PROOFS workshop: pre-proceedings

"Security Proofs for Embedded Systems"

UCSB, CA, USA
Saturday, August 24th, 2013

Preface

On behalf of the steering committee, we are glad to welcome you to this second edition of PROOFS. The first edition has been held on Thursday, September 13rd, 2012, *i.e.* on the day following CHES at the K.U.Leuven, Belgium.

The goal of the PROOFS workshop series is to promote methodologies that increase the confidence level in the security of embedded systems.

Embedded system security too frequently consists in security by obscurity solutions (except, of course, for high-security solutions produced by specialized firms, for instance in the smartcard industry). This has obvious drawbacks:

- it requires costly black-box evaluation,
- there is no certainty about the correctness of the security, etc.

Formal methods allow to increase the trust level of digital systems, especially those that embed cryptography. They are very appealing, for the following reasons:

- they are mature in theory, and there are off-the-shelf tried and tested methods and tools,
- they have been applied both on software and hardware for a long time, mainly for safety and conformance tests, but also sometimes for security assessment.

Some important security features (random number generation, physically unclonable functions, side-channel resistance, etc.) rely on analog devices. Their correct functioning can be ascertained by techniques such as physical modeling and unitary experimental testing. But in general, physical models are better evaluated by mathematical methods, which encompass "formal methods".

An important objective for the PROOFS workshop is to bridge the gap between both topics, and therefore to pave the way to "security by clarity" in the design and the evaluation of embedded systems.

The steering committee:
Sylvain Guilley,
Çetin Kaya Koç,
David Naccache,
Akashi Satoh,
Werner Schindler.

The general chair:
Çetin Kaya Koç.

Forewords

For this second edition, the PROOFS workshop will be held on one day, with a program that includes:

- one invited talk,
- four contributed standard talks,
- three contributed short talks,
- two short talks in the "WiP" (Work in Progress) session.

Enjoy the workshop!

Venue of the PROOFS 2013 workshop: UCSB campus.

Acknowledgements:
We are grateful to **Institut MINES/TELECOM – TELECOM-ParisTech** and **Secure-IC S.A.S.** for the generous sponsorship of the workshop, and to **UCSB** *Department of Computer Science* and *Campus Conference Services* for the perfect help in its organization.

The programme committee (PC, listed at page 133) reviewed the papers, using the easychair conference management system. Each submission has been evaluated by three PC members. The submissions that involved at least one PC member as co-author have been evaluated by four PC members (of course excluding those who cosigned the submission).
Amongst the 12 submissions:

- 4 have been accepted for a *regular talk* (30 minute speaking time slot),
- 3 have been accepted for a *short talk* (20 minute speaking time slot), and
- 5 have been rejected.

Up-to-date information can be found on the workshop permanent web site:

`http://www.proofs-workshop.org/`

Contents

Program of PROOFS

UCSB, USA, CA — Saturday August 24th, 2013

8h30–9h00	Registration, at Corwin Pavilion lobby (same location as the CHES registration), coffee. The workshop will take place in Flying A Room
9h00–9h15	Opening – Welcome, presentation of PROOFS 2013
9h15–10h15	Session 1: formal analysis of fault attacks Chair: Rob Bekkers.
• "**Formal verification of a software countermeasure against instruction skip attacks**", by *Karine Heydemann, Nicolas Moro, Emmanuelle Encrenaz and Bruno Robisson.*	
• "**A formal proof of countermeasures against fault injection attacks on CRT-RSA**", by *Pablo Rauzy and Sylvain Guilley.*	
10h15–10h30	Coffee break
10h30–11h30	Session 2: formal analysis of countermeasures against side-channel attacks Chair: Jean-Pierre Seifert.
• "**Towards Fresh Re-Keying with Leakage-Resilient PRFs: Cipher Design Principles and Analysis**", by *Sonia Belaid, Fabrizio De Santis, Johann Heyszl, Stefan Mangard, Marcel Medwed, Jörn-Marc Schmidt, François-Xavier Standaert and Stefan Tillich.*	
• "**Understanding the Limitations and Improving the Relevance of SPICE Simulations in Side-Channel Security Evaluations**", by *Dina Kamel, Mathieu Renauld, Denis Flandre and François-Xavier Standaert.*	
11h30–12h30	Invited keynote talk
• "**Better Provability through Computer Architecture**", by *Timothy Sherwood.*	
12h30–14h00	Lunch
14h00–15h00	Session 3: Formal design methods Chair: Kris Gaj.
• "**Formal Design of Composite Physically Unclonable Function**", by *Durga Prasad Sahoo, Debdeep Mukhopadhyay and Rajat Subhra Chakraborty.*	
• "**A hierarchical graph-based approach to generating formally-proofed Galois-field multipliers**", by *Kotaro Okamoto, Naofumi Homma and Takafumi Aoki.*	
• "**Trojan-Resilient Circuits**", by *Christoph Bayer and Jean-Pierre Seifert.*	
15h00–15h25	"Work in Progress" session
15h25–15h30	Wrap-up

Formal verification of a software countermeasure against instruction skip attacks

Karine Heydemann[1], Nicolas Moro[1,2], Emmanuelle Encrenaz[1], and Bruno Robisson[2]

[1] Laboratoire d'Informatique de Paris 6 (LIP6), UPMC Univ Paris 06
{karine.heydemann,emmanuelle.encrenaz}@lip6.fr, nicolas.moro@etu.upmc.fr
[2] Commissariat à l'Énergie Atomique et aux Énergies Alternatives (CEA)
{nicolas.moro,bruno.robisson}@cea.fr

Abstract. Fault attacks against embedded circuits enabled to define many new attack paths against secure circuits. Every attack path relies on a specific fault model which defines the type of faults that the attacker can perform. On embedded processors, a fault model in which an attacker is able to skip an assembly instruction is practical and has been obtained by using several fault injection means. To handle this issue, some countermeasure schemes which rely on temporal redundancy have been proposed. Nevertheless, double fault injection in a long enough time interval is practical and can bypass those countermeasure schemes. Some fine-grained other countermeasure schemes have been proposed for specific instructions. However, to the best of our knowledge, no approach that enables to secure a generic assembly program in order to make it fault-tolerant to instruction skip attacks has been formally proven yet. In this paper, we provide a fault-tolerant replacement sequence for every instruction of the whole Thumb2 instruction set and provide a formal proof of this fault tolerance. This simple transformation enables to add a reasonably good security level to an embedded program and makes practical fault injection attacks much harder to achieve.

Keywords: microcontroller, fault attack, instruction skip, countermeasure, formal proof

1 Introduction

Physical attacks were introduced in the late 1990s as a new way to break cryptosystems. Unlike classical cryptanalysis, they use some weaknesses in the cryptosystems' implementations as a way to break them. Among them, faults attacks were introduced in 1997 by Boneh *et al.* [1]. In this class of attacks, attackers try to modify a circuit's environment in order to change its behaviour or induce faults into its computations [2]. This attack principle was first introduced against cryptographic circuits but can be used against a larger set of embedded circuits. Many physical means can be used to induce such faults: laser shots [3] [4], overclocking [5], chip underpowering [6], temperature increase [7] or electromagnetic glitches [4] [8].

Among fault attacks, three subclasses can be distinguished: differential fault analysis, safe error and algorithm modifications. Differential fault analysis (DFA) aims at retrieving some ciphering keys by comparing correct ciphertexts with ciphertexts obtained from a faulted encryption [1] [9]. Safe-error attacks are based on the fact that a fault injection may have or not have an impact on the output [10]. Finally, algorithm modifications target an embedded processor and aim at injecting faults into an embedded program's control flow [11] [12].

Those attack schemes rely on an attacker's fault model which defines the set of faults an attacker can perform [13,14]. As a consequence, countermeasure schemes must take this fault model aspect into account. On microcontrollers and embedded processors, the fault model in which an attacker can skip an assembly instruction has been observed on different architectures [11] [15] and for different fault injection means [12] [8] [16]. As a consequence, this fault model is a realistic threat for an embedded program.

In this paper, we consider this instruction skip fault model and propose a countermeasure scheme that could enable to secure any assembly code against instruction skip faults. Some countermeasures based on multiple executions of a function have already been proposed and can theoretically handle this issue [2]. However, this kind of high granularity temporal redundancy is vulnerable to multiple fault attacks. Even with commonly-used low-cost fault injection means, a high temporal accuracy can be obtained by an attacker, and performing the same fault injection on several executions of an algorithm is practical [16]. On the contrary, performing a multiple fault on two instructions separated by a few clock cycles is significantly harder [17] while still possible. Indeed, it requires a much more costly fault injection equipment and very high synchronization capabilities. It is then not yet considered as a threat.

Our approach uses an instruction-scale temporal redundancy to ensure a fault-tolerant execution of an embedded program. It is based on the statement that performing two faults on two instructions separated by few clock cycles is hardly feasible. In this paper, we propose a fault-tolerant replacement sequence for each instruction of the whole Thumb2 instruction set. We also show how to formally prove the fault tolerance of replacement sequences by using a model-checking tool. By using such a fine-grained redundancy scheme, it is then possible to strengthen any assembly program against fault attacks without any specific knowledge about it. In the experimental results, we evaluate the overhead induced by fault tolerance and show that it can be reduced by only applying this countermeasure scheme to the sensitive parts of an implementation.

The rest of this paper is organized as follows. Section 2 introduces our fault model and gives details about some related research papers. Section 3 introduces our countermeasure scheme and details our replacement sequences. Section 4 explains the approach we use for the formal proof. Finally, section 5 evaluates the efficiency of our countermeasure scheme on an AES implementation.

2 Related works and fault model

This section is dedicated to related works. First, fault models are discussed in section 2.1. Countermeasure schemes that have previously been proposed are addressed in section 2.2. Section 2.3 presents some related research papers on formal verification.

2.1 Fault model

On embedded processors, a fault model in which an attacker can skip an assembly instruction or equivalently replace it by a `NOP` has been observed on several architectures and for several fault injection means [13]. On a 8-bit AVR microcontroller, Schmidt *et al.* [11] and Balasch *et al.* [12] obtained instruction skip effects by using clock glitches. Dehbaoui *et al.* obtained the same kind of effects on another 8-bit AVR microcontroller by using electromagnetic glitches [8]. On a 32-bit ARM9 processor, Barenghi *et al.* obtained some instruction skip effects by using voltage glitches. On a more recent 32-bit ARM Cortex-M3 processor, Trichina *et al.* were able to perform instruction skips by using laser shots [16]. Moreover, this fault model has also been used as a basis for several cryptanalytic attacks [18] [14]. As a consequence, it is considered as a common fault model an attacker may be able to perform [19].

A more generic fault model is the instruction replacement model, in which `NOP` replacements correspond to one possible case. In some previous experiments on an ARM Cortex-M3 processor by using electromagnetic glitches, we have observed a corruption of the instructions binary encodings during the bus transfers [20] leading to such instruction replacements. Actually, instruction skips correspond to specific cases of instruction replacements: replacing an instruction by another one that does not affect any useful register has the same effect as a `NOP` replacement and so is equivalent to an instruction skip. Many injection means enable to perform instruction replacement attacks [20] [12]. Nevertheless, even with very accurate fault injection means, being able to precisely control an instruction replacement is a very tough task and, to the best of our knowledge, no practical attack based on such a fault model has been published yet.

As a conclusion, we consider in this paper that an attacker is able to skip an instruction.

2.2 Countermeasure schemes

Several countermeasures schemes have been defined to protect embedded processor architectures against specific fault models. At a hardware level, many countermeasures have been proposed. As an example, Nguyen *et al.* [21] propose to use integrity checks to ensure that no instruction replacement took place.

Software-only countermeasure schemes, which aim at protecting the assembly code, are more flexible and avoid any modification of the hardware. Against fault

attacks, the most common software fault detection approach relies on function-level temporal redundancy [2]. For example, this principle applied to a cryptographic implementation can be achieved by calling twice the same encryption algorithm on the same input and then comparing the outputs. For encryption algorithms, an alternative way is to call the deciphering algorithm on the output of an encryption and to compare its output with the initial input. These approaches enable fault detection and involves doubling the execution time of the algorithm. Triplication approaches with voting enabling fault tolerance at the price of tripling the execution time of the whole algorithm has also been proposed [2].

At an algorithm level, in [22], Medwed *et al.* propose a generic approach based on the use of specific algebraic structures named $AN+B$ codes. Their approach enables to protect both the control and data flow. An application to an AES implementation has also been detailed in [23].

At an assembly level, in [17], Barenghi *et al.* propose three countermeasure schemes based on instruction duplication, instruction triplication and parity checking. Their approach enables to ensure a fault detection for the *load* instructions against instruction skip or transient data corruption fault models. Our scheme enables a fault tolerance only against the instruction skip fault model but for every instruction of the whole considered instruction set. Moreover, our countermeasure scheme has been formally proven fault tolerant.

2.3 Formal verification of software countermeasures

Formal methods and formal verification tools have been used for cryptographic protocols' verification of to check that an implementation could meet the Common Criteria security specifications [24]. However, to the best of our knowledge, very few formal proof approaches to check the correctness of software countermeasure schemes against fault attacks have been proposed yet. The most significant contribution has been proposed by Christofi *et al.* [25]. Their approach aims at performing a source code level verification of the effectiveness of a countermeasure scheme on a CRT-RSA implementation by using the Frama-C program analyzer. In this paper, we formally prove all our proposed countermeasures against an instruction skip fault model at an assembly level.

3 Countermeasure scheme

The proposed countermeasure scheme aims at ensuring a fault-tolerant execution of an assembly code against instruction skip faults. The approach we propose relies on providing a formally proven fault-tolerant replacement sequence for each assembly instruction of a whole instruction set. We chose the ARM Thumb2 instruction set [26] since ARM is a widely used target architecture for embedded processors. Thumb2 is actually the successor to both ARM and Thumb instruction sets, and contains both 16-bit and 32-bit instructions. In this section, we present some of the replacement sequences we have defined for each instruction of the Thumb2 instruction set. This fine-grained redundancy scheme enables to

strengthen any assembly code against fault attacks without any specific knowledge about it.

3.1 Instruction classes

We have defined a fault-tolerant replacement sequence for each instruction and each encoding of the Thumb2 instruction set. This instruction set contains 151 instructions, and each instruction has up to four different encodings. For many instructions, the replacement sequence is very simple. However, this sequence can become much more complex for some specific instructions. According to the replacement sequences found, the instructions in the Thumb2 instruction set can be divided into three classes. Every class is associated to one kind of replacement sequence. These three classes are summarized in table 1.

Table 1. Instruction classes in the Thumb2 instruction set

Instruction class	**Examples**	**Replacement scheme**
Idempotent instructions	`mov r1,r8` `add r3,r1,r2`	Instruction duplication
Separable instructions	`add r1,r1,#1` `push {r4,r5,r6}`	Use of extra registers and decomposition into an idempotent instruction sequence

Specific instructions	`bl <function>`
	`it` blocks
	`adcs r3,r1,r2`
		Replacement sequence specific to each instruction

The first class is composed of the idempotent instructions which only need to be duplicated to provide a fault tolerance. The second class gathers the instructions that are not idempotent but can be replaced by an equivalent sequence of idempotent instructions. The third class gathers some specific instructions that cannot easily be replaced by a list of idempotent instructions but for which a specific replacement sequence is possible. This last class also contains the instructions for which no replacement sequence that ensures a fault tolerance and a correct execution in any case can be provided. The solution for these instructions is either to avoid the compiler to use them or to use a fault detection approach. The following section gives more details about those classes. Moreover, it provides some examples of replacement sequences for every class.

3.2 Individual instruction replacement sequences

Idempotent instructions Idempotent instructions are the instructions that have the same effect when executed once or twice. If all the source operands are different from the destination operands, and if the value written into the destination operands does not depend on the instruction's location in the code, then the instruction is said to be idempotent. For such instructions, the countermeasure is a simple instruction duplication. The overhead for such a duplication is twofold: an overhead equals to the instruction size in terms of code size and a performance overhead that is equal to the time execution time for the instruction.

Table 2 gives some examples of idempotent instructions and their associated replacement sequence.

Table 2. Replacement sequences for some idempotent instructions

Instruction	Replacement sequence
`mov r1,r8` (copies `r8` into `r1`)	`mov r1, r8` `mov r1, r8`
`ldr r1, [r8, r2]` (loads the value at the address `r8+r2` into `r1`	`ldr r1, [r8, r2]` `ldr r1, [r8, r2]`
`str r3, [r2, #10]` (stores `r3` at the address `r2+10`)	`str r3, [r2, #10]` `str r3, [r2, #10]`
`add r3,r1,r2` (puts `r1+r2` into `r3`)	`add r3,r1,r2` `add r3,r1,r2`

Separable instructions In the considered instruction set, some instructions are not idempotent but can be replaced by a sequence of fully idempotent instructions whose execution gives the same result. Once this separation is performed, each idempotent instruction of the replacement sequence can then be duplicated. This class gathers the instructions whose destination register is also a source register. To replace these instruction by a sequence of fully idempotent instructions, an extra register has to be used. This register has to be available at this location in the code: any dead register can be used[3]. Listing 1.1 shows the replacement sequence for an `add r1, r3` instruction. For this class of instructions, the overhead cost brought by our countermeasure scheme depends on the instruction to replace. There is an overhead cost in code size, performance and register pressure (since the replacement sequence needs some extra registers). For the `add r1, r3` instruction example, one extra register is needed and 3 extra instructions, the overhead cost in terms of code size is between 6 and 10 bytes (depending on the encoding used for the `add` instruction).

Listing 1.1. Replacement sequence for the non idempotent `add r1, r3` instruction

```
1 ; we assume rx is an
2 ; available register
3 mov rx, r1
4 mov rx, r1
5 add r1, rx, r3
6 add r1, rx, r3
```

Listing 1.2. Replacement sequence for the `push {r1, r2, r3, lr}` instruction

```
1 ; the push{} instruction
2 ; is equivalent to the
3 ; stmdb sp!,{} instruction
4 stmdb sp, {r1, r2, r3, lr}
5 stmdb sp, {r1, r2, r3, lr}
6 sub.W   rx, sp, #16
7 sub.W   rx, sp, #16
8 mov sp, rx
9 mov sp, rx
```

[3] It turns out that, in the ARM calling conventions, the `r12` register can be used to hold intermediate values and does not need to be saved on the stack. Thus, this register can be used as a temporary register for such replacement scenarios.

Stack manipulation instructions Some memory access instructions update the address register before or after (`stmdb`[4]/`ldmia`[5]) a memory access. As a consequence, this address register is both a source and a destination register for such an instruction. This is notably the case of the stack manipulation instructions (`push` and `pop`). These instructions respectively write or read on the stack and decrement or increment the stack pointer. Such instructions can be separated into a sequence of instructions that only perform one operation at a time, either a memory access or an address register update. The `push` instruction can be decomposed into instructions that first write the register to save on the stack and then decrement the stack pointer. As decrementing the stack pointer implies reading and writing the same register, this operation is decomposed into two steps in order to get a sequence of idempotent instructions. Such a replacement sequence for the `push` instruction is detailed on listing 1.2. This replacement requires 1 extra register and has a code size and performance overhead of 5 instructions.

Listing 1.3. Replacement sequence for `umlal rlo, rhi, rn, rm` instruction that performs `rhi:rlo = rn*rm + rhi:rlo`

```
1  mrs   rt, APSR   ; save
2  mrs   rt, APSR   ; flags
3  umull rx, ry, rn, rm
4  umull rx, ry, rn, rm
5  adds  rz, rx, rlo
6  adds  rz, rx, rlo
7  addc  rx, ry, rhi
8  addc  rx, ry, rhi
9  mov   rlo, rz
10 mov   rlo, rz
11 mov   rhi, rx
12 mov   rhi, rx
13 msr   APSR, rt   ; restore
14 mrs   APSR, rt   ; flags
```

`umlal` instruction The `umlal` instruction multiplies two source registers and then adds the content of the concatenation of the two 32-bit destination registers. The final result is written into the two 32-bit destination registers. As a consequence, this instruction has registers that are both source and destination. However, it can be decomposed. First, a multiply instruction whose result is a 64-bit value can be performed. Then the 64-bit addition has to be decomposed into several instructions. This requires to propagate the carry set by adding the 32 least significant bits (by using an `adds` instruction) to the addition of the 32 most significant bits by using an `adc` instruction. However the `adds` instruction sets the flags whereas the `umlal` does not: this sequence of instructions is not strictly equivalent to the `umlal` instruction and may be wrong if the flags are used after the `umlal` instruction without being set. As a consequence, it is necessary to save

[4] `stmdb` stores multiple registers into the memory in a descending direction
[5] `ldmia` loads a memory segment into multiple registers in an ascending direction

the flags before the sequence and restore them after it. Performing such a saving requires 4 extra instructions. The corresponding replacement sequence for this instruction is given in listing 1.3. This countermeasure requires 4 extra registers and replaces the initial instruction by 14 instructions. This replacement sequence is actually the most costly one of the whole instruction set, both in term of extra registers and extra instructions.

