only with hash maps (**map**) and the case in which both convolution and verification is done with ADDs (**fujita**,

3.5.2 Experimental results

Table 3.4: Evaluation of different implementation choices. Values from third to sixth columns are in seconds.

sec. lev.	gadget	lil	fujita	map	best method
1	ti-1	1.89	6.70	1.94	1.89
	trichina-1	1.93	10.83	1.96	1.93
	isw-1	1.76	9.08	1.79	1.76
	dom-1	1.87	9.74	1.84	1.84
	Keccak-1	2.09	1.37	2.10	1.37
2	dom-2	0.91	2.44	0.84	0.84
	Keccak-2	44.6	5.19	30.89	5.19
3	dom-3	0.72	1.75	0.57	0.57
	Keccak-3	4214.74	34.76	1629.05	34.76
4	dom-4	1.02	1.43	0.56	0.56
median		1.88	5.94	1.89	1.80

using the Fujita method [61]). Table 3.4 reports the speed-ups of our method (**mapi**) with respect to all the others (**lil**, **map**, **fujita**) while the absolute execution times are shown in Figure 3.42. Overall **mapi**'s mixing of hash maps and ADDs improves with respect to all other methods (median 1.8x), except for the DOM benchmark. We suppose that this behavior is due to Walsh matrices being very sparse and thus not requiring a significant effort in verification.

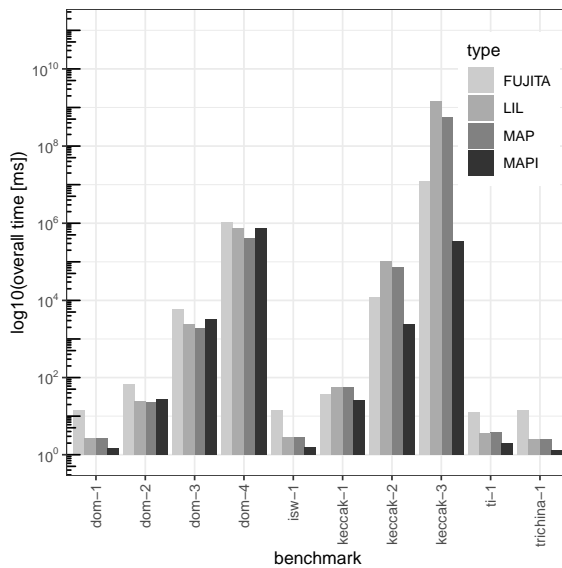


Figure 3.42: Comparison of overall computation times of the proposed method (**mapi**) and other implementations analysed in the experimental results.

Table 3.5: Comparison between **mapi** and the state of the art tools in [7], [26] and [75]. Values from third to sixth columns are in seconds.

sec. lev.	gadget	heuristic		exact	
		maskVerif	Bloem	SILVER	mapi
1	ti-1	0.01	≤ 1	–	0.0019
	trichina-1	0.01	≤ 1	–	0.0013
	isw-1	0.01	≤ 1	–	0.0016
	dom-1	0.01	≤ 1	0.0	0.0015
	Keccak-1	0.01	≤ 1	–	0.0263
2	dom-2	0.01	≤ 1	0.0	0.0273
	Keccak-2	0.2	$\leq 10^*$	–	2.3904
3	dom-3	0.04	≤ 4	3.7	3.2972
	Keccak-3	41	$\leq 240^*$	–	351.7129
4	dom-4	0.34	≤ 120	–	740.1740

We conclude by giving in Table 3.5 a comparison of **mapi** with other state of the art tools, and in particular **maskVerif** [7], the approximate technique proposed by Bloem et Al. in [26] (called Bloem in the Table) and SILVER [75]. Being exact, **mapi** implies obviously more computation time respect the first two heuristic methods, but not so much more, especially for Keccak-3. Instead, the comparison with SILVER is difficult, due to the different choice of benchmarks. In this case, for DOM algorithm, SILVER and **mapi** seem to need close processing time. Note that some results in Bloem column are marked by a *, because the benchmarks provided for their tool in [26] only concern the verification of one secret instead of all 5 secrets of elaborated by the gadget; also, their technique verifies probing security and not the strong non-interference.

3.6 Conclusion and further works

In this Chapter, we present our work in the context of the probing security. Two main cases have been inspected: the classic probing model, i.e., when the probes placed on the circuit allow to read only the values flowing inside the wires, and the robust probing model, i.e., when also glitches are considered.

In the first case, we investigate a formalization that is an alternative from those present in the state-of-the-art. It is based on the theory of Boolean functions, and is able to describe the multiple dependencies between outputs and inputs of a vectorial

function. This relation calculus is precise enough to prove and extend known compositional properties, without much semi-formal or verbal ratiocination. This reasoning has much room of improvement, for example considering other probing security and composition definitions, such as the t -PINI condition.

In the second case, we apply the relation calculus to the robust probing security, providing a spectral formalization also in the context of gadgets composition. Moreover, we derive formal conditions for t -probing security in the presence of glitches by further categorizing probes, to enable compositional reasoning of vulnerability profiles. Also in this case, we highlight that an improvement can be modelling t -PINI condition, as well as inquiring about the minimum number of randoms required to achieve robust t -strong non interference and/or investigating more efficient refresh layers/gadgets.

In the context of robust probing security, we also propose a low-latency multiplication gadget that is secure against probing attacks that exploit logic glitches in the circuit. This gadget is the first of its kind to present a 1-cycle input-to-output latency while belonging to the class of probing security by optimized gadgets. A possible extension of this work is find low-latency gadgets that allow higher probing security protection, with the number of randoms as low as possible.

Among the tools proposed to examine the probing security of gadgets, with or without glitches, we propose a new methodology that allows to exactly verify strong-non-interference properties. Our approach combines both hash maps and ADDs and provides, on a standard set of use cases, an interesting speed-up against other exact methods. Next steps can be comparing it with more other tools, such as the very new IronMask, and applying our tool to more complex gadgets, with higher security levels and by exploiting parallelization.

Part II

Protecting secrets during computations

CHAPTER 4

MULTIPLICATIVE COMPLEXITY, AUTOSYMMETRIC AND DIMENSION REDUCIBLE BOOLEAN FUNCTIONS

Nowadays, And Inverted Graph (AIG) is one of the most studied and exploited data structure in Logic Synthesis. An AIG is a directed acyclic graph of 2-input *and* nodes, with possibly inverted edges, that represents a Boolean function. The academic state of the art logic synthesis tool implementing the AIGs is called ABC [31]. Recently, the new logic networks representation of the *xor-and* Graphs (XAG) has been developed [69, 71, 112, 114], namely an AIG enriched with 2-input *xor* nodes, and then on the basis $\{and, xor, not\}$. The introduction of *xor* nodes in AIGs is mainly due to two different requirements: several proposed emerging technologies exploit *xor* gates [22, 69, 107, 113], and the growing relevance of cryptography-related applications has revived the interest in *xor* gates [44]. For example, in high-level cryptography protocols such as secure multi-party computation, processing *xor* gates is particularly convenient since their evaluation is possible without any communication cost (see Section 2.4.2.3).

In the context of logic synthesis for emerging technologies, the minimization of a XAGs mainly aims at reducing the number of *and/xor* nodes. On the contrary, in the cryptography-related applications, we are typically interested in reducing the number of *and* nodes, only. For example, for secure multi-party computation *and* nodes are the only nodes in a XAG with a communication cost. In this case, the minimization cost depends only on the number of *and* nodes, and then the main aim is the minimization of the number of *and* gates in a XAG [113, 112, 114].

In this work, the XAG model used is the one described in [114], where regular and complemented edges are used to connect the gates. Complemented edges indicate the inversion of the signals and replace inverters in the network (see Example 4.1.1).

In Section 4.1 we outline an overview on the state of the art about multiplicative complexity of circuits in XAG form, studying the particular cases of autosymmetric functions and D-reducible functions. In Section 4.2 we present some developments in the multiplicative complexity, which have been published in a paper of which the author of this thesis is co-author. Similarly, for paper on which Section 4.3 is focused on, which concerns some more investigations about the topic.

4.1 State of the art

At the beginning of this Section, we give two basic definitions.

Definition 4.1. The multiplicative complexity of a Boolean function is a measure defined as the minimum number of *and* gates (i.e., multiplications) that are sufficient to represent the function over the basis $\{and, xor, not\}$. More precisely, the *multiplicative complexity* $M(f)$ of a Boolean function f is the number of *and* gates, with fan-in 2, that are necessary and sufficient to implement f with a circuit over the basis $\{and, xor, not\}$.

Definition 4.2. The *multiplicative complexity* $M_C(f)$ of a circuit C implementing a Boolean function f over the basis $\{and, xor, not\}$ is the actual number of *and* gates in C .

Observe that the multiplicative complexity of a circuit for f only provides an upper bound for the multiplicative complexity of f , i.e., $M(f) \leq M_C(f)$.

Example 4.1.1. In Figure 4.1 there is an example of XAG. Note that the complemented edges are denoted by dashed lines. Since the number of *and* gates is 2, for this particular circuit implementation the $M_C(f)$ measure is equal to 2.

The multiplicative complexity measure plays a crucial role in cryptography applications, for various reasons. First, the minimization of the number of *and* gates is important for high-level cryptography protocols such as zero-knowledge protocols and secure multi-party computation, where processing *and* gates is more expensive than processing *xor* gates [1]. Moreover, the multiplicative complexity is an indicator of the degree of vulnerability of the circuits, as a small number of *and* gates in an XAG corresponds to a high vulnerability to algebraic attacks [65, 111]. Unfortunately, determining the exact value of the multiplicative complexity of a function f is

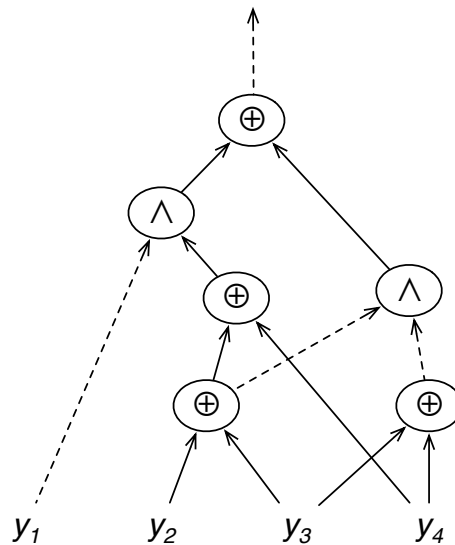


Figure 4.1: XAG representation of the 4-input Boolean function.

a computationally intractable problem [35]. Thus, the minimization of the number of *and* gates in any circuit implementation over the basis $\{and, xor, not\}$ becomes very important to assess the actual multiplicative complexity of the function.

4.1.1 Autosymmetric functions

Generally, the regularities of Boolean functions are exploited with the purpose to derive more compact circuits in shorter synthesis time. When a Boolean function has to be synthesized in a circuit, it is not always easy to find if exists a regularity that can be exploited for a faster and more performing synthesis. In the literature, some works have focused on structural regularities of Boolean functions based on the notion of affine spaces and easily expressed using *xor* gates. In this context, in order to decrease the multiplicative complexity of a XAG, it is possible to exploit the regularity called autosymmetry [18, 20, 81, 30].

A Boolean function f over n variables is k - **autosymmetric** if it can be projected onto a smaller function f_k that depends on $n-k$ variables. The regularity of a Boolean function f is then measured computing its autosymmetry degree k , with $0 \leq k \leq n$, where $k = 0$ means no regularity. For $k \geq 1$ the Boolean function f is said to be autosymmetric, and a new function f_k depending on $n-k$ variables only, called the restriction of f , is identified in polynomial time. Moreover, an expression for f can be simply built from the restriction f_k , indeed $f(x_1, x_2, \dots, x_n) = f_k(y_1, y_2, \dots, y_{n-k})$, where f_k is a Boolean function on $n-k$ variables $y_1 = \bigoplus(X_1)$, $y_2 = \bigoplus(X_2)$, \dots , $y_{n-k} = \bigoplus(X_{n-k})$ and each $\bigoplus(X_i)$ is a *xor* whose input is a set of variables X_i with $X_i \subseteq \{x_1, x_2, \dots, x_n\}$. Note that $\bigoplus(X_i)$ can be a single variable, i.e., $X_i = \{x_j\}$ and

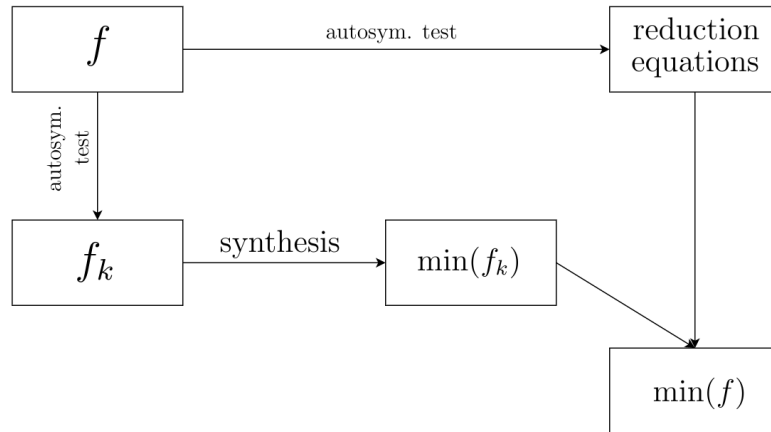


Figure 4.2: Synthesis process, when autosymmetry test is applied to an autosymmetric function f .

$$\bigoplus(X_i) = x_j .$$

The *autosymmetry test* consists of finding the value of k , the restriction f_k , and each single *xor* with its input variables X_i , i.e., the reduction equations. Note that a degenerate function, i.e., a function that does not depend on all the variables in the current Boolean space, is autosymmetric. The restriction f_k is “equivalent” to, but smaller than f , and has $\frac{|S(f)|}{2^k}$ minterms only, where $S(f)$ denotes the support of f , and thus $|S(f)|$ is the number of minterms of f . The synthesis of f can be reduced to the synthesis of its restriction f_k , which can be identified in polynomial time. As the new $n - k$ variables are *xor* combinations of some of the original ones, the reconstruction of f from f_k can be obtained with an additional logic level of *xor* gates, whose inputs are the original variables, and the outputs are the new $n - k$ variables and their complements given as inputs to a circuit for f_k . The restricted function f_k can be synthesized in any framework of logic minimization (for example, in XAG form).

The synthesis flow, with autosymmetric test, is depicted in Figure 4.2. The aim is to find the XAG for the Boolean function f with the minimum number of *and* gates, i.e., $\min(f)$. Directly synthesizing function f can be computationally hard, because it is an exponential time process. Instead, if f is autosymmetric, it is possible to reduce f to the restriction f_k and decrease the synthesis time. Note that the autosymmetric test can be computed in polynomial time, while the last step calculating $\min(f)$ from $\min(f_k)$ and the reduction equations is a linear time process.

Example 4.1.2. Consider the Boolean function f , the Karnaugh map of which is depicted in Figure 4.3.

4.1.1 Autosymmetric functions

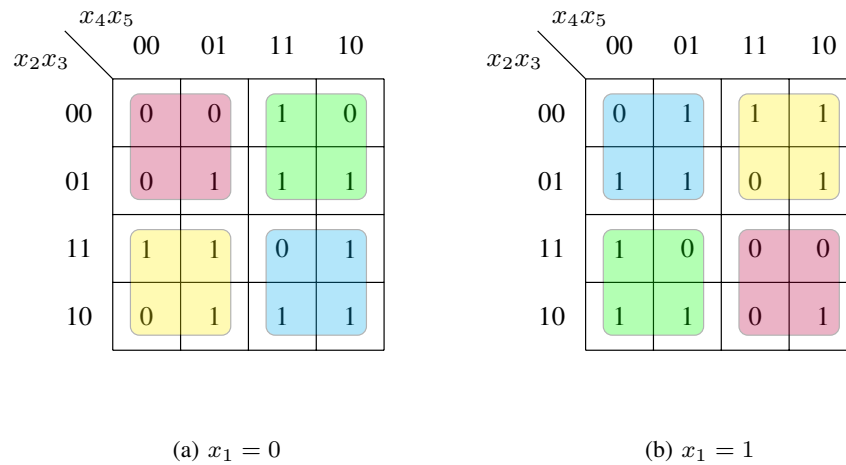


Figure 4.3: Karnaugh map of a Boolean function that depends on 5 Boolean variables x_1, x_2, x_3, x_4, x_5 .

The regularity of the function is highlighted by the colors in the figure. The autosymmetry degree of f is 1 (i.e., $k = 1$) and the reduction equations are $y_1 = x_1 \oplus x_2$, $y_2 = x_1 \oplus x_3$, $y_3 = x_1 \oplus x_4$, $y_4 = x_1 \oplus x_5$. Thus, the restriction f_1 depends on 4 variables and it is depicted in Figure 4.4.

Note that each point of the restriction corresponds to two points of the original function, as indicated by the colors in the maps. For example, the point $(0, 0, 0, 0)$ in the Karnaugh map of Figure 4.4 corresponds to the two points $(0, 0, 0, 0, 0)$ and $(1, 1, 1, 1, 1)$ in the Karnaugh map of Figure 4.3. This is due to the reduction equations, because considering the point $(x_1, x_2, x_3, x_4, x_5) = (0, 0, 0, 0, 0)$ and through the reduction equations $(y_1, y_2, y_3, y_4) = (x_1 \oplus x_2, x_1 \oplus x_3, x_1 \oplus x_4, x_1 \oplus x_5) = (0, 0, 0, 0)$. Exactly the same holds for the point $(x_1, x_2, x_3, x_4, x_5) = (1, 1, 1, 1, 1)$. It is easy to verify that it is possible to perform a similar computation for any couple of corresponding points depicted in Figure 4.3, obtaining the Karnaugh map in Figure 4.4.

Autosymmetric functions are just an exponentially smallest subset of the total number of Boolean functions. Indeed, while the number N_B of the Boolean functions of n variables is $N_B = 2^{2^n}$, the number of autosymmetric ones is $N_A = (2^n - 1)2^{2^{n-1}}$ [20].

The interest on autosymmetric functions is motivated mostly by their compact representation in terms of number of *and* gates, which consists of a *xor* layer that is the input to a XAG for the restriction. Moreover, in the classical sets of benchmark functions, the frequency of autosymmetric functions is relevant. Indeed, for example, in ESPRESSO benchmark suite [121] about 24% of the functions have at least one truly (i.e. non degenerate) autosymmetric outputs.

Figure 4.4: Karnaugh map of the restriction of the Boolean function in Figure 4.3, which depends on 4 Boolean variables y_1, y_2, y_3, y_4 .

		y_3y_4			
		00	01	11	10
y_1y_2	00	0	0	1	0
	01	0	1	1	1
	11	1	1	0	1
	10	0	1	1	1

4.1.2 D-reducible functions

In this Section, we present the Dimension Reducible Boolean functions (DRed functions) and some of their major properties.

D-reducible functions are Boolean functions whose minterms are all contained in an affine space A strictly smaller than the whole Boolean space $\{0, 1\}^n$ (see Section 2.1.1 for definitions). More precisely, a Boolean function $f : \{0, 1\}^n \rightarrow \{0, 1\}$ is D-reducible if $f \subseteq A$, where $A \subset \{0, 1\}^n$ is an affine space of dimension strictly smaller than n .

The minimal affine space A containing a D-reducible function f is unique, and it is called the associated affine space of f . The function f can be represented in the following way: $f = \chi_A \cdot f_A$, where $f_A \subseteq \{0, 1\}^{\dim A}$ is the projection of f onto A and χ_A is the characteristic function of A .

Moreover, as shown in [46], an affine space can be represented by a simple expression, called *pseudoproduct*, consisting in an *and* of *xors* of literals. In particular, an affine space of dimension \dim_A can be represented by a pseudoproduct containing $(n - \dim_A)$ *xor* factors.

Example 4.1.3. Consider the function $f : \{0, 1\}^4 \rightarrow \{0, 1\}$ with on-set $\{0001, 0110, 1101\}$ represented in the Karnaugh map on the left side of Figure 4.5.

The smallest affine space containing the on-set of f , depicted with dotted circles on the map, is $A = \{0001, 0110, 1010, 1101\}$. This affine space has dimension $\dim_A = 2$ and can be represented by the pseudoproduct $(x_1 \oplus x_2 \oplus \bar{x}_3)(x_1 \oplus x_2 \oplus x_4)$ that contains exactly $(n - \dim_A) = 2$ *xor* factors. If we project f onto the smaller space A , we obtain the function $f_A = \{00, 01, 11\}$, represented in the Karnaugh map on the right side of the figure.

4.1.3 Mockturtle tool

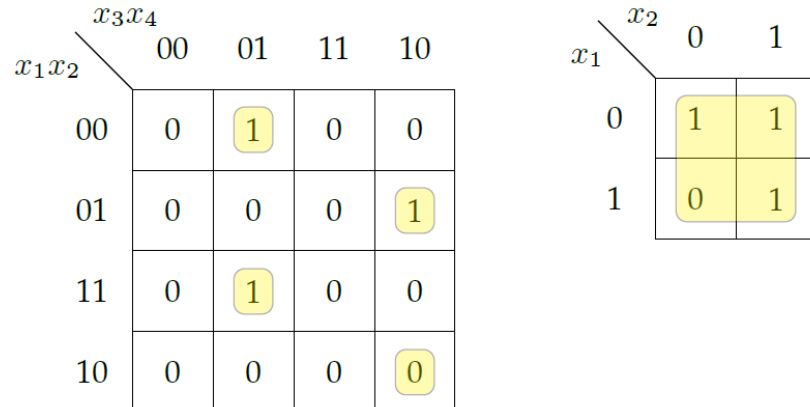


Figure 4.5: Karnaugh maps of a D-reducible function f and its corresponding projection f_A .

The D-reducibility of a function f can be exploited in the minimization process. The projection f_A is minimized instead of f . This approach requires two steps: (i) deriving the affine space A and the projection f_A ; (ii) minimizing f_A in any logic framework. The D-reducibility test, which establishes whether a function f is D-reducible, and the computation of A have polynomial complexity [15].

4.1.3 Mockturtle tool

Mockturtle is a C++ library and part of the EPFL logic synthesis libraries [108]. This library provides generic logic synthesis algorithms and logic network data structures. The main philosophy of Mockturtle is that all algorithms are generic in the sense that they are independent from the implementation of the logic network data structure. In order to achieve this, Mockturtle makes use of concept-based design using some modern C++-17 language features.

It can be integrated with other EPFL libraries, expanding its potential. For example, it allows to read various format files, when integrated with the library Lorina, and it can be augmented with SAT using the exact synthesis library Percy. For our work, we are interested in Mockturtle because it is a state-of-the-art tool that can be used to support and build algorithms for the optimization of the multiplicative complexity [112, 114].

The Mockturtle is based on a principle of 4 layers that depend on each other in a linear order. The bottom layer is provided by the Network interface API. It defines naming conventions for types and methods in classes that implement network interfaces, some of which are mandatory while others are optional. The network interface

API does not provide any implementations for a network though. Algorithms, the second layer, are implemented in terms of generic functions that takes as input an instance of some hypothetical network type and require that type to implement all mandatory and some optional interfaces. The algorithms do however make no assumption on the internal implementation of the input network. For instance, they make no assumption on how gates of the network are internally represented. The third layer consists of actual network implementations for some network types that implement the network interface API, e.g., And-inverter graphs, Majority-inverter graphs, or k-LUT networks. Algorithms from the second layer can be called on instances of these network types, if they implement the required interfaces. Static compile time assertions are guaranteeing that compilation succeeds only for those network implementations that do provide all required types and methods. Finally, to improve the performance, some algorithmic details may be specialized for some network types based on their internal implementation. This can be done for each network individually, without affecting the generic algorithm implementation nor the implementation of other network types.