Specific instructions Some instructions cannot easily be replaced by a list of idempotent instructions. These instructions can still be decomposed into an equivalent sequence of instructions that can be duplicated to enforce a robust execution. There are also some instructions for which no fault-tolerant countermeasure in any case can be found. Some of them can still be replaced by a fault-tolerant sequence under some constraints. In this section, we give details and provide some examples for both kinds of such specific instructions.

`bl` subroutine call instruction The subroutine call instruction (`bl`) performs a jump and writes the return pointer into the link register (`r14`). Duplicating a `bl` instruction would induce two subroutine calls if no attack is performed. A possible solution is to manually put the return address into the link register and then perform an unconditional jump. As the Thumb execution mode requires the last bit of an instruction address to be set, this bit must be set before the unconditional jump to the subroutine code, as shown on listing 1.4.

Listing 1.4. Replacement sequence for a `bl <function>` instruction.

```
; Thumb mode requires
; the last bit to be set
 adr   ry,<return_label>
 adr   ry,<return_label>
 add     lr, ry, 1
 add     lr, ry, 1
 b   <function>
 b   <function>
return_label:
```

Listing 1.5. Replacement sequence for `adcs r1, r2, r3` instruction

```
; This sequence is valid
; if the flags are not alive
mrs rx, APSR  ; save
mrs rx, APSR  ; flags
adcs r1, r2, r3
msr APSR, rx  ; restore
msr APSR, rx  ; flags
adcs r1, r2, r3
```

Instructions that both read and write the flags Instructions that read and write the flags cannot easily be replaced by a fault-tolerant sequence of instructions. For example, the `adsc` instruction performs an addition between two source operands (two registers or one register and an immediate value) and the carry flag. The result is written into a destination register and the flags (carry, negative, overflow and zero) are updated. Duplicating such an instruction is not correct since the second `adcs` would use the carry set by the first `adcs` instruction instead of the initial carry value. If the flags are read before being written whatever the execution path after the `adcs` instruction is, then no simple replacement sequence is possible, the code has to be modified. Otherwise, if the flags are written before being used again, a replacement sequence is possible. Such a sequence consists in

saving the flags values before the first `adcs` instruction and restoring these values before the second `adcs` instruction. This replacement sequence is illustrated in listing 1.5.

Listing 1.6. Example of `it` block

```
1    itte NE
2    addne r1, r2, 10
3    eorne r3, r5, r1
4    moveq r3, #10
```

Listing 1.7. Code equivalent to the `it` block above

```
1    b.eq else
2    add r1, r2, 10
3    eor r3, r5, r1
4    b continuation
5  else
6    mov r3, #10
7  continuation
```

Listing 1.8. Code of listing 1.7 strengthened with our individual instruction countermeasure scheme

```
1    b.eq else
2    b.eq else
3    add r1, r2, 10
4    add r1, r2, 10
5    eor r3, r5, r1
6    eor r3, r5, r1
7    b continuation
8    b continuation
9  else
10   mov r3, #10
11   mov r3, #10
12 continuation
```

`it` blocks Thumb2 provides conditional execution of instructions through `it` blocks. An `it` instruction specifies a condition and up to the 4 following instructions can be conditionally executed according to this condition or its inverse. `it` blocks correspond to if-then or if-then-else higher-level constructions and are useful when the branches of a conditional statement are composed of a limited number of instructions. Listing 1.6 gives an example of such an `it` block. The simplest solution for such blocks is to first transform the `it` block into an equivalent classical if-then-else structure such as the one presented on listing 1.8 and then apply the countermeasure scheme to each instruction, as illustrated on listing 1.7. However, we have defined a specific replacement sequence for `it` blocks but this replacement has some limitations and can quickly become more costly than its equivalent form with an if-then-else structure. Due to the lack of space, this replacement sequence is presented in appendix A.

4 Formal proof of fault tolerance

In this section, we present how we formally prove the fault tolerance specification for the countermeasure replacement sequences we presented in section 3. Details about the modelings we use for the proof approach are presented in 4.1 and proof examples for some replacement sequences are presented in 4.2.

4.1 State machine modeling and specification to prove

A program acts as the application of transformations of the values stored in the set of registers or in memory. Each instruction of the program acts like a function whose input is a registers and memory configuration and produces a new registers and memory configuration. The program can be represented as

a transition system whose states are registers and memory configurations and transitions mimic the state transformation applied by the instructions.

Individual instruction modeling Instead of proving the fault tolerance for a complete program, our model checking approach consists in proving the fault tolerance for each replacement sequence proposed in our countermeasure scheme. Indeed, it is sufficient to certify that the output state (register and memory configuration) after the replacement sequence execution (with or without a fault injection) is equivalent to the normal output state after the initial instruction execution. As this output state is also the input state for the following instruction, using such a proof approach certifies that the next instruction will start from the right configuration. Moreover, this enables to use model checking while avoiding state-explosion problem.

State machine modeling We can model the execution of a sequence of instructions by a transition system TS. We define this transition system as $TS = \{S, T, S_0, S_f, L\}$. S is the set of states, T the set of transitions $T : S \rightarrow S$, S_0 and S_f are the subsets of S which respectively gather the initial states and final states. The final states from S_f are absorbing states. A state from S is defined by the value of the different registers (from the set of registers R which includes the program counter) and processor flags (from the set of flags F). Each transition from T is defined by the effect of an instruction on the registers and processor flags. L is a set of labels which correspond to the values the program counter can take. An example of such a transition system for the `add r1, r2, r3]` instruction is shown in Fig. 1. To prove that a countermeasure for an instruction i is robust against a fault, we build two transition systems: a first one for the initial instruction $m(i)$ and another one for its strengthened replacement sequence $m_{cm}(i)$.

```
# input r2, r3, flags
# output r1, flags
pc_init  : add r1, r2, r3
pc_final : next_instruction
```

$pc\_init, pc\_final \in L$
(R, F) is the current state
(R', F') is the next state
$t : (R, F) \rightarrow (R', F')$ with
$R.pc = pc\_init$
$R'.r1 = R.r1 + R.r2$
$R'.pc = pc\_final$

Fig. 1. Transition system for the `add r1, r2, r3` instruction

Fault modeling In any transition system $m_{cm}(i)$, skip fault or data transient fault may occur. An instruction skip fault is modeled by a transition from a state to any following one. Such a faulty transition only modifies the program counter. We add to the whole transition system a skip instruction faulty transition between every pair of adjacent states. As we assume that only one skip instruction fault injection may occur, every fault transition is guarded with a boolean which identifies that a fault has already occurred.

Flags and registers modeling The set of registers is composed of the general-purpose registers (`r0-r12`), the stack pointer (`r13`), the link register (`r14`) and the program counter (`r15`). The 5 processor flags are: C (carry), N (negative), Z (zero), V (overflow), Q (saturation). These flags can be set by some instructions and are used by several others. The conditional jumps are among the instructions that use those flags. Each flag is modeled as a 1-bit register. All the other registers are modeled as 4-bit registers. This width is sufficient to model the arithmetic and logic operations as well as the flags computations and enables to keep a reasonable complexity for the model checker. Moreover, modeling all the registers is not necessary since an instruction only reads a subset of the registers and writes on the destination registers. Besides, according to our fault model, the registers that are not modified by an instruction cannot be modified by a fault. Thus, for a given $m(i)$ or $m_{cm}(i)$, the set of registers R is only composed of the subset of registers that are manipulated by i or its replacement sequence $cm(i)$. Newly introduced registers in $cm(i)$ are supposed to be dead after the occurrence of the instruction i in the initial program.

Memory modeling Since in our fault model we assume the memory cannot be corrupted, modeling the memory is not relevant. To ensure that a write to the memory took place, we only need to ensure that the corresponding instruction has been executed at least once. As explained later in this section, we add a counter variable to $m(i)$ and $m_{cm}(i)$ in order to achieve this. For the loads from the memory, we use symbolic values as the values cannot be corrupted and they also do not matter since the formal proof consists in checking the equivalence for any value. The important point is to give the same symbolic value to any loads at a given address for the transition system $m(i)$ (when i is a *load* instruction) and for $m_{cm}(i)$. This is achieved by adding a variable for each memory address that is read by i and $cm(i)$ to both transition systems. These variables contain the needed symbolic values.

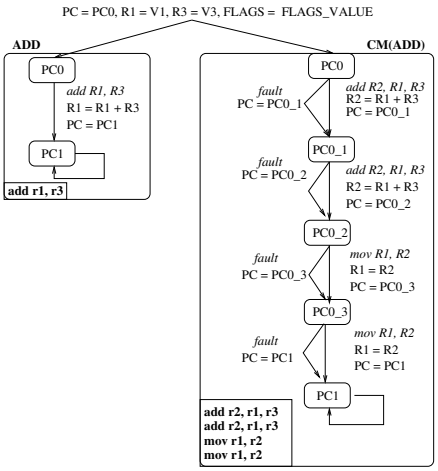
Vis model checker We used the Vis model checker[6] to prove the fault tolerance of our countermeasure scheme. This tool can take as input a transition system described with a subset of the Verilog hardware description language. Using Verilog is convenient to model transition systems which manipulate registers and bit vectors. The Vis model checker supports symbolic model checking techniques which enable to perform the proof in a symbolic way without having to enumerate each value for the registers. The proofs presented in this paper required less than one second to compute.

Specification to prove To prove the equivalence of the output of an instruction and its replacement sequence, we prove the validity of logic formulas on the two modelings. To perform such a proof, we use a specific construction in which the two transition systems $m(i)$ and $m_{cm}(i)$ have the same values for the set of registers R (except for the program counter), the set of flags F and the symbolic

[6] http://vlsi.colorado.edu/~vis/

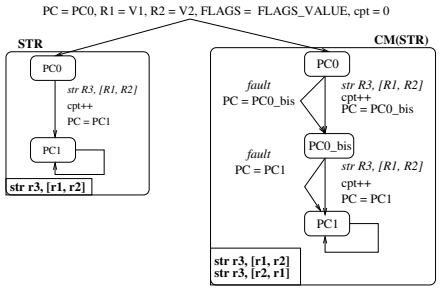
values (for the memory loads) in their initial states. Such constructions are presented in Fig. 2, 3 and 4. We need to prove that $m(i)$ and $m_{cm}(i)$ always reach a final absorbing state. Moreover, we also need to prove that, when $m(i)$ and $m_{cm}(i)$ reach a final state, the values for the set of alive registers R' (except for the program counter) and flags F' are similar. Such properties to check are expressed with the CTL temporal logic.

4.2 Formal proof of fault tolerance for some replacement sequences




```
P1: AF(ADD.PC = PC1)
P2: AF(CM(ADD).PC = PC1)
P3: AG(((ADD.PC=PC1)*(CM(ADD).PC=PC1)) =>
     ADD.R1 = CM(ADD).R1 &
     ADD.FLAGS = CM(ADD).FLAGS)
```

Fig. 2. Modeling one non idempotent instance of the `add` instruction and its countermeasure




```
P1 : AF(STR.PC=PC1)
P2 : AF(CM(STR).PC=PC1)
P3 : AG((STR.PC=PC1 * CM(STR).PC=PC1) =>
     (CM(STR).cpt = 2 + CM(STR).cpt = 1))
```

Fig. 3. Modeling one idempotent instance of the `str` instruction and its countermeasure

Idempotent and separable instructions The left part of Fig. 2 shows the state machine corresponding to the transition system for a non-idempotent `add r1,r3` instruction. The program counter is updated and depending on the instruction, the registers or the flags may be updated too. The replacement sequence uses a dead register `r2` and two extra `mov` instructions to write the result to the destination register `r1`. Its transition system is modeled by the state machine on the right part of Fig. 2. To prove that the replacement sequence is fault tolerant against a possible instruction skip, both state machines are fed with the same values for the source registers (`r1` and `r3`) and flags. Then, the validity of three CTL logic formulas has been checked with the Vis model checker. P1 and P2 express the fact that in both state machines any path from an initial state

goes to a final state. P3 expresses the fact that in this final state, for all possible values in the source registers, the values in `r1` and the flags are identical in $m(i)$ and $m_{cm}(i)$. Fig. 3 presents the transition systems for an idempotent memory write, an `str r3, [r1, r2]` instruction, and its replacement sequence. In this case, as the instruction writes the content of `r3` to the memory at the address `r1`+`r2`, no proof is needed on the value inside the registers. We only need to make sure that at least one `str` instruction has been executed. A counter variable is added to the definition of a state. This counter is set to 0 and is incremented by any transition which corresponds to a `str` instruction. P1 and P2 express the fact that any path goes to the last state. P3 expresses the fact that the number of writes made by the replacement sequence greater or equal to the number of writes made by the initial instruction (which is equal to 1).

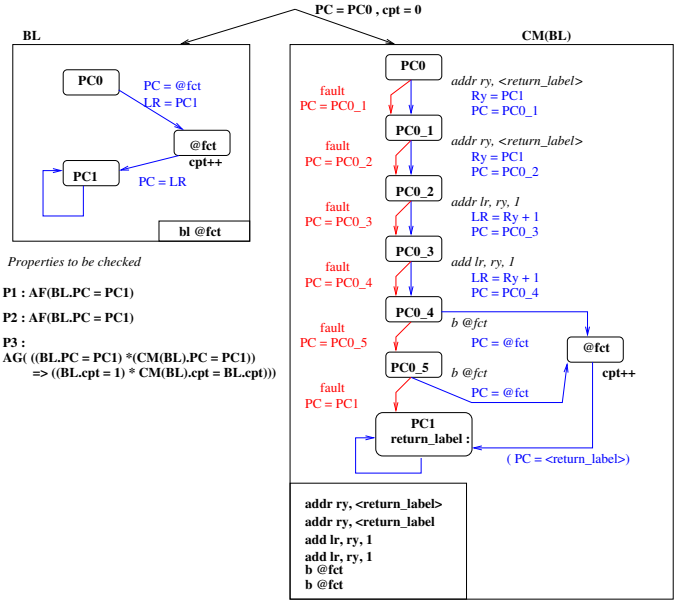


Fig. 4. Transition systems for the `bl` instruction and its replacement sequence

Specific instructions

Subroutine call: the `bl` instruction Figure 4 shows the state machines for the `bl` instruction and its replacement sequence. In both corresponding transition systems, we have added a label $@fct$ to model the target of the subroutine call. Transitions from a state in which $PC = @fct$ assign the link register to the PC. Such a transition models the return of the function and also increments a counter. Then, properties P1 and P2 to be checked by the model checker express that any path from an initial state goes to a final state. Property P3 expresses the fact that in a final state the number of calls to the function (the counter values) are the same. Validity of property P3 ensures that the function has been

executed only once while validity of P1 and P2 ensures that the control flow comes back to the calling function.

Instructions that read and write the flags For the `adcs` instruction and its replacement sequence, as presented in listing 1.5, the CTL properties are the same as the ones that were used for the `add` instruction. However, the property that deals with the equality of the destination register and the flags is not valid if a fault targets the last `adcs` instruction. Relaxing the constraint on flags equality (expressed as `LIGHT_RESULT` below) makes this property valid as shown with the output of the Vis model checker in Fig. 5. To sum up, this countermeasure can only be used if the flags are not used before being set again after the `adcs` instruction.

```
MC: formula passed - AG(AF(adcs.pc=PC1))
MC: formula passed - AG(AF(cm(adcs).pc=PC1))
MC: formula passed - AG(((adcs.pc=PC1*cm(adcs).pc=PC1)->LIGHT_RESULT=1))
MC: formula failed - AG(((adcs.pc=PC1*cm(adcs).pc=PC1)->RESULT=1))
```

Fig. 5. VIS Model Checker output for the equivalence checking of the `adcs` instruction

5 Application to an AES implementation

In this section, we applied our countermeasure scheme to an implementation of the AES-128 symmetric encryption algorithm. In our implementation, every round key is calculated before the associated `AddRoundKey` operation. We provide an estimation of the overhead cost brought by our countermeasure scheme and perform an exhaustive instruction skip simulation on an ARM Cortex-M3 microcontroller to confirm the effectiveness of our approach. The chosen target is an up-to-date 32-bit microcontroller based on the ARM Cortex-M3 processor [27]. This microcontroller uses an ARMv7-M Harvard architecture and runs the Thumb2 instruction set [26].

Estimation of the overhead cost The overhead cost in terms of clock cycles and code size for two implementations that use our countermeasure scheme is shown on table 3. In the first implementation, the whole code has been strengthened with our methodology. Both overhead costs are high with this implementation. Another approach consists in applying our countermeasure to the last two rounds. In terms of cryptanalysis, fault injections are supposed to be harder to exploit if the fault does not target the last two rounds. This last scenario is just an example of a possible optimization. It enables to reduce the clock cycles overhead by strengthening some specific parts of an algorithm but some cryptanalysis attacks may still exist against such an implementation.

The overhead cost brought by our countermeasure scheme is high, but remains comparable to the one brought by classical algorithm triplication or other software approaches for fault tolerance. However, unlike such classical algorithm duplication or triplication approaches, our countermeasure scheme should be resistant to double fault attacks in a time interval longer than a few clock cycles.

Table 3. Countermeasures overhead for an AES implementation

	Clock cycles	**Relative increase**	**Code size**	**Relative increase**
AES - without countermeasure	9595		490 bytes	
AES - whole code with CM	20503	113.7 %	1480 bytes	202 %
AES - last two rounds with CM	11374	18.6 %	1874 bytes	282.5 %

6 Conclusion

In this paper, we have presented a countermeasure scheme that enables to strengthen an embedded program and make it tolerant to instruction skip faults. In our countermeasure scheme, we have build a fault-tolerant replacement sequence for each instruction of the whole Thumb2 instruction set. The instructions can be separated into three classes, which all have their dedicated replacement sequences. We have also provided a formal proof in order to guarantee the correctness and the fault tolerance of our replacement sequences for each class of instructions.

Finally, we do not claim our scheme enables a full protection against fault attacks. Nevertheless, such an approach enables to add a reasonably good security level to an embedded program, without requiring any extra hardware countermeasure and any specific knowledge about the embedded program. The overhead cost brought by using such a countermeasure is comparable to the extra cost brought by using classical algorithm-level temporal redundancy approaches and can be reduced with a more accurate knowledge about the sensitive parts that should be protected. Moreover, using a very fine-grained redundancy at the instruction scale makes the multiple fault attacks less practical with a reasonable cost equipment.

In future works, we will try to extend the fault model to a global bus corruption fault model in which data loads from the memory can also be corrupted. This fault model extension will require some changes in our fault tolerance proofs. Furthermore, we also aim at performing a practical evaluation of our countermeasure scheme by trying to attack a secured implementation on a real microcontroller with real fault injection means.

References

1. Boneh, D., DeMillo, R.A., Lipton, R.J.: On the Importance of Checking Cryptographic Protocols for Faults. Journal of Cryptology **1233** (1997)
2. Bar-El, H., Choukri, H., Naccache, D., Tunstall, M., Whelan, C.: The Sorcerer's Apprentice Guide to Fault Attacks. Proceedings of the IEEE (February 2006)
3. Skorobogatov, S.P., Anderson, R.J.: Optical Fault Induction Attacks. Cryptographic Hardware and Embedded Systems - CHES 2002 **2523**(August) (2003)
4. Schmidt, J.M., Hutter, M.: Optical and EM Fault-Attacks on CRT-based RSA: Concrete Results. In: Austrochip 2007, Graz, Austria

5. Agoyan, M., Dutertre, J.m., Naccache, D., Robisson, B., Tria, A.: When Clocks Fail: On Critical Paths and Clock Faults. In: CARDIS 2010. (2010)
6. Zussa, L., Dutertre, J.m., Clédière, J., Robisson, B., Tria, A.: Investigation of timing constraints violation as a fault injection means. In: DCIS 2012
7. Skorobogatov, S.: Local Heating Attacks on Flash Memory Devices. In: IEEE International Workshop on Hardware-Oriented Security and Trust - HOST'09. (2009)
8. Dehbaoui, A., Dutertre, J.M., Robisson, B., Tria, A.: Electromagnetic Transient Faults Injection on a Hardware and a Software Implementations of AES. FDTC 2012
9. Biham, E., Shamir, A.: Differential Fault Analysis of Secret Key Cryptosystems. In: CRYPTO'1997, Santa Barbara, California, USA (1997)
10. Yen, S., Joye, M.: Checking before output may not be enough against fault-based cryptanalysis. Computers, IEEE Transactions on **49** (2000)
11. Schmidt, J.M., Herbst, C.: A Practical Fault Attack on Square and Multiply. In: FDTC 2008, IEEE (August 2008)
12. Balasch, J., Gierlichs, B., Verbauwhede, I.: An In-depth and Black-box Characterization of the Effects of Clock Glitches on 8-bit MCUs. In: FDTC 2011, IEEE
13. Verbauwhede, I., Karaklajic, D., Schmidt, J.M.: The Fault Attack Jungle - A Classification Model to Guide You. In: FDTC 2011, IEEE (2011)
14. Barenghi, A., Breveglieri, L., Koren, I., Naccache, D.: Fault Injection Attacks on Cryptographic Devices: Theory, Practice, and Countermeasures. Proceedings of the IEEE **100**(11) (November 2012) 3056–3076
15. Barenghi, A., Bertoni, G.M., Breveglieri, L., Pelosi, G.: A fault induction technique based on voltage underfeeding with application to attacks against aes and rsa. J. Syst. Softw. **86**(7) (July 2013) 1864–1878
16. Trichina, E., Korkikyan, R.: Multi Fault Laser Attacks on Protected CRT-RSA. In: 2010 Workshop on Fault Diagnosis and Tolerance in Cryptography, IEEE (2010)
17. Barenghi, A., Breveglieri, L., Koren, I., Pelosi, G., Regazzoni, F.: Countermeasures against fault attacks on software implemented AES. In: Proceedings of the 5th Workshop on Embedded Systems Security - WESS'10, New York, USA (2010)
18. Schmidt, J.M., Medwed, M.: A Fault Attack on ECDSA. In: FDTC 2009, IEEE
19. Karaklajić, D., Schmidt, J.M., Verbauwhede, I.: Hardware Designer's Guide to Fault Attacks. IEEE Transactions on Very Large Scale Integration Systems (2013)
20. Moro, N., Dehbaoui, A., Heydemann, K., Robisson, B., Encrenaz, E.: Electromagnetic fault injection: towards a fault model on a 32-bit microcontroller. In: FDTC 2013, Santa Barbara, California, USA (2013)
21. Nguyen, M.H., Robisson, B., Agoyan, M., Drach, N.: Low-cost recovery for the code integrity protection in secure embedded processors. In: 2011 IEEE International Symposium on Hardware-Oriented Security and Trust, IEEE (June 2011) 99–104
22. Medwed, M., Schmidt, J.M.: A Generic Fault Countermeasure Providing Data and Program Flow Integrity. In: FDTC 2008, IEEE (2008)
23. Medwed, M.: A Continuous Fault Countermeasure for AES Providing a Constant Error Detection Rate. In: FDTC 2010, IEEE (2010)
24. Chetali, B., Nguyen, Q.h.: Industrial Use of Formal Methods for a High-Level Security Evaluation. In: FM 2008: Formal Methods. (2008) 198–213
25. Christofi, M., Chetali, B., Goubin, L., Vigilant, D.: Formal verification of a CRT-RSA implementation against fault attacks. Journal of Crypt. Eng. (2013)
26. ARM: ARM Architecture Reference Manual - Thumb-2 Supplement (2005)
27. Yiu, J.: The Definitive Guide To The ARM Cortex-M3. Elsevier Science (2009)

A Replacement sequence for `it` blocks

Listing 1.9. Example of `it` block

```
1   itte NE
2   addne r1, r2, 10
3   eorne r3, r5, r1
4   moveq r3, #10
```

Listing 1.10. Second step for replacement of the `it` block given in listing 1.9

```
1   it??? NE
2   it??  NE
3   addne r1, r2, 10
4   addne r1, r2, 10
5   eorne r3, r5, r1
6   it??? NE
7   it??  NE
8   eorne r3, r5, r1
9   moveq r3, #10
10  moveq r3, #10
```

Listing 1.11. First step for replacement of the `it` block given in listing 1.9

```
1   it??? NE
2   it??  NE
3   addne r1, r2, 10
4   addne r1, r2, 10
5   eorne r3, r5, r1
6   eorne r3, r5, r1
7   moveq r3, #10
8   moveq r3, #10
```

Listing 1.12. Replacement sequence of the `it` block given in listing 1.9

```
1   itttt NE
2   ittt  NE
3   addne r1, r2, 10
4   addne r1, r2, 10
5   eorne r3, r5, r1
6   ittee NE
7   itee  NE
8   eorne r3, r5, r1
9   moveq r3, #10
10  moveq r3, #10
```

Thumb2 provides conditional execution of instructions through `it` blocks (`it` stands for *if true*). An `it` instruction specifies a condition and up to the 4 following instructions can be conditionally executed according to this condition or its inverse. `it` blocks correspond to if-then or if-then-else higher-level constructions. Such an instruction is useful when the branches of a conditional statement are composed of a limited number of instructions. Listing 1.9 gives an example of such an `it` block. If the condition NE is set, (*i.e.* if the Z flag has been set by previous instructions), then the two following instructions (`addne` and `eorne`) are executed. Otherwise, the last two instructions (`subeq` and `moveq`) are executed. The whole `it` block needs to be considered in order to propose a countermeasure. The solution we propose is to first apply our countermeasure scheme to every instruction of the `it` block. Every instruction of a replacement sequence keeps the same condition as the initial instruction. The first `it` instruction is duplicated. The second `it` instructions will specify one instruction less than the first one. Moreover, they are to be updated depending on the instructions that will result of the replacement sequence of the initial `it` block. This step is presented in listing 1.11. The second step consists in adding some `it` instructions,

since `it` blocks cannot contain more than 4 instructions, as illustrated in 1.10. Finally, the conditions set in the `it` instructions need to be updated to match with the instructions of the `it` block they define. Listing 1.12 shows the secure code corresponding to the `it` block code example given in listing 1.9.

Note that an `it` instruction should not appear in an `it` block. Thus, in case of a fault targeting one of the duplicated `it` instructions, the code behaves as if there was only one `it` instruction. Otherwise, the second `it` instruction is executed in the `it` block defined by the first `it` instruction. The second `it` instruction has actually no effect and is considered as a `NOP`. However, some compilers may not accept such a construction. In this case, we have to use traditional conditional sequences for if-then or if-then-else constructions and apply our countermeasure scheme to each individual instruction of the resulting code as presented in Section3.2. Moreover, transforming first the `it` block into a classical if-then-else structure and then applying the countermeasure scheme may induce a smaller overhead cost. Such a construction has been previously presented in listings 1.7 and 1.8.

A formal proof of countermeasures against fault injection attacks on CRT-RSA

Pablo Rauzy, Sylvain Guilley
firstname.lastname@telecom-paristech.fr

Télécom ParisTech

Abstract. In this article, we describe a methodology that aims at either breaking or proving the security of CRT-RSA algorithms against fault injection attacks. In the specific case-study of BellCoRe attacks, our work bridges a gap between formal proofs and implementation-level attacks. We apply our results to three versions of CRT-RSA, namely the naive one, that of Shamir, and that of Aumüller *et al.* Our findings are that many attacks are possible on both the naive and the Shamir implementations, while the implementation of Aumüller *et al.* is resistant to all fault attacks with one fault. However, we show that the countermeasure is not minimal, since two tests out of seven are redundant and can simply be removed.

Keywords: RSA (*Rivest, Shamir, Adleman* [13]), CRT (*Chinese Remainder Theorem*), fault injection, BellCoRe (*Bell Communications Research*) attack, formal proof, OCaml.

1 Introduction

It is known since 1997 that injecting faults during the computation of CRT-RSA could yield to malformed signatures that expose the prime factors (p and q) of the public modulus ($N = p \cdot q$). Notwithstanding, computing without the fourfold acceleration conveyed by the CRT is definitely not an option in practical applications. Therefore, many countermeasures have appeared that consist in step-wise internal checks during the CRT computation. To our best knowledge, none of these countermeasures have been proven formally. Thus without surprise, some of them have been broken, and then patched. The current state-of-the-art in computing CRT-RSA without exposing p and q relies thus on algorithms that have been carefully scrutinized by cryptographers. Nonetheless, neither the hypotheses of the fault attack nor the security itself have been unambiguously modeled.

This is the purpose of this paper. The difficulties are *a priori* multiple: in fault injection attacks, the attacker has an extremely high power because he can fault any variable. Traditional approaches thus seem to fall short in handling this problem. Indeed, there are two canonical methodologies: *formal* and *computational* proofs. Formal proofs (*e.g.*, in the so-called Dolev-Yao model) do not

capture the requirement for faults to preserve some information about one of the two moduli; indeed, it considers the RSA as a black-box with a key pair. Computational proofs are way too complicated since the handled numbers are typically $2,048$ bit long.

The state-of-the-art contains one reference related to the formal proof of CRT-RSA: it is the work of Christofi, Chetali, Goubin and Vigilant [6]. For tractability purposes, the proof is conducted on reduced versions of the algorithms parameters. One fault model is chosen authoritatively (the zeroization of a complete intermediate data), which is a strong assumption. In addition, the verification is conducted on a pseudo-code, hence concerns about its portability after compilation into machine-level code. Another reference related to formal proofs against fault injection attacks is the work of Guo, Mukhopadhyay and Karri. In [8], they explicit an AES implementation that is provably protected against differential fault analyses [3]. The approach is purely combinational, because the faults propagation in AES concerns 32-bit words called columns; consequently, all fatal faults (and thus all innocuous faults) can be enumerated.

Contributions. Our contribution is also to reach a full fault coverage on CRT-RSA algorithm, thereby keeping the proof even if the code is transformed (*e.g.*, compiled or partitioned in software/hardware). To this end we developed tools based of symbolic computation in the framework of modular arithmetic, which enable formal analysis of CRT-RSA and its countermeasures against fault injection attacks. We apply our methods on three implementations of CRT-RSA: an unprotected one, one protected by Shamir countermeasure, and one protected by Aumüller *et al.* countermeasure. We find many possible fault injections which enable BellCoRe attacks on an unprotected implementation of the CRT-RSA computation, as well as on one protected by Shamir countermeasure. We formally prove the security of the Aumüller *et al.* countermeasure against the BellCoRe attack, under a fault model that considers *permanent faults* (in memory) and *transient faults* (one-time faults, even on copies of the secret key parts), with or without forcing at zero, and with possibly faults at various locations. We also simplify Aumüller *et al.* countermeasure by proving that two out of the seven tests it consists of are redundant and can be removed.

Organization of the paper. We recall CRT-RSA cryptosystem and the BellCoRe attack in Sec. 2; still from an historical perspective, we explain how the CRT-RSA implementation has been amended to withstand more or less efficiently the BellCoRe attack. Then, in Sec. 3, we define our approach. Sec. 4, Sec. 5, and Sec. 6 are case studies using the methods developed in Sec. 3 of respectively an unprotected version of the CRT-RSA computation, a version protected by Shamir countermeasure, and a version protected by Aumüller *et al.* countermeasure. Conclusions and perspectives are in Sec. 7. To improve the readability of the article, the longest code portions have been consigned in appendix.

2 CRT-RSA and the "BellCoRe" attack

This section recaps known results about fault attacks on CRT-RSA (see also [12] and [15, Chap. 3]). Its purpose is to settle the notions and the associated notations that will be used in the later sections (that contain novel contributions).

2.1 CRT-RSA

RSA is both an *encryption* and a *signature* scheme. It relies on the identity that for all message $0 \leq m < N$, $(m^d)^e \equiv m \mod N$, where $d \equiv e^{-1} \mod \varphi(N)$, by the Euler theorem. In this equation, φ is the Euler totient function, equal to $\varphi(N) = (p-1)\cdot(q-1)$ when $N = p \cdot q$ is a composite number, product of two primes p and q. For example, if Alice generates the signature $S = m^d \mod N$, then Bob can verify it by computing $S^e \mod N$, which must be equal to m unless Alice is pretending to know d although she does not. Therefore (N, d) is called the private key, and (N, e) the public key. In this paper, we are not concerned about the key generation step of RSA, and simply assume that d is an unknown number in $[\![1, \varphi(N) = (p-1)\cdot(q-1)[\![$. Actually, d can also be chosen equal to the smallest value $e^{-1} \mod \lambda(n)$, where $\lambda(n) = \frac{(p-1)\cdot(q-1)}{\gcd(p-1,q-1)}$ is the Carmichael function. The computation of $m^d \mod N$ can be speeded-up by a factor four by using the Chinese Remainder Theorem (CRT). Indeed, figures modulo p and q are twice as short as those modulo N. For example, for $2,048$ bit RSA, p and q are $1,024$ bit long. The CRT-RSA consists in computing $S_p = m^d \mod p$ and $S_q = m^d \mod q$, which can be recombined into S with a limited overhead. Due to the little Fermat theorem (special case of the Euler theorem when the modulus is a prime), $S_p = (m \mod p)^{d \mod (p-1)} \mod p$. This means that in the computation of S_p, the processed data have $1,024$ bit, and the exponent itself has $1,024$ bits (instead of $2,048$ bits). Thus the multiplication is four times faster and the exponentiation eight times faster. However, as there are two such exponentiations (modulo p and q), the overall CRT-RSA is roughly speaking four times faster than RSA computed modulo N.

This acceleration justifies that CRT-RSA is always used if the factorization of N as $p \cdot q$ is known. In CRT-RSA, the private key is a more rich structure than simply (N, d): it is actually comprised of the 5-tuple (p, q, d_p, d_q, i_q), where:

- $d_p \doteq d \mod (p-1)$,
- $d_q \doteq d \mod (q-1)$,
- $i_q \doteq q^{-1} \mod p$.

The "naive" CRT-RSA algorithm is presented in Alg. 1. It is straightforward to check that the signature computed at line 3 belongs to $[\![0, p\cdot q-1]\!]$. Consequently, no reduction modulo N is necessary before returning S.

2.2 BellCoRe attack on CRT-RSA

In 1997, an dreadful remark has been made by Boneh, DeMillo and Lipton [4], three staff of BellCoRe: Alg. 1 could reveal the secret primes p and q if the

Algorithm 1: Naive CRT-RSA

Input : Message m, key (p, q, d_p, d_q, i_q)
Output: Signature $m^d \mod N$

1 $S_p = m^{d_p} \mod p$ /* Signature modulo p */
2 $S_q = m^{d_q} \mod q$ /* Signature modulo q */
3 $S = S_q + q \cdot (i_q \cdot (S_p - S_q) \mod p)$ /* Recombination */
4 **return** S

computation is faulted, even in a very random way. The attack can be expressed as the following proposition.

Proposition 1 (Orignal BellCoRe attack). *If the intermediate variable S_p (resp. S_q) is returned faulted as $\widehat{S_p}$ (resp. $\widehat{S_q}$)*[1]*, then the attacker gets an erroneous signature $\widehat{S}$, and is able to recover p (resp. q) as* $\gcd(N, S - \widehat{S})$.

Proof. For all integer x, $\gcd(N, x)$ can only take 4 values:

- 1, if N and x are coprime,
- p, if x is a multiple of p,
- q, if x is a multiple of q,
- N, if x is a multiple of both p and q, *i.e.*, of N.

In Alg. 1, if S_p is faulted (*i.e.*, replaced by $\widehat{S_p} \neq S_p$), then $S - \widehat{S} = q \cdot \left((i_q \cdot (S_p - S_q) \mod p) - (i_q \cdot (\widehat{S_p} - S_q) \mod p)\right)$, and thus $\gcd(N, S - \widehat{S}) = q$. If S_q is faulted (*i.e.*, replaced by $\widehat{S_q} \neq S_q$), then $S - \widehat{S} \equiv (S_q - \widehat{S_q}) - (q \mod p) \cdot i_q \cdot (S_q - \widehat{S_q}) \equiv 0 \mod p$ because $(q \mod p) \cdot i_q \equiv 1 \mod p$. Thus $S - \widehat{S}$ is a multiple of p. Additionally, $S - \widehat{S}$ is not a multiple of q. So, $\gcd(N, S - \widehat{S}) = p$. □

This version of the BellCoRe attack requires that two identical messages with the same key can be signed; indeed, one signature yields the genuine S while the other one is perturbed, and thus returns $\widehat{S}$. Little later, the BellCoRe attack has been improved by Joye, Lenstra and Quisquater [10]. This time, the attacker can recover p or q with one only faulty signature, provided the input m of RSA is known.

Proposition 2 (One faulty signature BellCoRe attack). *If the intermediate variable S_p (resp. S_q) is returned faulted as $\widehat{S_p}$ (resp. $\widehat{S_q}$), then the attacker gets an erroneous signature $\widehat{S}$, and is able to recover p (resp. q) as* $\gcd(N, m - \widehat{S}^e)$ *(*with an overwhelming probability*).*

[1] In other papers related to faults, the faulted variables (such as X) are noted either with a star (X^*) or a tilde ($\tilde{X}$); in this paper, we use a hat, as it can stretch, hence cover the adequate portion of the variable. For instance, it allows to make an unambiguous difference between a faulted data raised at some power and a fault on a data raised at a given power (contrast $\widehat{X}^e$ with $\widehat{X^e}$).

Proof. By proposition 1, if a fault occurs during the computation of S_p, then $\gcd(N, S - \widehat{S}) = q$ (*most likely*). This means that:

- $S \not\equiv \widehat{S} \mod p$, and thus $S^e \not\equiv \widehat{S}^e \mod p$ (*indeed, if the congruence was true, we would have* $e|p-1$*, which is very unlikely*);
- $S \equiv \widehat{S} \mod q$, and thus $S^e \equiv \widehat{S}^e \mod q$;

As $S^e \equiv m \mod N$, this proves the result. A symmetrical reasoning can be done if the fault occurs during the computation of S_q. □

2.3 Protection of CRT-RSA against BellCoRe attacks

Several protections against the BellCoRe attacks have been proposed. A non-exhaustive list is given below, and then, the most salient features of these countermeasures are described:

- Naive;
- Obvious countermeasures: no CRT, or with signature verification;
- Shamir [14];
- Aumüller *et al.* [1];
- Vigilant, original [16] and with some corrections by Coron *et al.* [7];
- Kim *et al.* [11].

Obvious countermeasures Fault attacks on RSA can be thwarted simply by refraining from implementing the CRT. If this is not affordable, then the signature can be verified before being outputted. Such protection is efficient in practice, but is criticized for two reasons. First of all, it requires an access to e; second, the performances are incurred by the extra exponentiation needed for the verification. This explains why other countermeasures have been devised.

Shamir The CRT-RSA algorithm of Shamir builds on top of the CRT and introduces, in addition to the two primes p and q, a third factor r. This factor r is random and small (less than 64 bit long), and thus co-prime with p and q. The computations are carried out modulo $p' = p \cdot r$ (resp. modulo $q' = q \cdot r$), which allows for a retrieval of the intended results by reducing them modulo p (resp. modulo q), and for a verification by a reduction modulo r. Alg. 2 describes one version of Shamir's countermeasure.

Aumüller The CRT-RSA algorithm of Aumüller *et al.* is a variation of that of Shamir, that is primarily intended to fix two shortcomings. First it removes the need for d in the signature process, and second, it also checks the recombination step. The countermeasure, given in Alg. 3, introduces, in addition to p and q, a third prime t. The computations are done modulo $p' = p \cdot t$ (resp. modulo $q' = q \cdot t$), which allows for a retrieval of the intended results by reducing them modulo p (resp. modulo q), and for a verification by a reduction modulo t.

Algorithm 2: Shamir CRT-RSA

Input : Message m, key (p, q, d, i_q),
32-bit random prime r
Output: Signature $m^d \mod N$,
or error if some fault injection has been detected.

1 $p' = p \cdot r$
2 $d_p = d \mod (p-1) \cdot (r-1)$
3 $S'_p = m^{d_p} \mod p'$ `/* Signature modulo` p' `*/`
4 $q' = q \cdot r$
5 $d_q = d \mod (q-1) \cdot (r-1)$
6 $S'_q = m^{d_q} \mod q'$ `/* Signature modulo` q' `*/`
7 $S_p = S'_p \mod p$
8 $S_q = S'_q \mod q$
9 $S = S_q + q \cdot (i_q \cdot (S_p - S_q) \mod p)$ `/* Same as in line 3 of Alg. 1 */`
10 **if** $S'_p \not\equiv S'_q \mod r$ **then**
11 | **return** error
12 **else**
13 | **return** S
14 **end**

However, the verification is more subtle than for the case of Shamir. In Shamir's CRT-RSA (Alg. 2), the verification is *symmetrical*, in that the computations modulo $p \cdot r$ and $q \cdot r$ operate on the same object, namely m^d. In Aumüller *et al.*'s CRT-RSA (Alg. 3), the verification is *asymmetrical*, since the computations modulo $p \cdot t$ and $q \cdot t$ operate on two different objects, namely $m^{d_p \mod (t-1)}$ and $m^{d_q \mod (t-1)}$. The verification consists in an identity that resembles that of ElGamal for instance: $(m^{d_p \mod (t-1)})^{d_q \mod (t-1)} \equiv (m^{d_q \mod (t-1)})^{d_p \mod (t-1)} \mod t$. Specifically, if we note S'_p the signature modulo p', then $S_p = S \mod p$ is equal to $S'_p \mod p$. Furthermore, let us denote $S_{pt} = S'_p \mod t$, $S_{qt} = S'_q \mod t$, $d_{pt} = d_p \mod (t-1)$ and $d_{qt} = d_q \mod (t-1)$. It can be checked that those figures satisfy the identity: $S_{pt}^{d_{qt}} \equiv S_{qt}^{d_{pt}} \mod t$, because both terms are equal to $m^{d_{pt} \cdot d_{qt}} \mod t$. The prime t is referred to as a security parameter, as the probability to pass the test (at line 23 of Alg. 3) is equal to $1/t$ (*i.e.*, about 2^{-32}), assuming a uniform distribution of the faults. Indeed, this is the probability to find a large number that, once reduced modulo t, matches a predefined value.