4.2 Multiplicative Complexity of Autosymmetric Functions: Theory and Applications to Security

In this Section we investigate some properties that Boolean functions assume when their multiplicative complexity is explored [13]. In particular, we study a specific structure regularity of Boolean functions, called autosymmetry, and exploit it to decrease the number of *ands* in *xor*-and Graphs (XAGs), i.e., Boolean networks composed by *ands*, *xors*, and inverters. The interest in autosymmetric functions is motivated by the fact that a considerable amount of standard Boolean functions of practical interest presents this regularity. Indeed, about 24% of the functions in the classical ESPRESSO benchmark suite have at least one autosymmetric output.

4.2.1 Multiplicative Complexity of Autosymmetric Functions

In this Section we study the relationships between the multiplicative complexity of an autosymmetric function and the multiplicative complexity of its restriction.

First of all, observe that a XAG representation of a k -autosymmetric function f can be easily obtained composing a XAG for the restriction f_k with an additional layer of *xor* gates implementing the reduction equations. The inputs to the new

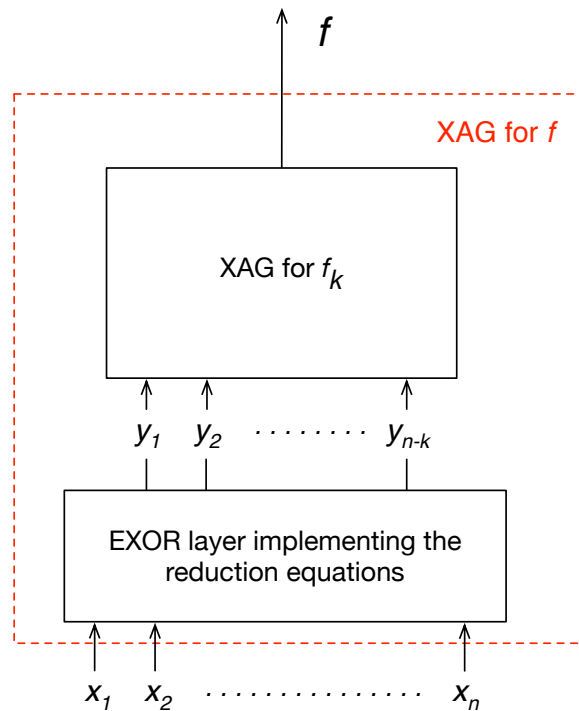


Figure 4.6: A XAG for an autosymmetric function f obtained adding a xor level implementing the reduction equations to a XAG for the restriction f_k .

layer are the original variables x_1, x_2, \dots, x_n and the outputs are the new variables y_1, y_2, \dots, y_{n-k} , that become the inputs to the XAG for f_k , as shown in Figure 4.6. Since the new layer contains only xor gates, we immediately conclude that $M(f) \leq M(f_k)$, as formally stated in the following lemma.

Lemma 4.2.1. *The multiplicative complexity of an autosymmetric function f is less or equal to the multiplicative complexity of its restriction f_k .*

Proof. First recall that the multiplicative complexity of a XAG implementation for a function f provides an upper bound for the multiplicative complexity of the function itself. Thus, the thesis follows since we can construct a XAG for f with exactly $M(f_k)$ *and* nodes. This can be done adding to a XAG for f_k , containing a minimum number $M(f_k)$ of *and* nodes, a layer consisting only of xor nodes, as shown in Figure 4.6. \square

Actually, a much stronger result holds: f and f_k have exactly the same multiplicative complexity. To prove this result, we need to recall some properties of autosymmetric functions and of their restrictions. As shown in [17, 21], any k -autosymmetric function f is associated to a k -dimensional vector space L_f , defined as the set of all minterms w s.t. $f(v) = f(v \oplus w)$ for all $v \in \{0, 1\}^n$. The k variables that are truly

independent onto L_f , i.e., the variables that assume all the possible combinations of $\{0, 1\}$ values in the minterms in L_f , are called *canonical variables* and are used to construct the restriction f_k . In fact, f_k corresponds to the projection of f onto the subspace $\{0, 1\}^{n-k}$ where all the canonical variables assume value 0 (see [17, 21] for more details).

Consider for instance the 1-autosymmetric benchmark *rd53* (second output) that correspond to the case discussed in Example 4.1.2. Its associated vector space is the 1-dimensional space $L_f = \{00000, 11111\}$, whose canonical variable is x_1 (all other variables must be equal to x_1 on L_f), and the restriction corresponds to the projection of the function onto the space where $x_1 = 0$, as it can be noted from Figures 4.3 and 4.4.

Exploiting this characterization for the restriction of an autosymmetric function, we can prove the following theorem.

Theorem 4.2.2. *Let f be a k -autosymmetric function and let f_k be its restriction. Then,*

$$M(f) = M(f_k).$$

Proof. We have proved in Lemma 4.2.1 that $M(f) \leq M(f_k)$. Thus, it is enough to show that $M(f) \geq M(f_k)$. By contradiction suppose that the multiplicative complexity of the whole function f is strictly less than the multiplicative complexity of its restriction f_k , i.e., $M(f) < M(f_k)$. This assumption means that any XAG for f_k requires strictly more than $M(f)$ *and* nodes, i.e., $M_X(f_k) > M(f)$, where $M_X(f_k)$ denotes the multiplicative complexity of a XAG for f_k . Since the restriction f_k corresponds to the projection of f onto the subspace $\{0, 1\}^{n-k}$ where all the canonical variables of f have value 0, we can derive a XAG representation for f_k starting from a XAG for f and substituting all canonical input variables with the constant value 0. Note that the constant value 0 can be obtained computing the *xor* of any non-canonical variable with itself. Such a transformation can only decrease the number of *ands* in the original XAG, as all *and* nodes that receive in input the constant value 0 can be removed from the circuit and substituted with the value 0. Therefore, if we start from a XAG implementation of f with the minimum number $M_X(f) = M(f)$ of *and* nodes, we can derive a XAG for f_k with $M_X(f_k) \leq M(f)$ *ands*, in contradiction with the initial assumption $M(f) < M(f_k)$. \square

Since the restriction f_k is a smaller function, depending on less variables, computing a XAG representation and minimizing the number of *ands* become easier problems, whose solutions allow to better assess the actual multiplicative complexity of the original function f . For instance, for our running example concerning the benchmark *rd53* (second output), we can derive the XAG representation shown in

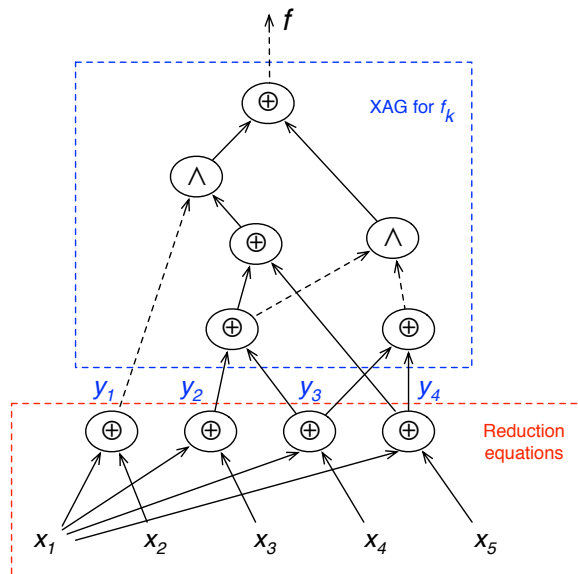


Figure 4.7: XAG representation for the benchmark *rd53* (second output), derived exploiting the autosymmetry of the function.

Figure 4.7 simply adds four *xor* nodes to the XAG for f_k of Figure 4.1 (that is, the XAG representation of its reduction), that contains 4 *xors* and only 2 *ands*. Notice that the direct XAG minimization of *rd53* performed using the software from [112] would produce a bigger circuit, with 12 *ands* and 23 *xors*.

4.2.2 Experimental results

The approach presented above has been applied to the ESPRESSO and LGSynth'89 benchmark suite [120], running on a Pentium INTEL(R) CORE(TM) i5-5200U 2.20 GHz processor with 4.00 GB RAM, on a virtual machine running OS Ubuntu 64-bit. The experiments consider the subset of single outputs that are autosymmetric. The main aim of the experiments is to compare the synthesized XAG computed starting from an autosymmetric function f and the synthesized XAG computed starting from the corresponding restriction f_k , after the autosymmetry test. Recall that the autosymmetry test computes the autosymmetry degree k of a Boolean function and outputs: 1) the reduction equations, which form the *xor* layer, and 2) the corresponding restriction f_k . We performed the autosymmetry test described in [17, 21] considering the on-set of the benchmarks. The functions f and f_k are synthesized in XAG form using the heuristic approach proposed in [112] and described at the end of Section 4.1. We then compare the number of *and* nodes of the XAGs for f and f_k in order to understand how the autosymmetry test can enable the XAG minimization

Table 4.1: Experimental comparison of autosymmetric benchmarks, considering a XAG after the autosymmetry test and the standard XAG computed without the autosymmetry test.

Benchmark	in	k	standard XAG		XAG with autosym. test		gain
			$M_X(f)$	time (s)	$M_X(f_k)$	time (s)	
add6(0)	12	11	3	0.01	0	0.01	100%
bcc(32)	26	11	26	17.15	21	19.45	19%
bcc(33)	26	11	61	56.00	54	85.34	11%
exep(1)	30	18	17	7.33	15	9.36	12%
in5(3)	24	5	31	14.13	27	12.46	13%
in7(1)	26	10	21	15.68	15	11.48	29%
in7(5)	26	5	34	24.94	30	35.27	12%
mainpla(27)	27	3	180	147.01	147	130.00	18%
mish(4)	94	91	2	0.01	2	0.01	0%
opa(24)	17	11	9	2.15	5	1.98	44%
opa(25)	17	2	39	26.51	31	22.84	21%
pdc(13)	16	8	108	8.67	7	2.96	94%
pdc(26)	16	2	25	22.57	12	6.16	52%
t1(19)	21	18	5	0.18	2	0.04	60%
t2(6)	17	4	17	12.49	14	11.6	18%
t2(9)	17	10	21	14.87	18	10.35	14%
vg2(6)	25	11	17	12.15	15	8.37	12%
x2dn(33)	82	79	2	0.01	2	0.01	0%
x6dn(0)	39	11	82	62.37	66	54.57	20%
x6dn(4)	39	10	93	74.83	82	77.12	12%
x7dn(1)	66	51	20	10.64	20	9.26	0%
xparc(0)	41	17	105	84.82	93	75.19	11%

4.2.2 Experimental results

Table 4.2: Summary of the experimental evaluation, considering the number of *ands* in the XAGs for autosymmetric functions and non-degenerate autosymmetric functions.

	$M_X(f_k) < M_X(f)$	$M_X(f_k) = M_X(f)$	$M_X(f_k) > M_X(f)$
autosymmetric functions	11.34%	68.2%	20.46%
non-degenerate autosymmetric functions	74.2%	19.35%	6.45%

of autosymmetric functions.

For the sake of brevity, we report in Table 4.1 only a significant subset of the results. The first column reports the name of the function considered (benchmark function and output number). The following one provides its input size. Next column refers to the autosymmetry degree (i.e., k) of the function. The following two pairs of columns report the multiplicative complexity of the XAG (M_X) after applying the heuristic in [112] and the time in seconds required to obtain it, for the entire function f (first couple) and for the corresponding restriction f_k (second couple). Finally, the last column reports the gain in applying the autosymmetry test before XAG synthesis.

Table 4.2 shows a summary of the overall experimental results. We first consider the set of all autosymmetric functions (degenerate* and non-degenerate), we then study the truly autosymmetric (i.e., the non-degenerate) ones. In Table 4.2, we denote with $M_X(f_k) < (=, >, \text{resp.}) M_X(f)$ the number of benchmarks where the number of *ands* of the XAG for f_k is less than (equal to, greater than, resp.) the number of *ands* of the XAG for f . We notice that the XAG minimization algorithm proposed in [112] is sensible to degenerate functions as shown in the first row of Table 4.1, where the number of benchmarks where f_k and f have the same number of *ands* is the majority (i.e., about 68%). Nevertheless, if we concentrate on non-degenerate autosymmetric functions (i.e., second row of the table) we notice that the number of benchmarks where $M_X(f_k) < M_X(f)$ is about 74%. Moreover, in this set the average gain is about 61%, while the overall gain in the entire set of autosymmetric functions (in the case $M_X(f_k) < M_X(f)$) is about 31%. Finally, the results on computational times are not very interesting, since the two compared approaches have similar synthesis times.

From these experiments, we can conclude that, when a function is truly autosymmetric (i.e., non-degenerate), we can obtain better results computing the XAG on the restriction f_k instead of computing the XAG directly on the function f .

*Recall that degenerate functions are, by definition, autosymmetric.

4.3 Multiplicative Complexity of Regular Functions

This work [14] is an extended version of the conference paper presented in Section 4.2, with an added investigation of the properties of the *D-reducible* functions in the context of the multiplicative complexity. Moreover, all the experimental results reached in here are computed making use of a the newer mockturtle’s version [114].

With the new tool, for tested *autosymmetric functions*, we are able to get a better estimate of the multiplicative complexity in about 52% of the studied cases, with an average reduction of the number of *ands* of about 44%. The experiments for *D-reducible functions* show that the XAG minimization can benefit from the D-reducible decomposition of the function in about the 43% of the D-reducible benchmarks, with an average reduction of the number of *ands* of about 35%. Moreover, the computational time for XAG synthesis is reduced in average by the 24%. Finally, for functions that are both autosymmetric and D-reducible, we get a better estimate of the multiplicative complexity in about 90% of the cases, with an average reduction of the number of *ands* of about 24%.

4.3.1 Multiplicative Complexity of D-reducible Functions

In this Section we focus on the class of D-reducible functions with the aim of verifying if the decomposition that characterizes these functions can be exploited to estimate their multiplicative complexity, in analogy to what we have seen for autosymmetric functions.

Recall, from Section 4.1, that a D-reducible function f , with associated affine space A , can be decomposed in the form

$$f = \chi_A \cdot f_A$$

where χ_A is the characteristic function of A and $f_A \subseteq \{0, 1\}^{\dim A}$ is the projection of f onto A . Thus, an XAG representation for f can be derived combining, via an *and* gate, an XAG representation for the affine space A with an XAG representation for f_A (as shown in Figure 4.8).

This decomposition immediately allows to upper bound the multiplicative complexity of f with the sum of the multiplicative complexity of χ_A and of f_A . More precisely we have:

$$M(f) \leq M(\chi_A) + M(f_A) + 1, \tag{4.1}$$

where we also account for the *and* gate on top of the overall XAG for f .

The advantages of this approach should derive from the fact that f_A is a function that depends on fewer variables, so we can reasonably expect its XAG representation to be more compact, and also easier to derive from a computational point of view.

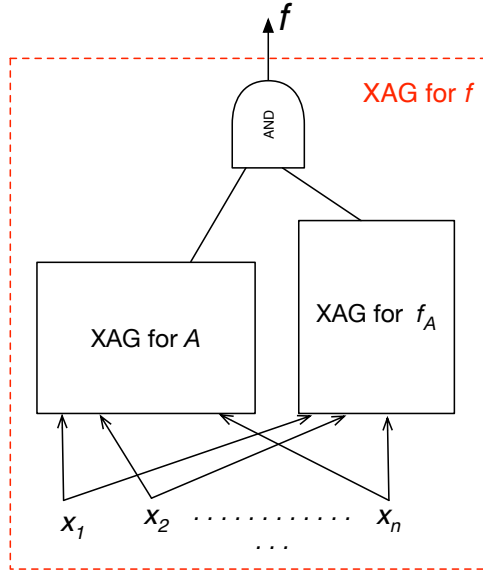


Figure 4.8: An XAG representation for a D-reducible function f obtained combining an XAG for the affine space A with a XAG for the projection f_A . Notice that only $\dim A$ of the n input variables are actual inputs for f_A .

Moreover, χ_A is not just any function, but the characteristic function of an affine space, thus we can take advantage of its structural properties to derive an XAG possibly optimal in number of *and* gates. While the first aspect can only be verified and evaluated experimentally, the XAG representation of affine spaces can be fully investigated from a theoretical point of view.

To this aim, we first mention a result concerning the relation between the multiplicative complexity of a function and its *algebraic degree*. Recall that any function f depending on n binary variables can be represented as a multilinear polynomial over \mathbb{F}_2 , i.e., a *xor* of conjunctions of literals.

This representation, which is unique, can be obtained taking the modulo 2 sum of all the minterms in the on-set of the function, each represented as a conjunction of literals (see [117] for more details). Literals corresponding to negated variables are replaced with the modulo-2 sum of the variable and the constant 1, e.g., $\bar{x}_i = (1 \oplus x_i)$.

Definition 4.3.1. The *algebraic degree* $\deg(f)$ of a Boolean function f is the degree of the unique multilinear polynomial that represents f over \mathbb{F}_2 .

Observe that $\deg(f)$ corresponds to the number of variables in the longest products of this polynomial.

Consider for instance the function $f : \{0, 1\}^4 \rightarrow \{0, 1\}$ with on-set $\{0001, 0110, 1101\}$

represented in the Karnaugh map on the left side of Figure 4.5. The polynomial representation of f over \mathbb{F}_2 is given by the expression $f = x_4 \oplus x_1x_4 \oplus x_2x_3 \oplus x_2x_4 \oplus x_3x_4 \oplus x_1x_2x_3 \oplus x_1x_3x_4 \oplus x_1x_2x_3x_4$. Thus, f has algebraic degree $\deg(f) = 4$.

The algebraic degree turns out to be related to several complexity measures, and in particular to the multiplicative complexity [29, 102].

Lemma 4.3.2. ([102]) *Any circuit computing a Boolean function f over the basis $\{and, xor, not\}$ contains at least $\deg(f) - 1$ and gates with fan-in 2.*

Corollary 4.3.3. ([102]) *If a function f has algebraic degree $\deg(f)$, then its multiplicative complexity is at least $\deg(f) - 1$.*

Let us now focus on the multiplicative complexity of affine spaces. Consider an affine subspace A of $\{0, 1\}^n$. As discussed in [16, 45], we can partition the set of binary variables $\{x_1, x_2, \dots, x_n\}$ into two subsets: the subset of the *canonical variables* and the subset of the *non-canonical variables*. The canonical variables are the truly independent variables in the space A , in the sense that they can assume all possible combinations of 0-1 values, and their number is exactly $\dim A$. On the contrary, the remaining $n - \dim A$ non-canonical variables are not independent of A because they can be defined as linear combinations (i.e., *xors*) of the canonical ones. This fact is clearly expressed by the characteristic function χ_A of the affine space A , which can always be represented by a special pseudoproduct containing exactly $(n - \dim A)$ *xor* factors such that

- each non-canonical variable appears in exactly one *xor* factor,
- each *xor* factor is composed by one non-canonical variable and possibly some canonical ones.

This particular pseudoproduct representation is called the *canonical (CEX) expression* of A .

Consider, for example, the function f represented in Figure 4.5, and its affine space $A = \{0001, 0110, 1010, 1101\}$. We can observe that the first two variables, x_1 and x_2 assume on A all possible combinations of values, i.e., 00, 01, 10, and 11. On the contrary, x_3 and x_4 can be defined on A in terms of x_1 and x_2 as follows:

$$\begin{aligned} x_3 &= x_1 \oplus x_2 \\ \bar{x}_4 &= x_1 \oplus x_2. \end{aligned}$$

Thus, the two canonical variables of A are x_1 and x_2 , while the non-canonical ones are x_3 and x_4 . The CEX expression of this affine space is given by the pseudoproduct $(x_1 \oplus x_2 \oplus \bar{x}_3)(x_1 \oplus x_2 \oplus x_4)$, that encodes the two linear combinations defining x_3 and

4.3.1 Multiplicative Complexity of D-reducible Functions

x_4 on A . Indeed, this pseudoproduct states that on A the two factors $(x_1 \oplus x_2 \oplus \bar{x}_3)$ and $(x_1 \oplus x_2 \oplus x_4)$ must be equal to 1, i.e., $x_1 \oplus x_2 \oplus \bar{x}_3 = 1$ and $x_1 \oplus x_2 \oplus x_4 = 1$, from which we can immediately derive the two equalities (1) and (2). Note that each non-canonical variable occurs in one and only one *xor* factor.

We now show how the CEX expression allows to precisely characterize the multiplicative complexity of any affine space.

Theorem 4.3.4. *The multiplicative complexity of the characteristic function χ_A of an affine subspace $A \subseteq \{0, 1\}^n$ is exactly $M(\chi_A) = n - \dim A - 1$.*

Proof. We first prove that $M(\chi_A) \geq n - \dim A - 1$. To this aim, let us consider a CEX representation for A . This expression is composed by an *and* of $n - \dim A$ *xor* factor. Each *xor* factor contains a different non-canonical variable. Thus, the polynomial representation of χ_A , that can be immediately derived from the CEX expression, has degree $\deg(\chi_A) = n - \dim A$ since it certainly contains the term corresponding to the product of all the non-canonical variables. Thus, Corollary 4.3.3 immediately implies that $M(\chi_A) \geq n - \dim A - 1$. Now, observe that we can immediately derive an XAG representation from the CEX expression: we use *xor* gates for each *xor* factor, and then exactly $n - \dim A - 1$ *and* gates to compute the product of all factors. Thus, we have $M(\chi_A) \leq n - \dim A - 1$, and the thesis immediately follows. \square

Thus, in the overall XAG representation for D-reducible functions proposed in Figure 4.8, we can always represent the affine space A with the XAG derived from its CEX expression, which is optimal in the number of *and* gates. We therefore derive the following upper bound for the multiplicative complexity of any D-reducible function:

Corollary 4.3.5. *Let $f : \{0, 1\}^n \rightarrow \{0, 1\}$ be a D-reducible function with affine space A , and let f_A be its projection onto A . Then*

$$M(f) \leq n - \dim A + M(f_A).$$

Proof. Follows immediately from Equation (4.1) since, by Theorem 4.3.4, we have $M(\chi_A) = n - \dim A - 1$. \square

Given a D-reducible function f and an XAG representation for its projection f_A , with $M_X(f_A)$ *and* gates, we can then exploit the decomposition of Figure 4.8 to derive the estimate

$$M_{dec}(f) = M(\chi_A) + M_X(f_A) + 1 = n - \dim A + M_X(f_A) \quad (4.2)$$

for the multiplicative complexity of f .

4.3.2 Multiplicative complexity of D-reducible autosymmetric functions

In this section we provide an analysis of the multiplicative complexity of Boolean functions that are both D-reducible and autosymmetric. First of all, we prove that if a function f is autosymmetric and D-reducible with associated affine space A , then its projection onto A is also autosymmetric.

Theorem 4.3.6. *Let f be a k -autosymmetric Boolean function depending on n binary variables. If f is D-reducible with associated affine space A , then the projection f_A of f onto A is k -autosymmetric.*

Proof. Recall that any k -autosymmetric function f is associated to a k -dimensional vector space L_f , defined as the set of all minterms α s.t. $f(x) = f(x \oplus \alpha)$ for all $x \in \{0, 1\}^n$. We prove that f_A is k -autosymmetric by showing that for any $\alpha \in L_f$ it holds that $f_A(y \oplus \alpha_A) = f_A(y)$ for all $y \in \{0, 1\}^{\dim A}$, where α_A denote the projections of α onto A , i.e., the minterm obtained from α by keeping only the literals corresponding to the canonical variables of A .

First of all, we observe that the set L_f is a subspace of the linear vector space V associated to A . Indeed, let $\alpha \in L_f$, and let x be any on-set minterm of f . Then, $f(x \oplus \alpha) = f(x) = 1$, and therefore both x and $x \oplus \alpha \in A$. This in turns implies that $\alpha \in (x \oplus A)$, i.e., $\alpha \in V$, since $x \oplus A = V$ for any $x \in A$ (we refer the reader to [45] for more details on affine spaces and their properties).

Since $L_f \subseteq V$, we have that for all $x \in \{0, 1\}^n$ and for all $\alpha \in L_f$, $x \in A$ if and only if $x \oplus \alpha \in A$. Indeed,

$$\begin{aligned} x \in A &\Leftrightarrow x = a \oplus v, \text{ for some } a \in A \text{ and some } v \in V \\ &\Leftrightarrow x \oplus \alpha = a \oplus v \oplus \alpha \\ &\Leftrightarrow x \oplus \alpha \in A \end{aligned}$$

since $v \oplus \alpha \in V$.

Now, let α be any vector in L_f . Then, for all $x \in \{0, 1\}^n$, we have $f(x \oplus \alpha) = f(x)$, i.e.,

$$\chi_A(x \oplus \alpha) f_A(x_A \oplus \alpha_A) = \chi_A(x) f_A(x_A),$$

where x_A and α_A denote the projections of x and α onto A . If we consider only the vectors $x \in A$, the fact that $x \oplus \alpha$ is also in A implies $\chi_A(x) = \chi_A(x \oplus \alpha) = 1$. Thus, we have

$$f_A(x_A \oplus \alpha_A) = f_A(x_A),$$

which in turns implies that $\alpha_A \in L_{f_A}$. □

4.3.3 Experimental results

This result is very interesting from a practical point of view, indeed it implies that we can run the autosymmetry test onto f_A instead of f , with a reduced computational effort guaranteed by the fact that f_A depends on less variables than f .

As for the estimation of the multiplicative complexity of f , we can observe that, since f is autosymmetric and D-reducible, we can upper bound its multiplicative complexity by first projecting f onto A , and then by estimating the multiplicative complexity of the restriction $f_{A,k}$ of f_A . Indeed, we have that

$$M(f) = M(f_k) \leq (n - \dim A) + M(f_A),$$

and, since $M(f_A) = M(f_{A,k})$, we finally obtain

$$M(f) \leq (n - \dim A) + M(f_{A,k}).$$

Thus, we can estimate the multiplicative complexity of f by computing and minimizing the restriction $f_{A,k}$ of f_A in XAG form. Since $f_{A,k}$ contains $|S(f)|/2^k$ minterms only (recall that $|S(f)|$ is the number of minterms of f), and depends on $\dim A - k < n - k$ variables only, its XAG minimization should be easier and might provide a final circuit with a reduced number of *and* gates. This expectation has been confirmed by our experiments.

4.3.3 Experimental results

In this section we report and discuss the experimental results of the evaluation of the multiplicative complexity of the two classes of autosymmetric and D-reducible functions*.

4.3.3.1 Autosymmetric functions

The approach presented in Section 4.2.1 has been applied to the ESPRESSO and LGSynth'89 benchmark suite [120] and to some functions from cryptography benchmarks in the context of multi-party computation (MPC) and fully homomorphic encryption (FHE) [114, 112]. Notice that autosymmetry is a property of single outputs, i.e., different outputs of the same benchmark can have different autosymmetry degrees. Thus, we perform the autosymmetry test on each single output of the considered benchmark suites. Finally, in ESPRESSO benchmark suite about 24% of the functions have at least one truly (i.e. non degenerate) autosymmetric outputs, while

*A GitHub repository with the experimental data can be found at <https://github.com/MariaChiaraMC/IEEETC-experiments>

in the cryptographic benchmark the percentage is about the 50% of the tested functions. Due to a current limitations in terms of scalability on larger benchmarks, the tested cryptographic functions are a small part of the whole benchmark (about 9%).

The experiments have been run on a Pentium INTEL(R) CORE(TM) i5-5200U 2.20 GHz processor with 4.00 GB RAM, on a virtual machine running OS Ubuntu 64-bit.

The experiments consider the subset of single outputs that are autosymmetric. The main aim of the experiments is to compare the synthesized XAG computed starting from an autosymmetric function f and the synthesized XAG computed starting from the corresponding restriction f_k , after the autosymmetry test. Recall that the autosymmetry test computes the autosymmetry degree k of a Boolean function and outputs: 1) the reduction equations, which form the *xor* layer, and 2) the corresponding restriction f_k .

We performed the autosymmetry test described in [19, 21] considering the onset of the benchmarks. The functions f and f_k are first minimized in SOP form (using Espresso [84]), and then synthesized in XAG form using the heuristic approach proposed in [114] and briefly described at the end of Section 4.1.3.

We then compare the number of *and* nodes of the XAGs for f and f_k in order to understand how the autosymmetry test can enable the XAG minimization of autosymmetric functions.

For the sake of brevity, we report in Table 4.3 only a significant subset of the results. The first column reports the name of the function considered (benchmark function and output number). The following one provides its input size. Next column refers to the autosymmetry degree (i.e., k) of the function. The following two pairs of columns report the multiplicative complexity of the XAG (M_X) after applying the heuristic in [114] and the time in seconds required to obtain it, for the entire function f (first couple) and for the corresponding restriction f_k (second couple). Finally, the last column reports the gain in applying the autosymmetry test before XAG synthesis. Note that there are also some (rare) cases in which the application of the autosymmetry test before the XAG synthesis does not imply a gain. In Table 4.3, two representative functions are reported, i.e., $p1(6)$ and $pd(10)$. This unexpected result is due to the heuristic nature of the XAG minimizer.

Table 4.4 shows a summary of the overall results, i.e., the results obtained for all the circuits that have been processed in our experimental evaluation. We first consider the set of all autosymmetric functions (degenerate and non-degenerate), we then study the truly autosymmetric (i.e., the non-degenerate) ones. Recall that degenerate functions are, by definition, autosymmetric. In Table 4.4, we denote with $M_X(f_k) < (=, >, \text{resp.}) M_X(f)$ the number of benchmarks where the number of *ands*

4.3.3 Experimental results

Table 4.3: Experimental comparison of autosymmetric benchmarks, considering an XAG after the autosymmetry test and the standard XAG computed without the autosymmetry test.

Benchmark	in	k	standard XAG		XAG with autosym. test		gain
			$M_X(f)$	time (s)	$M_X(f_k)$	time (s)	
add6(5)	12	1	8	0.97	5	1.77	38%
addm4(5)	9	2	32	3.98	32	3.93	0%
amd(0)	14	2	53	10.19	53	10.03	0%
apla(5)	10	1	11	0.96	8	0.68	27%
b2(15)	16	1	118	22.00	118	22.32	0%
ex5(21)	8	1	7	0.59	6	0.43	14%
exep(46)	30	9	20	0.92	20	0.87	0%
exps(19)	8	1	10	2.30	6	0.45	40%
in0(3)	15	1	14	3.03	14	2.77	0%
mainpla(20)	27	5	119	21.37	119	21.80	0%
max46(1)	10	2	36	6.74	10	1.80	72%
max46(5)	10	1	305	50.10	156	26.22	49%
opa(26)	17	9	16	2.51	16	2.44	0%
opa(48)	17	3	22	4.21	22	5.74	0%
p1(6)	8	1	6	1.03	8	0.79	-33%
pdc(10)	16	3	12	2.05	14	1.66	-17%
rd53(1)	5	1	5	0.38	2	0.11	60%
spla(39)	16	3	16	1.27	12	0.80	25%
tial(2)	14	2	174	29.69	174	29.83	0%
xparc(6)	41	16	83	10.90	21	1.80	75%
z5xp1(6)	7	3	2	0.00	0	0.00	100%
ctrl(13)	7	3	3	0.02	3	0.02	0%
dec(39)	8	1	6	0.53	6	0.42	0%
voting_N_2_M_2(1)	8	1	81	9.88	22	3.04	73%
voting_N_2_M_3(1)	16	1	9261	739.29	5208	468.86	44%
int2float(6)	11	2	8	0.59	8	0.55	0%

Table 4.4: Summary of the experimental evaluation, considering the number of *ands* in the XAGs for autosymmetric functions and non-degenerate autosymmetric functions.

	$M_X(f_k) < M_X(f)$	$M_X(f_k) = M_X(f)$	$M_X(f_k) > M_X(f)$
autosymmetric functions	6.06%	92.31%	1.63%
non-degenerate autosymmetric functions	52.09%	40.10%	7.81%

of the XAG for f_k is less than (equal to, greater than, resp.) the number of *ands* of the XAG for f .

We notice that the XAG minimization algorithm proposed in [114] is sensible to degenerate functions as shown in the first row of Table 4.4, where the number of benchmarks where f_k and f have the same number of *ands* is the majority (i.e., about 92.31%) and only 6.06% of them is such that $M_X(f_k)$ is less than $M_X(f)$. However, when we concentrate on non-degenerate autosymmetric functions (i.e., second row of the table), we notice that the number of benchmarks where $M_X(f_k) < M_X(f)$ considerably increases, reaching about the 52.09%, and the number of benchmarks where f_k and f have the same number of *ands* remains high (about 40.1%). Moreover, in this set the average gain is about 44%. Interestingly enough, the two compared approaches have similar synthesis times.

From these experiments, we can conclude that, when a function is truly autosymmetric (i.e., non-degenerate), we can obtain better results computing the XAG on the restriction f_k instead of computing the XAG directly on the function f .

4.3.3.2 D-reducible functions

We now analyze the experimental results conducted in order to evaluate the multiplicative complexity of D-reducible functions exploiting the XAG decomposition discussed in Section 4.3.1.

The experiments have been run on a CPU Intel i7 2.60GHz processor, with a virtual machine running Ubuntu 18.04, and benchmarks are taken from the ESPRESSO and LGSynth'89 benchmark suite [120]. We considered each output as a separate Boolean function, and analyzed a total of 406 D-reducible functions. As before, the functions and their projections have been synthesized in XAG form using the heuristic approach proposed in [114].

We report in Table 4.5 a significant subset of functions as representative indicators of our experiments. The first two columns report the name and the number of the considered output of each benchmark, and the number of its input variables. The following pair of columns reports the multiplicative complexity of the XAG for the entire function f ($M_X(f)$), obtained running the heuristic in [114], and the time in seconds required to obtain it. The next group of four columns reports (i) the multiplicative complexity of the XAG for the projection f_A ($M_X(f_A)$); (ii) the multiplicative complexity of the affine space χ_A computed applying Theorem 4.3.4 ($M(\chi_A)$); (iii) the overall estimate of the multiplicative complexity of f ($M_{dec}(f)$) derived using Equation (4.2); and (iv) the time in seconds required to obtain this overall estimate. Finally, the last column reports the gain (or loss) in applying the proposed decomposition method.

4.3.3 Experimental results

Table 4.5: Experimental comparison of D-reducible benchmarks, considering XAGs computed exploiting the D-reducibility property with standard XAGs.

Benchmark	in	standard XAG		XAG with D-red. test				gain
		$M_X(f)$	time (s)	$M_X(f_A)$	$M(\chi_A)$	$M_{dec}(f)$	time (s)	
alu2(6)	10	6	2.69	5	0	6	2.29	0%
amd(15)	14	6	2.51	0	5	6	2.33	0%
amd(16)	14	7	2.47	0	6	7	2.28	0%
apla(8)	10	17	3.97	4	2	7	2.31	59%
b10(2)	15	17	3.78	13	4	18	3.19	-6%
dk48(16)	15	16	3.58	1	10	12	2.20	25%
in2(6)	19	66	9.19	48	1	50	6.27	24%
in5(8)	24	46	8.09	47	0	48	7.66	-4%
m181(6)	15	8	2.89	6	1	8	2.56	0%
newcond(1)	11	2	2.28	0	1	2	2.22	0%
spla(2)	16	14	2.90	0	13	14	2.24	0%
spla(16)	16	10	2.95	1	8	10	2.26	0%
spla(29)	16	19	3.47	5	9	15	2.33	21%
spla(42)	16	10	2.88	0	6	7	2.19	30%
t1(1)	21	10	3.89	9	0	10	3.08	0%
t2(0)	17	14	3.44	12	1	14	3.14	0%
t2(4)	17	5	2.65	3	1	5	2.44	0%
t3(2)	12	16	3.72	9	1	11	3.10	31%
t4(0)	12	17	3.87	5	0	6	2.42	65%
vg2(2)	25	30	4.26	14	5	20	3.34	33%

Table 4.6: Summary of the experimental evaluation, considering the number of *ands* in the XAGs for D-reducible functions.

	$M_{dec}(f) < M_X(f)$	$M_{dec}(f) = M_X(f)$	$M_{dec}(f) > M_X(f)$
D-reducible functions	42.61%	49.02%	8.37%

Table 4.7: Summary of the experimental evaluation, considering the number of *ands* in the XAGs for autosymmetric and D-reducible functions.

	$M_X(f_k) < M_X(f)$	$M_X(f_k) = M_X(f)$	$M_X(f_k) > M_X(f)$
autosymmetric and D-reducible functions	89.54%	3.27%	7.19%

In Table 4.6, we report a summary of the overall experimental results conducted on all the 406 D-reducible benchmarks' outputs. We denote with $M_{dec}(f) < M_X(f)$, $M_{dec}(f) = M_X(f)$, and $M_{dec}(f) > M_X(f)$ the number of benchmarks' outputs where the number of *ands* obtained applying the decomposition based on the D-reducibility property is less than, equal to, and greater than the number of *ands* of the XAG for f . The number of functions where the XAG minimization can benefit from the D-reducible decomposition is about the 43% of the whole set of functions, with an average reduction of the number of *ands* of about 35%. The number of functions where the estimates of the multiplicative complexity are the same is about 49%, while for the remaining 8% of the functions the method provides a worst estimate.

We finally observe how the proposed decomposition method guarantees a reduction of the average computational time for XAG synthesis of about 24%.

4.3.3.3 Autosymmetric and D-reducible functions

We conclude this experiments section analysing the results reached applying both the autosymmetry test and the D-reducible decomposition to Boolean functions in the benchmarks from ESPRESSO and LGSynth'89 benchmark suite [120]. These last experiments have been run on a Pentium INTEL(R) CORE(TM) i5-5200U 2.20 GHz processor with 4.00 GB RAM, on a virtual machine running OS Ubuntu 64-bit.

Table 4.7 shows a summary of the results for functions that are both autosymmetric and D-reducible (about 9% on the total). We applied the autosymmetry test to the D-reducible functions discussed in Section 4.3.3.2. We note that we did not find D-reducible benchmarks that are also non-degenerate and autosymmetric.

From Table 4.7, we note that the functions where the XAG minimization can benefit from autosymmetry and D-reducibility are about 90%, with an average reduction of the number of *ands* of about 24%. The number of functions where the estimates of the multiplicative complexity are the same is about 3%, while for the remaining 7% of the functions the method provides a worst result.

4.4 Conclusion and further works

In this Chapter we present our research in the context of logic synthesis. In particular, since we are interested in the cryptographic-related applications (e.g. in the multi-party computation), we work on the reduction of the number of *and* nodes in XAG graphs. With this aim, we propose two articles in which we investigate how the properties of autosymmetry and D-reducibility can be exploited to better estimate the multiplicative complexity of the Boolean functions. We have conducted our experimentations on a wide set of standard Boolean benchmarks. We plan to extend the autosymmetry algorithms to detect all linear structures of a function, thus generalizing our approach. We also will work on the current limitations in terms of scalability on larger functions.

CHAPTER 5

MULTIPLE-VALUED LOGIC

In Section 2.1.5 we introduce the Boolean logic: the truth values *true* (corresponding to the binary value 1) and *false* (corresponding to the binary value 0) can be combined by the operations *not*, *and*, *or* and *xor*. In this Chapter, we present the concept of Multiple-valued logic (MVL), i.e., a logic in which more than two elements are managed. The most studied in literature, and then also in this work, is the three-valued logic (3VL): this is a logic with one element more than *true* (1) and *false* (0), called unknown (2). In Section 5.1, we propose an overview on the state of the art about this topic. In Section 5.2, we present some new results: a new reasoning about the 3VL and the proof of *xor* free transfer in the MVL context.

5.1 State of the art

Until the beginning of the 20-th century, logic was almost mostly viewed as binary, but mathematicians, logicians and philosophers needed to represent uncertainty and error, so they started investigating systems that allowed more options than just *true* or *false*. Their aim was to solve some logical paradoxes, looking for a third or more truth values, and applied them to problems related to representability of functions. After a first push during the 1920s (with Emil Post and Jan Łukasiewicz), several mathematicians, logicians and philosophers (Godel, Bochvar, Kleene as example) started working with this concept and developed several forms of MVL during the 1930s [63].

5.1.1 MVL as generalization of the Boolean Logic

Multiple-valued logics are similar to Boolean logic because they accept the principle of truth-functionality, i.e., the truth of a compound sentence is determined by the truth values of its component sentences. But they differ from Boolean logic by the fundamental fact that they do not restrict the number of truth values to only two, but they allow for a larger set of values. Indeed, considering the natural generalization of the classical Boolean logic, it is possible to define a MVL as follows: let P_i be the finite subset of natural numbers $P_i = \{0, 1, \dots, |P_i| - 1\}$ with $|P_i| > 1$, and $x \in P_i$ is a truth variable of this MVL [86, 56]. Note that, if $|P_i| = 2$, the MVL coincides with the Boolean logic, if $|P_i| = 3$ it is a 3 valued logic, and so on.

A multiple-valued function F is a function such that $F : P_1, P_2, \dots, P_n \rightarrow P_F$. In particular, when $P_1 = P_2 = \dots = P_n = P_F = P$ we have that $F : P^n \rightarrow P$. Note that, in this case, there are $|P|^{|P|^n}$ possible different MVL functions.

In this case, MVL gates are a direct generalization of standard Boolean gates. The number of two-input gates grows exponentially with the dimension of P . In particular, the standard one and two-inputs Boolean operations in the multiple-valued logic are reported in Table 5.1 [86].

Name	Notation	Definition
Not	$\neg x$	$(P - 1) - x$
Min	$x \cdot y$	x if $x < y$, y otherwise
Max	$x + y$	x if $x > y$, y otherwise
Mod-sum	$x \oplus y$	$(x + y) \bmod_{ P }$
Mod-difference	$x \ominus y$	$(x - y) \bmod_{ P }$
Truncated sum	$x +_t y$	$\min(P - 1, x + y)$

Table 5.1: Two-inputs MVL operations.

Note that the *and* gate corresponds to the Min operation, the *or* gate can be generalized to the Max or to the Truncated sum, the *xor* gate can be generalized to the Mod-sum or to the Mod-difference. A MVL circuit is a circuit composed by MVL gates.

This is not the only one possible definition of multiple-valued logic, but many other forms have been developed and studied in the last century, mostly when 3 truth values are taken into account.

Lukasiewicz's logics. The first multiple-valued logic was proposed by Jan Lukasiewicz in the 1920s [82]. His first intention was to use a third, additional truth value for

“possible”, and to model in this way the modalities “it is necessary that” and “it is possible that”. Starting from these investigations, he defined his three-valued logic L_3 , with the truth values $0, \frac{1}{2}, 1$, on which operate the following functions:

$$\begin{aligned}\neg_L u &= 1 - u \\ u \rightarrow_L v &= \min\{1, 1 - u + v\}\end{aligned}$$

The outcomes of these investigations are, however, the two Łukasiewicz systems L_m and L_∞ . The former is defined on some finite set of rationals within the real unit interval, i.e., in the set $\{\frac{k}{m-1} | 0 \leq k \leq m-1\}$. The latter is defined on the whole unit interval, i.e., in the set $[0, 1] = \{x \in \mathbb{R} | 0 \leq x \leq 1\}$. In both the systems, 1 is the only designated truth value. In addition to negation and implication just defined for L_3 , other two operations are given:

$$\begin{aligned}u \& v &= \max\{0, u + v - 1\} \\ u \wedge v &= \min\{u, v\}\end{aligned}$$

Moreover, two disjunction connectives are defined in terms of $\&$ and \wedge , via the usual de Morgan laws using \neg . Finally, the two quantifiers \forall and \exists are the infimum and the supremum of all the values in a considered subset, respectively.

Gödel’s logics. In the 1930s, Kurt Gödel tried to understand intuitionistic logic in terms of many truth values [62]. The outcome was the family of Gödel systems G_m and G_∞ . The former is defined on the finite set of rationals within the real unit interval, i.e., $\{\frac{k}{m-1} | 0 \leq k \leq m-1\}$, and the latter on the whole unit interval $[0, 1] = \{x \in \mathbb{R} | 0 \leq x \leq 1\}$. The value 1 is the only designated as truth value. The two main functions on the elements in these sets are

$$\begin{aligned}u \wedge v &= \min\{u, v\} \\ u \vee v &= \max\{u, v\}\end{aligned}$$

while the implication and negation operations are

$$\begin{aligned}u \rightarrow v &= \begin{cases} 1 & \text{if } u \leq v \\ v & \text{if } u > v \end{cases} \\ \neg u &= \begin{cases} 1 & \text{if } u = 0 \\ 0 & \text{if } u \neq 0 \end{cases}\end{aligned}$$

The two quantifiers \forall and \exists are the infimum and the supremum of all the values in a considered subset, respectively.

Kleene's (strong) logic. A mathematical application of 3-valued logic to partial functions and relations was proposed by the American logician Stephen Cole Kleene [48]. The Kleene's (strong) logic K_3 introduce to the set of Boolean values *true* (T) and *false* (F) a third element *undefined* (called also *indeterminate*) (I). The functions of negation (\neg), conjunction (\wedge), disjunction (\vee) and implication (\rightarrow_K) are given in Figure 5.1.

\neg	
T	F
I	I
F	T

\wedge	T	I	F
T	T	I	F
I	I	I	F
F	F	F	F

\vee	T	I	F
T	T	T	T
I	T	I	I
F	T	I	F

\rightarrow_K	T	I	F
T	T	I	F
I	T	I	I
F	T	T	T

Figure 5.1: Operations in the Kleene's logic K_3 .

In Kleene's logic, only T is a designed truth value, and is interpreted as being *underdetermined* (neither *true* nor *false*). Similar to the Kleen's logic, the Graham Priest logic P_3 has the same set of values $\{T, F, I\}$ and the same operations, but both T and I are considered with *true* value, and I is interpreted as *overdetermined* (both *true* and *false*).

Bochvar's logic. A philosophical application of 3-valued logic to the discussion of paradoxes was proposed by the Russian logician D.A. Bochvar [27]. His logic B_3 (also called Kleene's weak three-valued logic), has the same negation as Kleene's strong logic, but all the other tables are different (Figure 5.2).

\neg	
T	F
I	I
F	T

\wedge_{B_3}	T	I	F
T	T	I	F
I	I	I	I
F	F	I	F

\vee_{B_3}	T	I	F
T	T	I	T
I	I	I	I
F	T	I	F

\rightarrow_{B_3}	T	I	F
T	T	I	F
I	I	I	I
F	T	I	T

Figure 5.2: Operations in the Bochvar's logic B_3 .

The main difference between the Kleene's strong logic and the weak one is that in the Bochvar's logic the intermediate truth value propagates in a formula regardless of the value of any other variables.

Belnap's 4-valued logic. This 4-valued system has an interesting interpretation in the context of information bases stored in a computer, which was explained by Nuel Belnap [12]. Its truth values set is $\{\emptyset, \{\perp\}, \{\top\}, \{\perp, \top\}\}$, and the elements in

5.1.1 MVL as generalization of the Boolean Logic

the set are interpreted as indicating (e.g. with respect to a database query for some particular state of affairs) that there is: no information concerning this state of affairs (\emptyset), information saying that the state of affairs fails ($\{\perp\}$), information saying that the state of affairs obtains ($\{\top\}$), conflicting information saying that the state of affairs obtains as well as fails ($\{\perp, \top\}$).

This set of truth values has two natural (lattice) orderings: a *truth ordering* which has $\{\top\}$ on the top, $\{\perp\}$ on the bottom and incomparable \emptyset and $\{\perp, \top\}$ (Figure 5.3, on the left), and an *information ordering* which has $\{\perp, \top\}$ on the top, \emptyset on the bottom and incomparable $\{\perp\}$ and $\{\top\}$ (Figure 5.3, on the right).