Alg. 3 does some verifications during the computations, and reports an error in case a fault injection can cause a malformed signature susceptible of unveiling p and q. More precisely, an error is returned in either of these seven cases:

1. p' is not a multiple of p (*because this would amount to faulting p in the naive algorithm*)
2. $d'_p = d_p + \mathsf{random}_1 \cdot (p-1)$ is not equal to $d_p \mod (p-1)$ (*because this would amount to faulting d_p in the naive algorithm*)

Algorithm 3: Aumüller CRT-RSA

Input : Message m, key (p, q, d_p, d_q, i_q),
32-bit random prime t
Output: Signature $m^d \mod N$,
or error if some fault injection has been detected.

1 $p' = p \cdot t$
2 $d'_p = d_p + \mathsf{random}_1 \cdot (p-1)$ `/* Against DPA, not fault attacks */`
3 $S'_p = m^{d'_p} \mod p'$ `/* Signature modulo` p' `*/`
4 **if** $(p' \mod p \neq 0)$ **or** $(d'_p \not\equiv d_p \mod (p-1))$ **then**
5 | **return** error
6 **end**
7 $q' = q \cdot t$
8 $d'_q = d_q + \mathsf{random}_2 \cdot (q-1)$ `/* Against DPA, not fault attacks */`
9 $S'_q = m^{d'_q} \mod q'$ `/* Signature modulo` q' `*/`
10 **if** $(q' \mod q \neq 0)$ **or** $(d'_q \not\equiv d_q \mod (q-1))$ **then**
11 | **return** error
12 **end**
13 $S_p = S'_p \mod p$
14 $S_q = S'_q \mod q$
15 $S = S_q + q \cdot (i_q \cdot (S_p - S_q) \mod p)$ `/* Same as in line 3 of Alg. 1 */`
16 **if** $(S - S'_p \not\equiv 0 \mod p)$ **or** $(S - S'_q \not\equiv 0 \mod q)$ **then**
17 | **return** error
18 **end**
19 $S_{pt} = S'_p \mod t$
20 $S_{qt} = S'_q \mod t$
21 $d_{pt} = d'_p \mod (t-1)$
22 $d_{qt} = d'_q \mod (t-1)$
23 **if** $S_{pt}^{d_{qt}} \not\equiv S_{qt}^{d_{pt}} \mod t$ **then**
24 | **return** error
25 **else**
26 | **return** S
27 **end**

3. q' is not a multiple of q (*because this would amount to faulting q in the naive algorithm*)
4. $d'_q = d_q + \mathsf{random}_2 \cdot (q-1)$ is not equal to $d_q \mod (q-1)$ (*because this would amount to faulting d_q in the naive algorithm*)
5. $S - S'_p \mod p$ is nonzero (*because this would amount to faulting the recombination modulo p in the naive algorithm*)
6. $S - S'_q \mod q$ is nonzero (*because this would amount to faulting the recombination modulo q in the naive algorithm*)
7. $S_{pt}^{d_q} \mod t$ is not equal to $S_{qt}^{d_p} \mod t$ (*this checks simultaneously for the integrity of S'_p and S'_q*)

Notice that the last verification could not have been done on the naive algorithm, and constitutes the added value for the Aumüller algorithm. These seven cases are *informally* assumed to protect the algorithm against the BellCoRe attacks. The criteria for fault detection is not to detect all faults; for instance, a fault on the final return of S (line 26) is not detected. However, of course, such a fault is not exploitable by a BellCoRe attack.

Remark 1. Some parts of the Aumüller algorithm are actually not intended to protect against fault injection attacks, but against side-channel analysis, such as differential power analysis (DPA). This is the case of lines 2 and 8 in Alg. 3. They can be removed if a minimalist protection against only fault injection attacks is looked for; but as they do not introduce weaknesses, they are simply kept as such.

Vigilant The CRT-RSA algorithm of Vigilant [16] also considers computations in a larger ring than $\mathbb{Z}_p$ (abbreviation for $\mathbb{Z}/p\mathbb{Z}$) and $\mathbb{Z}_q$, to enable verifications. In this case, a small random number r is cast, and computations are carried out in $\mathbb{Z}_{p\times r^2}$ and $\mathbb{Z}_{q\times r^2}$. In addition, the computations are now conducted not on the plain message m, but on an encoded message m', built using the CRT as the solution of those two requirements:

i: $m' \equiv m \mod N$, and
ii: $m' \equiv 1 + r \mod r^2$.

This system of equations has a single solution modulo $N \times r^2$, because N and r^2 are coprime. Such a representation allows to conduct in parallel the functional CRT-RSA (line i) and a verification (line ii). The verification is elegant, as it leverages this remarkable equality: $(1+r)^{d_p} = \sum_{i=0}^{d_p} \binom{d_p}{i} \cdot r^i \equiv 1 + d_p \cdot r \mod r^2$. Thus, as opposed to Aumüller *et al.*'s CRT-RSA, that requires one exponentiation (line 23 of Alg. 3), the verification of Vigilant's algorithm adds only one affine computation (namely $1 + d_p \mod r^2$).

The original description of Vigilant's algorithm involves some trivial computations on p and q, such as $p-1$, $q-1$ and $p \times q$. Those can be faulted, in such a way the BellCoRe attack becomes possible despite all the tests. Thus, a patch by Coron *et al.* has been released in [7] to avoid the reuse of $\widehat{p-1}$, $\widehat{q-1}$ and $\widehat{p \cdot q}$ in the algorithm.

Kim The authors Kim, Kim, Han and Hong propose in [11] a CRT-RSA algorithm that is based on a collaboration between a customized modular exponentiation and verifications at the recombination level based on Boolean operations. The underlying protection concepts being radically different from the algorithms of Shamir, Aumüller and Vigilant, we choose not to detail this interesting countermeasure.

In this paper, we will focus on three implementations, namely the naive one (Sec. 4), the one protected by Shamir countermeasure (Sec. 5), and the one with Aumüller *et al.* countermeasure (Sec. 6).

3 Formal Methods

For all the countermeasures presented in the previous section (Sec. 2), we can see that no formal proof of resistance against attacks is claimed. Informal arguments are given, that convince that for some attack scenarii, the attack attempts are detected hence harmless. Also, an analysis of the probability that an attack succeeds is carried out, however, this analysis strongly relies on assumptions on the faults distribution. Last but not least, the algorithms include protections against both passive side-channel attacks (SPA, DPA) and against active side-channel attacks, which makes it difficult to analyze for instance the minimal code to be added for the countermeasure to be correct.

Our goal is to prove that the proposed countermeasures work, *i.e.*, that they deliver a result that does leak information about neither p nor q (if the implementation is subject to fault injection) exploitable in a BellCoRe attack. In addition, we wish to reach this goal with the two following assumptions:

- our proof applies to a very general attacker model, and
- our proof applies to any implementation that is a (strict) refinement of the abstract algorithm.

First, we must define what computation is done, and what is our threat model.

Definition 1 (CRT-RSA). *The CRT-RSA computation takes as input a message m, assumed known by the attacker, and a secret key (p, q, d_p, d_q, i_q). Then, the implementation is free to instantiate any variable, but must return a result equal to: $S = S_q + q \cdot (i_q \cdot (S_p - S_q) \mod p)$, where:*

- $S_p = m^{d_p} \mod p$, *and*
- $S_q = m^{d_q} \mod q$.

Definition 2 (fault injection). *An attacker is able to request RSA computations, as per Definition 1. During the computation, the attacker can modify any intermediate value by setting it to either a random value or zero. At the end of the computation the attacker can read the result.*

Of course, the attacker cannot read the intermediate values used during the computation, since the secret key and potentially the modulus factors are used. Such "whitebox" attack would be too powerful; nonetheless, it is very hard in practice for an attacker to be able to access intermediate variables, due to protections and noise in the side-channel leakage (*e.g.*, power consumption, electromagnetic emanation). Remark that our model only take into account fault injection on data; the control flow is supposed not to be modifiable.

As a side remark, we notice that the fault injection model of Definition 2 corresponds to that of Vigilant ([16]), with the exception that the conditional tests can also be faulted. To summarize, an attacker can:

- modify a value in the global memory (*permanent fault*), and
- modify a value in a local register or bus (*transient fault*),

but cannot

- inject a permanent fault in the input data (message and secret key), nor
- modify the control flow graph.

The independence of the proofs on the algorithm implementation demands that the algorithm is described at a high level. The two properties that characterize the relevant level are as follows:

1. The description should be low level enough for the attack to work if protections are not implemented.
2. Any additional intermediate variable that would appear during refinement could be the target of an attack, but such a fault would propagate to an intermediate variable of the high level description, thereby having the same effect.

From those requirements, we deduce that:

1. The RSA description must exhibit the computation modulo p and q and the CRT recombination; typically, a completely blackbox description, where the computations would be realized in one go without intermediate variables, is not conceivable.
2. However, it can remain abstract, especially for the computational parts[2].

In our approach, the protections must thus be considered as an augmentation of the unprotected code, *i.e.*, a derived version of the code where additional variables are used. The possibility of an attack on the unprotected code attests that the algorithm is described at the adequate level, while the impossibility of an attack (to be proven) on the protected code shows that added protections are useful in terms of resistance to attacks.

[2] For instance a fault in the implementation of the multiplication (or the exponentiation) is either inoffensive, and we don't need to care about it, or it affects the result of the multiplication (or the exponentiation), and our model take it into account without going into the details of how the multiplication (or exponentiation) is computed.

Remark 2. The algorithm only exhibit evidence of safety. If after a fault injection, the algorithm does not simplify to an error detection, then it might only reveal that some simplification is missing. However, if it does not claim safety, it produces a *simplified* occurrence of a possible weakness to be investigated further.

Several tools are *a priori* suitable for a formalization of CRT-RSA. PARI/GP is a specialized computer algebra system, primarily aimed at solving number theory problems. Although PARI/GP can do a fair amount of symbolic manipulation, it remains limited compared to systems like Axiom, Magma, Maple, Mathematica, Maxima, or Reduce. Those last software also fall short to implement automatically number theoretic results like the Euler theorem. This explains why we developed from scratch a system to reason on modular numbers from a formal point of view. Our system is not general, in that it cannot for instance factorize terms in an expression. However, it is simply able to simplify recursively what is simplifiable from a set of unambiguous rules. This behavior happens to be suitable to the problem of resistance to fault attacks, because the redundancy that is added in the computation is meant to be simplified at the end (if no fault happened).

We describe the computation by a recursively defined term and we model it in OCaml [9][3] with an algebraic data type:

```
type term =
| Zero                          (* the constant zero *)
| One                           (* the constant one *)
| Named of string               (* a named number with no properties *)
| Num of int                    (* a number with no properties *)
| Prime of string               (* a named prime number *)
| Sum of term list              (* a sum of terms *)
| Prod of term list             (* a product of terms *)
| Opp of term                   (* the opposite of a term *)
| Inv of term                   (* the inverse of a term *)
| Mod of term * term            (* the remainder of division of a term by another *)
| Pow of term * term            (* the exponentiation of a term by another *)
| Let of string * term * term   (* the definition of a variable *)
| Let_ of string * term * term (* the safe definition of a variable *)
| Var of string                 (* a reference to a variable *)
| If of term * term * term      (* a condition *)
```

There is no difference between `Named` and `Num` other than cosmetic, for display purpose. The `Num` constructor takes an `int` to ensure that OCaml sees each of them as a different value[4].

The definition of a variable (`Let`) consists of a string for the name of that variable, a term for its value, and a term in which the variable is defined (it writes

[3] We can only guarantee the validity of our tools by the simplicity of its code, and by making it free software (in the near future) so that it can be subject to review.

[4] Using a constructor with no argument, `Random` for instance, is not possible because OCaml would return *true* when comparing a `Random` with a `Random`.

like the `let x = v in e` form, which binds `x` to the value of the expression `v` in the expression `e`, in the OCaml programming language).

The safe definition of a variable (`Let_`) is the same thing except that we assume that there will be no fault in the value term.

An `If` conditional consists of three terms. Its value is the value of the second term if the first one is not zero, or of the third term otherwise[5].

Such a description of the computation, while abstracting the computational parts, allows to simplify the defined terms using rules from arithmetic and properties that we can deduce on the terms, such as being null, being null modulo another term, or being a multiple of another term.

We implement simplification as an OCaml function based on pattern-matching on the term. It applies most of the rules from arithmetic in the $\mathbb{Z}$ ring, and from modular arithmetic in the $\mathbb{Z}/n\mathbb{Z}$ rings. We omit factorization and expansion as they are not confluent operations in general. We also implement a few theorems such as the little Fermat's theorem and its generalization, *i.e.*, Euler's theorem. Of course we cannot do integer factorization to compute φ in our model so we raise an exception to handle cases where the exponent is not a product of prime numbers, but this actually never happens in well formed CRT-RSA computations, including computations with Shamir or Aumüller *et al.* countermeasure.

The simplification function is a recursive traversal of the term tree, and each step is very simple and easily verifiable, thus making it trustworthy. In particular, it is able to prove Proposition 1 and 2.

Injecting a fault in a computation amounts to replacing a subterm by zero or by a number with no properties, according to Definition 2. In the former case, the fault consists in zeroizing an intermediate variable (like in [6]). In the latter case, the fault consists in assuming that all the properties of the subterm are lost. Indeed, numbers with remarkable properties are extremely rare and thus the probability to create a property by a randomizing fault is negligible.

In our model, a fault can occur at any place in the computation. This is modeled by creating a faulted version of the term for each possible fault. To compute the nth faulted version, we traverse the term tree incrementing a counter at each recursive call, and when this counter's value is n, we return the fault (either `Zero` or `Num(`n`)`). When the computation of the faulted version is finished, the faulted term is compared to the original one. If they are the same, it means that the recursive traversing of the term tree was complete and that n is too large, which means we have generated all possible faulted versions of the term.

The faulted versions of the term we want to study are then used to check whether the properties required for the BellCoRe attack to be effective are respected.

Example 1. If we have the following term t which represents the computation $a + b \times c$: `Sum([ Named("a") ; Prod([ Named("b") ; Named("c") ]) ])`, it can be faulted in five different ways (using the randomizing fault):

[5] Remark: we do not apply any restrictions on the possible fault injections in any of the three arguments of the `If` constructor.

1. `Num(1)`, the final result is faulted;
2. `Sum([ Num(2) ; Prod([ Named("b") ; Named("c") ]) ])`, a is faulted;
3. `Sum([ Named("a") ; Num(3) ])`, the result of $b \times c$ is faulted;
4. `Sum([ Named("a") ; Prod([ Num(4) ; Named("c") ]) ])`, b is faulted;
5. `Sum([ Named("a") ; Prod([ Named("b") ; Num(5) ]) ])`, c is faulted.

If the properties that interest us is to know whether t is congruent with a modulo b, we can check if `Mod(Sum([ t, Opp(a) ]), b)` simplifies to `Zero`. Of course it will be true for t, but it will only be true for the fifth version of faulted t. If we had used the zeroing fault, it would also have been true for the third and fourth versions.

4 Study of an Unprotected CRT-RSA Computation

Here is the description of the naive CRT-RSA computation (Alg. 1). For readability reasons, `Named("x")` and `Prime("x")` have been replaced by `x`. `p` and `q` are prime numbers; `m` and `e` are numbers with no properties):

```
Let_("dp", Mod(Pow(e, Opp(One)), Sum([ p ; Opp(One) ])),
Let_("dq", Mod(Pow(e, Opp(One)), Sum([ q ; Opp(One) ])),
Let_("iq", Mod(Pow(q, Opp(One)), p),
Let("sp", Mod(Pow(m, Var("dp")), p),
Let("sq", Mod(Pow(m, Var("dq")), q),
Sum([ Var("sq")
    ; Prod([ q
           ; Mod(Prod([ Var("iq")
                      ; Sum([ Var("sp") ; Opp(Var("sq")) ]) ]),
                 p) ]) ]) )))))
```

The first three lines define d_p, d_q, and i_q. As we can see we use `Let_` rather than `Let` for these definitions, so the computation of the values of these variables cannot be faulted (since they are seen as inputs of the algorithm). After that, S_p and S_q are computed and then recombined in the last expression, as in Definition 1.

To test if the BellCoRe attack works on a faulted version $\widehat{S}$, we perform the following tests (we note $|S|$ for the simplified version of S):

1. Is $|S|$ equal to $|\widehat{S}|$?
2. Is $|S \mod p|$ equal to $|\widehat{S} \mod p|$?
3. Is $|S \mod q|$ equal to $|\widehat{S} \mod q|$?

If the first test is false and at least one of the second and third is true, we have a BellCoRe attack, as seen in Sec. 2.

There are 27 different possible faults in our model of the unprotected CRT-RSA, 17 of which allows a BellCoRe attack using the randomizing fault, and 19 with the zeroing fault. These results are obtained almost instantaneously by our tool.

As an example, replacing the intermediate variable holding the value of $i_q \cdot (S_p - S_q) \mod p$ in the final expression with zero or a random value makes the first and second tests false, and the last one true, and thus allows a BellCoRe attack.

5 Study of the Shamir Countermeasure

The description of the computation of CRT-RSA with Shamir countermeasure (Alg. 2) can be found in App. A.

Using the same method as for the unprotected implementation of CRT-RSA, we can prove that on the 75 different possible faults, 21 allows a BellCoRe attack, whether using a randomizing fault or a zeroing fault (these results are obtained almost instantaneously by our tool). This is not really surprising, as the test which is done on line 10 of Alg. 2 does not verify if a fault is injected during the computations of S_p and S_q, nor during their recombination in S. For instance zeroing or randomizing the intermediate variable holding the result of $S_p - S_q$ during the computation of S (line 9 of Alg. 2) result in a BellCoRe attack.

During our study of this countermeasure, we remarked that if the attacker can modify the value of an intermediate variable only for one use of this variable (*transient fault*), we can do more attacks. In practice, it would translate into faulting the variable when it is read (*e.g.*, in a register or on a bus), rather than in (persistent) memory. This behavior could also be the effect of a fault injection in cache, which is later replaced with the good value when it is read from memory again. To the authors knowledge, these are not impossible situations. Nonetheless, growing the power of the attacker to take that into account break some very important assumptions that are classical (sometimes even implicit) in the literature. It does not matter that the parts of the secret key are stored in a secure "key container" if their values can be a faulted at read time. Indeed, allowing this kind of fault enable even more BellCoRe attacks on a CRT-RSA computation protected by the Shamir countermeasure. For instance, if the value of p is randomized for the computation of the value of S_p (line 7 of Alg. 2), then we have $S \neq \widehat{S}$, but also $S \equiv \widehat{S} \mod q$, which enables a BellCoRe attack, as seen in Sec. 2.

It is often asserted that the countermeasure of Shamir is unpractical due to its need for d (as mentioned in [1] and [16]), and because there is a possible fault attack on the recombination, *i.e.*, line 9 of Alg. 2 (as mentioned in [16]). However, the attack on the recombination can easily be checked, by testing that $S - S_p \not\equiv 0 \mod p$ and $S - S_q \not\equiv 0 \mod q$ before returning the result. Notwithstanding, to our best knowledge, it is difficult to detect the attack our tool found (described in the previous paragraph), and so the existence of this attack (new, in the sense it has not been described previously) is a compelling reason for not implementing Shamir's CRT-RSA.

6 Study of the Aumüller *et al.* Countermeasure

The description of the computation of CRT-RSA with Aumüller *et al.* countermeasure (Alg. 3) is quite large and can thus be found in the App. B.

Using the same method as before, we can prove that on the 145 different possible faults, *none* allows a BellCoRe attack, whether the fault is zero or random (these results are obtained in very few seconds by our tools). This is a proof that the Aumüller *et al.* countermeasure works when there is one fault[6].

We also tried to remove some of the tests that the countermeasure consists of. It appears that *two of them are unnecessary*: the first ones of lines 4 and 10 in the Aumüller CRT-RSA as presented in Alg. 3. These two tests are actually redundant with the two tests of line 16 and the test of line 23 of Alg. 3 which also verify the integrity of p' and q' by using them indirectly. Removing these tests is not very useful in terms of performances, but their uselessness shows the need for formal studies in the field of implementation security, even if they might appear unnatural at first.