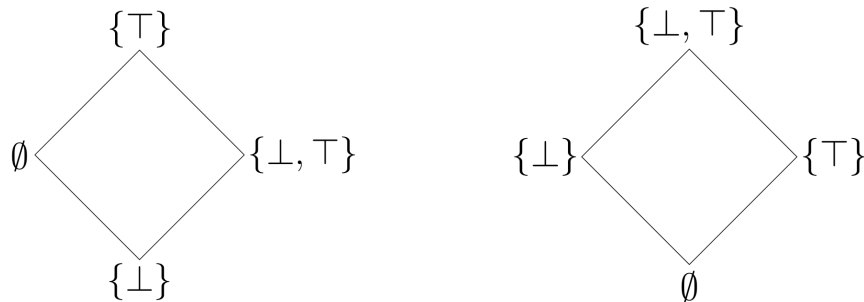


Figure 5.3: Orders of the truth values in Belnap's logic.

The negation function is determined exchanging the values $\{\top\}$ and $\{\perp\}$, and which leaves the degrees $\{\perp, \top\}$ and \emptyset fixed. Moreover, given the infimum and the supremum under the truth ordering, conjunction and disjunction operations can be defined (Figure 5.4). Actually, there is no standard candidate for the implication, and the choice depends on the intended applications: for computer science applications, it is natural to have $\{\top\}$ as the only designated degree, for applications to relevance logic, the choice of $\{\top\}$, $\{\perp, \top\}$ as designated degrees proved to be adequate. The choice of suitable entailment relations is still an open research topic.

\wedge_{B_4}	{T}	{⊥, T}	∅	{⊥}
{T}	{T}	{⊥, T}	∅	{⊥}
{⊥, T}	{⊥, T}	{⊥, T}	{⊥}	{⊥}
∅	∅	{⊥}	∅	{⊥}
{⊥}	{⊥}	{⊥}	{⊥}	{⊥}

\neg	
{T}	{⊥}
{⊥, T}	{⊥, T}
∅	∅
{⊥}	{T}

\vee_{B_4}	{T}	{⊥, T}	∅	{⊥}
{T}	{T}	{T}	{T}	{T}
{⊥, T}	{T}	{⊥, T}	{T}	{⊥, T}
∅	{T}	{T}	∅	∅
{⊥}	{T}	{⊥, T}	∅	{⊥}

 Figure 5.4: Operations in the Kleene's logic K_3 .

5.1.2 Lindell and Yanai's 3VL approach

In [80], the authors present a 3 valued logic approach (called FTU) based on the Kleene's logic. Then, the set of the truth values is $\{true, false, unknown\}$, and *unknown* is the result of a sort of uncertainty in the inference process. The 3VL inference rules as in the FTU approach are presented in Figure 5.5, where F, T, U refers to *false*, *true*, *unknown* respectively. Note that substantially the operations *not*, *and* and *or* are the same as in Figure 5.1, substituting I (indeterminate) with U (unknown) and with the adjunct of the *xor* operation.

\neg'_3	
F	T
U	U
T	F

\wedge'_3	F	U	T
F	F	F	F
U	F	U	U
T	F	U	T

\vee'_3	F	U	T
F	F	U	T
U	U	U	T
T	T	T	T

\oplus'_3	F	U	T
F	F	U	T
U	U	U	U
T	T	U	F

Figure 5.5: Operations in the 3VL with the FTU approach.

The main idea in [80] is to try to find encodings in the FTU-3VL that are better than the *naive garbling*, that is described below. Let g_3 be a 3VL gate with inputs x, y and output z , where each wire takes a value in $\{T, F, U\}$. Following the basic garbling scheme by Yao et Al. [122], for each wire $\alpha \in \{x, y, z\}$, the random keys $k_\alpha^T, k_\alpha^F, k_\alpha^U$. Then, for every combination of $\beta_x, \beta_y \in \{F, T, U\}$ the encryption $f_z^{g(\beta_x, \beta_y)}$ with the keys $k_x^{\beta_x}, k_y^{\beta_y}$ is computed. The complete garbled table of gate g is reported in Table 5.2, where E notation refers to the encryption function. This garbled table has 9 rows, and then the cost to garble in this way a 3VL gate is proportional to it.

5.1.2 Lindell and Yanai's 3VL approach

1	$E_{k_x^T}(E_{k_y^T}(k_z^{g(T,T)}))$	4	$E_{k_x^F}(E_{k_y^T}(k_z^{g(F,T)}))$	7	$E_{k_x^U}(E_{k_y^T}(k_z^{g(U,T)}))$
2	$E_{k_x^T}(E_{k_y^F}(k_z^{g(T,F)}))$	5	$E_{k_x^F}(E_{k_y^F}(k_z^{g(F,F)}))$	8	$E_{k_x^U}(E_{k_y^F}(k_z^{g(U,F)}))$
3	$E_{k_x^T}(E_{k_y^U}(k_z^{g(T,U)}))$	6	$E_{k_x^F}(E_{k_y^U}(k_z^{g(F,U)}))$	9	$E_{k_x^U}(E_{k_y^U}(k_z^{g(U,U)}))$

Table 5.2: Naive garbling for a 3VL gate.

Since the aim of Lindell et Al. is to find ways to garble 3VL functions more efficiently than the naive method, they propose an encoding method that implies first an encoding function from 3VL to the Boolean domain, the application of the state of the art garbling schemes for Boolean functions [97, 124], and then a decoding from Boolean to 3VL. These garbling schemes have the property that *and* gate is garbled using three ciphertexts, and *xor* is garbled for free. This allows a faster and cheaper communication of garbled gates, as show in [78].

The notation is shown in Figure 5.6:

1. F_3 is the set of all 3VL functions (i.e., all functions of the form $\{F,T,U\}^* \rightarrow \{F,T,U\}^*$) and F_2 is the set of all Boolean functions (i.e., all functions of the form $\{0, 1\}^* \rightarrow \{0, 1\}^*$).
2. $Tr_{2 \rightarrow 3}$ is the set of transformations from the set of all elements in the 3VL to those in Boolean logic, and $Tr_{3 \rightarrow 2}$ the set of inverse transformations. A transformation can be functional, namely at each element in the 3VL corresponds to a couple of Boolean values, or non-functional, namely at each element in the 3VL can correspond more than one couple of Boolean values. Any transformation must be injective, to allow the definition of the inverse.
3. Tr_F is the set of all transformations from operations in F_3 to operations in F_2 .
4. Observe that $\forall f_2 \in F_2 \exists f_3 \in F_3, tr_{2 \rightarrow 3} \in Tr_{2 \rightarrow 3}$ such that, $\forall x$

$$tr_{2 \rightarrow 3}(f_2(tr_{3 \rightarrow 2}(x))) = f_3(x) \quad (5.1)$$

Three different encodings are described in [80]: a natural encoding, an encoding through a functional relation and a non-functional one. These encodings belong in the set of those with the minimum number of *ands* in F_2 , as explained in [80].

Natural encoding. This encoding is defined by the input transformation

$$(x_L, x_R) = tr_{3 \rightarrow 2}(x) = \begin{cases} (0, 1) & x = T \\ (0, 0) & x = F \\ (1, 0) & x = U \end{cases}$$

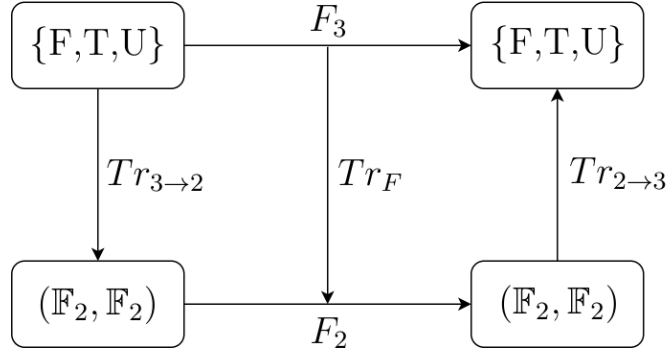


Figure 5.6: Encoding steps.

and it is called natural because "naturally" transform a 3VL value x in a couple of Boolean values (x_L, x_R) , such that x_L signals whether the value is known or unknown, and x_R signals whether the value is true or false.

This transformation translates each function in F_3 to a function in F_2 as follows.

1. Operation \neg'_3 is translated in

$$Tr_F(\neg'_3) = \neg_2(x_L, x_R) = (x_L, \neg x_R).$$

2. Operation \wedge'_3 is translated in

$$Tr_F(\wedge'_3) = \wedge_2(x_L, x_R, y_L, y_R) = ((x_L \wedge y_L) \vee (x_L \wedge y_R) \vee (x_R \wedge y_L), x_R \wedge y_R).$$

3. Operation \oplus'_3 is translated in

$$Tr_F(\oplus'_3) = \oplus_2(x_L, x_R, y_L, y_R) = (x_L \vee y_L, x_R \oplus y_R)$$

The cost of each operation in Boolean logic is shown in Table 5.3. Note that, thanks to the De Morgan's law, $a \vee b = \neg((\neg a) \wedge (\neg b))$, and then \vee costs $3 \cdot \neg + 1 \cdot \wedge$. From Table 5.4, it is then clear that \oplus'_3 costs 1 \wedge , 1 \oplus and 3 \neg , thus it cannot be evaluated for free.

About the efficiency of this encoding, when using the garbling scheme of [124] that incorporates free-*xor* and requires two ciphertexts for \wedge and \vee gates, the cost of garbling \wedge'_3 is 12 ciphertexts, and the cost of garbling \oplus'_3 is 2 ciphertexts. In comparison, recall that the naive garbling scheme requires 8 ciphertexts for both \wedge'_3 and \oplus'_3 . To see which is better, let C be a 3VL circuit and denote by C_\wedge and C_\oplus the number of \wedge'_3 and \oplus'_3 gates in C , respectively. Then, the natural 3VL-Boolean encoding is better than the naive approach if and only if $12 \cdot C_\wedge + 2 \cdot C_\oplus < 8 \cdot C_\wedge + 8 \cdot C_\oplus$, which holds if and only if $C_\wedge < 1.5 \cdot C_\oplus$.

5.1.2 Lindell and Yanai's 3VL approach

Table 5.3: FTU approach: cost of each translated operations in the Boolean logic, in case of the natural encoding.

3VL Function	cost
\neg'_3	$1 \cdot \neg$
\wedge'_3	$6 \cdot \wedge + 6 \cdot \neg$
\oplus'_3	$1 \cdot \wedge + 1 \cdot \oplus + 3 \cdot \neg$

Functional encoding. This encoding is defined by the input transformation

$$(x_L, x_R) = tr_{3 \rightarrow 2}(x) = \begin{cases} (1, 1) & x = T \\ (0, 0) & x = F \\ (1, 0) & x = U \end{cases}$$

This transformation translates each function in F_3 to a function in F_2 as follows.

1. Operation \neg'_3 is translated in

$$Tr_F(\neg'_3) = \neg_2(x_L, x_R) = (\neg x_R, \neg x_L).$$

2. Operation \wedge'_3 is translated in

$$Tr_F(\wedge'_3) = \wedge_2(x_L, x_R, y_L, y_R) = (x_L \wedge y_L, x_R \wedge y_R).$$

3. Operation \oplus'_3 is translated in

$$Tr_F(\oplus'_3) = \oplus_2(x_L, x_R, y_L, y_R) = (z'_L \oplus aux, z'_R \oplus aux)$$

where $z'_L = (x_L \oplus y_L) \oplus ((x_L \oplus x_R) \wedge (y_L \oplus y_R))$ and $z'_R = x_R \oplus y_R$ and $aux = \neg z'_L \wedge z'_R$.

The cost of each operation in Boolean logic is shown in Table 5.4. Also in this case it is possible to note that \oplus'_3 cannot be evaluated for free.

About the efficiency of this encoding, when using the garbling scheme of [124] that incorporates free-*xor* and requires two ciphertexts for \wedge and \vee gates, the cost of garbling \wedge'_3 is 4 ciphertexts, and the cost of garbling \oplus'_3 is 4 ciphertexts. This is far more efficient than the naive garbling for all gate types. Next, recall that the natural encoding previously presented required 12 ciphertexts for \wedge_3 gates and 2 ciphertexts for \oplus_3 gates. Thus, denoting by C_\wedge and C_\oplus the number of \wedge'_3 and \oplus'_3

Table 5.4: FTU approach: cost of each translated operations in the Boolean logic, in case of the functional encoding.

3VL Function	cost
\neg'_3	$2 \cdot \neg$
\wedge'_3	$2 \cdot \wedge$
\oplus'_3	$2 \cdot \wedge + 7 \cdot \oplus$

gates, respectively, in a 3VL circuit C , we have that the scheme in this section is more efficient if and only if $4 \cdot C_\wedge + 4 \cdot C_\oplus < 12 \cdot C_\wedge + 2 \cdot C_\oplus$, which holds if and only if $C_\oplus < 4 \cdot C_\wedge$. Thus, the natural encoding is only better if the number of \oplus'_3 gates is over four times the number of \wedge_3 gates in the circuit.

Non-functional encoding. This encoding is such that value U is encoded with both (1,0) and (0,1). This means that it is defined by two input transformations, both mapping T to (1, 1) and F to (0, 0); one of them maps U to (1, 0) the other maps U to (0, 1).

$$tr_{3 \rightarrow 2}^1(x) = \begin{cases} (1, 1) & x = T \\ (0, 0) & x = F \\ (0, 1) & x = U \end{cases} \quad tr_{3 \rightarrow 2}^2(x) = \begin{cases} (1, 1) & x = T \\ (0, 0) & x = F \\ (1, 0) & x = U \end{cases}$$

The function transformation Tr_F needs to work for both the input transformations $tr_{3 \rightarrow 2}^1$ and $tr_{3 \rightarrow 2}^2$; The transformation Tr_F for each gate type is given below.

1. Operation \neg'_3 is translated in

$$Tr_F(\neg'_3) = \neg_2(x_L, x_R) = (\neg x_L, \neg x_R).$$

2. Operation \wedge'_3 is translated in

$$Tr_F(\wedge'_3) = \wedge_2(x_L, x_R, y_L, y_R) = (x_L \wedge y_L, (x_R \wedge y_R) \oplus ((x_L \oplus x_R) \wedge (y_L \oplus y_R) \wedge (\neg(x_R \oplus y_L)))).$$

3. Operation \oplus'_3 is translated in

$$Tr_F(\oplus'_3) = \oplus_2(x_L, x_R, y_L, y_R) = ((x_L \oplus y_L) \oplus ((x_L \oplus x_R) \wedge (y_L \oplus y_R)), x_R \oplus y_R).$$

The cost of each operation in Boolean logic is shown in Table 5.5. Also in this case it is possible to note that \oplus'_3 cannot be evaluated for free.

5.1.2 Lindell and Yanai's 3VL approach

Table 5.5: FTU approach: cost of each translated operations in the Boolean logic, in case of the non-functional encoding.

3VL Function	cost
\neg'_3	$2 \cdot \neg$
\wedge'_3	$4 \cdot \wedge + 4 \cdot \oplus + 1 \cdot \neg$
\oplus'_3	$1 \cdot \wedge + 5 \cdot \oplus$

About the efficiency of this encoding, when using the garbling scheme of [124] that incorporates *free-xor*, the cost of garbling \wedge'_3 is 8 ciphertexts, and the cost of garbling \oplus'_3 is 2 ciphertexts. Denote by C_\wedge and C_\oplus the number of \wedge'_3 and \oplus'_3 gates in the 3VL circuit, then the encoding of this section is better than the functional one if and only if $8 \cdot C_\wedge + 2 \cdot C_\oplus < 4 \cdot C_\wedge + 4 \cdot C_\oplus$ which holds if and only if $C_\oplus > 2 \cdot C_\wedge$. Observe also that the non-functional encoding is always at least as good as the natural encoding. In particular, it has the same cost for \oplus'_3 gates and is strictly cheaper for \wedge'_3 gates.

No free-xor transmission. In [80], the authors show that no one among all the possible encodings in their FTU approach from the 3VL to the Boolean logic can exploit the *free-xor* optimization, because each one translates \oplus'_3 in a Boolean function that involves at least one \wedge . Indeed, they prove that any garbling scheme for 3VL-*xor* can be used to garble Boolean-*and* gates at the exact same cost. This result descends from the fact that the truth table for \oplus'_3 embeds the truth table of both the Boolean logic operations \wedge, \vee, \oplus (in Figures 5.7 and 5.8 the tables of the embedded Boolean operations are circled). Now, [124] proved that at least 2 ciphertexts are required for garbling *and* gates using any linear garbling method. By reducing to this result, they show that 3VL-*xor* cannot be garbled with less than two ciphertexts using any linear garbling method (with the FTU approach).

\oplus'_3	F	U	T
F	F	U	T
U	U	U	U
T	T	U	F

Figure 5.7: Embedded *and* and *or*.

\oplus'_3	F	U	T
F	F	U	T
U	U	U	U
T	T	U	F

Figure 5.8: Embedded *xor*.

5.1.3 Cimato et Al.'s 3VL approach

In [43], the authors present a 3 valued logic approach (called 012), that descends from the natural generalization of the Boolean logic, with 3 truth values [86]. In this approach, 0 corresponds to the *false* value and 1 to the *true* value, as in the Boolean logic, and 2 corresponds to the *unknown* value. In Figure 5.9 the gates operations for this 3VL approach are reported. In particular, for the *xor* gate the mod-sum operation is chosen, and for the *or* gate the max operation.

\neg_3	
0	2
1	1
2	0

\wedge_3	0	1	2
0	0	0	0
1	0	1	1
2	0	1	2

\vee_3	0	1	2
0	0	1	2
1	1	1	2
2	2	2	2

\oplus_3	0	1	2
0	0	1	2
1	1	2	0
2	2	0	1

Figure 5.9: Operations in the 3VL with the 012 approach.

The main result in [43] is that, since this approach descends from the Boolean logic, also the property described for the transmission of *xor* gates for free in [78] is valid. Indeed, following the model of proof also reported in Section 2.4.2.3, the authors in [43] prove that the *xor* gate can be sent for free also in their 3VL.

Let G be a *xor* gate with two input wires W_a and W_b and output wire W_c . Garble the wire values as follows. Randomly choose $w_a^0, w_b^0, R_1, R_2 \in 0, 1, 2$, such that $R_1 \oplus R_2 = 0$, $R_1 \oplus R_1 = R_2$ and $R_2 \oplus R_2 = R_1$. Set $w_c^0 = w_a^0 \oplus w_b^0$, and $\forall i \in \{a, b, c\}$: $w_i^1 = w_i^0 \oplus R_1$ and $w_i^2 = w_i^0 \oplus R_2$. Then, the garbled output can be simply obtained by summing the garbled gate inputs:

$$\begin{aligned} w_c^0 &= w_a^0 \oplus w_b^0 = (w_a^0 \oplus R_1) \oplus (R_2 \oplus w_b^0) = w_a^1 \oplus w_b^2 = (w_a^0 \oplus R_2) \oplus (R_1 \oplus w_b^0) = \\ &= w_a^2 \oplus w_b^1 \end{aligned}$$

$$\begin{aligned} w_c^1 &= w_c^0 \oplus R_1 = w_a^0 \oplus (w_b^0 \oplus R_1) = w_a^0 \oplus w_b^1 = (w_a^0 \oplus R_1) \oplus w_b^0 = w_a^1 \oplus w_b^0 = \\ &= (w_a^0 \oplus R_1 \oplus R_1) \oplus (R_2 \oplus w_b^0) = w_a^2 \oplus w_b^0 \end{aligned}$$

$$\begin{aligned} w_c^2 &= w_c^0 \oplus R_2 = w_a^0 \oplus (w_b^0 \oplus R_2) = w_a^0 \oplus w_b^2 = (w_a^0 \oplus R_2) \oplus w_b^0 = w_a^2 \oplus w_b^0 = \\ &= (w_a^0 \oplus R_2 \oplus R_2) \oplus (R_1 \oplus w_b^0) = w_a^1 \oplus w_b^1 \end{aligned}$$

5.2 A Multiple Valued Logic Approach for the Synthesis of Garbled Circuits

We submitted the work presented in this Section to IEEE Transactions on Information Forensics & Security. In this paper we focus on the Garbled Circuit technique and explore the possibility to extend the design and the representation of the circuits considering *multiple valued logic*, that is a generalization of the Boolean logic [32, 57]. Indeed, instead of considering $\{0, 1\}$ values, we allow variables to assume values in the finite domain $P = \{0, 1, \dots, |P| - 1\}$. Reasoning in the extended domain, usually allows a more compact representation of the circuit and in a more efficient evaluation. The process of transforming values from one domain to the other is called *encoding*, and in some cases it allows some optimizations that directly affect the overall efficiency of the garbling process [43].

In particular, we first focus on *3-valued logic (3VL)*, where variables assume values in $\{0, 1, 2\}$ and operations are consequently defined. One of our main goals is to find “good” encodings of 3-valued logic that allow efficient garbling of the associated Boolean representation. We study the possible encodings, in the same spirit of the approach described in Lindell et Al. [80], but we achieve different results. The overall contribution of our work can be summarized as follows:

- We explore alternative 3-valued logic representations for garbling circuits;
- We define different encodings from our construction in the 3VL setting to the Boolean logic, elaborating a comparison with the state of the art paradigms;
- We give an exhaustive description of the obtained results, defining also some metrics to choose the best encodings;
- We show that some optimizations for garbling work also in the multiple valued logic, focusing on the *xor*-free property and proving its validation in our multiple-valued logic setting;
- We define the new *Mixed Logic*, that allows to drastically reduce the costs needed to garble a 3VL circuit;
- We present an exhaustive case in which our proposed encodings and logic are applied to a well known circuit, analysing the more convenient solution in the costs’ context.

5.2.1 Multiple Valued approaches: a comparison

In classical Boolean logic, the truth values are generally denoted by *True* and *False*, which correspond to the Boolean Algebra elements 1 and 0, respectively. It is possible to extend this algebra including a third element and creating a 3-valued logic. The semantic values set can be $\{False, True, Unknown\}$ or $\{0, 1, 2\}$, a subset of \mathbb{N} . We focus our attention on two different extensions, namely the approach in [80] that we call FTU, and the approach in [43] that we call 012. These approaches have been presented in Sections 5.1.

It is possible to develop a comparison between these two approaches, matching the logic elements through the following bijective map:

$$\begin{aligned}
 f : \{False, Unknown, True\} &\rightarrow \{0, 1, 2\} \\
 False &\mapsto 0 \\
 Unknown &\mapsto 1 \\
 True &\mapsto 2
 \end{aligned} \tag{5.2}$$

This map implies that all the coded operations with three values are the same in the two approaches, except the *xor* definitions (see the definitions of \oplus_3 and \oplus'_3 in Figures 5.9 and 5.5). Indeed, we can note that $1 \oplus_3 1 = 2$ and $1 \oplus_3 2 = 0$, but $U \oplus'_3 U = U$ and $U \oplus'_3 T = U$.

5.2.1.1 Free *xor* gates

As already discussed in Section 5.1, in order to reduce the cost of a garbled Boolean circuit, paper [78] defines a new starting point that allows to reduce drastically the amount of shared information exploiting *xor* gates. In the multiple valued context, papers [43] and [80] analyze *xor* gates, leading to two different conclusions.

Indeed, the 012 approach leads to define the operation \oplus_3 in $P_3 = \{0, 1, 2\}$ that allows a free evaluation of it in garbled circuits. On the other hand, FTU approach does not allow any free evaluation of the proposed \oplus'_3 operation, as described in [80].

5.2.1.2 Encodings in the FTU approach

In [80] the authors show a method that implies first an encoding function from 3VL to the Boolean domain, the application of the state of the art garbling schemes for Boolean functions [97, 124], and then a decoding from Boolean to 3VL. We present it in Section 5.1.2.

Until now, the encoding from 3VL to the Boolean domain has been studied only in the context of FTU approach. We discuss now, for the first time, encodings for the 012 approach.

5.2.2 Encodings in the 012 approach

Table 5.6: A comparison between multiple valued approaches.