Since it allowed more attacks on the Shamir countermeasure, we also tested the Aumüller *et al.* countermeasure against *transient fault* such as described in Sec. 5. It happens that Aumüller *et al.* is resistant against such fault injections too.

Our methods also confirmed that the computation of d_p, d_q, and i_q (in terms of p, q, and d) must not be part of the algorithm. The countermeasure effectively needs these three variables to be inputs of the algorithm to work properly. For instance there is a BellCoRe attack if d_q happens to be zeroed. However, even with d_p, d_q, and i_q as inputs, we can still attempt to attack a CRT-RSA implementation protected by the Aumüller *et al.* countermeasure by doing more than one fault.

Our results are as follows. With more than one fault it is obvious that the countermeasure can be dodged if one of the fault is a zeroing of an intermediate variable which is used as condition in one of the useful tests. However we were able to prove that Aumüller *et al.* countermeasure is still efficient if there is two or even three randomizing faults. The computations for two and three faults took respectively a few minutes and a few dozen of minutes.

7 Conclusions and Perspectives

We have formally proven the resistance of the Aumüller *et al.* countermeasure against the BellCoRe attack by fault injection on CRT-RSA. To our knowledge, it is the first time that a formal proof of security is done for a BellCoRe countermeasure.

[6] This result is worthwhile some emphasis: the genuine algorithm of Aumüller is thus *proved* resistant against single fault attacks. At the opposite, the CRT-RSA algorithm of Vigilant is not immune to single fault attacks (refer to [7]), and the corrections suggested in the same paper by Coron *et al.* have not yet been proved.

The proof enables us to show that two out of seven of the tests done by Aumüller *et al.* countermeasure are unnecessary, thereby simplifying the protected computation of CRT-RSA.

During our research, we have raised several questions about the assumptions traditionally made by countermeasures. The possibility of fault at read time is, in particular, responsible for many vulnerabilities. The possibility of such fault means that part of the secret key can be faulted (even if only for one computation). It allows interesting BellCoRe attacks on a computation of CRT-RSA protected by Shamir countermeasure.

The first of these two points demonstrates the lack of formal studies of fault injection attack and their countermeasures, while the second one shows the importance of formal proofs in the field of implementation security.

As a first perspective, we would like to address the hardening of software codes of CRT-RSA under the threat of a bug attack. This attack has been introduced by Biham, Carmeli and Shamir [2] at CRYPTO 2008. It assumes that a hardware has been trapped in such a way that there exists two integers a and b, for which the multiplication is incorrect. In this situation, Biham, Carmeli and Shamir mount an explicit attack scenario where the knowledge of a and b is leveraged to produce a faulted result, that can lead to a remote BellCoRe attack. For sure, testing for the correct functionality of the multiplication operation is impractical (it would amount to an exhaustive verification of 2^{128} multiplications on 64 bit computer architectures). Thus, it can be imagined to use a countermeasure, like that of Aumüller, to detect a fault (caused logically). Our aim would be to assess in which respect our fault analysis formal framework allows to validate the security of the protection. Indeed, a fundamental difference is that the fault is not necessarily injected at *one* random place, but can potentially show up at *several* places.

As another perspective, we would like to handle the repaired countermeasure of Vigilant [7] and the countermeasure of Kim [11]. Regarding Vigilant, the difficulty our verification framework in OCaml shall overcome is to decide how to inject the remarkable identity $(1+r)^{d_p} \equiv 1 + d_p \cdot r \mod r^2$: either it is kept as such such, like an *ad hoc* theorem (but we need to make sure it is called only at relevant places, since it is not confluent), or it is made more general (but we must ascertain that the verification remains tractable). However, this effort is worthwhile, because the authors themselves say in the conclusion of their article [7] that:

> *"Formal proof of the FA-resistance of Vigilant's scheme including our countermeasures is still an open (and challenging) issue."*

Regarding the CRT-RSA algorithm from Kim, the computation is very detailed (it goes down to the multiplication level), and involves Boolean operations (and, xor, *etc.*), so more expertise about both arithmetic and logic must be added to our software.

Eventually, we wish to answer a question raised by Vigilant [16] about the prime t involved in Aumüller *et al.* countermeasure:

"Is it fixed or picked at random in a fixed table?"

The underlying issue is that of *replay* attacks on CRT-RSA, that are more complicated to handle; indeed, they would require a formal system such as ProVerif [5], that is able to prove interactive protocols.

Concerning the tools we developed during our research, they currently only allow to study fault injection in the data, and not in the control flow, it would be interesting to enable formal study of fault injections affecting the control flow. We would also like to make these tools usable by anyone, which will require the creation of a better DSL for describing computations and attacks, as well as a nice user interface to our code, which is still in "research code" stage for now.

References

1. Christian Aumüller, Peter Bier, Wieland Fischer, Peter Hofreiter, and Jean-Pierre Seifert. Fault Attacks on RSA with CRT: Concrete Results and Practical Countermeasures. In Burton S. Kaliski, Jr., Çetin Kaya Koç, and Christof Paar, editors, *CHES*, volume 2523 of *Lecture Notes in Computer Science*, pages 260–275. Springer, 2002.
2. Eli Biham, Yaniv Carmeli, and Adi Shamir. Bug attacks. In *CRYPTO*, volume 5157 of *LNCS*, pages 221–240. Springer, 2008. Santa Barbara, CA, USA.
3. Eli Biham and Adi Shamir. Differential Fault Analysis of Secret Key Cryptosystems. In *CRYPTO*, volume 1294 of *LNCS*, pages 513–525. Springer, August 1997. Santa Barbara, California, USA. DOI: 10.1007/BFb0052259.
4. Dan Boneh, Richard A. DeMillo, and Richard J. Lipton. On the Importance of Checking Cryptographic Protocols for Faults. In *Proceedings of Eurocrypt'97*, volume 1233 of *LNCS*, pages 37–51. Springer, May 11-15 1997. Konstanz, Germany. DOI: 10.1007/3-540-69053-0_4.
5. Bruno Blanchet. ProVerif: Cryptographic protocol verifier in the formal model. http://prosecco.gforge.inria.fr/personal/bblanche/proverif/.
6. Maria Christofi, Boutheina Chetali, Louis Goubin, and David Vigilant. Formal verification of an implementation of CRT-RSA Vigilant's algorithm. *Journal of Cryptographic Engineering*, 3(3), 2013. DOI: 10.1007/s13389-013-0049-3.
7. Jean-Sébastien Coron, Christophe Giraud, Nicolas Morin, Gilles Piret, and David Vigilant. Fault Attacks and Countermeasures on Vigilant's RSA-CRT Algorithm. In Luca Breveglieri, Marc Joye, Israel Koren, David Naccache, and Ingrid Verbauwhede, editors, *FDTC*, pages 89–96. IEEE Computer Society, 2010.
8. Xiaofei Guo, Debdeep Mukhopadhyay, and Ramesh Karri. Provably secure concurrent error detection against differential fault analysis. Cryptology ePrint Archive, Report 2012/552, 2012. http://eprint.iacr.org/2012/552/.
9. INRIA. OCaml, a variant of the Caml language. http://caml.inria.fr/ocaml/index.en.html.
10. Marc Joye, Arjen K. Lenstra, and Jean-Jacques Quisquater. Chinese Remaindering Based Cryptosystems in the Presence of Faults. *J. Cryptology*, 12(4):241–245, 1999.
11. Sung-Kyoung Kim, Tae Hyun Kim, Dong-Guk Han, and Seokhie Hong. An efficient CRT-RSA algorithm secure against power and fault attacks. *J. Syst. Softw.*, 84:1660–1669, October 2011.
12. Çetin Kaya Koç. High-Speed RSA Implementation, November 1994. Version 2, ftp://ftp.rsasecurity.com/pub/pdfs/tr201.pdf.
13. Ronald L. Rivest, Adi Shamir, and Leonard M. Adleman. A Method for Obtaining Digital Signatures and Public-Key Cryptosystems. *Commun. ACM*, 21(2):120–126, 1978.
14. Adi Shamir. Method and apparatus for protecting public key schemes from timing and fault attacks, November 1999. Patent Number 5,991,415; also presented at the rump session of EUROCRYPT '97.
15. Mohammad Tehranipoor and Cliff Wang, editors. *Introduction to Hardware Security and Trust*. Springer, 2012. ISBN 978-1-4419-8079-3.
16. David Vigilant. RSA with CRT: A New Cost-Effective Solution to Thwart Fault Attacks. In Elisabeth Oswald and Pankaj Rohatgi, editors, *CHES*, volume 5154 of *Lecture Notes in Computer Science*, pages 130–145. Springer, 2008.

A Description of Shamir Implementation of CRT-RSA

For readability purpose, `Named("x")` and `Prime("x")` have been replaced by `x`; `p`, `q`, and `r` are prime numbers; `m`, `d`, and `ERROR` are numbers with no properties):

```
Let_("iq", Mod(Pow(q, Opp(One)), p),
Let("p'", Prod([ p ; r ]),
Let("dp", Mod(d, Prod([ Sum([ p ; Opp(One) ]) ; Sum([ r ; Opp(One) ]) ])),
Let("S'p", Mod(Pow(m, Var("dp")), Var("p'")),
Let("q'", Prod([ q ; r ]),
Let("dq", Mod(d, Prod([ Sum([ q ; Opp(One) ]) ; Sum([ r ; Opp(One) ]) ])),
Let("S'q", Mod(Pow(m, Var("dq")), Var("q'")),
Let("Sp", Mod(Var("S'p"), p),
Let("Sq", Mod(Var("S'q"), q),
Let("S", Sum([ Var("Sq")
             ; Prod([ q
                    ; Mod(Prod([ Var("iq")
                               ; Sum([ Var("Sp") ; Opp(Var("Sq")) ]) ]),
                          p) ]) ]),
If(Mod(Sum([ Var("S'p") ; Opp(Var("S'q")) ]), r),
   ERROR,
Var("S")))))))))))))
```

B Description of Aumüller *et al.* Implementation of CRT-RSA

For readability purpose `Named("x")` and `Prime("x")` have been replaced by `x` (`m`, `e`, `Random1`, `Random2`, and `ERROR` are numbers with no properties; and `p`, `q`, and `t` are prime numbers).

```
Let_("dp", Mod(Pow(e, Opp(One)), Sum([ p ; Opp(One) ])),
Let_("dq", Mod(Pow(e, Opp(One)), Sum([ q ; Opp(One) ])),
Let_("iq", Mod(Pow(q, Opp(One)), p),
Let("p'", Prod([ p ; t ]),
Let("d'p", Sum([ Var("dp") ; Prod([ Random1 ; Sum([ p ; Opp(One) ]) ]) ]),
Let("s'p", Mod(Pow(m, Var("d'p")), Var("p'")),
If(Mod(Var("p'"), p),
   ERROR,
If(Mod(Sum([ Var("d'p") ; Opp(Var("dp")) ]), Sum([ p ; Opp(One) ])),
   ERROR,
Let("q'", Prod([ q ; t ]),
Let("d'q", Sum([ Var("dq") ; Prod([ Random2 ; Sum([ q ; Opp(One) ]) ]) ]),
Let("s'q", Mod(Pow(m, Var("d'q")), Var("q'")),
If(Mod(Var("q'"), q),
   ERROR,
If(Mod(Sum([ Var("d'q") ; Opp(Var("dq")) ]), Sum([ q ; Opp(One) ])),
   ERROR,
Let("sp", Mod(Var("s'p"), p),
Let("sq", Mod(Var("s'q"), q),
```

```
Let("S", Sum([ Var("sq")
             ; Prod([ q
                    ; Mod(Prod([ Var("iq")
                               ; Sum([ Var("sp")
                                     ; Opp(Var("sq")) ]) ]), p) ]) ]),
If(Mod(Sum([ Var("S") ; Opp(Var("s'p")) ]), p),
   ERROR,
If(Mod(Sum([ Var("S") ; Opp(Var("s'q")) ]), q),
   ERROR,
Let("spt", Mod(Var("s'p"), t),
Let("sqt", Mod(Var("s'q"), t),
Let("dpt", Mod(Var("d'p"), Sum([ t ; Opp(One) ])),
Let("dqt", Mod(Var("d'q"), Sum([ t ; Opp(One) ])),
If(Mod(Sum([ Pow(Var("spt"), Var("dqt"))
           ; Opp(Pow(Var("sqt"), Var("dpt"))) ]), t),
   ERROR,
Var("S")))))))))))))))))))))))))
```

Towards Fresh Re-Keying with Leakage-Resilient PRFs: Cipher Design Principles and Analysis

Sonia Belaïd[1], Fabrizio De Santis[2,3], Johann Heyszl[4], Stefan Mangard[3], Marcel Medwed[5], Jörn-Marc Schmidt[6], François-Xavier Standaert[7], Stefan Tillich[8]

[1] Ecole Normale Supérieure and Thales Communications, France.
[2] Technische Universität München, Munich, Germany.
[3] Infineon Technologies AG, Neubiberg, Germany.
[4] Fraunhofer Research Institution AISEC, Munich, Germany.
[5] NXP Semiconductors, Gratkorn, Austria.
[6] IAIK, Graz University of Technology, Austria.
[7] ICTEAM/ELEN/Crypto Group, Université catholique de Louvain, Belgium.
[8] Department of Computer Science, University of Bristol, UK.

Abstract. Leakage-resilient cryptography aims at developing new algorithms for which physical security against side-channel attacks can be formally analyzed. Following the work of Dziembowski and Pietrzak at FOCS 2008, several symmetric cryptographic primitives have been investigated in this setting. Most of them can be instantiated with a block cipher as underlying component. Such an approach naturally raises the question whether certain block ciphers are better suited for this purpose. In order to answer this question, we consider a leakage-resilient re-keying function, and evaluate its security at different abstraction levels. That is, we study possible attacks exploiting specific features of the algorithmic description, hardware architecture and physical implementation of this construction. These evaluations lead to two main outcomes. First, we complement previous works on leakage-resilient cryptography and further specify the conditions under which they actually provide physical security. Second, we take advantage of our analysis to extract new design principles for block ciphers to be used in leakage-resilient primitives. While our investigations focus on side-channel attacks in the first place, we hope these new design principles will trigger the interest of symmetric cryptographers to design new block ciphers combining good properties for secure implementations and security against black box cryptanalysis.

1 Introduction

Securing embedded devices against side-channel attacks is an important challenge in modern cryptography. Because of their technology-dependent nature, protections against these attacks usually require combining ideas at different abstraction levels, e.g. exploiting noise in physical processes and randomness in hardware/software designs [21]. In the context of symmetric cryptography, a recent and concurrent trend has investigated the opportunities to analyze new primitives, better suited for physically-secure implementations. Dziembowski and Pietrzak's leakage-resilient cryptography is one of the most investigated models for this purpose [7], and several proposals of pseudorandom generators

(PRGs)/stream ciphers, pseudorandom functions (PRFs) and pseudorandom permutations (PRPs) have been considered in this setting [6,9,27,35,39,40]. These new constructions naturally raise interesting open questions regarding the practical relevance of formal models for physical security analysis. Yet, they are all based on some kind of re-keying strategies (i.e. reminiscent from Kocher's early patents [17]). Hence, and somewhat independent of these modeling issues, it may very well be that (small variations of) ideas proposed in such theoretical works actually provide significantly enhanced security against large categories of "practical attacks". Since another possible drawback of leakage-resilient cryptography is its significant performance overheads, it naturally suggests an intermediate line of research, where the security of leakage-resilient primitives is analyzed in front of actual side-channel adversaries, in order to mitigate these overheads. This approach has been recently followed by Medwed et al. for the case of leakage-resilient PRFs [26]. In this paper, we embrace a similar strategy and further study the possibilities to design secure and efficient leakage-resilient PRFs. In particular, we focus on their instantiation using block ciphers, which is motivated by the large literature on side-channel attacks and countermeasures for this type of building blocks. In this context, our main goal is to investigate new design principles that would be best suited for the secure implementation of such primitives. For this purpose and as a starting point, we analyze the physical security of a generic block cipher construction, aimed to be used in the re-keying scheme represented in Fig. 1. This re-keying essentially uses a function g to re-key a block cipher f with a master key k and a public random nonce r. For each block of message, a fresh key is computed as $k^\star = g_k(r)$, and then used to generate the ciphertext $c = f_{k^\star}(m)$. One important advantage of this scheme (put forward and analyzed in [25]) is that (informally): (i) from the mathematical point of view, f has to be cryptographically strong while g only requires some minimum diffusion properties, (ii) by contrast, from the implementation point of view f only needs to be secure against Simple Power Analysis (SPA) , while g has to resist both SPA and Differential Power Analysis (DPA) . The solution previously proposed in [25] was to use a modular multiplication for g, which benefits from the feature of being easy to mask [5,11]. Yet, and despite being promising from a security and performance point of view, this proposal is quite specific to one countermeasure (namely, masking) that has proved to be quite effective in software [32], but may turn out to be difficult to implement in hardware [23]. As a result, we propose to investigate alternative candidates for the g function, and focus on hardware implementation issues, in order to increase the versatility of the design space for fresh re-keying.

Our contributions. The CHES 2012 work of Medwed et al. is based on a new assumption that identical components (e.g. S-boxes) in parallel hardware implementations leak similarly. It also suggests that the AES may not be the best block cipher for integration in a leakage-resilient PRF and left a number of questions open regarding the security of this proposal. In this paper, we contribute to these issues in two main directions. On one hand, we extend the leakage-resilient security analysis of [26], paying attention to three different abstraction levels.

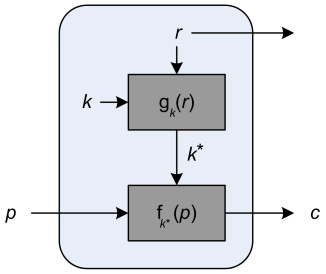


Fig. 1. Fresh re-keying: basic principle.

At the algorithm level, we investigate generic side-channel attacks targeting the first and second rounds of a re-keying function (and check how much they can help to break the "identical leakage assumption"). We also use our analysis to provide a discussion of the tradeoff between the time and data complexity of such attacks. At the architecture level, we exhibit a possible weakness in the (realistic) case where an implementation would leak according to a distance-based leakage model (e.g. the Hamming distance one). We then put forward different solutions to mitigate the issue. Eventually, at the implementation level, we study the impact of localized Electro-Magnetic Analysis (EMA) [10,30] for distinguishing the leakage of different components of our constructions. We use an FPGA case study to highlight that the resulting (key-dependent) algorithmic noise remains difficult to exploit by actual adversaries. On the other hand, we take advantage of our security evaluations in order to specify the components of a block cipher that would be better suited to leakage-resilience than the AES. Starting from a PRESENT-like structure [2] (a natural candidate for hardware implementations), the results of our algorithmic-level security analysis allows determining the size of S-boxes in this cipher, while the result of our architecture-level security analysis leads to new criteria for the permutation layer. The latter example is interesting from a methodological point of view, as it suggests that low-level issues in physical security can sometimes be more efficiently solved at higher abstraction levels. We claim that the resulting cipher integrated in the leakage-resilient PRF construction from [26] can lead to secure and efficient implementations of the fresh re-keying scheme in Fig. 1.

Organization. We start the paper with the description of a generic block cipher for use in leakage-resilient schemes, leaving some parameters open (e.g. the previously mentioned S-box size and permutation layer). Following, Sec. 3, 4 and 5 contain our security analyzes at different abstraction levels and fix the open parameters progressively. Eventually, we specify an instance of block cipher based re-keying function in Sec. 6, and detail an open source Hardware Description Language (HDL) code for an instance of hardware architecture. We hope that this open source code will stimulate further research and practical security evaluations of our proposal. Conclusions are in Sec. 7.

Cautionary note. Our focus is on the side-channel resistance of the proposed construction. In this context, the first/last encryption rounds of a block cipher implementation are usually the most critical. We consequently investigate these rounds as an important step in validating the interest of leakage-resilient PRFs based on block ciphers. By contrast, we *do not* make any specific claim regarding the fact that our proposal is a secure PRF yet. Our hope and belief naturally is that combining enough of the iterations proposed in this paper can lead to mathematical security at lower cost than previous proposals, and our performance evaluations in Sec. 6 provide good indications that this could indeed be the case. Meanwhile, we specify our constructions up to the point where its physical security can be analyzed, and suggest to use it as a possible instance for the function $\mathbf{g}$ in Fig. 1, since it has relaxed mathematical requirements.

2 Towards efficient leakage-resilient PRF designs

The block diagram of our instance of re-keying function $\mathbf{g}$ is given in Fig. 2, where $r[0]$ denotes the 0th word of the public random nonce r in Fig. 1, and the word size is determined by the S-box size of the underlying cipher used in the re-keying steps. In the CHES 2012 proposal, each step corresponds to the execution of the AES Rijndael and the words are 8-bit wide. In the rest of this paper, we will consider an alternative (generic) cipher design represented in the right part of the picture, in which the iterations combine a bitwise key addition, an S-box layer and a permutation layer (aka wire crossing). Intuitively, the improved physical security of this re-keying function comes from the careful selection of this plaintexts that can be enforced in tree-based PRFs. Namely, the block cipher (i.e. the steps) in Fig. 2 can only be queried with inputs of a very specific format, where each word of r bears the same value (i.e. $r[0]||r[0]||\ldots||r[0]$ for the first round). This implies that any divide-and-conquer DPA trying to exploit the leakage will be affected by a key-dependent algorithmic noise. Besides, if the leakage functions corresponding to all the S-boxes are identical, they will only provide information about the master key (e.g. k_0) up to a permutation of its words (see [26] for a detailed analysis of this claim).

As previously mentioned, using this construction for re-keying rather than directly as a PRF (which would require a secure block cipher) allows relaxing its mathematical security requirements, leading to the following advantages. First, the number of rounds in the block cipher can possibly be reduced since this block cipher essentially needs to fulfill the diffusion criteria detailed in [25]. Second, the output of the re-keying function will be used as a fresh session key k^* that is not public. Hence, the output whitening step of the CHES 2012 construction is not necessary. For performance reasons, we will also consider a very minimum key scheduling (inspired by [3,12]), which allows that the recovery of any ith step key k_i does not directly translate into a master key recovery. Given these choices, the main design questions we will consider in the next sections are:

1. How to select the S-boxes number N_s and bit size b?
2. How to choose the permutation layer?

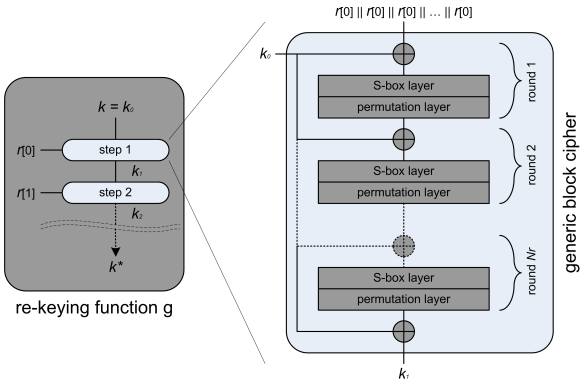


Fig. 2. Our instance of re-keying function g.

3. How many block cipher rounds per step are necessary?
4. How many steps are necessary?

The analysis of Sec. 3 will answer the first question. The analysis of Sec. 4 will allow answering the second question. As for the number of rounds and steps, we will discuss minimum requirements for fresh re-keying applications in Sec. 4.3.

3 Physical security analysis at the algorithm level

In this section, we investigate the physical security of our generic block cipher construction in a simple model where the leakage produced by each S-box is assumed identical. We first refine the security levels provided in [26], by relaxing the simplifying hypothesis that all key words are pairwise different. Our results show that efficient design choices still allow preventing low-complexity attacks targeting the first S-box layer. Next, we focus on the second block cipher round and highlight possible attacks with practical time complexities. Finally, we exhibit in Sec. 3.3 that despite its limited time complexity, DPA taking advantage of the second-round leakages may remain difficult because of the bounded data complexity that is guaranteed by our leakage-resilient construction.

3.1 Analysis of the first S-box layer

Our substitution layer is composed of N_s b-bit S-boxes operating in parallel, as illustrated in Fig. 3 for $N_s = 4$ and $b = 4$. Intuitively, this parallelism combined with a careful selection of the plaintexts improves security against DPA, since an attacker may succeed in recovering the set $\mathcal{S}$ of the N_s key words, but has no information to order them as long as the leakage functions L_i's are identical. As a result, the security analysis of [26] suggests that successful attacks should have at least the (super-exponential) time complexity of an enumeration over N_s S-boxes. Yet, in practice one should additionally consider that several key words in $\mathcal{S}$ may share the same value in $[0...2^b - 1]$. In this case, the optimistic complexity $N_s!$ has to be divided by the (factorial of the) multiplicities of each value in $\mathcal{S}$.

Details about the computations of these multiplicities are given in Appendix A. The improved attack complexities are given in the left part of Table 1.

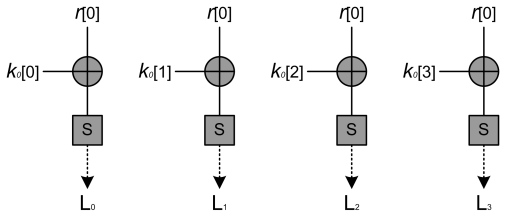


Fig. 3. Attacks against the first S-box layer.

3.2 Analysis of the second S-box layer

One important argument in the previous subsection is that it can be computationally difficult to distinguish the different key words in the set $\mathcal{S}$ when the leakage functions L_i's are identical. In this context, a natural question is to know whether the second round leakage could not be used to discriminate these key words with lower complexities. To answer it, we use the exemplary design of Fig. 4 (given for $N_s = 4$ and $b = 4$). For now, we use the permutation of

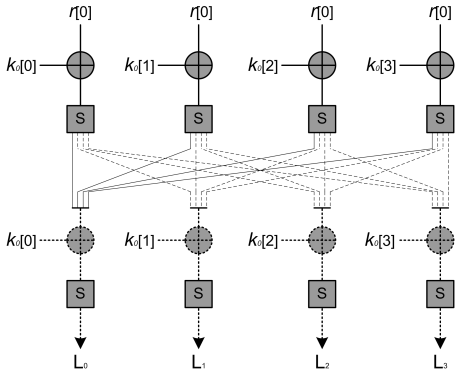


Fig. 4. Attacks against the second S-box layer.

`Small-Present` in our analysis [18]. In this case, the second-round S-box inputs depend on b key words from the multiset $\mathcal{S}$. So an adversary essentially has to pick these b key words and determine their order. Assuming no key addition in the second round, the first step is equivalent to the enumeration of the b combinations of a multiset of cardinal N_s, which complexity is given by [19]:

$$\sum_{p=0}^{N_s} (-1)^p \sum_{1 \leq i_1 \leq i_2 \leq \cdots \leq i_p \leq N_s} \binom{N_s + b - m_{i_1} - m_{i_2} - \cdots - m_{i_p} - p - 1}{N_s - 1},$$

with the m_i's standing for the multiplicities of the values in $\mathcal{S}$. The complexity of second step is determined as in the previous subsection. The resulting attack complexities are given in the right part of Table 1. Additionally considering a key addition in the second round would multiply them by 2^b. We conclude that the large time complexities obtained when only taking advantage of first round leakages vanish if the second round is targeted.

Table 1. Expected time complexities of attacks targeting the first (left) and the second (right) S-box layer estimated with Monte Carlo sampling (in $\log_2$ scale).

N_s	16	24	32
$b = 4$	39	66	95
$b = 8$	44	78	116
$b = 12$	44	79	118
$b = 16$	44	79	118

N_s	16	24	32
$b = 4$	13.4	14.8	15.5
$b = 8$	28.8	34.4	38.1
$b = 12$	39.7	50.2	56.5
$b = 16$	44.3	63.7	73.4

3.3 Time complexity vs. data complexity tradeoff

Since the best attacks exploiting second round leakages do not have a sufficiently high time complexity for ensuring practical security, we finally investigate the exploitation of this leakage in the context of practical adversaries with data complexity bounded to 2^b, as guaranteed by design in our leakage-resilient construction. In this case, the main goal is to solve the estimation issue that is typical from side-channel distinguishers. We will focus on Brier et al.'s Correlation Power Analysis (CPA) to illustrate our claims [4]. Yet, we note that in a first-order DPA scenario, this distinguisher is actually equivalent to worst-case template attacks as long as both distinguishers use the same leakage models [16]. Since our following analyzes consider perfect leakage models anyway, our conclusions are reflective of most actual strategies that could be used in practice [22]. In general, a successful CPA requires that an adversary can distinguish a correlation coefficient estimated for the correct key candidate (denoted as ρ_g) from correlation coefficients estimated for wrong key candidates (denoted as ρ_w). In order to simplify analyses, a usual assumption is to consider $\rho_w = 0$ (i.e. that wrong key candidates give rise to uncorrelated signals) [21]. Further assuming that the adversary obtains noiseless leakages and that she perfectly knows the leakage model, we can additionally approximate the maximum correlation obtained for the correct key candidate as $\rho_g \approx \frac{1}{\sqrt{N_s}}$. In this simple setting, the number of traces required to distinguish both distributions is given by [20]:

$$N_t = 3 + 8 * \frac{z_{1-\alpha}^2}{\ln^2 \frac{1+\rho_g}{1-\rho_g}}, \tag{1}$$

with $z_{1-\alpha}$ the quantile value. When testing N_k key candidates, we typically set the confidence α to $\frac{1}{N_k}$. This number of key candidates to test for the attack

strategies described in Sec. 3.2 is given in the left part of Table 2. It directly leads to the minimum data complexities required for a CPA to be successful for various parameters N_s and b, given in the right part of the table. For $b = 4$, $b = 8$ and the combination $b = 12$ $N_s = 32$, the data complexity needed is larger than the available 16, 256 and 4096 tolerated by our construction. For $b = 12$ combined with $N_s \leqslant 24$ and $b = 16$, a sufficient number of traces is available to mount a successful attack. This naturally suggests that $b = 4$ is the preferred solution for security reasons (which comes at the cost of lower performances, since less bits of r will be operated per step in Fig. 2). The next sections will stick with this design choice and consider $N_s = 32$ to prevent first-round attacks[1].

Table 2. Left: number of key hypotheses to test for a known key words set (in $\log_2$ scale). Right: Minimum number of traces to a successful CPA.

N_s	16	24	32
$b = 4$	13.4	14.8	15.5
$b = 8$	28.8	34.4	38.1
$b = 12$	39.7	50.2	56.5
$b = 16$	44.3	63.7	73.4

N_s	16	24	32
$b = 4$	432	741	1051
$b = 8$	1060	1966	2954
$b = 12$	1513	2969	4526
$b = 16$	1705	3831	5977

4 Physical security analysis at the architecture level

The previous analyzes are essentially independent of the architecture used to implement our re-keying scheme. In this section, we move towards a lower abstraction level and investigate possible attacks taking advantage of a typical hardware implementation that would implement the operations of our block cipher round in parallel. In this context, an important observation is that CMOS devices usually leak proportional to the Hamming distance of values which appear subsequently in a part of the hardware module, e.g. a data bus or register. As a result, an attack can take advantage of extra information provided by the combined leakage of the two values (which would not be available in two separate attacks on the individual values). We first show that such attacks exist in a reasonable implementation context, and then discuss how to mitigate them with an appropriate choice of permutation layer. Finally, we conclude the section with a short discussion of the minimum number of rounds required for re-keying.

[1] Since for $b = 4$, N_t might be not large enough for the formula of Equation 1 to be accurate, we also performed the following experiment. We uniformly sampled a 16-tuple of 4-bit values as hypothesis for the correct key (A) and simulated the observed signal by adding 15 more random 16-tuples to the first one (B). Then, we sampled 2^{16} tuples of 4-bit values for the incorrect key hypotheses (C_i). Finally, we applied a Hamming weight leakage function and calculated the 2^{16} correlation coefficients between (A) and (B), and (B) and (C_i) respectively. The resulting coefficients for the wrong hypotheses lied between -0.85 and 0.85. Furthermore, over 100 experiments we observed that on average 18 000 wrong hypotheses yielded a higher ρ than the correct key. The observed minimum of favored wrong keys was 209 and the maximum 64 800. This experiment identically suggests that a b of no more than four should be chosen.

4.1 The Hamming distance issue

As our leakage-resilient design requires the parallel execution of all the S-boxes, a natural architecture for implementing a re-keying step would consist of a single-round unit performing key addition, substitution and permutation in a single clock cycle, whose result is fed back until the required number of rounds is reached. In a device leaking the Hamming distance, this would mean that there is combined leakage of two values occurring at the same point in subsequent rounds, e.g. two round inputs or two S-box outputs. Assuming that the permutation layer used in the rounds is exactly the one of `Small-Present` as proposed in Sec. 3.1, such a Hamming distance leakage would directly lead to improved attacks. The main issue is that such a permutation layer has the property that the relative position of a bit within a word after the permutation is dependent on the index of the S-box the bit originated from. For example for $N_s = 4$, the first bit of each word after the permutation originates exclusively from the first S-box. Considering (as in Sec. 3.1) that the values of the key words are known and only their order remains unknown, an attacker could further identify (e.g.) the first key word in the following way. Derive the S-box output using the value of each key word and calculate the Hamming distance with a word consisting of the first bit of the input replicated four times. When attacking the power traces with these hypotheses, only the one for the actual first key word will succeed. This process can be repeated for all other key word positions using different bits from the nonce word to calculate the Hamming distance hypothesis. For $N_s > b$, the position of the key word cannot be determined uniquely by this attack. However, their number of possible orderings will be reduced significantly, even in the optimistic case where these words are all pairwise independent. Before the attack, each key word could potentially appear in each position of the key, giving $N_s!$ possible candidates (in this optimistic case). After the attack there will be b mutually exclusive groups of N_s/b candidates, each belonging to fixed parts of the key. So only the ordering within the b groups will remain unknown, leading to $((N_s/b)!)^b$ possible candidates (again in the optimistic case). A straightforward solution to avoid this issue is to deal with it at the architecture level, i.e. design an implementation where such Hamming distance leakages do not appear. For example, one could use multiple registers for this purpose (so that the output of a round never erases its input). In the next subsection, we change the permutation layer to mitigate the issue algorithmically.

4.2 Mitigating distance-based leakages with the permutation layer

The described Hamming distance attack is enabled by the structure of the permutation layer. It is therefore interesting to examine alternative permutations which avoid this particular property but retain the desired diffusion properties. This means that for each bit of the output of the new permutation, the offset within its word should not depend on the index of the S-box the bit originated from. Put another way, all S-box output bits of a specific offset (e.g. all first bits of the S-box outputs) should end up in the same position of a word after

the permutation (e.g. the first bit of a word). The diffusion of the permutation should still be optimal (as for the permutation of `Small-Present`). Optimal diffusion means that full diffusion (i.e. each output bit depends on each input bit) is reached after at most $\lceil log_b(N_s) \rceil$ rounds of the SP-network[2]. We have constructed several such permutations. For example, a fairly general variant for arbitrary values of N_s and b (with the requirement $N_s \equiv 0 \pmod b$) is given by:

$$P(i) = ((i \bmod b) * (N_s + 1) + (\lfloor i/b \rfloor \bmod b) * N_s + \lfloor i/b^2 \rfloor * b) \bmod (b * N_s).$$

This permutation connects the first bit of each S-box output to the first bit of a word after the permutation, the second bit of each S-box output to the second bit of a word after the permutation, ... Hence, an attack using the Hamming distance yields no extra information about the location of the key words.

4.3 Number of rounds per step

In order to keep our construction efficient, it is naturally desirable to minimize the number of rounds per step. In this respect, let us assume that an adversary can use two consecutive chunks of r to recover the input and the output of a step up to a permutation over the S-boxes. If one step does not have full diffusion (e.g. if it has too few rounds), she should again be able to exclude some positions for the key bytes and thus reduce the complexity of finding their order. By contrast, a step with full diffusion would then require to guess the permutation in the first place (so that nothing can be gained by such an attack anymore). In the following, we will consequently set as minimum criterion that one step should have complete diffusion. By using a permutation with optimal diffusion, $\lceil log_b(N_s) \rceil$ rounds are necessary to reach full diffusion. For 4-bit S-boxes ($b = 4$), a choice of $4 < N_s \leq 16$ would require at least two rounds and $16 < N_s \leq 64$ would require at least three rounds per step. A security margin of one or two rounds could be added depending on the number of S-boxes. Such parameters are sufficient for ensuring the security of the re-keying scheme in Fig. 1 (since they fulfill the six conditions stated in [25]). Besides, we note that they also provide a better mathematical security level than the modular multiplication of the Africacrypt 2010 proposal (e.g. some non-linearity is provided by the use of S-boxes). As mentioned in introduction, it is an interesting open problem to determine the number of rounds required for our construction to become a mathematically strong PRF.

5 Physical security analysis at the implementation level

In this section, we further move down to low abstraction levels and investigate the practicality of the "similar leakage" assumption that is probably the most important one to validate in practice. For this purpose, we consider a prototype implementation of our leakage-resilient construction on a FPGA, and evaluate

[2] Under the assumption that the S-box does not contain structural weaknesses.

its security against localized Electro-Magnetic (EM) field analysis [13], which was left as an open problem in [26]. The architecture of the design is detailed in Appendix B. It implements the first round of our construction in the first step (as described in Sec. 2), using 32 parallel PRESENT S-boxes and the permutation layer presented in Sec. 4.2. In order to allow worst-case analysis of our re-keying function, the architecture additionally provides two operational modes. In a first (open) mode, it is possible to change each single word of both the master key k and the random nonce r, keeping all the other words constant. While this is exactly what is prevented by construction (i.e. only carefully selected plaintexts are observable by the adversary), this mode was investigated in order to allow profiling without the impediment of the key-dependent algorithmic noise. In the second (fixed) mode, the master key is fixed and the word $r[i]$ of the nonce in the i^{th} step is replicated 32 times to cover the length of the nonce register. This corresponds to the actual circumstances that an adversary would face when attacking our leakage-resilient construction. In the rest of the section, we describe the worst-case profiling together with the selection of points of interests in the EM maps. Next, we present the results of attacks against our implementation in fixed mode taking advantage of these worst-case profiles. Finally, we discuss the relevance of worst-case evaluations and the interpretation of our results.

5.1 Worst-case profiling in open mode

In open mode, the adversary is able to independently observe the EM leakage characteristic of each subkey at different locations over the chip surface, without the influence of the key-dependent algorithmic noise (since the untargeted words can be set to random values). Hence, she can directly profile a leakage model of each subkey, just as in any other parallel implementation. In order to identify the univariate leakage of individual subkeys, we recorded 2^{16} measurements and computed the signal-to-noise ratio (SNR) for each word j, at each location (x, y) and for each time instant t. That is, $\mathrm{SNR}_j(x, y, t) = \frac{\hat{\sigma}(\hat{\mu}_{0\to 0}, \hat{\mu}_{0\to 1}, \ldots, \hat{\mu}_{F\to F})}{\hat{\mu}(\hat{\sigma}_{0\to 0}, \hat{\sigma}_{0\to 1}, \ldots, \hat{\sigma}_{F\to F})}$, where $\hat{\mu}_{u\to w}$ and $\hat{\sigma}_{u\to w}$ are the maximum likelihood estimators of the mean value and standard deviation of the leakages at time instant t conditioned on the transition from the value u to the value w of the target S-box. The 4-bit inputs to the key and nonce registers were carefully chosen from a 16-bit LFSR, in order to produce all the possible 256 transitions of a word in the state register exactly 256 times each. As a result, we obtained 32 SNR maps which are provided in Appendix C. It can be observed that the leakage of individual key words are clearly bounded to some confined regions on the chip surface. However, if we consider the leakage of each subkey as occurring simultaneously during an actual attack, then all the SNRs overlap significantly, as shown in the left part of Fig. 5. This result can be easily explained by looking at the placement of our design on the floorplan, which is shown in the right part of the figure. In fact, contrary to [14] where constraints on the placement were set, in our case the logic cells on the floorplan of the FPGA are located only in one large fuzzy region (due to an unconstrained placement). This region overlaps with the region of high SNR.

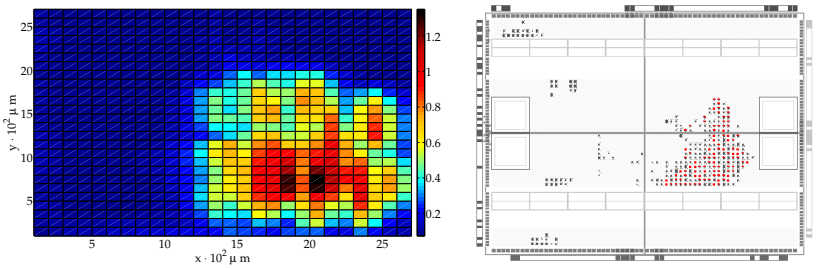


Fig. 5. Left: SNR over the 27×27 chip surface. Right: Placement on the floorplan

Given these preliminary results, the next question is to determine how to select the Points Of Interests (POIs) that will be used in our attacks. Quite naturally, the previous SNRs considered individually are not optimal in this respect, since they are based on the implicit assumption of independent noise. Therefore, we considered two additional criteria in order to better reflect the activity of individual key words considering the presence of key-dependent algorithmic noise:

$$C_2 = \max \frac{\mathrm{SNR}_j(x,y,t)}{\sum_{i \neq j} \mathrm{SNR}_i(x,y,t)}, \quad C_3 = \max \frac{\mathrm{SNR}_j(x,y,t)}{\max_{i \neq j} \mathrm{SNR}_i(x,y,t)}. \tag{2}$$

The intuition behind these criteria is that the best POIs should isolate one target S-box from either all the other S-boxes (on average) or from the "closest" S-box.