Method	MV approach for Secure TPC	Free xor gates	MV logic	Encoding	Mixed encoding
012	yes	yes	three valued	no	no
FTU	yes	no	three valued	yes	no
our method	yes	yes	multiple valued	yes	yes

5.2.2 Encodings in the 012 approach

In order to compare the FTU and 012 approaches, in this section we study all the possible encodings for the 012 approach. First of all, we observe that the encoding steps are the ones described in Figure 5.6, but for the 012 approach the 3VL set is $\{0, 1, 2\}$ (instead of $\{T, F, U\}$). Second, we can notice that there are 24 possible functional encodings $tr_{3 \rightarrow 2}$, which are listed in Table 5.7. In this table, $P_3 = \{0, 1, 2\}$ and $0_L, 0_R, 1_L, 1_R, 2_L, 2_R \in P_2 = \{0, 1\}$.

For each $tr_{3 \rightarrow 2}$, our aim is to identify the transformations in T_F for the three operations \oplus_3, \wedge_3 and \neg_3 , such that \oplus_2, \wedge_2 and \neg_2 have the minimum number of \wedge s. We work on this step making use of logic synthesis toolbox presented in [114], which minimizes the number of \wedge s in a logic network composed of \wedge, \oplus and \neg gates. For small Boolean functions (i.e., with few input variables), this tool gives the exact result with the minimum number of \wedge s. Indeed, for Boolean function with at most six inputs, the database for logic rewriting in [114] reports the exact representation in terms of multiplicative complexity, because based on [111, 35]. Since we are working with functions with at most four inputs, this allows us to find the exact transformations with the less multiplicative complexity for all of them.

In detail, for each $tr_{3 \rightarrow 2}$ in Table 5.7 and for each operation $\star_3 \in F_3$, we define the correspondent $\star_2 \in F_2$ such that, if $z = \star_3(x, y)$ with $x, y, z \in \{0, 1, 2\}$, then

$$tr_{3 \rightarrow 2}(z)^{side} = \star_2^{side}(tr_{3 \rightarrow 2}(x), tr_{3 \rightarrow 2}(y))$$

where *side* is equal to L or R . This step is computed through the ABC software [31], which, starting from the truth table of a Boolean function, creates a Boolean circuit implementing it. Then, we minimize the number of \wedge s in all these circuits thanks to the tool in [114]. The outcomes of this process lead us to the following results.

Result 1. For all $tr_{3 \rightarrow 2}$, it is always possible to define $\neg_2 = (\neg_2^L, \neg_2^R)$ (image of \neg_3) without any \wedge .

Result 2. For all $tr_{3 \rightarrow 2}$, there exists an $\wedge_2 = (\wedge_2^L, \wedge_2^R)$ (image of \wedge_3) such that both \wedge_2^L and \wedge_2^R have only one \wedge . (The total number of \wedge gates is 2.)

Table 5.7: $Tr_{3 \rightarrow 2}$ functions, describing all functional encoding in the 012 approach.

$\{0, 1, 2\}$	\mapsto	$\{$	$(0_L, 0_R),$	$(1_L, 1_R),$	$(2_L, 2_R)$	$\}$
$\{0, 1, 2\}$	\mapsto	$\{$	$(0, 0),$	$(0, 1),$	$(1, 0)$	$\}$
$\{0, 1, 2\}$	\mapsto	$\{$	$(0, 0),$	$(0, 1),$	$(1, 1)$	$\}$
$\{0, 1, 2\}$	\mapsto	$\{$	$(0, 0),$	$(1, 0),$	$(0, 1)$	$\}$
$\{0, 1, 2\}$	\mapsto	$\{$	$(0, 0),$	$(1, 0),$	$(1, 1)$	$\}$
$\{0, 1, 2\}$	\mapsto	$\{$	$(0, 0),$	$(1, 1),$	$(0, 1)$	$\}$
$\{0, 1, 2\}$	\mapsto	$\{$	$(0, 0),$	$(1, 1),$	$(1, 0)$	$\}$
$\{0, 1, 2\}$	\mapsto	$\{$	$(0, 1),$	$(0, 0),$	$(1, 0)$	$\}$
$\{0, 1, 2\}$	\mapsto	$\{$	$(0, 1),$	$(0, 0),$	$(1, 1)$	$\}$
$\{0, 1, 2\}$	\mapsto	$\{$	$(0, 1),$	$(1, 0),$	$(0, 0)$	$\}$
$\{0, 1, 2\}$	\mapsto	$\{$	$(0, 1),$	$(1, 0),$	$(1, 1)$	$\}$
$\{0, 1, 2\}$	\mapsto	$\{$	$(0, 1),$	$(1, 1),$	$(0, 0)$	$\}$
$\{0, 1, 2\}$	\mapsto	$\{$	$(0, 1),$	$(1, 1),$	$(1, 0)$	$\}$
$\{0, 1, 2\}$	\mapsto	$\{$	$(1, 0),$	$(0, 0),$	$(0, 1)$	$\}$
$\{0, 1, 2\}$	\mapsto	$\{$	$(1, 0),$	$(0, 0),$	$(1, 1)$	$\}$
$\{0, 1, 2\}$	\mapsto	$\{$	$(1, 0),$	$(0, 1),$	$(0, 0)$	$\}$
$\{0, 1, 2\}$	\mapsto	$\{$	$(1, 0),$	$(0, 1),$	$(1, 1)$	$\}$
$\{0, 1, 2\}$	\mapsto	$\{$	$(1, 0),$	$(1, 1),$	$(0, 0)$	$\}$
$\{0, 1, 2\}$	\mapsto	$\{$	$(1, 0),$	$(1, 1),$	$(0, 1)$	$\}$
$\{0, 1, 2\}$	\mapsto	$\{$	$(1, 1),$	$(0, 0),$	$(0, 1)$	$\}$
$\{0, 1, 2\}$	\mapsto	$\{$	$(1, 1),$	$(0, 0),$	$(1, 0)$	$\}$
$\{0, 1, 2\}$	\mapsto	$\{$	$(1, 1),$	$(0, 1),$	$(0, 0)$	$\}$
$\{0, 1, 2\}$	\mapsto	$\{$	$(1, 1),$	$(0, 1),$	$(1, 0)$	$\}$
$\{0, 1, 2\}$	\mapsto	$\{$	$(1, 1),$	$(1, 0),$	$(0, 0)$	$\}$
$\{0, 1, 2\}$	\mapsto	$\{$	$(1, 1),$	$(1, 0),$	$(0, 1)$	$\}$

5.2.2 Encodings in the 012 approach

Result 3. For all $tr_{3 \rightarrow 2}$, there exists an unique $\oplus_2 = (\oplus_2^L, \oplus_2^R)$ (image of \oplus_3) such that both \oplus_2^L and \oplus_2^R have only one \wedge . (The total number of \wedge gates is 2.)

This means that, for each functional encoding, both \wedge_3 and \oplus_3 operations in the 3VL can be transformed in some Boolean logic expressions with at least two \wedge . That has consequence in how much is advantageous to transform a circuit (or a gate) from the 3VL to the Boolean logic, because a comparison between $\wedge_3(\oplus_3)$ cost and two \wedge cost in the Boolean logic is necessary.

Example 5.2.1 (An example of encoding in 012 approach). Considering the fourth encoding in Table 5.7, the transformation function is:

$$(x_L, x_R) = tr_{3 \rightarrow 2}(x) = \begin{cases} (0, 0) & x = 0 \\ (1, 0) & x = 1 \\ (1, 1) & x = 2 \end{cases} \quad (5.3)$$

1. Operation \neg_3 is translated in

$$Tr_F(\neg_3) = \neg_2(x_L, x_R) = (\neg x_R, \neg x_L)$$

2. Operation \wedge_3 is translated in

$$\begin{aligned} Tr_F(\wedge_3) &= \wedge_2(x_L, x_R, y_L, y_R) \\ &= (\wedge_2^L(x_L, x_R, y_L, y_R), \wedge_2^R(x_L, x_R, y_L, y_R)) \end{aligned}$$

There are three possible \wedge_2^L with only one *and*:

- $\wedge_2^L(x_L, x_R, y_L, y_R) = x_L \wedge y_L$
- $\wedge_2^L(x_L, x_R, y_L, y_R) = \neg y_L \wedge (x_L \oplus y_R) \oplus x_L$
- $\wedge_2^L(x_L, x_R, y_L, y_R) = x_L \wedge (x_R \oplus y_L) \oplus x_R$

There are three possible \wedge_2^R with only one *and*:

- $\wedge_2^R(x_L, x_R, y_L, y_R) = x_R \wedge y_R$
- $\wedge_2^R(x_L, x_R, y_L, y_R) = y_R \wedge \neg(x_R \oplus y_L)$
- $\wedge_2^R(x_L, x_R, y_L, y_R) = x_R \wedge \neg(x_L \oplus y_R)$

3. Operation \oplus_3 is translated in

$$\begin{aligned} Tr_F(\oplus_3) &= \oplus_2(x_L, x_R, y_L, y_R) \\ &= (\oplus_2^L(x_L, x_R, y_L, y_R), \oplus_2^R(x_L, x_R, y_L, y_R)) \end{aligned}$$

where $\oplus_2^L(x_L, x_R, y_L, y_R) = \neg(\neg(x_L \oplus y_L) \wedge \neg(x_R \oplus y_L \oplus y_R))$ and $\oplus_2^R(x_L, x_R, y_L, y_R) = (x_L \oplus y_R) \wedge (x_R \oplus y_L)$

Table 5.8: 012 approach: cost of translated \oplus_3 operation in the Boolean logic, in case of the example functions.

Function	cost
\neg_3	$2 \cdot \neg$
\wedge_3	$2 \cdot \wedge$
\oplus_3	$2 \cdot \wedge + 5 \cdot \oplus + 3 \cdot \neg$

The cost for each operation is reported in Table 5.8. For \wedge_3 we choose \wedge_2^L and \wedge_2^R without any \oplus .

5.2.2.1 Encodings' equivalences

The encodings in Table 5.7 can be divided into couples, such that for each couple the following Proposition holds.

Proposition 5.2.2. *Let $\{(0_L, 0_R), (1_L, 1_R), (2_L, 2_R)\}$ and $\{(0_L^*, 0_R^*), (1_L^*, 1_R^*), (2_L^*, 2_R^*)\}$ be two encodings.*

*Let \star_3 be an operation in the set $\{\wedge_3, \oplus_3, \neg_3\}$, and (\star_2^L, \star_2^R) the couple of transformations from 3VL to Boolean logic for the first encoding, $(\star_2^{*L}, \star_2^{*R})$ the couple of transformations for the second encoding.*

If, for the two encodings and $\forall a \in \{0, 1, 2\}$, the following equalities hold:

$$a_L^* = a_R \quad (5.4)$$

$$a_R^* = a_L \quad (5.5)$$

*then the couples of transformations (\star_2^L, \star_2^R) and $(\star_2^{*L}, \star_2^{*R})$ are such that*

$$\star_2^{*L}(x_L, x_R, y_L, y_R) = \star_2^R(x_R, x_L, y_R, y_L) \quad (5.6)$$

$$\star_2^{*R}(x_L, x_R, y_L, y_R) = \star_2^L(x_R, x_L, y_R, y_L) \quad (5.7)$$

Proof. Equations 5.4 and 5.5 imply that, for the two considered encodings,

$$(0_L^*, 0_R^*) = (0_R, 0_L)$$

$$(1_L^*, 1_R^*) = (1_R, 1_L)$$

$$(2_L^*, 2_R^*) = (2_R, 2_L).$$

5.2.2 Encodings in the 012 approach

Then, if for the encoding $\{(0_L, 0_R), (1_L, 1_R), (2_L, 2_R)\}$ the transformation $TrF(\star_3)$ is described through the equations

$$\begin{aligned} z_L &= \star_2^L(x_L, x_R, y_L, y_R) \\ z_R &= \star_2^R(x_L, x_R, y_L, y_R) \end{aligned}$$

then for the encoding $\{(0_L^*, 0_R^*), (1_L^*, 1_R^*), (2_L^*, 2_R^*)\}$ the transformation $TrF(\star_3^*)$ is such that

$$\begin{aligned} z_L &= \star_2^{*L}(x_L, x_R, y_L, y_R) = \star_2^R(x_R, x_L, y_R, y_L) \\ z_R &= \star_2^{*R}(x_L, x_R, y_L, y_R) = \star_2^L(x_R, x_L, y_R, y_L) \end{aligned}$$

□

Example 5.2.3 (Encodings' couple). The first and third encodings in Table 5.7, denoted as $\{(0, 0), (0, 1), (1, 0)\}$ and $\{(0, 0), (1, 0), (0, 1)\}$ respectively, are such that $(0_L, 0_R) = (0_R^*, 0_L^*)$, $(1_L, 1_R) = (1_R^*, 1_L^*)$ and $(2_L, 2_R) = (2_R^*, 2_L^*)$. For the first encoding, \neg_3 operation is transformed as follows:

$$\begin{aligned} \neg_2^L(x_L, x_R) &= \neg(x_L \oplus x_R) \\ \neg_2^R(x_L, x_R) &= x_R \end{aligned}$$

while, for the second encoding:

$$\begin{aligned} \neg_2^{*L}(x_L, x_R) &= x_L = \neg_2^R(x_R, x_L) \\ \neg_2^{*R}(x_L, x_R) &= \neg(x_R \oplus x_L) = \neg_2^L(x_R, x_L) \end{aligned}$$

For the first encoding, \wedge_3 operation is transformed as follows:

$$\begin{aligned} \wedge_2^L(x_L, x_R, y_L, y_R) &= x_L \wedge y_L \\ \wedge_2^R(x_L, x_R, y_L, y_R) &= (x_R \oplus y_R) \wedge \neg(x_L \oplus y_L \oplus y_R) \oplus x_R \end{aligned}$$

while, for the second encoding:

$$\begin{aligned} \wedge_2^{*L}(x_L, x_R, y_L, y_R) &= (x_L \oplus y_L) \wedge \neg(x_L \oplus y_R \oplus y_L) \oplus x_L \\ &= \wedge_2^R(x_R, x_L, y_R, y_L) \\ \wedge_2^{*R}(x_L, x_R, y_L, y_R) &= x_R \wedge y_R \\ &= \wedge_2^L(x_R, x_L, y_R, y_L) \end{aligned}$$

For the first encoding, \oplus_3 operation is transformed as follows:

$$\begin{aligned} \oplus_2^L(x_L, x_R, y_L, y_R) &= \neg(x_R \oplus y_R) \wedge (x_L \oplus y_L \oplus y_R) \\ \oplus_2^R(x_L, x_R, y_L, y_R) &= \neg(x_L \oplus y_L) \wedge (x_R \oplus y_L \oplus y_R) \end{aligned}$$

while, for the second encoding:

$$\begin{aligned}\oplus_2^{*L}(x_L, x_R, y_L, y_R) &= \neg(x_R \oplus y_R) \wedge (x_L \oplus y_R \oplus y_L) \\ &= \oplus_2^R(x_R, x_L, y_R, y_L) \\ \oplus_2^{*R}(x_L, x_R, y_L, y_R) &= \neg(x_L \oplus y_L) \wedge (x_R \oplus y_R \oplus y_L) \\ &= \oplus_2^L(x_R, x_L, y_R, y_L)\end{aligned}$$

5.2.2.2 More convenient encodings

In previous sections we show that, for each functional encoding $tr_{3 \rightarrow 2}$ and for all operations \oplus_3 , \wedge_3 and \neg_3 , there is at least one transformation in T_F such that the number of \wedge in \oplus_2 , \wedge_2 , \neg_2 is minimized. Among them, we are also interested to define which encoding $tr_{3 \rightarrow 2}$ requires the minimum number of \neg and \oplus . Indeed, even if \oplus and \neg gates cost 0 in the garbled circuits context, it would be better to reduce the overall number of gates. In Tables 5.9, 5.10 and 5.11 the number of Boolean gates needed to describe each operation is reported. The last column shows the total number of gates needed for both left and right encoding. For some encodings, there are more expressions that define the left or right part of an operation in Boolean logic with only one \wedge . In these cases, among them we consider the expression containing the minimum number of \oplus and \neg .

Depending on the number and type of gates in a 3VL function, we choose an encoding rather than another one, to ensure not only a low number of \wedge , but also the minimum number of \oplus and \neg (see Section 5.2.4).

Remark 5.2.4. In Tables 5.9, 5.10 and 5.11 stands out that all the encodings can be divided into couples, such that the encodings in each couple have the same number of \oplus and \neg in all the Tables. This feature descends directly from the considerations pointed out in Proposition 5.2.2.

Example 5.2.5 (Encodings' couple - same number of \neg and \oplus). For this Example, we consider the same encodings presented in Example 5.2.3, i.e., $\{(0, 0), (0, 1), (1, 0)\}$ and $\{(0, 0), (1, 0), (0, 1)\}$. From Tables 5.9, 5.10 and 5.11, we recover that for both of them:

- in transformation \neg_2 , the total number of \neg is 1 and of \oplus is 1,
- in transformation \wedge_2 the total number of \neg is 1 and of \oplus is 4,
- in transformation \oplus_2 the total number of \neg is 2 and of \oplus is 6.

5.2.2 Encodings in the 012 approach

Table 5.9: Boolean gates needed to compute the operation \wedge_2 for every functional encoding.

	\wedge_2^L			\wedge_2^R			\wedge_2		
	\wedge	\oplus	\neg	\wedge	\oplus	\neg	\wedge	\oplus	\neg
000110	1	0	0	1	4	1	2	4	1
000111	1	0	0	1	0	0	2	0	0
001001	1	4	1	1	0	0	2	4	1
001011	1	0	0	1	0	0	2	0	0
001101	1	4	0	1	0	0	2	4	0
001110	1	0	0	1	4	0	2	4	0
010010	1	0	0	1	0	3	2	0	3
010011	1	0	0	1	4	0	2	4	0
011000	1	4	0	1	0	3	2	4	3
011011	1	0	0	1	4	1	2	4	1
011100	1	4	1	1	0	3	2	4	4
011110	1	0	0	1	0	3	2	0	3
100001	1	0	3	1	0	0	2	0	3
100011	1	4	0	1	0	0	2	4	0
100100	1	0	3	1	4	0	2	4	3
100111	1	4	1	1	0	0	2	4	1
101100	1	0	3	1	4	1	2	4	4
101101	1	0	3	1	0	0	2	0	3
110001	1	0	3	1	4	1	2	4	4
110010	1	4	1	1	0	3	2	4	4
110100	1	0	3	1	0	3	2	0	6
110110	1	4	0	1	0	3	2	4	3
111000	1	0	3	1	0	3	2	0	6
111001	1	0	3	1	4	0	2	4	3

Table 5.10: Boolean gates needed to compute the operation \oplus_2 for every functional encoding.

	\oplus_2^L			\oplus_2^R			\oplus_2		
	\wedge	\oplus	\neg	\wedge	\oplus	\neg	\wedge	\oplus	\neg
000110	1	3	1	1	3	1	2	6	2
000111	1	2	0	1	3	3	2	5	3
001001	1	3	1	1	3	1	2	6	2
001011	1	3	3	1	2	0	2	5	3
001101	1	2	0	1	3	3	2	5	3
001110	1	3	3	1	2	0	2	5	3
010010	1	2	2	1	3	1	2	5	3
010011	1	3	2	1	3	2	2	6	4
011000	1	2	2	1	3	1	2	5	3
011011	1	3	3	1	2	3	2	5	6
011100	1	3	2	1	3	2	2	6	4
011110	1	3	3	1	2	3	2	5	6
100001	1	3	1	1	2	2	2	5	3
100011	1	3	2	1	3	2	2	6	4
100100	1	3	1	1	2	2	2	5	3
100111	1	2	3	1	3	3	2	5	6
101100	1	3	2	1	3	2	2	6	4
101101	1	2	3	1	3	3	2	5	6
110001	1	3	1	1	2	1	2	5	2
110010	1	2	1	1	3	1	2	5	2
110100	1	3	1	1	2	1	2	5	2
110110	1	3	3	1	3	3	2	6	6
111000	1	2	1	1	3	1	2	5	2
111001	1	3	3	1	3	3	2	6	6

5.2.2 Encodings in the 012 approach

Table 5.11: Boolean gates needed to compute the operation \neg_2 for every functional encoding.

	\neg_2^L			\neg_2^R			\neg_2		
	\wedge	\oplus	\neg	\wedge	\oplus	\neg	\wedge	\oplus	\neg
000110	0	1	1	0	0	0	0	1	1
000111	0	0	1	0	0	1	0	0	2
001001	0	0	0	0	1	1	0	1	1
001011	0	0	1	0	0	1	0	0	2
001101	0	0	0	0	1	1	0	1	1
001110	0	1	1	0	0	0	0	1	1
010010	0	0	0	0	0	0	0	0	0
010011	0	1	0	0	0	0	0	1	0
011000	0	0	0	0	1	1	0	1	1
011011	0	1	0	0	0	0	0	1	0
011100	0	0	0	0	1	1	0	1	1
011110	0	0	0	0	0	0	0	0	0
100001	0	0	0	0	0	0	0	0	0
100011	0	0	0	0	1	0	0	1	0
100100	0	1	1	0	0	0	0	1	1
100111	0	0	0	0	1	0	0	1	0
101100	0	1	1	0	0	0	0	1	1
101101	0	0	0	0	0	0	0	0	0
110001	0	1	0	0	0	0	0	1	0
110010	0	0	0	0	1	0	0	1	0
110100	0	0	1	0	0	1	0	0	2
110110	0	0	0	0	1	0	0	1	0
111000	0	0	1	0	0	1	0	0	2
111001	0	1	0	0	0	0	0	1	0

5.2.3 Free *xor* Evaluation in Multiple Valued Logic

From work by Kolesnikov et Al. [78], in which free transfer for *xor* gates in GC protocol is established, to find some improvements in this sense has been a fruitful area. Indeed, in a later work by Cimato et Al. [43], the authors declare that also transfer *xors* in the 3-valued logic with 012 approach has no cost. Now, we finally prove that this holds for any multiple valued logic with truth values in the set $P = \{0, 1, \dots, |P| - 1\}$.

Theorem 5.2.6. *Let G have two input wires w_a and w_b and output wire w_c . Garble the wire values as follows: randomly choose $w_a^0, w_b^0, \{R_1, \dots, R_{n-1}\}$ and $i, j, h \in \mathbb{Z}_n$, with the properties that*

- $R_i \oplus R_j = 0$, if $[i + j]_{\text{mod}(n)} = 0$
- $R_i \oplus R_j = R_m$, otherwise, where $m = [i + j]_{\text{mod}(n)}$

Set $w_c^0 = w_a^0 \oplus w_b^0$, and $\forall k \in (a, b, c)$: $w_k^m = w_k^0 \oplus R_m$.

It is easy to see that the garbled gate output is simply obtained by Xoring garbled gate inputs:

- if $m = 0$ then

$$w_c^0 = w_a^0 \oplus w_b^0 = (w_a^0 \oplus R_i) \oplus (w_b^0 \oplus R_j) = w_a^i \oplus w_b^j$$

where $[i + j]_{\text{mod}(n)} = 0$

- if $m \neq 0$ then

$$\begin{aligned} w_c^m &= w_c^0 \oplus R_m = w_a^0 \oplus w_b^0 \oplus R_m = \\ &= (w_a^0 \oplus R_i) \oplus (w_b^0 \oplus R_j) = w_a^i \oplus w_b^j = \\ &= (w_a^0 \oplus R_j) \oplus (w_b^0 \oplus R_i) = w_a^j \oplus w_b^i \end{aligned}$$

where $[i + j]_{\text{mod}(n)} = m$

Example 5.2.7 ($n = 6$). Let $P_6 = \{0, 1, 2, 3, 4, 5\}$ be the set of multi-valued variables with $n = 6$, and \oplus_6 the *xor* function in it. The \oplus_6 inference rules are shown in Table 5.12.

We give now some examples about how it is possible to garble for free the *xor* gates also in a multiple valued logic (in this case, a 6-valued logic), as proved in Theorem 5.2.6.

Table 5.12: Definition of \oplus_6 using a truth table.

\oplus_6	0	1	2	3	4	5
0	0	1	2	3	4	5
1	1	2	3	4	5	0
2	2	3	4	5	0	1
3	3	4	5	0	1	2
4	4	5	0	1	2	3
5	5	0	1	2	3	4

If $m = 0$ then:

$$\begin{aligned}
 w_c^0 &= w_a^0 \oplus_6 w_b^0 = (w_a^0 \oplus_6 R_1) \oplus_6 (w_b^0 \oplus_6 R_5) = w_a^1 \oplus_6 w_b^5 \\
 &= (w_a^0 \oplus_6 R_2) \oplus_6 (w_b^0 \oplus_6 R_4) = w_a^2 \oplus_6 w_b^4 \\
 &= (w_a^0 \oplus_6 R_3) \oplus_6 (w_b^0 \oplus_6 R_3) = w_a^3 \oplus_6 w_b^3 \\
 &= (w_a^0 \oplus_6 R_4) \oplus_6 (w_b^0 \oplus_6 R_2) = w_a^4 \oplus_6 w_b^2 \\
 &= (w_a^0 \oplus_6 R_5) \oplus_6 (w_b^0 \oplus_6 R_1) = w_a^5 \oplus_6 w_b^1
 \end{aligned}$$

If $m \neq 0$, let us take $m = 2$, as an instance:

$$\begin{aligned}
 w_c^2 &= w_c^0 \oplus_6 R_2 = (w_a^0 \oplus_6 R_0) \oplus_6 (w_b^0 \oplus_6 R_2) = w_a^0 \oplus_6 w_b^2 \\
 &= (w_a^0 \oplus_6 R_1) \oplus_6 (w_b^0 \oplus_6 R_1) = w_a^1 \oplus_6 w_b^1 \\
 &= (w_a^0 \oplus_6 R_2) \oplus_6 (w_b^0 \oplus_6 R_0) = w_a^2 \oplus_6 w_b^0 \\
 &= (w_a^0 \oplus_6 R_3) \oplus_6 (w_b^0 \oplus_6 R_5) = w_a^3 \oplus_6 w_b^5 \\
 &= (w_a^0 \oplus_6 R_5) \oplus_6 (w_b^0 \oplus_6 R_3) = w_a^5 \oplus_6 w_b^3 \\
 &= (w_a^0 \oplus_6 R_4) \oplus_6 (w_b^0 \oplus_6 R_4) = w_a^4 \oplus_6 w_b^4
 \end{aligned}$$

5.2.4 From 3VL to Boolean logic: costs comparison

The results in [43], generalized in Section 5.2.3, show that garbling a \oplus_3 gate in the 012 approach has no cost. This allows to conclude that, in the 3VL with 012 approach, garbling \oplus_3 costs 0 and \wedge_3 costs 8 rows of the garbled table [97].

In Section 5.2.2 we show that, for each of the 24 functional encodings in Table 5.7, the operations \wedge_3 and \oplus_3 can be transformed in the functions \wedge_2 and \oplus_2 with two \wedge both (at least).

Given these assumptions, we note that it is more convenient to garble \oplus_3 in the 3VL, since this does not imply any cost, while it is better to translate \wedge_3 from the 3VL (where it costs 8 rows) to Boolean logic (where it costs 6 rows, 3 for each \wedge in \wedge_2).

Since the garbled protocol is a software implementation, it implies the concept of circuits but does not involve the physical construction of them. Moreover, in the garbled protocol all garbled tables are generated and transferred from A player to B player *one-by-one*. Starting from this assumption, we give a definition of a new logic.

Definition 5.2.8 (Mixed Logic (ML)). The *Mixed Logic (ML)* is a logic in which some gates are represented in the 3VL, and some other in the Boolean logic. In particular, for our purpose, we work with \oplus_3 and \neg_3 in the 3VL, while we translate \wedge_3 in its Boolean logic form \wedge_2 .

Summarizing, when the A player has to garble a 3VL function, in the ML for each gate she computes the following:

- if the operation is \oplus_3 or \neg_3 , she garbles the gate in the 3VL, since she can send its garbled table for free;
- if the operation is \wedge_3 , she chooses a cheap encoding from the 3VL to the Boolean logic and transforms the gate in its Boolean form \wedge_2 , decreasing the cost to send the correspondent garbled table.

Starting from this analysis, in this Section we define the cost functions in the three possible scenarios: 1) garbling the circuit in the 3VL, 2) garbling the circuit in the Boolean logic, 3) garbling the circuit in the Mixed Logic.

Definition 5.2.9 (Cost functions). The cost necessary to transfer each gate in the garble circuit protocol is the number of rows in the corresponding garbled table.

The number of rows for each gate in the three scenarios are reported in Table 5.13.

Example 5.2.10 (garbling costs). Consider the 3VL function f :

$$y = f(x_1, x_2, x_3) = x_1 \oplus_3 (\neg_3 x_2 \wedge_3 x_3).$$

where $x_1, x_2, x_3 \in \{0, 1, 2\}$. It is easy to define the correspondent circuit in 3VL (Figure 5.10).

1. *Cost of garbling the circuit in 3VL.* The cost of the garbled protocol in 3VL is the sum of the costs of all the gates, i.e., 8 rows (first column in Table 5.14).

5.2.4 From 3VL to Boolean logic: costs comparison

Table 5.13: Number of rows in the garbled tables for each gate (columns) in all the three scenarios (rows).

	\oplus	\wedge	\neg
Boolean logic	0	3	0
3-Valued logic	0	8	0
Mixed Logic	0	6	0

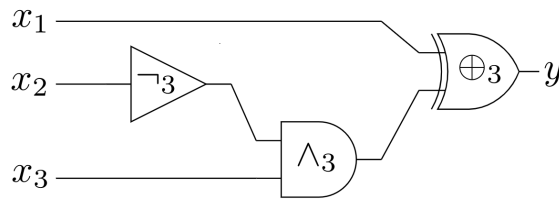


Figure 5.10: Synthesis of function f in a 3VL circuit.

2. *Cost of garbling the circuit in Boolean logic.* This time, we transform all the gates from 3VL to Boolean logic through the encoding $tr_{3 \rightarrow 2}$ in Eq. 5.3 (Figure 5.11). From the example in Section 5.2.1, we know that translation of \neg_3 cost as two \neg , \wedge_3 as two \wedge and \oplus_3 as two \wedge , five \oplus and three \neg (see Table 5.8). In this case, garbling the circuit costs a total of 12 rows (second column in Table 5.14).
3. *Cost of garbling the circuit in the Mixed Logic.* In this case, we apply $tr_{3 \rightarrow 2}$ only to \wedge_3 , since this is the single case when the transformation produces a gain. Garbling the circuit costs 6 rows, and it is clear that this case is the one with minimum cost (third column in Table 5.14).

Table 5.14: Comparison of costs.

	3VL	Bool. logic	ML
\oplus_3	0	6	0
\wedge_3	8	6	6
\neg_3	0	0	0
Tot.	8	12	6

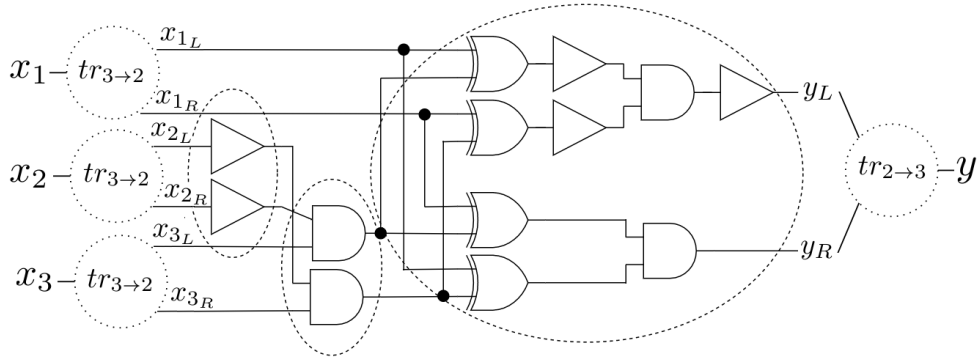


Figure 5.11: Synthesis of function f in the Boolean logic, after the application of the encoding in Eq. 5.3. In each dashed circle, there is an operation among \neg_2 , \wedge_2 or \oplus_2 .

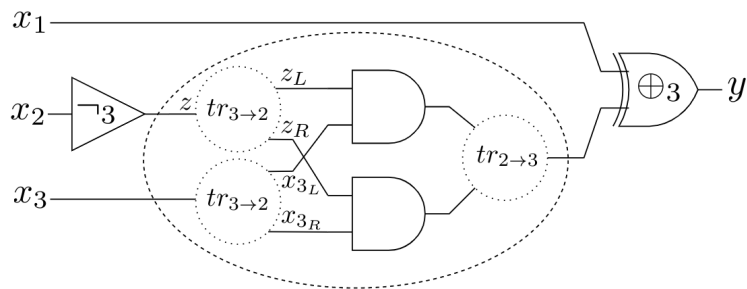


Figure 5.12: Synthesis of function f in the Mixed logic, after the application of the encoding in Eq. 5.3 only for the gate \wedge_3 , that is transformed into \wedge_2 (dashed circle).

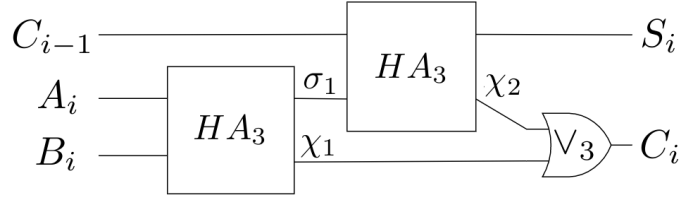


Figure 5.13: Full Adder circuit: function FA_3 has as inputs C_{i-1} , A_i and B_i , and as outputs S_i and C_i .

5.2.5 Applied case: Adder

In this section, we describe the Adder function in the 3VL, comparing the costs in the three scenarios (3VL, Boolean logic and Mixed Logic), considering different encodings.

Let $A = A_{n-1}A_{n-2}\dots A_0$ and $B = B_{n-1}B_{n-2}\dots B_0$ are such that $A_i, B_j \in \{0, 1, 2\} \forall i, j \in \{0, 1, \dots, n-1\}$.

The result of the Adder function is a sequence $S = S_{n-1}S_{n-2}\dots S_0$ of elements in the 3VL set. In computations we exploit the carry sequence $C = C_{n-1}C_{n-2}\dots C_0$ (assuming $C_{-1} = 0$).

We describe the adder in the 3VL through the Full Adder function FA_3 in Figure 5.13 and the Half Adder function HA_3 in Figure 5.14. In particular,

$$FA_3 : (A_i, B_i, C_{i-1}) \mapsto (S_i, C_i)$$

$$HA_3 : (\alpha, \beta) \mapsto (\sigma, \chi)$$

where

$$\sigma = \alpha \oplus_3 \beta$$

$$\chi = [(\neg_3(\alpha \oplus_3 \beta)) \wedge_3 (\alpha \vee_3 \beta)] \wedge_3 [(\neg_3(\alpha \oplus_3 \beta) \vee_3 (\alpha \vee_3 \beta)) \oplus_3 2]$$

and i refers to the i -th iteration of the function needed to complete the sum in 3VL.

Example 5.2.11. Let us consider the sum $15 + 17 = 32$ between two natural numbers in \mathbb{N} . In the 3VL, 15 corresponds to the sequence $A = 0120$ and 17 to the sequence $B = 0122$. To compute the sum between A and B , we perform the computations reported in table 5.15. We denote as σ_1 , χ_1 and χ_2 the intermediate values of FA_3 . The final result of computation is the sequence $S = 1012$, that corresponds to the natural number 32.

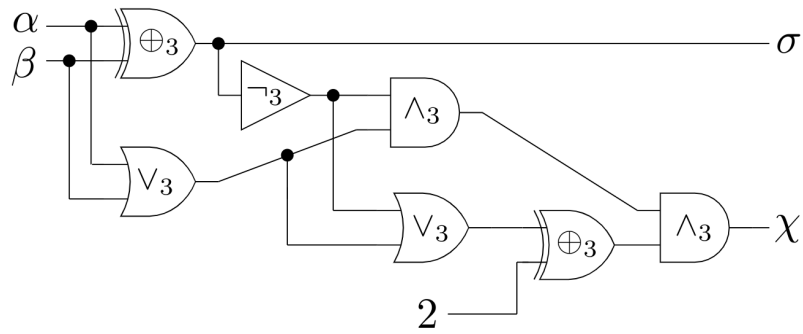


Figure 5.14: Half Adder circuit: function HA_3 has as inputs α and β , and as outputs σ and χ .

Table 5.15: Steps performed to sum the sequences $A = 0120$ and $B = 0122$.

i	A_i	B_i	σ_1	χ_1	χ_2	S_i	C_i
0	0	2	2	0	0	2	0
1	2	2	1	1	0	1	1
2	1	1	2	0	1	0	1
3	0	0	0	0	0	1	0

5.2.5.1 Costs for garbling the Full Adder circuit

Let us analyze and compare the cost for garbling the circuit in Fig 5.13, in all the three considered scenarios. For this purpose, since we are considering circuits on the basis $\{and, xor, not\}$, we replace any $x \vee_3 y$ with the equivalent formula $\neg_3(\neg_3 a \wedge_3 \neg_3 b)$.

1) *Cost of garbling the circuit in the 3VL:* in this case, \neg_3 and \oplus_3 have cost 0. In the HA_3 function there are 4 \wedge_3 , then the total cost to garble this function is $4 \cdot 8 = 32$ rows. In the FA_3 there are 2 HA_3 and \wedge_3 . Thus, the cost is $2 \cdot 32 + 8 = 72$ rows.

2) *Cost of garbling the circuit in the Boolean logic:* in this case, any minimal encoding we choose, the cost for both \oplus_2 and \wedge_2 is 6 rows (see Tables 5.10 and 5.9). In the HA_3 function there are 4 \wedge_2 and 2 \oplus_2 , then the total cost to garble this function is $4 \cdot 6 + 2 \cdot 6 = 36$. In the FA_3 there are 2 HA_3 and a \wedge_2 , then the total cost is $2 \cdot 36 + 6 = 78$ rows.

3) *Cost of garbling the circuit in the Mixed logic:* in order to compute the total cost, we sum only the cost of the \wedge_2 . In the HA_3 function there are 4 \wedge_2 , and then the cost is $4 \cdot 6 = 24$. In the FA_3 there are 2 HA_3 and a \wedge_2 , then the total cost is $2 \cdot 24 + 6 = 54$ rows.

5.2.5.2 More convenient encodings for the Full Adder

Also for the Adder function, among all the encodings that allow the minimum cost after the transformation of circuits from the 3VL to the Boolean logic, we are also interested to find those composed of the minimum number of gates (then also \neg and \oplus).

For this purpose, we give to all the gates a rate, i.e., for the \wedge gate we give 3 points (it costs in the garbling process), for the \oplus gate 2 points (it is for free in the garbling process, but it costs in sense of area), for the \neg gate 1 point (it is for free, and it is also small in sense of area).

In Table 5.16, for each encoding $tr_{3 \rightarrow 2}$ is reported the total points computed summing the points assigned to each gate, when the Adder function is completely translated from the 3VL to the Boolean logic.

In the same way as done in Section 5.2.2.2, if there are more expressions that define the left or right part of an operation in Boolean logic with only one \wedge , we consider the expression with also the less number of \oplus and \neg .

From Table 5.16 we note that, through our analysis, the best encodings that transform the Adder function from 3VL to Boolean logic are 001110 and 100001.

Remark 5.2.12. Note that in Table 5.16 the subdivision of encodings in couples (discussed in Section 5.2.2.1) is evident: indeed, the points assigned to the two encodings

Table 5.16: Total points awarded to the Half Adder and the Full Adder, for each encoding, following the rule: 1 point to a \neg gate, 2 points to a \oplus gate and 3 points to an \wedge gate.

Encoding	HA points	FA points
000110	121	266
000111	76	164
001001	121	266
001011	76	164
001101	115	253
001110	115	253
010010	74	157
010011	114	248
011000	127	280
011011	118	257
011100	137	301
011110	80	169
100001	74	157
100011	114	248
100100	127	280
100111	118	257
101100	137	301
101101	80	169
110001	118	259
110010	122	268
110100	98	214
110110	130	283
111000	98	214
111001	130	283

in a couple for both HA and FA are equal, because the Boolean operations in the transformed circuits are the same.

5.2.6 Appendix: Cheapest transformations for all the functional encodings

In this appendix we report, for all the functional encodings in Table 5.7, the transformations in T_F for the three operations \oplus_3 , \wedge_3 and \neg_3 , such that \oplus_2 , \wedge_2 and \neg_2 have the minimum number of \wedge (see Section 5.2.2).

Each functional encoding $tr_{2 \rightarrow 3}$ is notated as $0_L 0_R 1_L 1_R 2_L 2_R$, such that:

$$0_L 0_R 1_L 1_R 2_L 2_R : \{0, 1, 2\} \mapsto \{(0_L, 0_R), (1_L, 1_R), (2_L, 2_R)\}$$

where $0_L, 0_R, 1_L, 1_R, 2_L, 2_R \in P_2$.

Tables for \neg_3 . The operation $y = \neg_3 x$ in P_3 is transformed in the operation:

$$(y_L, y_R) = (\neg_2^L(x_L, x_R), \neg_2^R(x_L, x_R))$$

For each functional encoding, we write in Table 5.17 all the transformations $\neg_2 = (\neg_2^L, \neg_2^R)$ where both \neg_2^L and \neg_2^R have only one \wedge .

Tables for \wedge_3 . The operation $z = x \wedge_3 y$ in P_3 is transformed in the operation:

$$(z_L, z_R) = (\wedge_2^L(x_L, x_R, y_L, y_R), \wedge_2^R(x_L, x_R, y_L, y_R))$$

For each functional encoding, we write in Tables 5.18 - 5.19 all the transformations $\wedge_2 = (\wedge_2^L, \wedge_2^R)$ where both \wedge_2^L and \wedge_2^R have only one \wedge .

Tables for \oplus_3 . The operation $z = x \oplus_3 y$ in P_3 is transformed in the operation:

$$(z_L, z_R) = (\oplus_2^L(x_L, x_R, y_L, y_R), \oplus_2^R(x_L, x_R, y_L, y_R))$$

For each functional encoding, we write in Table 5.20 all the transformations $\oplus_2 = (\oplus_2^L, \oplus_2^R)$ where both \oplus_2^L and \oplus_2^R have only one \wedge .

5.3 Conclusion and further works

In this Chapter we present our work in the context of the Multiple-valued logic. We present an exhaustive comparison between two state-of-arts three-valued logics, proving that one of them, once extended to more then three truth elements, can

Table 5.17: Cheapest transformations for \neg_3 .

$0_L 0_R 1_L 1_R 2_L 2_R$	\neg_2
000110	$z_L = \neg(x_L \oplus x_R)$ $z_R = x_R$
000111	$z_L = \neg x_R$ $z_R = \neg x_L$
001001	$z_L = x_L$ $z_R = \neg(x_L \oplus x_R)$
001011	$z_L = \neg x_R$ $z_R = \neg x_L$
001101	$z_L = x_L$ $z_R = \neg(x_L \oplus x_R)$
001110	$z_L = \neg(x_L \oplus x_R)$ $z_R = x_R$
010010	$z_L = x_R$ $z_R = x_L$
010011	$z_L = x_L \oplus x_R$ $z_R = x_R$
011000	$z_L = x_L$ $z_R = \neg(x_L \oplus x_R)$
011011	$z_L = x_L \oplus x_R$ $z_R = x_R$
011100	$z_L = x_L$ $z_R = \neg(x_L \oplus x_R)$
011110	$z_L = x_R$ $z_R = x_L$
100001	$z_L = x_R$ $z_R = x_L$
100011	$z_L = x_L$ $z_R = x_L \oplus x_R$
100100	$z_L = \neg(x_L \oplus x_R)$ $z_R = x_R$
100111	$z_L = x_L$ $z_R = x_L \oplus x_R$
101100	$z_L = \neg(x_L \oplus x_R)$ $z_R = x_R$
101101	$z_L = x_R$ $z_R = x_L$
110001	$z_L = x_L \oplus x_R$ $z_R = x_R$
110010	$z_L = x_L$ $z_R = x_L \oplus x_R$
110100	$z_L = \neg x_R$ $z_R = \neg x_L$
110110	$z_L = x_L$ $z_R = x_L \oplus x_R$
111000	$z_L = \neg x_R$ $z_R = \neg x_L$
111001	$z_L = x_L \oplus x_R$ $z_R = x_R$

5.3. Conclusion and further works

Table 5.18: Cheapest transformations for \wedge_3 , for encodings with $0_L = 0$.

$0_L 0_R 1_L 1_R 2_L 2_R$	\wedge_2
000110	$z_L = x_L \wedge y_L$ $z_L = x_L \wedge (x_R \oplus y_L)$ $z_L = y_L \wedge (x_L \oplus y_R)$ $z_R = (x_R \oplus y_R) \wedge \neg(x_L \oplus y_L \oplus y_R) \oplus x_R$
000111	$z_L = x_L \wedge y_L$ $z_L = x_L \wedge \neg(x_R \oplus y_L)$ $z_L = y_L \wedge \neg(x_L \oplus y_R)$ $z_R = x_R \wedge y_R$ $z_R = x_R \wedge (x_L \oplus y_R) \oplus x_L$ $z_R = \neg y_R \wedge (x_R \oplus y_L) \oplus x_R$
001001	$z_L = (x_L \oplus y_L) \wedge \neg(x_R \oplus y_L \oplus y_R) \oplus x_L$ $z_R = y_R \wedge (x_R \oplus y_L)$ $z_R = x_R \wedge (x_L \oplus y_R)$ $z_R = x_R \wedge y_R$
001011	$z_L = x_L \wedge y_L$ $z_L = \neg y_L \wedge (x_L \oplus y_R) \oplus x_L$ $z_L = x_L \wedge (x_R \oplus y_L) \oplus x_R$ $z_R = y_R \wedge \neg(x_R \oplus y_L)$ $z_R = x_R \wedge \neg(x_L \oplus y_R)$ $z_R = x_R \wedge y_R$
001101	$z_L = (x_L \oplus y_L) \wedge (x_R \oplus y_L \oplus y_R) \oplus x_L$ $z_R = x_R \wedge y_R$ $z_R = x_R \wedge (x_L \oplus y_R) \oplus x_L$ $z_R = \neg y_R \wedge (x_R \oplus y_L) \oplus x_R$
001110	$z_L = x_L \wedge y_L$ $z_L = \neg y_L \wedge (x_L \oplus y_R) \oplus x_L$ $z_L = x_L \wedge (x_R \oplus y_L) \oplus x_R$ $z_R = (x_R \oplus y_R) \wedge (x_L \oplus y_L \oplus y_R) \oplus x_R$
010010	$z_L = x_L \wedge y_L$ $z_L = x_L \wedge (x_R \oplus y_L)$ $z_L = y_L \wedge (x_L \oplus y_R)$ $z_R = \neg(\neg x_R \wedge \neg y_R)$ $z_R = y_R \wedge \neg(x_R \oplus y_L) \oplus x_R$ $z_R = \neg(\neg x_R \wedge \neg(x_L \oplus y_R) \oplus x_L)$
010011	$z_L = x_L \wedge y_L$ $z_L = y_L \wedge \neg(x_L \oplus y_R)$ $z_L = x_L \wedge \neg(x_R \oplus y_L)$ $z_R = (x_R \oplus y_R) \wedge (x_L \oplus y_L \oplus y_R) \oplus x_R$
011000	$z_L = (x_L \oplus y_L) \wedge (x_R \oplus y_L \oplus y_R) \oplus x_L$ $z_R = \neg(\neg x_R \wedge \neg y_R)$ $z_R = y_R \wedge \neg(x_R \oplus y_L) \oplus x_R$ $z_R = \neg(\neg x_R \wedge \neg(x_L \oplus y_R) \oplus x_L)$
011011	$z_L = x_L \wedge y_L$ $z_L = \neg y_L \wedge \neg(x_L \oplus y_R) \oplus x_L$ $z_L = x_L \wedge \neg(x_R \oplus y_L) \oplus x_R$ $z_R = (x_R \oplus y_R) \wedge \neg(x_L \oplus y_L \oplus y_R) \oplus x_R$
011100	$z_L = (x_L \oplus y_L) \wedge \neg(x_R \oplus y_L \oplus y_R) \oplus x_L$ $z_R = \neg(\neg x_R \wedge \neg y_R)$ $z_R = \neg(\neg y_R \wedge \neg(x_R \oplus y_L))$ $z_R = \neg(\neg x_R \wedge \neg(x_L \oplus y_R))$
011110	$z_L = x_L \wedge y_L$ $z_L = \neg y_L \wedge \neg(x_L \oplus y_R) \oplus x_L$ $z_L = x_L \wedge \neg(x_R \oplus y_L) \oplus x_R$ $z_R = \neg(\neg y_R \wedge (x_R \oplus y_L))$ $z_R = \neg(\neg x_R \wedge (x_L \oplus y_R))$ $z_R = \neg(\neg x_R \wedge \neg y_R)$

Table 5.19: Cheapest transformations for \wedge_3 , for encodings with $0_L = 1$.