5.2 Attacks exploiting worst-case profiles in fixed mode

For the different selections of POIs in the previous section (including the basic SNR), we built leakage models and then performed 32 CPA attacks in fixed mode (i.e. for a fixed key, with the nonces defined in Sec. 2), using a fresh set of measurements. In this context, the data complexity for each attack is bounded to 16. Yet, nothing prevents an adversary to repeatedly measure each of its allowed input queries in order to get rid of physical noise. Hence, we performed attacks exploiting increasing number of traces (from 2^8 to 2^{16}) and first observed that the results were stable from 2^{12} traces on. Next, we had a look at the subkey ranks (i.e. the position of the correct subkeys in the 32 vectors of 16 candidates as provided by the attacks). For illustration, we list the ones obtained for the worst criteria (SNR) and the best one (i.e. C_2 or C_3 depending on the S-boxes):

Subkey ranks (SNR): [1 5 14 7 6 8 3 1 2 1 1 14 14 1 7 1 6 9 6 15 6 1 1 3 6 16 7 14 8 2 1 1 1].
Subkey ranks (best): [1 1 1 4 2 3 4 1 1 2 1 1 7 14 1 7 1 6 6 6 12 1 1 1 3 3 6 2 1 2 8 1 7 1].

One can directly observe that for a number of S-boxes (namely 9 for the worst and 13 for the best cases), the correct subkey is ranked first - hence suggesting that the localized EM profiling indeed allows improved attacks. Yet, looking at the CPA results more precisely, we also observed that firstly ranked subkeys were usually slightly better correlated than other candidates. By contrast for badly ranked subkeys, some of them showed very poor correlation results. The main consequence of this observation is that enumerating the master key remains a

computationally intensive task, even in the context where worst-case profiling is possible. To illustrate this claim, first observe that an underestimated time complexity for the enumeration can be obtained by computing the product of the subkey ranks. From the two previous lists, we obtain 2^{64} and 2^{46}, respectively. Improving this lower bound can be done by merging the lists, e.g. the 16 subkey ranks for 8-bit bytes corresponding to the same two attacks are given by:

Subkey ranks (SNR): [9 202 59 9 7 68 78 26 90 159 6 11 142 112 80 78].
Subkey ranks (best): [1 76 19 1 7 27 78 26 50 107 1 11 35 50 43 36].

leading to refined bounds of 2^{86} and 2^{66}, respectively. Intuitively, the better bounds derive from the fact that when merging dimensions (as an optimal key enumeration algorithm does [37]), the time complexity significantly increases every time both subkeys are not highly ranked. Using the rank estimation algorithm in [38], we finally obtained tight bounds for the master key rank as $[2^{115} - 2^{118}]$ and $[2^{99} - 2^{102}]$. Quite naturally, one could further consider that the knowledge of which subkeys are "easy to recover" is an additional outcome of the worst-case profiling[3]. In this conservative scenario, the adversary could reduce the dimension of her enumeration problem (down to 23 and 19, respectively), but our experiments still lead to security bounds of $[2^{89} - 2^{90}]$ and $[2^{69} - 2^{70}]$.

5.3 Interpretation of the results

The previous results are encouraging, as they suggest that the master key of our construction remains hard to enumerate, even in conditions where worst-case profiling is possible. Despite being difficult to compare (since based on totally different hardware assumptions), it is worth to note that under similar conditions, the security of a masked implementation would most likely be quite weak (since the localized electromagnetic measurements would allow obtaining low-noise leakages for each of the shares [36]). Nevertheless, it is also important to consider these results with care, as they only correspond to a single implementation context. In this respect, we emphasize the large number of factors that could have impact on our conclusions, such as the manufacturing technology, the distribution of active logic cells on the floorplan, the resolution of the coil, and the distance and materials between the probe and the leaking circuitry. We now briefly discuss the interpretation of our experiments with respect to two important axes, namely (i) what are the possible improvements and (ii) how representative is worst-case profiling. As far as improvements are concerned, they could certainly go in two directions. On the one hand, improved attacks could be considered. The most natural proposal would be to take advantage of multivariate leakages in order to better discriminate the S-boxes. It raises many interesting open problems. For example, the selection of POIs could not be based on SNRs anymore in this case. Best exploiting multivariate information would

[3] This is realistic as this information mainly depends on the placement of the S-boxes in the implementation. By contrast, the information of the correct subkey ranking depends on the key-dependent algorithmic noise and cannot be considered as constant for all attacks.

require to perform the information theoretic evaluations advertised in [34] and to exploit dimensionality reductions such as, e.g. [1,33]. These evaluations may turn out to become challenging in view of the huge data sets considered in our experiments (more than one week of measurements and 400GB of traces). On the other hand, several solutions to improve the countermeasure could be studied as well. In this respect, a starting observation is that the discrimination of S-boxes in our leakage-resilient construction inherently requires some profiling. Therefore, general questions about the portability of templates (e.g. in front of nanoscale devices with variability [31]) are particularly relevant in this case. Besides, the investigation of place-and-route constraints that best allow "interleaving" the S-boxes in our design is certainly another interesting scope for further research. Moving from FPGAs to ASICs could also reveal additional opportunities to improve the countermeasure. Eventually and if needed, taking advantage of space randomization such as proposed, e.g. in [28], is yet another possible track for security enhancement. As far as worst-case profiling is concerned, the main question is whether similar results could be obtained in the more realistic scenario where the implementation is in fixed mode for profiling as well. One direct problem is that in this context, the first-round leakages cannot be exploited as in this section. In fact, the transitions used to compute our SNRs would all be equivalent up to a permutation in this case, making it impossible to select POIs for different subkeys. Nevertheless, alternative profiling paths also exist. One solution would be to "group" similar transitions thanks to a non-bijective transformation. But the choice of a transformation that adequately captures the similarity of different transitions is not straightforward (and we can anyway only loose information by profiling in this way). For example, experiments performed under a Hamming distance transformation exhibited significantly reduced SNRs for our prototype. Another solution is to profile second-round leakages. But this would require building more templates and could also be limited by the bounded data complexity issues discussed in Sec. 3.2. Other options certainly exist and are an interesting scope for further research. Meanwhile, we conclude that although fixed-key profiling may be more annoying to perform in practice, considering worst-case profiling for reference is certainly a relevant choice for the evaluation of our countermeasure in view of the improved attacks that could be designed.

6 An open source and generic VHDL code

In order to estimate the costs of our method in terms of speed and size in silicon, we implemented a leakage-resilient re-keying function in VHDL. We decided to keep the design as flexible as possible to allow realizing and testing different configurations. The parameters a designer can set before synthesizing the re-keying function comprise the number of rounds within a step and the number of steps to generate a fresh key. Furthermore, both 4-bit PRESENT S-boxes and 8-bit AES S-boxes can be selected. The designer can additionally choose the desired bit-size of the data path and hence the size of the key-material generated. Finally, and as a complement to functional parameters, a tradeoff between speed and required area can be configured. That is, the implementation

supports unrolling of rounds, where the overall number of rounds must be a multiple of those performed in a single clock cycle. Thus, the latency to generate a fresh key using our construction can be computed as (number of steps) $*$ (number of rounds) / (unrolled rounds) + (one initialization cycle). An overview of this architecture is given in Fig. 6. The dotted lines in the figure depict possible extensions of the design that were not used in this paper. For example, the design is ready for including a final step like a Davies-Meyer transformation and/or a key expansion that transforms the key between each round. Our synthesis results for implementations with different configuration options are shown in Table 3. We targeted the UMC 0.18 μm FSA0A_C standard-cell library [8] and did not perform timing optimizations. The first two implementations use our recommended configuration with two different degrees of round unrolling, while the third implementation features the absolute minimal options which could still yield a moderate degree of security. All implementations use the PERSENT S-box, the linear layer proposed in Sec. 4.2, no key expansion, and no final transformation in the step[4].

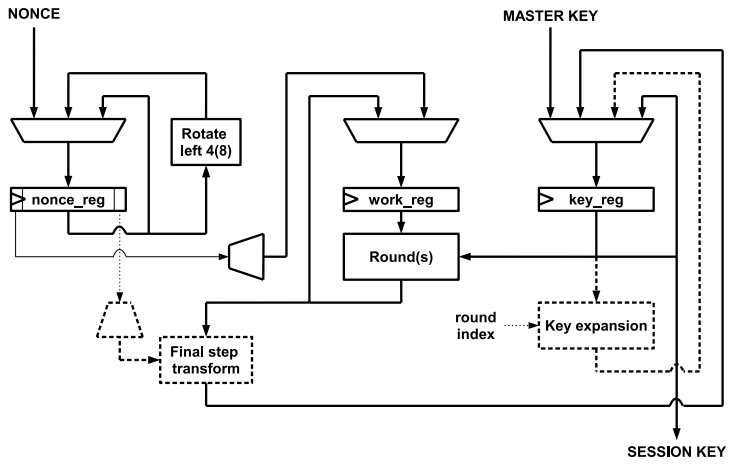


Fig. 6. Overview of our open source hardware architecture for fresh re-keying.

It is important to note that 562 cycles for 7,300 GEs correspond to the cost of a first-order masked implementation for the modular multiplication in [25]. So the performances of our architecture already compare favorably with this one (moving to higher-order masking naturally makes the comparison even better). Besides and most importantly, our implementation is a parallel one while the Africacrypt 2010 one is only 8-bit wide. This means that reaching acceptable noise levels for the masking countermeasure to become effective requires additional shuffling, e.g. as proposed in [24] and leading to significant performance

[4] Our source codes are available under an open source license on the authors' home pages.

overheads (in the 10th of thousands cycles). These preliminary investigations suggest with good confidence that in a hardware context, the fresh re-keying based the construction we describe in this paper had good potential to lead to a better performance vs. security tradeoff than a masked modular multiplication.

Table 3. Synthesis results using the UMC 0.18 μm FSA0A_C library.

	Latency	**Area**	**Clock freq.**
S-boxes/steps/rounds/unrolled rounds	cycles	GE	MHz
32/32/5/1	161	7,300	338
32/32/5/5	33	16,828	210
24/20/3/1	61	5,302	354

7 Conclusion

In this paper, we presented an exploratory analysis of the design space for fresh re-keying opened by the use of leakage-resilient PRFs to prevent side-channel attacks. Our results provide new guidelines for the choice of block cipher components to consider in this context and for their implementation. Admittedly, the understanding and security evaluation of this type of constructions is still far from the one of standard protections such as masking and shuffling. Yet, the preliminary investigations we describe in this paper are promising and suggest new solutions to build physically secure hardware devices. From the side-channel security point of view, further optimizing the localized electromagnetic measurements by exploiting multivariate attacks is certainly worth further investigations. Depending on the strength of these advanced attacks, space-randomized implementations (where the localization of the S-box executions would vary over time) could then be designed as well. From a more theoretical point of view, it would be interesting to investigate whether a PRF could be directly obtained by extending the number of rounds of our new construction. It would allow to use it directly as a leakage-resilient primitive rather than for re-keying the AES, and maybe to obtain additional performance gains.

Acknowledgements. This work has been funded in part by the European Commissions ECRYPT-II NoE (ICT-2007-216676), by the 7th framework European project TAMPRES, by the ERC project 280141 (acronym CRASH) and by the German Federal Ministry of Education and Research project 01IS11035Y (acronym ARAMiS). François-Xavier Standaert is a Research Associate of the Belgian Fund for Scientific Research (FNRS-F.R.S).

References

1. C. Archambeau, E. Peeters, F.-X. Standaert, and J.-J. Quisquater. Template attacks in principal subspaces. In L. Goubin and M. Matsui, editors, *CHES*, volume 4249 of *LNCS*, pages 1–14. Springer, 2006.
2. A. Bogdanov, L. R. Knudsen, G. Leander, C. Paar, A. Poschmann, M. J. B. Robshaw, Y. Seurin, and C. Vikkelsoe. Present: An ultra-lightweight block cipher. In P. Paillier and I. Verbauwhede, editors, *CHES*, volume 4727 of *LNCS*, pages 450–466. Springer, 2007.
3. A. Bogdanov, L. R. Knudsen, G. Leander, F.-X. Standaert, J. P. Steinberger, and E. Tischhauser. Key-alternating ciphers in a provable setting: Encryption using a small number of public permutations. In D. Pointcheval and T. Johansson, editors, *EUROCRYPT*, volume 7237 of *LNCS*, pages 45–62. Springer, 2012.
4. E. Brier, C. Clavier, and F. Olivier. Correlation power analysis with a leakage model. In M. Joye and J.-J. Quisquater, editors, *CHES*, volume 3156 of *LNCS*, pages 16–29. Springer, 2004.
5. S. Chari, C. S. Jutla, J. R. Rao, and P. Rohatgi. Towards sound approaches to counteract power-analysis attacks. In M. J. Wiener, editor, *CRYPTO*, volume 1666 of *LNCS*, pages 398–412. Springer, 1999.
6. Y. Dodis and K. Pietrzak. Leakage-resilient pseudorandom functions and side-channel attacks on feistel networks. In T. Rabin, editor, *CRYPTO*, volume 6223 of *LNCS*, pages 21–40. Springer, 2010.
7. S. Dziembowski and K. Pietrzak. Leakage-resilient cryptography. In *FOCS*, pages 293–302. IEEE Computer Society, 2008.
8. Faraday Technology Corporation. Faraday FSA0A_C 0.18 μm ASIC Standard Cell Library, 2004. Details available online at `http://www.faraday-tech.com`.
9. S. Faust, K. Pietrzak, and J. Schipper. Practical leakage-resilient symmetric cryptography. In Prouff and Schaumont [29], pages 213–232.
10. K. Gandolfi, C. Mourtel, and F. Olivier. Electromagnetic analysis: Concrete results. In Çetin Kaya Koç, D. Naccache, and C. Paar, editors, *CHES*, volume 2162 of *LNCS*, pages 251–261. Springer, 2001.
11. L. Goubin and J. Patarin. Des and differential power analysis (the "duplication" method). In Çetin Kaya Koç and C. Paar, editors, *CHES*, volume 1717 of *LNCS*, pages 158–172. Springer, 1999.
12. J. Guo, T. Peyrin, A. Poschmann, and M. J. B. Robshaw. The led block cipher. In B. Preneel and T. Takagi, editors, *CHES*, volume 6917 of *LNCS*, pages 326–341. Springer, 2011.
13. J. Heyszl, S. Mangard, B. Heinz, F. Stumpf, and G. Sigl. Localized electromagnetic analysis of cryptographic implementations. In O. Dunkelman, editor, *CT-RSA*, volume 7178 of *LNCS*, pages 231–244. Springer, 2012.
14. J. Heyszl, D. Merli, B. Heinz, F. D. Santis, and G. Sigl. Strengths and limitations of high-resolution electromagnetic field measurements for side-channel analysis. In S. Mangard, editor, *CARDIS*, LNCS. Springer Berlin Heidelberg, 2012.
15. A. Joux, editor. *Advances in Cryptology - EUROCRYPT 2009, 28th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Cologne, Germany, April 26-30, 2009. Proceedings*, volume 5479 of *LNCS*. Springer, 2009.
16. B. S. K. Jr., Çetin Kaya Koç, and C. Paar, editors. *Cryptographic Hardware and Embedded Systems - CHES 2002, 4th International Workshop, Redwood Shores, CA, USA, August 13-15, 2002, Revised Papers*, volume 2523 of *LNCS*. Springer, 2003.

17. P. C. Kocher. Leak resistant cryptographic indexed key update. US Patent.
18. G. Leander. Small scale variants of the block cipher present. Cryptology ePrint Archive, Report 2010/143, 2010.
19. P. A. MacMahon. *Percy Alexander MacMahon: Collected Papers - Vol. 1: Combinatorics*. MIT Press, 1978.
20. S. Mangard. Hardware countermeasures against dpa ? a statistical analysis of their effectiveness. In T. Okamoto, editor, *CT-RSA*, volume 2964 of *LNCS*, pages 222–235. Springer, 2004.
21. S. Mangard, E. Oswald, and T. Popp. *Power analysis attacks - revealing the secrets of smart cards*. Springer, 2007.
22. S. Mangard, E. Oswald, and F.-X. Standaert. One for all – all for one: unifying standard differential power analysis attacks. *IET Information Security*, 5(2):100–110, 2011.
23. S. Mangard, T. Popp, and B. M. Gammel. Side-channel leakage of masked CMOS gates. In A. Menezes, editor, *CT-RSA*, volume 3376 of *LNCS*, pages 351–365. Springer, 2005.
24. M. Medwed, C. Petit, F. Regazzoni, M. Renauld, and F.-X. Standaert. Fresh re-keying ii: Securing multiple parties against side-channel and fault attacks. In E. Prouff, editor, *CARDIS*, volume 7079 of *LNCS*, pages 115–132. Springer, 2011.
25. M. Medwed, F.-X. Standaert, J. Großschädl, and F. Regazzoni. Fresh re-keying: Security against side-channel and fault attacks for low-cost devices. In D. J. Bernstein and T. Lange, editors, *AFRICACRYPT*, volume 6055 of *LNCS*, pages 279–296. Springer, 2010.
26. M. Medwed, F.-X. Standaert, and A. Joux. Towards super-exponential side-channel security with efficient leakage-resilient prfs. In Prouff and Schaumont [29], pages 193–212.
27. K. Pietrzak. A leakage-resilient mode of operation. In Joux [15], pages 462–482.
28. F. Poucheret, L. Barthe, P. Benoit, L. Torres, P. Maurine, and M. Robert. Spatial EM jamming: A countermeasure against EM analysis? In *VLSI-SoC*, pages 105–110. IEEE, 2010.
29. E. Prouff and P. Schaumont, editors. *Cryptographic Hardware and Embedded Systems - CHES 2012 - 14th International Workshop, Leuven, Belgium, September 9-12, 2012. Proceedings*, volume 7428 of *LNCS*. Springer, 2012.
30. J.-J. Quisquater and D. Samyde. Electromagnetic analysis (EMA): Measures and counter-measures for smart cards. In I. Attali and T. P. Jensen, editors, *E-smart*, volume 2140 of *LNCS*, pages 200–210. Springer, 2001.
31. M. Renauld, F.-X. Standaert, N. Veyrat-Charvillon, D. Kamel, and D. Flandre. A formal study of power variability issues and side-channel attacks for nanoscale devices. In K. G. Paterson, editor, *EUROCRYPT*, volume 6632 of *LNCS*, pages 109–128. Springer, 2011.
32. M. Rivain and E. Prouff. Provably secure higher-order masking of AES. In S. Mangard and F.-X. Standaert, editors, *CHES*, volume 6225 of *LNCS*, pages 413–427. Springer, 2010.
33. F.-X. Standaert and C. Archambeau. Using subspace-based template attacks to compare and combine power and electromagnetic information leakages. In E. Oswald and P. Rohatgi, editors, *CHES*, volume 5154 of *LNCS*, pages 411–425. Springer, 2008.
34. F.-X. Standaert, T. Malkin, and M. Yung. A unified framework for the analysis of side-channel key recovery attacks. In Joux [15], pages 443–461.

35. F.-X. Standaert, O. Pereira, Y. Yu, J.-J. Quisquater, M. Yung, and E. Oswald. Leakage resilient cryptography in practice. In A.-R. Sadeghi and D. Naccache, editors, *Towards Hardware-Intrinsic Security*, Information Security and Cryptography, pages 99–134. Springer Berlin Heidelberg, 2010.
36. F.-X. Standaert, N. Veyrat-Charvillon, E. Oswald, B. Gierlichs, M. Medwed, M. Kasper, and S. Mangard. The world is not enough: Another look on second-order dpa. In M. Abe, editor, *ASIACRYPT*, volume 6477 of *LNCS*, pages 112–129. Springer, 2010.
37. N. Veyrat-Charvillon, B. Gerard, M. Renauld, and F.-X. Standaert. An optimal key enumeration algorithm and its application to side-channel attacks. Cryptology ePrint Archive, Report 2011/610, 2011.
38. N. Veyrat-Charvillon, B. Gerard, and F.-X. Standaert. Security evaluations beyond computing power. In T. Johansson and P. Q. Nguyen, editors, *Advances in Cryptology EUROCRYPT 2013*, volume 7881 of *LNCS*, pages 126–141. Springer Berlin Heidelberg, 2013.
39. Y. Yu and F.-X. Standaert. Practical leakage-resilient pseudorandom objects with minimum public randomness. In E. Dawson, editor, *CT-RSA*, volume 7779 of *LNCS*, pages 223–238. Springer, 2013.
40. Y. Yu, F.-X. Standaert, O. Pereira, and M. Yung. Practical leakage-resilient pseudorandom generators. In E. Al-Shaer, A. D. Keromytis, and V. Shmatikov, editors, *ACM CCS*, pages 141–151. ACM, 2010.