$0_L 0_R 1_L 1_R 2_L 2_R$	\wedge_2	
100001	$z_L = \neg(\neg x_L \wedge \neg y_L)$	
	$z_L = y_L \wedge \neg(x_L \oplus y_R) \oplus x_L$	
	$z_L = \neg x_L \wedge \neg(x_R \oplus y_L) \oplus x_R$	
	$z_R = y_R \wedge (x_R \oplus y_L)$	
	$z_R = x_R \wedge (x_L \oplus y_R)$	
	$z_R = x_R \wedge y_R$	
100011	$z_L = (x_L \oplus y_L) \wedge (x_R \oplus y_L \oplus y_R) \oplus x_L$	
	$z_R = y_R \wedge \neg(x_R \oplus y_L)$	
	$z_R = x_R \wedge \neg(x_L \oplus y_R)$	
100100	$z_L = \neg(\neg x_L \wedge \neg y_L)$	
	$z_L = y_L \wedge \neg(x_L \oplus y_R) \oplus x_L$	
	$z_L = \neg x_L \wedge \neg(x_R \oplus y_L) \oplus x_R$	
	$z_R = (x_R \oplus y_R) \wedge (x_L \oplus y_L \oplus y_R) \oplus x_R$	
	100111	$z_L = (x_L \oplus y_L) \wedge \neg(x_R \oplus y_L \oplus y_R) \oplus x_L$
		$z_R = x_R \wedge y_R$
$z_R = \neg y_R \wedge \neg(x_R \oplus y_L) \oplus x_R$		
	$z_R = \neg(x_R \wedge \neg(x_L \oplus y_R) \oplus x_L)$	
	101100	$z_L = \neg(\neg x_L \wedge \neg(x_R \oplus y_L))$
		$z_L = \neg(\neg x_L \wedge \neg y_L)$
$z_L = \neg(\neg y_L \wedge \neg(x_L \oplus y_R))$		
	$z_R = (x_R \oplus y_R) \wedge \neg(x_L \oplus y_L \oplus y_R) \oplus x_R$	
	101101	$z_L = \neg(\neg x_L \wedge \neg y_L)$
		$z_L = \neg(\neg y_L \wedge (x_L \oplus y_R))$
$z_L = \neg(\neg x_L \wedge (x_R \oplus y_L))$		
	$z_R = x_R \wedge y_R$	
	$z_R = \neg y_R \wedge \neg(x_R \oplus y_L) \oplus x_R$	
	$z_R = \neg(x_R \wedge \neg(x_L \oplus y_R) \oplus x_L)$	
110001	$z_L = \neg(\neg x_L \wedge \neg y_L)$	
	$z_L = y_L \wedge (x_L \oplus y_R) \oplus x_L$	
	$z_L = \neg x_L \wedge (x_R \oplus y_L) \oplus x_R$	
	$z_R = (x_R \oplus y_R) \wedge \neg(x_L \oplus y_L \oplus y_R) \oplus x_R$	
	110010	$z_L = (x_L \oplus y_L) \wedge \neg(x_R \oplus y_L \oplus y_R) \oplus x_L$
		$z_R = \neg(\neg x_R \wedge \neg y_R)$
$z_R = y_R \wedge (x_R \oplus y_L) \oplus x_R$		
	$z_R = \neg x_R \wedge (x_L \oplus y_R) \oplus x_L$	
	110100	$z_L = \neg(\neg x_L \wedge \neg y_L)$
		$z_L = y_L \wedge (x_L \oplus y_R) \oplus x_L$
$z_L = \neg x_L \wedge (x_R \oplus y_L) \oplus x_R$		
	$z_R = \neg(\neg x_R \wedge \neg y_R)$	
	$z_R = \neg(\neg y_R \wedge \neg(x_R \oplus y_L))$	
	$z_R = \neg(\neg x_R \wedge \neg(x_L \oplus y_R))$	
110110	$z_L = (x_L \oplus y_L) \wedge (x_R \oplus y_L \oplus y_R) \oplus x_L$	
	$z_R = \neg(\neg y_R \wedge (x_R \oplus y_L))$	
	$z_R = \neg(\neg x_R \wedge (x_L \oplus y_R))$	
	$z_R = \neg(\neg x_R \wedge \neg y_R)$	
	111000	$z_L = \neg(\neg x_L \wedge \neg(x_R \oplus y_L))$
		$z_L = \neg(\neg x_L \wedge \neg y_L)$
$z_L = \neg(\neg y_L \wedge \neg(x_L \oplus y_R))$		
	$z_R = \neg(\neg x_R \wedge \neg y_R)$	
	$z_R = y_R \wedge (x_R \oplus y_L) \oplus x_R$	
	$z_R = \neg x_R \wedge (x_L \oplus y_R) \oplus x_L$	
111001	$z_L = \neg(\neg x_L \wedge \neg y_L)$	
	$z_L = \neg(\neg y_L \wedge (x_L \oplus y_R))$	
	$z_L = \neg(\neg x_L \wedge (x_R \oplus y_L))$	
	$z_R = (x_R \oplus y_R) \wedge (x_L \oplus y_L \oplus y_R) \oplus x_R$	

5.3. Conclusion and further works

Table 5.20: Cheapest transformations for \oplus_3 .

$0_L 0_R 1_L 1_R 2_L 2_R$	\oplus_2
000110	$z_L = \neg(x_R \oplus y_R) \wedge (x_L \oplus y_L \oplus y_R)$ $z_R = \neg(x_L \oplus y_L) \wedge (x_R \oplus y_L \oplus y_R)$
000111	$z_L = (x_L \oplus y_R) \wedge (x_R \oplus y_L)$ $z_R = \neg(\neg(x_R \oplus y_R) \wedge \neg(x_L \oplus y_L \oplus y_R))$
001001	$z_L = \neg(x_R \oplus y_R) \wedge (x_L \oplus y_L \oplus y_R)$ $z_R = \neg(x_L \oplus y_L) \wedge (x_R \oplus y_L \oplus y_R)$
001011	$z_L = \neg(\neg(x_L \oplus y_L) \wedge \neg(x_R \oplus y_L \oplus y_R))$ $z_R = (x_L \oplus y_R) \wedge (x_R \oplus y_L)$
001101	$z_L = (x_L \oplus y_R) \wedge (x_R \oplus y_L)$ $z_R = \neg(\neg(x_R \oplus y_R) \wedge \neg(x_L \oplus y_L \oplus y_R))$
001110	$z_L = \neg(\neg(x_L \oplus y_L) \wedge \neg(x_R \oplus y_L \oplus y_R))$ $z_R = (x_L \oplus y_R) \wedge (x_R \oplus y_L)$
010010	$z_L = \neg(x_L \oplus y_R) \wedge \neg(x_R \oplus y_L)$ $z_R = \neg(x_R \oplus y_R) \wedge (x_L \oplus y_L \oplus y_R)$
010011	$z_L = \neg(x_R \oplus y_R) \wedge \neg(x_L \oplus y_L \oplus y_R)$ $z_R = \neg(\neg(x_L \oplus y_L) \wedge (x_R \oplus y_L \oplus y_R))$
011000	$z_L = \neg(x_L \oplus y_R) \wedge \neg(x_R \oplus y_L)$ $z_R = \neg(x_R \oplus y_R) \wedge (x_L \oplus y_L \oplus y_R)$
011011	$z_L = \neg(\neg(x_L \oplus y_L) \wedge \neg(x_R \oplus y_L \oplus y_R))$ $z_R = \neg(\neg(x_L \oplus y_R) \wedge \neg(x_R \oplus y_L))$
011100	$z_L = \neg(x_R \oplus y_R) \wedge \neg(x_L \oplus y_L \oplus y_R)$ $z_R = \neg(\neg(x_L \oplus y_L) \wedge (x_R \oplus y_L \oplus y_R))$
011110	$z_L = \neg(\neg(x_L \oplus y_L) \wedge \neg(x_R \oplus y_L \oplus y_R))$ $z_R = \neg(\neg(x_L \oplus y_R) \wedge \neg(x_R \oplus y_L))$
100001	$z_L = \neg(x_L \oplus y_L) \wedge (x_R \oplus y_L \oplus y_R)$ $z_R = \neg(x_L \oplus y_R) \wedge \neg(x_R \oplus y_L)$
100011	$z_L = \neg(\neg(x_R \oplus y_R) \wedge (x_L \oplus y_L \oplus y_R))$ $z_R = \neg(x_L \oplus y_L) \wedge \neg(x_R \oplus y_L \oplus y_R)$
100100	$z_L = \neg(x_L \oplus y_L) \wedge (x_R \oplus y_L \oplus y_R)$ $z_R = \neg(x_L \oplus y_R) \wedge \neg(x_R \oplus y_L)$
100111	$z_L = \neg(\neg(x_L \oplus y_R) \wedge \neg(x_R \oplus y_L))$ $z_R = \neg(\neg(x_R \oplus y_R) \wedge \neg(x_L \oplus y_L \oplus y_R))$
101100	$z_L = \neg(\neg(x_R \oplus y_R) \wedge (x_L \oplus y_L \oplus y_R))$ $z_R = \neg(x_L \oplus y_L) \wedge \neg(x_R \oplus y_L \oplus y_R)$
101101	$z_L = \neg(\neg(x_L \oplus y_R) \wedge \neg(x_R \oplus y_L))$ $z_R = \neg(\neg(x_R \oplus y_R) \wedge \neg(x_L \oplus y_L \oplus y_R))$
110001	$z_L = \neg(x_L \oplus y_L) \wedge (x_R \oplus y_L \oplus y_R)$ $z_R = \neg((x_L \oplus y_R) \wedge (x_R \oplus y_L))$
110010	$z_L = \neg((x_L \oplus y_R) \wedge (x_R \oplus y_L))$ $z_R = \neg(x_R \oplus y_R) \wedge (x_L \oplus y_L \oplus y_R)$
110100	$z_L = \neg(x_L \oplus y_L) \wedge (x_R \oplus y_L \oplus y_R)$ $z_R = \neg((x_L \oplus y_R) \wedge (x_R \oplus y_L))$
110110	$z_L = \neg(\neg(x_R \oplus y_R) \wedge \neg(x_L \oplus y_L \oplus y_R))$ $z_R = \neg(\neg(x_L \oplus y_L) \wedge \neg(x_R \oplus y_L \oplus y_R))$
111000	$z_L = \neg((x_L \oplus y_R) \wedge (x_R \oplus y_L))$ $z_R = \neg(x_R \oplus y_R) \wedge (x_L \oplus y_L \oplus y_R)$
111001	$z_L = \neg(\neg(x_R \oplus y_R) \wedge \neg(x_L \oplus y_L \oplus y_R))$ $z_R = \neg(\neg(x_L \oplus y_L) \wedge \neg(x_R \oplus y_L \oplus y_R))$

allows the transmission of *xor* gates for free. In the context of software computations for the Garbled circuit protocol, we also propose the new Mixed Logic, which reduces the transfer circuit's cost. In future we'll work on some other applications of this logic to more complex 3VL circuits, and on a deeper analysis of the MVL case, i.e., when more than three truth elements are considered.

CHAPTER 6

CONCLUSION AND OPEN PROBLEMS

In this thesis, we discuss how to protect cryptographic circuits. In particular, we identify two different types of attacker, i.e., external and internal. Following this reasoning, we divided the work in two parts, depending on the two natures of malicious entities in a cryptographic system: *protecting the secret from an external attacker* (Part I) and *protecting the secret during computation* (Part II).

In the first part, the attacker is external to the communication, and she tries to steal some sensitive information when the other entities send each other secret messages. In this context, probing a circuit is an useful technique through which an attacker can derive information correlated with the secret manipulated by the cryptographic circuit. Probing security is the branch of research of which we discussed in Chapter 3. In particular, the treatment of the argument is divided into two cases, when physical defaults that can happen in a circuit (as glitches) are not taken into account (Section 3.1.1) or when they are considered (Section 3.1.2). About this topic, we present many original works. For all of them, we define the conclusions and some future works.

- *A relation calculus for reasoning about t -probing security.* We originally started this research to extend our understanding of t -probing security. We have discovered a new relation calculus of shares which exploits the conventional Walsh transform. This calculus is precise enough to prove and extend known compositional properties without much semi-formal or verbal ratiocination. We believe that the underlying linear algebra, while providing a more intuitive understanding, but will allow for an easier mechanization of probing security proofs. We believe that more work must still be done towards an unifying approach that encompasses circuit glitches and new composability definitions such as the t -PINI

condition [42].

- *On the spectral features of robust probing security.* This work provided an alternative yet comprehensive view of robust probing security which, we argue, addresses more clearly the issues associated with composability of robust-probing secure gadgets. To achieve our goal, we introduced further distinctions for dealing with extended probes; in particular, these must be admitted to participate in a unique way during composition much like conventional outputs. We believe we have provided sufficient evidence that this new mathematical framework could work for analysis and synthesis of such gadgets. Further work is needed to make the underlying computations more efficient as they are based on computation of the Walsh spectrum which incurs exponential cost. We believe that sparse matrix representation might be a tool worth investigating to improve correlation matrix computation. Another possible further extension of this work could be modeling t -PINI as well as inquiring about the minimum number of randoms required to achieve robust t -strong non interference and/or investigating whether the ring structure of multiplication gadgets can be replaced by potentially more efficient refresh layers.
- *On robust strong-non-interferent low-latency multiplications.* In this work, we have derived a new robust t -SNI construction for multiplying two secrets in a robust strongly non interferent way. The novel construction has 1-cycle input-to-output latency and, for low security degrees t , the randomness complexity is comparable with conventional, 2-cycle-latency approaches. As a future work, we plan to study the use of the proposed gadget in the S-boxes of known cryptographic algorithms as well as the randomness requirements for higher t . In particular, preliminary work shows that a scheme that involves 42 randoms for $t = 5$ is possible, but we believe this not to be lowest bound achievable.
- *ADD-based Spectral Analysis of Probing Security.* In this work, we propose a new methodology that allows to exactly verify strong-non-interference properties of a gadget; our approach combines both hash maps and ADDs and provides, on a standard set of use cases, a median speed-up of 1.88x against other exact methods. Timing-wise, the results are also not dramatically far from *non-exact* approaches appeared in literature. We reserve for the future the more detailed inspection about the improvement's gap between Keccak and DOM algorithms with **mapi** method. Another possible future work is the application of our tool to more complex gadgets, with higher security levels and by exploiting parallelization. Moreover, also we expect to include the verification of other probing security properties (i.g. PINI [64]).

In the second part, the malicious entity is internal to the communication: each participant is interested to protect own sensitive information from all the others. In particular, two parties secure computation is the theory developed when the involved parties are only two. A solution that allows to protect secrets when the entities are two is the garbled circuit protocol. Correlated to the garbled circuit protocol, in this work we develop two main branches. The former, is the study of multiplicative complexity of Boolean functions, in particular autosymmetric and D-reducible (Chapter 4). The latter, is the application of multiple-valued logic theory to the involved circuits (Chapter 5). Also in this case, we present some original works, and we define brief conclusions and some future works.

- *Multiplicative Complexity of Autosymmetric Functions: Theory and Applications to Security.* In this paper we have considered the class of autosymmetric functions and we have shown how autosymmetry can be exploited to better estimate their multiplicative complexity. Moreover, the experimental results show that autosymmetry test can enable the XAG minimization of autosymmetric functions. We show that *xor* nodes are costless in multiparty computation. In general, the computation between parties can be applied to any Boolean functions; for this reason, we have conducted our experimentation on a wide set of standard Boolean benchmarks. As a future work, we plan to investigate benchmarks related to more general security and cryptographic applications.
- *Multiplicative Complexity of XOR Based Regular Functions.* In this paper we have considered the classes of autosymmetric and D-reducible functions and we have shown how these regularities can be exploited to better estimate their multiplicative complexity. Moreover, the experimental results show that these tests of regularity can enable the XAG minimization of Boolean functions. In this work, we have considered *xor*-based regularities that are frequent in classical benchmark functions. Nevertheless, it would be interesting to study the multiplicative complexity of other regular functions such as, for example, symmetric Boolean functions or self-dual functions, to better understand the multiplicative complexity and further improve the XAG minimization. Moreover, we plan to extend the autosymmetry algorithms to detect all linear structures of a function, thus generalizing our approach to all the functions with linearity dimension strictly less than n . This could further enable the XAG minimization of the corresponding circuits. All proposed methods could be extended to multi-output functions, potentially sharing the *xor* layer among multiple outputs. Finally, we will work on the current limitations in terms of scalability on larger functions, that mostly impacted when testing the cryptographic benchmarks.

- *A Multiple Valued Logic Approach for the Synthesis of Garbled Circuits.* In this paper we have given a comparison between two different 3-valued logic approaches. We have extended one of them, for which we have defined, in the context of the garbled circuit protocols, some functional encodings that allow to transform values and functions from the 3-valued logic to the Boolean logic. We have then studied the costs in both logics, leading to the definition of the (new) Mixed Logic: this new logic allows to exploit the cheapest solution in all the garbling situations. We also have proved that, in the multiple valued logic as defined in one of the two studied approaches, the *xor* can be evaluated for free, i.e. without any transfer cost in a GC protocol. In future we'll work on some other applications of the Mixed Logic to more complex 3VL circuits, such as the S-boxes of current cryptographic solutions. Moreover, a deeper analysis of the MVL case (with more than 3 elements) can be a prolific area to explore, especially now that we have proved the free transfer of the *xor* gate.

BIBLIOGRAPHY

- [1] Martin R. Albrecht, Christian Rechberger, Thomas Schneider, Tyge Tiessen, and Michael Zohner. Ciphers for mpc and fhe. In Elisabeth Oswald and Marc Fischlin, editors, *Advances in Cryptology – EUROCRYPT 2015*, pages 430–454, Berlin, Heidelberg, 2015. Springer Berlin Heidelberg.
- [2] Henrik Reif Andersen. *An Introduction to Binary Decision Diagrams*. 1996.
- [3] Kuhn M. Anderson, R. Tamper resistance - a cautionary note. 2nd Workshop on Electronic Commerce.
- [4] D. W. Archer, D. Bogdanov, B. Pinkas, and P. Pullonen. Maturity and performance of programmable secure computation. *IEEE Security Privacy*, 14(5):48–56, 2016.
- [5] R. Iris Bahar, Erica A. Frohm, Charles M. Gaona, Gary D. Hachtel, Enrico Macii, Abelardo Pardo, and Fabio Somenzi. Algebraic decision diagrams and their applications, 1993.
- [6] Gilles Barthe, Sonia Belaïd, Gaëtan Cassiers, Pierre-Alain Fouque, Benjamin Grégoire, and François-Xavier Standaert. *maskVerif: Automated Verification of Higher-Order Masking in Presence of Physical Defaults*, pages 300–318. 09 2019.
- [7] Gilles Barthe, Sonia Belaïd, Pierre-Alain Fouque, and Benjamin Grégoire. maskVerif: automated analysis of software and hardware higher-order masked implementations. Technical Report 562, 2018.

- [8] Gilles Barthe, Sonia Belaïd, François Dupressoir, Pierre-Alain Fouque, and Benjamin Grégoire. Compositional verification of higher-order masking: Application to a verifying masking compiler. *IACR Cryptology ePrint Archive*, 2015:506, 2015.
- [9] Gilles Barthe, Sonia Belaïd, François Dupressoir, Pierre-Alain Fouque, Benjamin Grégoire, Pierre-Yves Strub, and Rébecca Zucchini. Strong non-interference and type-directed higher-order masking. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security, CCS '16*, page 116–129, New York, NY, USA, 2016. Association for Computing Machinery.
- [10] Gilles Barthe, Sonia Belaïd, François Dupressoir, Pierre-Alain Fouque, Benjamin Grégoire, and Pierre-Yves Strub. Verified proofs of higher-order masking. In Elisabeth Oswald and Marc Fischlin, editors, *Advances in Cryptology – EUROCRYPT 2015*, pages 457–485, Berlin, Heidelberg, 2015. Springer Berlin Heidelberg.
- [11] Ali Galip Bayrak, Francesco Regazzoni, David Novo, and Paolo Ienne. Sleuth: Automated verification of software power analysis countermeasures. In Guido Bertoni and Jean-Sébastien Coron, editors, *Cryptographic Hardware and Embedded Systems - CHES 2013*, pages 293–310, Berlin, Heidelberg, 2013. Springer Berlin Heidelberg.
- [12] Nuel Belnap. How a computer should think. In G. Ryle, editor, *Contemporary Aspects of Philosophy*. Oriel Press, 1977.
- [13] Anna Bernasconi, Stelvio Cimato, Valentina Ciriani, and Maria Chiara Molteni. Multiplicative complexity of autosymmetric functions: Theory and applications to security. In *57th ACM/IEEE Design Automation Conference, DAC 2020, San Francisco, CA, USA, July 20-24, 2020*, pages 1–6. IEEE, 2020.
- [14] Anna Bernasconi, Stelvio Cimato, Valentina Ciriani, and Maria Chiara Molteni. Multiplicative complexity of xor based regular functions. *IEEE Transactions on Computers*, pages 1–1, 2022.
- [15] Anna Bernasconi and Valentina Ciriani. Dimension-reducible boolean functions based on affine spaces. *ACM Trans. Des. Autom. Electron. Syst.*, 16(2), April 2011.

BIBLIOGRAPHY

- [16] Anna Bernasconi and Valentina Ciriani. Dimension-Reducible Boolean Functions Based on Affine Spaces. *ACM Trans. Design Autom. Electr. Syst.*, 16(2):13:1–13:21, 2011.
- [17] Anna Bernasconi, Valentina Ciriani, Fabrizio Luccio, and Linda Pagli. Three-Level Logic Minimization Based on Function Regularities. *IEEE Trans. on CAD of Integrated Circuits and Systems*, 22(8):1005–1016, 2003.
- [18] Anna Bernasconi, Valentina Ciriani, Fabrizio Luccio, and Linda Pagli. Exploiting regularities for boolean function synthesis. *Theory Comput. Syst.*, 39(4):485–501, 2006.
- [19] Anna Bernasconi, Valentina Ciriani, Fabrizio Luccio, and Linda Pagli. Exploiting regularities for boolean function synthesis. *Theory Comput. Syst.*, 39(4):485–501, 2006.
- [20] Anna Bernasconi, Valentina Ciriani, Fabrizio Luccio, and Linda Pagli. Synthesis of autosymmetric functions in a new three-level form. *Theory Comput. Syst.*, 42(4):450–464, 2008.
- [21] Anna Bernasconi, Valentina Ciriani, Fabrizio Luccio, and Linda Pagli. Synthesis of autosymmetric functions in a new three-level form. *Theory Comput. Syst.*, 42(4):450–464, 2008.
- [22] Anna Bernasconi, Valentina Ciriani, Gabriella Trucco, and Tiziano Villa. Boolean minimization of projected sums of products via boolean relations. *IEEE Trans. Computers*, 68(9):1269–1282, 2019.
- [23] Guido Bertoni, Marco Martinoli, and Maria Chiara Molteni. A methodology for the characterisation of leakages in combinatorial logic. *J. Hardw. Syst. Secur.*, 1(3):269–281, 2017.
- [24] Tim Beyne. Block cipher invariants as eigenvectors of correlation matrices (full version). Cryptology ePrint Archive, Report 2018/763, 2018.
- [25] Begül Bilgin, Svetla Nikova, Ventzislav Nikov, Vincent Rijmen, and Georg Stütz. Threshold implementations of all 3×3 and 4×4 s-boxes. In Emmanuel Prouff and Patrick Schaumont, editors, *Cryptographic Hardware and Embedded Systems – CHES 2012*, pages 76–91, Berlin, Heidelberg, 2012. Springer Berlin Heidelberg.

-
- [26] Roderick Bloem, Hannes Groß, Rinat Iusupov, Bettina Könighofer, Stefan Mangard, and Johannes Winter. Formal verification of masked hardware implementations in the presence of glitches. In *EUROCRYPT (2)*, pages 321–353. Springer, 2018.
- [27] Dmitri A Bochvar. Ob odnom trechznacnom iscislenii i ego primenenii k analizu paradoksov klassiceskogo funkcional'nogo iscislenija. *Mat. Sbornik*, 4(46):287–308, 1938. [English translation: Bochvar, D.A., On a three-valued logical calculus and its application to the analysis of the paradoxes of the classical extended functional calculus, *History and Philosophy of Logic* 2, 87-112.].
- [28] Dan Bogdanov, Liina Kamm, Sven Laur, Pille Pruulmann-Vengerfeldt, Riivo Talviste, and Jan Willemson. Privacy-preserving statistical data analysis on federated databases. In Bart Preneel and Demosthenes Ikononou, editors, *Privacy Technologies and Policy*, pages 30–55, Cham, 2014. Springer International Publishing.
- [29] Joan Boyar, Magnus Gausdal Find, and René Peralta. On various nonlinearity measures for boolean functions. *Cryptogr. Commun.*, 8(3):313–330, 2016.
- [30] Joan Boyar and Rene Peralta. Tight bounds for the multiplicative complexity of symmetric functions. (396), 2008-04-28 00:04:00 2008.
- [31] Robert Brayton and Alan Mishchenko. Abc: An academic industrial-strength verification tool. In Tayssir Touili, Byron Cook, and Paul Jackson, editors, *Computer Aided Verification*, pages 24–40, Berlin, Heidelberg, 2010. Springer Berlin Heidelberg.
- [32] Robert K Brayton, Richard Rudell, Alberto Sangiovanni-Vincentelli, and Albert R Wang. Mis: A multiple-level logic optimization system. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 6(6):1062–1081, 1987.
- [33] Eric Brier, Christophe Clavier, and Francis Olivier. Correlation power analysis with a leakage model. In *Lecture Notes in Computer Science*, volume 3156, pages 135–152. Springer Berlin / Heidelberg, 2004.
- [34] Randal E. Bryant. Symbolic boolean manipulation with ordered binary-decision diagrams. *ACM Comput. Surv.*, 24(3):293–318, September 1992.
- [35] Cagdas Calik, Meltem Sonmez, and Rene Peralta. The multiplicative complexity of 6-variable boolean functions. 2018-04-03 2018.

BIBLIOGRAPHY

- [36] Claude Carlet. Vectorial boolean functions for cryptography. *Boolean Models and Methods in Mathematics, Computer Science, and Engineering*, 12 2006.
- [37] Claude Carlet. Boolean functions for cryptography and error correcting codes. 11 2007.
- [38] Claude Carlet. Vectorial Boolean Functions for Cryptography. In Yves Crama and Peter L. Hammer, editors, *Boolean Models and Methods in Mathematics, Computer Science, and Engineering*, pages 398–470. Cambridge University Press, Cambridge, 2010.
- [39] G. Cassiers, B. Gregoire, I. Levi, and F.-X. Standaert. Hardware Private Circuits: From Trivial Composition to Full Verification. *IEEE Transactions on Computers*, page (Preprint), 2020.
- [40] Gaëtan Cassiers, Sebastian Faust, Maximilian Orlt, and François-Xavier Standaert. Towards Tight Random Probing Security. Technical Report 880, 2021.
- [41] Gaëtan Cassiers and François-Xavier Standaert. Trivially and efficiently composing masked gadgets with probe isolating non-interference. *IEEE Trans. Inf. Forensics Secur.*, 15:2542–2555, 2020.
- [42] Gaëtan Cassiers and François-Xavier Standaert. Towards globally optimized masking: From low randomness to low noise rate: or probe isolating multiplications with reduced randomness and security against horizontal attacks. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, 2019(2):162–198, Feb. 2019.
- [43] Stelvio Cimato, Valentina Ciriani, Ernesto Damiani, and Maryam Ehsanpour. A multiple valued logic approach for the synthesis of garbled circuits. In *2017 IFIP/IEEE 25th International Conference on Very Large Scale Integration, VLSI-SoC 2017, ABU Dhabi, UAE, October 22-25, 2017*, pages 232–236, 2017.
- [44] Stelvio Cimato, Valentina Ciriani, Ernesto Damiani, and Maryam Ehsanpour. An obdd-based technique for the efficient synthesis of garbled circuits. In Sjouke Mauw and Mauro Conti, editors, *Security and Trust Management - 15th International Workshop, STM 2019, Luxembourg City, Luxembourg, September 26-27, 2019, Proceedings*, volume 11738 of *Lecture Notes in Computer Science*, pages 158–167. Springer, 2019.
- [45] V. Ciriani. Synthesis of SPP Three-Level Logic Networks using Affine Spaces. *IEEE Trans. on CAD of Integrated Circuits and Systems*, 22(10):1310–1323, 2003.

- [46] Valentina Ciriani. Synthesis of SPP three-level logic networks using affine spaces. *IEEE Trans. Comput. Aided Des. Integr. Circuits Syst.*, 22(10):1310–1323, 2003.
- [47] Thomas De Cnudde, Oscar Reparaz, Begul Bilgin, Svetla Nikova, Ventzislav Nikov, and Vincent Rijmen. Masking aes with $d+1$ shares in hardware. In *CHES*, pages 194–212. Springer, 2016.
- [48] Stephen Cole Kleene. On notation for ordinal numbers. *Journal Symbolic Logic*, (3):150–155, 1938.
- [49] Irving M. Copilowish. Matrix development of the calculus of relations. *The Journal of Symbolic Logic*, 13(04):193–203, December 1948.
- [50] Jean-Sébastien Coron, Emmanuel Prouff, Matthieu Rivain, and Thomas Roche. Higher-order side channel security and mask refreshing. *IACR Cryptology ePrint Archive*, 2015:359, 2015.
- [51] Jean-Sébastien Coron. Formal verification of side-channel countermeasures via elementary circuit transformations. In Bart Preneel and Frederik Vercauteren, editors, *Applied Cryptography and Network Security - 16th International Conference, ACNS 2018, Leuven, Belgium, July 2-4, 2018, Proceedings*, volume 10892 of *Lecture Notes in Computer Science*, pages 65–82. Springer, 2018.
- [52] Jean-Sébastien Coron, Emmanuel Prouff, Matthieu Rivain, and Thomas Roche. Higher-Order Side Channel Security and Mask Refreshing. In Shiho Moriai, editor, *Fast Software Encryption*, Lecture Notes in Computer Science, pages 410–424. Springer Berlin Heidelberg, 2014.
- [53] Joan Daemen, René Govaerts, and Joos Vandewalle. Correlation matrices. In Bart Preneel, editor, *Fast Software Encryption*, Lecture Notes in Computer Science, pages 275–285. Springer Berlin Heidelberg, 1995.
- [54] Joan Daemen and Vincent Rijmen. *The Design of Rijndael: AES - The Advanced Encryption Standard*. 01 2002.
- [55] Brandon Dravie, Jérémy Parriaux, Philippe Guillot, and Gilles Millérioux. Matrix representations of vectorial boolean functions and eigenanalysis. *Cryptography and Communications - Discrete Structures, Boolean Functions and Sequences*, 8(4):555–577, October 2016.
- [56] Elena Dubrova. *Multiple-Valued Logic Synthesis and Optimization*, pages 89–114. Springer US, Boston, MA, 2002.

BIBLIOGRAPHY

- [57] Elena Dubrova. Multiple-valued logic synthesis and optimization. In Soha Hassoun and Tsutomu Sasao, editors, *Logic Synthesis and Verification*, pages 89–114. 2002.
- [58] Hassan Eldib, Chao Wang, and Patrick Schaumont. Formal verification of software countermeasures against side-channel attacks. *ACM Trans. Softw. Eng. Methodol.*, 24(2), December 2014.
- [59] Sebastian Faust, Vincent Grosso, Santos Merino Del Pozo, Clara Paglialonga, and François-Xavier Standaert. Composable masking schemes in the presence of physical defaults and the robust probing model. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, 2018(3):89–120, Aug. 2018.
- [60] Sebastian Faust, Vincent Grosso, Santos Merino Del Pozo, Clara Paglialonga, and François-Xavier Standaert. Composable masking schemes in the presence of physical defaults & the robust probing model. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2018(3):89–120, 2018.
- [61] M. Fujita, J. Chih-Yuan Yang, E.M. Clarke, Zudong Zhao, and P. McGeer. Fast spectrum computation for logic functions using binary decision diagrams. In *Proceedings of IEEE International Symposium on Circuits and Systems - ISCAS '94*, volume 1, pages 275–278, London, UK, 1994. IEEE.
- [62] K. Gödel. Zum intuitionistischen aussagenkalkül. *Anzeiger der Akademie der Wissenschaften in Wien*, 69:65–66, 1932.
- [63] Siegfried Gottwald. Many-valued logic. *Stanford Encyclopedia of Philosophy*, 03 2015.
- [64] Dahmun Goudarzi, Thomas Prest, Matthieu Rivain, and Damien Vergnaud. Probing security through input-output separation and revisited quasilinear masking. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, 2021(3):599–640, Jul. 2021.
- [65] Dahmun Goudarzi and Matthieu Rivain. On the multiplicative complexity of boolean functions and bitsliced higher-order masking. In *CHES*, pages 457–478. Springer, 2016.
- [66] Hannes Gross and Stefan Mangard. A unified masking approach. *Journal of Cryptographic Engineering*, 8(2):109–124, June 2018.
- [67] Hannes Gross, Stefan Mangard, and Thomas Korak. Domain-oriented masking: Compact masked hardware implementations with arbitrary protection order. In

-
- Proceedings of the 2016 ACM Workshop on Theory of Implementation Security, TIS '16*, page 3, New York, NY, USA, 2016. Association for Computing Machinery.
- [68] Hannes Gross, David Schaffenrath, and Stefan Mangard. Higher-Order Side-Channel Protected Implementations of Keccak. Technical Report 395, 2017.
- [69] Ivo Háleček, Petr Fiser, and Jan Schmidt. Towards AND/XOR balanced synthesis: Logic circuits rewriting with XOR. *Microelectron. Reliab.*, 81:274–286, 2018.
- [70] Christoph Herbst, Elisabeth Oswald, and Stefan Mangard. An aes smart card implementation resistant to power analysis attacks. In Jianying Zhou, Moti Yung, and Feng Bao, editors, *Applied Cryptography and Network Security*, pages 239–252, Berlin, Heidelberg, 2006. Springer Berlin Heidelberg.
- [71] Ivo Háleček, Petr Fišer, and Jan Schmidt. Are xors in logic synthesis really necessary? In *2017 IEEE 20th International Symposium on Design and Diagnostics of Electronic Circuits Systems (DDECS)*, pages 134–139, 2017.
- [72] Roger A. Horn and Charles R. Johnson. *Topics in Matrix Analysis*. Cambridge University Press, Cambridge; New York, 1994.
- [73] Yuval Ishai, Amit Sahai, and David Wagner. Private circuits: Securing hardware against probing attacks. In Dan Boneh, editor, *Advances in Cryptology - CRYPTO 2003*, pages 463–481, Berlin, Heidelberg, 2003. Springer Berlin Heidelberg.
- [74] David Knichel, Amir Moradi, Nicolai Müller, and Pascal Sasdrich. Automated generation of masked hardware. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, 2022(1):589–629, Nov. 2021.
- [75] David Knichel, Pascal Sasdrich, and Amir Moradi. Silver - statistical independence and leakage verification. 2020.
- [76] Donald E. Knuth. *The Art of Computer Programming, Volume 4, Fascicle 1: Bitwise Tricks & Techniques; Binary Decision Diagrams*. Addison-Wesley Professional, 12th edition, 2009.
- [77] Paul Kocher, Joshua Jaffe, and Benjamin Jun. Differential power analysis. In Michael Wiener, editor, *Advances in Cryptology — CRYPTO' 99*, pages 388–397, Berlin, Heidelberg, 1999. Springer Berlin Heidelberg.

BIBLIOGRAPHY

- [78] Vladimir Kolesnikov and Thomas Schneider. Improved garbled circuit: Free xor gates and applications. In Luca Aceto, Ivan Damgård, Leslie Ann Goldberg, Magnús M. Halldórsson, Anna Ingólfssdóttir, and Igor Walukiewicz, editors, *Automata, Languages and Programming*, pages 486–498, Berlin, Heidelberg, 2008. Springer Berlin Heidelberg.
- [79] Yehuda Lindell. Secure multiparty computation (mpc). Cryptology ePrint Archive, Report 2020/300, 2020.
- [80] Yehuda Lindell and Avishay Yanai. Fast garbling of circuits over 3-valued logic. pages 620–643, 2018.
- [81] Fabrizio Luccio and Linda Pagli. On a new boolean function with applications. *IEEE Trans. Comput.*, 48(3):296–310, March 1999.
- [82] Jan Łukasiewicz. O logice trójwartociowej. *Studia Filozoficzne*, 270(5), 1988.
- [83] Stefan Mangard, Maria Elisabeth Oswald, and Thomas Popp. *Power Analysis Attacks - Revealing the Secrets of Smart Cards*. Springer, 1 edition, 2007. XXIII, 337 S.
- [84] P. McGeer, J. Sanghavi, R. Brayton, and A. Sangiovanni-Vincentelli. ESPRESSO-SIGNATURE: A New Exact Minimizer for Logic Functions. *IEEE Transactions on VLSI*, 1(4):432–440, 1993.
- [85] Lauren De Meyer, Begül Bilgin, and Oscar Reparaz. Consolidating Security Notions in Hardware Masking. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, pages 119–147, May 2019.
- [86] D. Michael Miller and Mitchell Aaron Thornton. *Multiple Valued Logic - Concepts and Representations*, volume 12 of *Synthesis lectures on digital circuits and systems*. Morgan & Claypool Publishers, 2008.
- [87] James S. Milne. Fields and galois theory (v4.51), 2015. Available at www.jmilne.org/math/.
- [88] Maria Chiara Molteni, Jürgen Pulkus, and Vittorio Zaccaria. On robust strong-non-interferent low-latency multiplications. *IET Information Security*.
- [89] Maria Chiara Molteni and Vittorio Zaccaria. On the spectral features of robust probing security. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, pages 24–48, August 2020.

- [90] Maria Chiara Molteni and Vittorio Zaccaria. A relation calculus for reasoning about t-probing security. *Journal of Cryptographic Engineering*, February 2022.
- [91] J. Donald Monk. *Mathematical Logic*, pages 141–161. Springer New York, 1976.
- [92] Thorben Moos, Amir Moradi, Tobias Schneider, and François-Xavier Standaert. Glitch-resistant masking revisited: or why proofs in the robust probing model are needed. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, 2019(2):256–292, Feb. 2019.
- [93] Andrew Moss, Elisabeth Oswald, Dan Page, and Michael Tunstall. Compiler assisted masking. In Emmanuel Prouff and Patrick Schaumont, editors, *Cryptographic Hardware and Embedded Systems – CHES 2012*, pages 58–75, Berlin, Heidelberg, 2012. Springer Berlin Heidelberg.
- [94] Svetla Nikova, Christian Rechberger, and Vincent Rijmen. Threshold implementations against side-channel attacks and glitches. In *Information and Communications Security*, pages 529–545. Springer, December 2006.
- [95] Svetla Nikova, Vincent Rijmen, and Martin Schl affer. Secure Hardware Implementation of Nonlinear Functions in the Presence of Glitches. volume vol. 24, pages 292–322. Springer, 2011.
- [96] J eremy Parriaux, Philippe Guillot, and Gilles Mill erioux. Towards a spectral approach for the design of self-synchronizing stream ciphers. *Cryptography and Communications*, 3(4):259–274, December 2011. C.
- [97] Benny Pinkas, Thomas Schneider, Nigel P. Smart, and Stephen C. Williams. Secure two-party computation is practical. In Mitsuru Matsui, editor, *Advances in Cryptology – ASIACRYPT 2009*, pages 250–267, Berlin, Heidelberg, 2009. Springer Berlin Heidelberg.
- [98] Oscar Reparaz, Beg ul Bilgin, Svetla Nikova, Benedikt Gierlichs, and Ingrid Verbauwhede. Consolidating masking schemes. In Rosario Gennaro and Matthew Robshaw, editors, *Advances in Cryptology – CRYPTO 2015*, pages 764–783, Berlin, Heidelberg, 2015. Springer Berlin Heidelberg.
- [99] Matthieu Rivain and Emmanuel Prouff. Provably Secure Higher-Order Masking of AES. Technical Report 441, 2010.
- [100] Ronald L. Rivest. Cryptography. In Jan van Leeuwen, editor, *Handbook of Theoretical Computer Science, Volume A: Algorithms and Complexity*, pages 717–755. Elsevier and MIT Press, 1990.

BIBLIOGRAPHY

- [101] Francisco Rodríguez-Henríquez, Arturo Díaz Pérez, Nazar Abbas Saqib, and Çetin Kaya Koç. *Cryptographic Algorithms on Reconfigurable Hardware*. 2007.
- [102] Claus-Peter Schnorr. The multiplicative complexity of boolean functions. In Teo Mora, editor, *Applied Algebra, Algebraic Algorithms and Error-Correcting Codes, 6th International Conference, AAECC-6, Rome, Italy, July 4-8, 1988, Proceedings*, volume 357 of *Lecture Notes in Computer Science*, pages 45–58. Springer, 1988.
- [103] Adel S. Sedra and Kenneth C. Smith. *Microelectronic Circuits*. Oxford University Press, fifth edition, 2004.
- [104] P. Selinger. A Survey of Graphical Languages for Monoidal Categories. In Bob Coecke, editor, *New Structures for Physics*, volume 813, pages 289–355. Springer Berlin Heidelberg, Berlin, Heidelberg, 2010.
- [105] Claude E. Shannon. A mathematical theory of communication. *Bell Syst. Tech. J.*, 27(3):379–423, 1948.
- [106] Peter Snyder. Yao’s garbled circuits: Recent directions and implementations, 2014.
- [107] Mathias Soeken, Luca Gaetano Amarù, Pierre-Emmanuel Gaillardon, and Giovanni De Micheli. Exact synthesis of majority-inverter graphs and its applications. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 36(11):1842–1855, 2017.
- [108] Mathias Soeken, Heinz Rienner, Winston Haaswijk, Eleonora Testa, Bruno Schmitt, Giulia Meuli, Fereshte Mozafari, and Giovanni De Micheli. The epl logic synthesis libraries, 2019.
- [109] Fabio Somenzi. Binary decision diagrams. In *Calculational System Design, volume 173 of NATO Science Series F: Computer and Systems Sciences*, pages 303–366. IOS Press, 1999.
- [110] Fabio Somenzi. Cudd: Cu decision diagram package - release 2.4.1, 2005.
- [111] Meltem Sonmez and Rene Peralta. The multiplicative complexity of boolean functions on four and five variables. *Lightweight Cryptography for Security and Privacy (Lecture Notes in Computer Science)*, Istanbul, -1, 2015-03-17 2015.
- [112] Eleonora Testa, Mathias Soeken, Luca G. Amarù, and Giovanni De Micheli. Reducing the multiplicative complexity in logic networks for cryptography and

- security applications. In *Proceedings of the 56th Annual Design Automation Conference 2019, DAC 2019, Las Vegas, NV, USA*, page 74, 2019.
- [113] Eleonora Testa, Mathias Soeken, Luca Gaetano Amarù, and Giovanni De Micheli. Logic synthesis for established and emerging computing. *Proc. IEEE*, 107(1):165–184, 2019.
- [114] Eleonora Testa, Mathias Soeken, Heinz Riener, Luca Amaru, and Giovanni De Micheli. A logic synthesis toolbox for reducing the multiplicative complexity in logic networks. In *2020 Design, Automation Test in Europe Conference Exhibition (DATE)*, pages 568–573, 2020.
- [115] Elena Trichina. Combinational logic design for aes subbyte transformation on masked data, 2003. Not published elsewhere. e.v.trichina@samsung.com 12368 received 11 Nov 2003.
- [116] Elena Trichina, Tymur Korkishko, and Kyung Hee Lee. Small size, low power, side channel-immune aes coprocessor: Design and synthesis results. In *Advanced Encryption Standard – AES*, pages 113–127, Berlin, Heidelberg, 2005. Springer.
- [117] Ingo Wegener. *The Complexity of Boolean Functions*. John Wiley & Sons, 1987.
- [118] Clifford Wolf. Yosys open synthesis suite. <http://www.clifford.at/yosys/>.
- [119] G. Z. Xiao and J. L. Massey. A spectral characterization of correlation-immune combining functions. *IEEE Transactions on Information Theory*, 34(3):569–571, May 1988.
- [120] S. Yang. Logic synthesis and optimization benchmarks user guide version 3.0. User guide, Microelectronic Center, 1991.
- [121] Saeyang Yang. Logic synthesis and optimization benchmarks user guide version 3.0, 1991.
- [122] Andrew C. Yao. Protocols for secure computations. In *Proceedings of the 23rd Annual Symposium on Foundations of Computer Science*, SFCS '82, page 160–164, USA, 1982. IEEE Computer Society.
- [123] Vittorio Zaccaria, Filippo Melzani, and Guido Bertoni. Spectral features of higher-order side-channel countermeasures. *IEEE Trans. Computers*, 67(4):596–603, 2018.

BIBLIOGRAPHY

- [124] Samee Zahur, Mike Rosulek, and David Evans. Two halves make a whole. In Elisabeth Oswald and Marc Fischlin, editors, *Advances in Cryptology - EUROCRYPT 2015*, pages 220–250, Berlin, Heidelberg, 2015. Springer Berlin Heidelberg.
- [125] Jun Zhang, Pengfei Gao, Fu Song, and Chao Wang. Scinfer: Refinement-based verification of software countermeasures against side-channel attacks. In Hana Chockler and Georg Weissenbacher, editors, *Computer Aided Verification*, pages 157–177, Cham, 2018. Springer International Publishing.

ACKNOWLEDGEMENTS

I would like to thank my supervisor Prof. Cimato Stelvio, who helps and supports me during my PhD. My gratitude extends to Prof.s Zaccaria Vittorio and Valentina Ciriani, for their advices and professionalism. Work with them has been a pleasure.

Additionally, I would like to thank the reviewers of this thesis, Amarù Luca, Belaïd Sonia and Lionel Brunie. They gave me constructive comments to improve it, making this work something for which I feel proud.

My appreciation also goes out to my family and friends, for their encouragement and support all through my studies.