A Impact of key words repetitions

Let us denote by $\mathcal{S}$ a multiset of N_s key words uniformly distributed in $[0..2^b-1]$. The number of permutations of these key words, or equivalently the complexity to order them, depends on the multiplicities of these key words in $\mathcal{S}$. We denote by m_j (with $1 \leq j \leq 2^b - 1$) the multiplicity of value j, i.e. the number of times this value appears in the multiset $\mathcal{S}$. For instance, with $\mathcal{S} = \{3, 3, 5, 8, 8, 8\}$ ($N_s = 6$), we have $m_3 = 2$, $m_5 = 1$, $m_8 = 3$ and $m_j = 0, \forall j \in [0, 2^4-1] \backslash \{3, 5, 8\}$. Let us additionally denote by M_i^q the random variable representing the number of multiplicities equal to q when selecting the i^{th} key word (with $1 \leqslant i \leqslant N$). We can then write the following recursion formula that, under relevant boundary conditions, gives us the desired probabilities $\forall i, q, k \in [0..N_s]$:

$$\Pr[M_{i+1}^q = k] = \frac{k+1}{2^b} \Pr[M_i^q = k+1] + \sum_{l=0}^{N} \left(\Pr[M_i^q = k-1] \frac{l}{2^b} + \Pr[M_i^q = k] \left(1 - \frac{k+l}{2^b}\right) \right) \Pr[M_i^{q-1} = l].$$

From these probabilities, we can deduce those of the time complexities of attacks for various parameters N_s and b. In practice, we used Monte Carlo sampling to evaluate the mean complexities thanks to the multiplicities distribution. That is, we drew a large (i.e. sufficient to have accurate estimates) number of independent random variables following a specific law to estimate its expectation using the law of large numbers.

B Architecture's Design on a FPGA

Our analysis was conducted on a Xilinx Spartan 3 FPGA device manufactured in a 90 nm technology. We performed localized magnetic field measurements using a coil with a resolution of 100 μm very closely positioned to the depackaged circuit's front side surface. We performed 27×27 measurements covering the surface area confined by the conjunctions of the bonding wires. The architecture of the design is shown in Fig. 7.

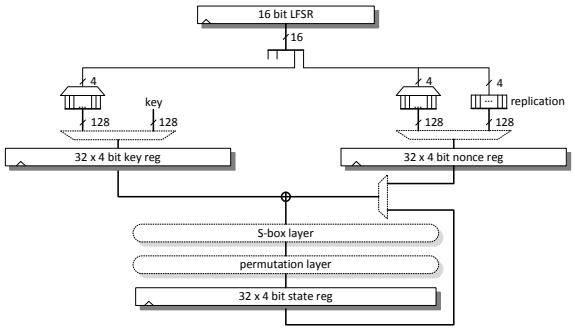
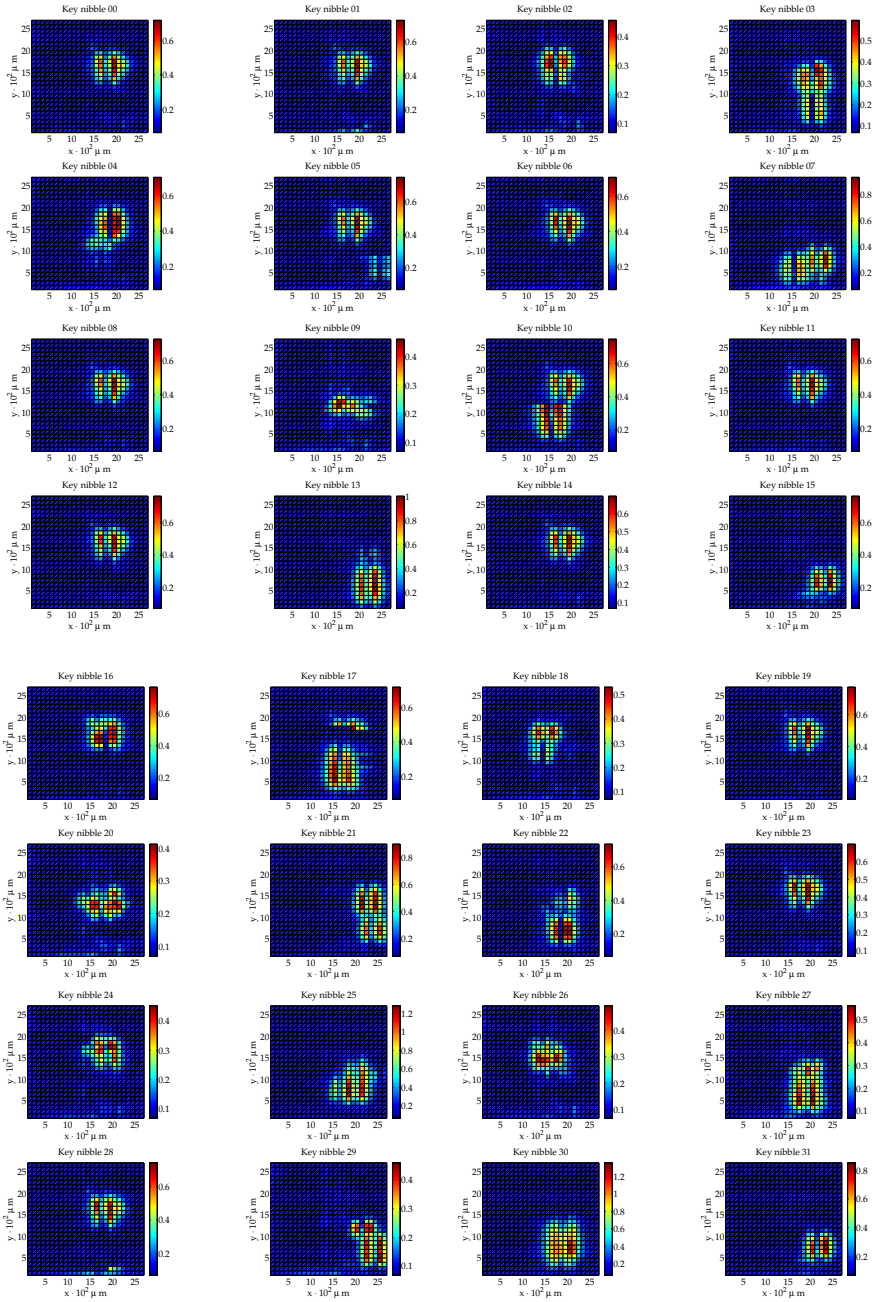


Fig. 7. Prototype architecture for worst-case EM profiling.

C SNR maps of the 32 key words over the chip surface



Understanding the Limitations and Improving the Relevance of SPICE Simulations in Side-Channel Security Evaluations

Dina Kamel, Mathieu Renauld, Denis Flandre, François-Xavier Standaert.
ICTEAM/ELEN/Crypto Group, Université catholique de Louvain, Belgium.

Abstract. Simulation is a very powerful tool for hardware designers. It generally allows the preliminary evaluation of a chip's performances before its final tape out. As security against side-channel attacks is an increasingly important issue for cryptographic devices, simulation also becomes a desirable option for preliminary evaluation in this case. However, its relevance highly depends on the proper modeling of all the attack peculiarities. For example, several works in the literature directly exploit SPICE-like simulations without considering measurement peripherals. But the outcome of such analyses may be questionable, as witnessed by the recent results of Renauld et al. at CHES 2011, which showed how far the power traces of an AES S-box implemented using a dynamic and differential logic style fabricated in 65 nm CMOS can lie from their post-layout simulations. One important difference was found in the linear dependencies between the (simulated and actual) traces and the S-box input/output bits. While simulations exhibited highly non-linear traces, actual measurements were much more linear. As linearity is a crucial parameter for the application of non-profiled side-channel attacks (which are only possible under the assumption of "sufficiently linear leakages"), this observation motivated us to study the reasons of such differences. Consequently, this work discusses the relevance of simulation in security evaluations, and highlights its dependency on the proper modeling of measurement setups. For this purpose, we present a generic approach to build an adequate model to represent measurements artifacts, based upon real data from equipment providers for our AES S-box case study. Next, we illustrate the transformation of simulated leakages, from highly non-linear to reasonably linear, exploiting our model and regression-based side-channel analysis. While improving the relevance of simulations in security evaluations, our results also raise doubts regarding the possibility to design dual-rail implementations with highly non-linear leakages.

1 Introduction

Side-channel attacks are considered an escalating threat to the physical security of cryptographic devices. These attacks are worrisome for actual applications since they are relatively cheap, easy to conduct and can be extremely powerful (e.g. leading to key recoveries). Generally, side-channel attacks are classified as profiled or non-profiled [9]. Profiled attacks, such as using templates [2], are based upon the assumption that the adversary has prior access to the target device and is capable of accurately profiling it. By contrast, non-profiled attacks

(e.g. Correlation Power Analysis (CPA) [1]) represent another group of (suboptimal but sometimes more realistic) adversaries who do not have the required capabilities to profile the leakages of the target. This difference is important as it was shown in a recent work by Whitnall et al. that we have a strict separation between such attacks [22]. That is, there (theoretically) exist leaking devices that can only be attacked under some a priori assumptions obtained through profiling. In practice though, the actual scenarios where only profiled attacks can succeed usually correspond to so-called "highly non-linear" leakage functions [21] - of which the existence remains an open question. Informally, a leakage function is considered non-linear if it cannot be accurately estimated by a linear combination of some intermediate bits processed by the target device (e.g. with Schindler et al.'s stochastic approach [16]). We say that it is highly non-linear if the resulting models do not allow successful key recoveries.

Security against side-channel attacks is usually obtained through a combination of countermeasures. As such countermeasures imply significant performance overheads, the best tradeoff between efficiency and security for small embedded devices has become an important research topic. But as usual in hardware design, the final performances of a taped out chip is not the only cost criteria. In particular, it has been frequently observed that countermeasures looking sound on a mathematical basis could be less effective than expected, because of physical artifacts (the problem of masking and glitches is a well-known example of this concern [10]). Hence, avoiding such limitations as early as possible in the development of a cryptographic implementation is also desirable. As a result, the improvement of simulation-based side-channel security evaluations has become another topic of interest [8,13], with the long-term goal to develop integrated design flows, with physical security as part of the optimization criteria [12,20]. Quite naturally, such a goal also raises questions regarding the relevance of simulated leakages, as noted by Tiri and Verbauwhede [19], or more recently by Renauld et al. [14]. The latter used state-of-the-art evaluation tools for quantifying information leakages, and put forward a difference of linearity between actual and simulated traces, raising questions about both the relevance of simulations for this criteria, and the possibility to design circuits with non-linear leakages.

In this paper, we contribute to these two important issues. Namely, we first narrow the gap between simulations and actual measurements. For this purpose, we develop a flexible model that captures measurement artifacts in side-channel attacks and integrate it into SPICE simulations. In order to validate it, we then analyze a Dynamic and Differential Swing-Limited Logic (DDSLL [3]) implementation of the AES S-box, in a 65nm CMOS technology (implementation details of the DDSLL S-box are beyond the scope of this study and can be found in [7]). This target circuit was chosen because it includes a circuit-level countermeasure for which non-linear leakages are expected (and have been previously observed in simulations [14]). But as highlighted in this previous work, such a non-linearity was not found in actual measurements. We repeat this comparison with gradually improved simulation models, that include elements such as the cables, socket and package that potentially affect the leakage traces, eventually leading to ac-

curate approximations of our measurements. By performing a regression-based information theoretic evaluation of our target implementation, we demonstrate how certain elements of the model can explain the transformation of a leakage function, from highly non-linear to fairly linear. Since this transformation is quite independent of the logic style design, our study consequently raises doubts regarding the possibility to design a dual-rail implementation with highly non-linear (and hard to linearize) leakages. Eventually, and despite the fact that the model we propose is specific and adapted to the measurement environment that we used in our experiments, we mention that our approach is rather generic. That is, the contribution of each element in the model will of course differ depending on the type of package, socket (if used), type of cable, etc. But we expect the way we include these elements in a model and their relative importance to remain meaningful for a wide range of implementations and setups.

2 Preliminaries

2.1 Notations

In this work, capital letters are assigned to random variables, while lower case letters refer to samples of these random variables. For example, L is the random variable representing a leakage and l is an actual power trace picked up from this distribution. The power trace is composed of t time samples. Generally, the leakage function has two input arguments: the discrete random variable X which denotes the value of the processed data under investigation, and the continuous random variable N which represents the noise in the measurements. The leakage function variable denoted by $L(\ ,\ )$ contains either random variable arguments or fixed arguments. For example, $L(x, N)$ is a random variable representing the noisy traces corresponding to a fixed processed data x. We also denote the t^{th} time sample in a leakage trace as $L_t(\ ,\ )$. In the measurement environment, we finally define (noise-free) mean traces as:

$$\overline{L_t^{meas}}(X) = \underset{n}{\hat{\mathbf{E}}}\, L(X, n), \tag{1}$$

where $\hat{\mathbf{E}}$ denotes the sample mean operator[1]. Note that in the simulation environment, the provided traces are noise-free by default. In this case, and in order to analyze the impact of noise on the security of the S-box implementation, we added a Gaussian noise to the simulations[2] (this is a usual assumption in side-channel attacks, which will also be confirmed in Section 3). As a result, our investigations considered three types of leakage traces. The first case is the simulated leakage function which is given by:

$$L_t^1(X, N) = L_t^{sim}(X) + N. \tag{2}$$

[1] The noise-freeness naturally depends on the sampling, but in view of our low-noise measurements, we were able to extract well estimated means in our experiments.

[2] Gaussian noise is added to the simulated traces in a post processing step assuming the noise-free simulated traces to provide the means of our leakages.

The second case is when the actual power traces are considered. Here, the noise is directly present in the measurements obtained from the oscilloscope and the corresponding leakage function is given by:

$$L_t^2(X, N) = L_t^{meas}(X, N). \tag{3}$$

The third case is a hybrid leakage function combining the noise-free mean traces of Equation 1 with Gaussian noise:

$$L_t^3(X, N) = \overline{L_t^{meas}}(X) + N. \tag{4}$$

2.2 SCA evaluation metrics

In order to evaluate the leakage of the measured and simulated traces of the DDSLL S-box and assess their linearity, we estimated the information theoretic metric put forward in [17] and refined in [15]. That is, we computed the Perceived Information (PI) that corresponds to the amount of information that can be exploited by a side-channel adversary given a certain leakage model, namely:

$$\hat{\mathrm{PI}}(X; L) = \mathrm{H}[X] - \sum_{x \in X} \Pr[x] \sum_{l \in L} \Pr_{chip}[l|x] \; . \; \log_2(\hat{\Pr}_{model}[x|l]).$$

It captures the accuracy of the adversary's leakage model estimate (given by $\hat{\Pr}_{model}[x|l]$) at predicting the true (unknown) leakage function of an implementation (denoted as $\Pr_{chip}[l|x]$). In case these two distributions are identical (e.g. in a simulated environment), we have a perfect evaluation and the PI is equivalent to the standard definition of mutual information (i.e. it captures the worst-case information leakages). By contrast, if these distributions deviate - because of practical limitations in the number of traces used for profiling, or because of a simplified (e.g. linear) leakage model - the PI is the (best available) estimate of the target device's leakage, biased by this slightly incorrect model.

2.3 Estimation tools

As previously mentioned, computing the PI first requires to estimate the leakage distribution with a model $\hat{\Pr}_{model}[x|l]$. Then, the evaluator just has to sample values l from the true distribution $\Pr_{chip}[l|x]$ and estimate the previous equation from it. In this section, we briefly describe the statistical tool we used for this purpose, namely Schindler et al.'s profiled stochastic approach [16]. It essentially aims to approximate the leakage samples with a linear combination of some base vectors. That is, during profiling the adversary chooses the base vectors $g_0(x)$, $g_1(x)$, ..., $g_d(x)$. These base vectors represent monomials in x (the input and/or output and/or intermediate bits of the DDSLL S-box under attack), e.g. d=8 in case of a linear model for an 8-bit S-box. Then, the adversary performs a regression in order to build a model $\hat{L}_t(x, N) = \Sigma_i \; \beta_{i,t} \; . \; g_i(x) + N$ that best suits the true leakages. Of course, the stochastic approach with a linear model cannot be perfect if the leakage function is non-linear. But by increasing the

number of elements in the basis, non-linearities can additionally be captured (e.g. with a d=256-element basis, a stochastic model is equivalent to the templates in [2], excepted that it only estimates a single variance for N). In the rest of this paper, the use of the stochastic approach was naturally motivated by our goal to analyze the linearity of different (simulated and real) leakage functions.

2.4 Test chip and measurement setup

The chip was fabricated using a low-power 65 nm technology. All the input, output and clock signals of the DDSLL S-box are buffered. The S-box is powered by its own supply rail which is different from that of the buffers in order to directly assess the security of the DDSLL S-box implementation by itself. The chip is packaged in a 44-pin CQFP package. To study the power traces, the voltage drop on a resistor (1kΩ) introduced in the path of the power supply of the measured S-box is monitored using a differential probe and adequately manipulated in a post processing step to get the power traces. We used a high sampling rate oscilloscope (1 Gsample/second), while running the chips at 2 MHz (motivated by interface constraints of our prototype board). Measurements were repeated 100 times to assess the security of DDSLL S-box under real noise. On the other hand, simulated power traces are obtained using post-layout SPICE simulations. Only parasitic capacitances to GND are extracted from the layout using Synopsys Star-RCXT. Both simulations and measurements are done at ambient temperature using a nominal supply voltage of 1.2 V. Figure 1 shows the floor plan of the test chip and all the PADs description are given in Table 1.

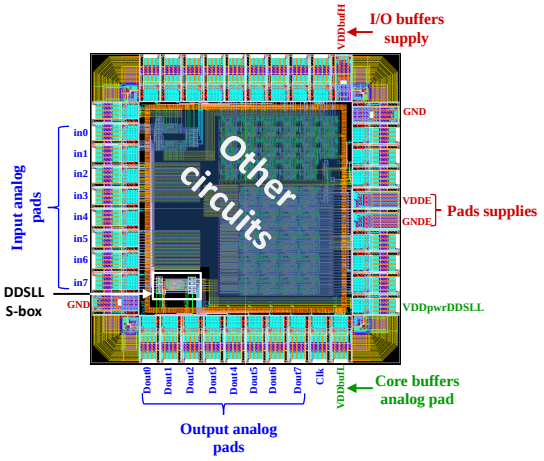


Fig. 1. Floor plan of the test chip.

Table 1. Description of the different PADs shown in Fig. 1.

Model	Description
in[0-7]	Input analog PADs
Dout[0-7]	Output analog PADs
Clk	Clock analog PAD
$VDDpwrDDSLL$	Supply analog PAD for the DDSLL AES S-box
$VDDBUFL$	Supply analog PAD for core buffers
$VDDBUFH$	Supply PAD for I/O buffers
GND	2 Ground analog PADs
$VDDE$	Supply PAD for the PADs
$GNDE$	Ground PAD for the PADs

3 Simulated traces vs. actual measurements

In this section, we exploit a linear regression based information theoretic analysis to analyze simulated and actual leakage traces, and to evaluate the practical relevance of certain assumptions regarding the physical behavior of our measurements. For this purpose, we essentially compute the PI between 256 events x given their leakage, using a model $\hat{\Pr}_{model}[x|l]$ obtained thanks to the stochastic approach[3]. Our analyses are limited to the evaluation phase of the DDSLL S-box, which have been previously shown to leak the most information [14].

3.1 Direct analysis

In the direct analysis, we compare preliminary post-layout SPICE simulations and actual measurements. For the preliminary simulations, we simply probed the current flowing from the supply voltage without adding a resistor in its path and without any model representing the measurement specificities. The upper part of Figure 2 plots the current traces of both preliminary simulations and measurements, allowing to highlight two significant differences. First, we observe the different scales in the X-axes: computations end in about 8 ns for preliminary simulations, whereas in measurements, they last around 100 ns. Next, there exist high frequency components resulting from the computations of the different blocks of the S-box in preliminary simulations (assumably related to the high switching activity in strong cryptographic functions). By contrast in measurements, these high frequency components are filtered out due to the presence of different elements related to the chip, PCB, cables and equipments. Besides, it is also noticeable that measured traces oscillate at a changing frequency that is not observed in simulations. This point can be explained by the fact that in

[3] Strictly speaking, there are 256^2 transitions that could be considered. To reduce the cost of our analysis, we only considered transitions between 0 and a value between 0 and 255. From past experiments, we do not expect this restriction to have a strong impact on our conclusions, in particular for the part related to the leakages linearity.

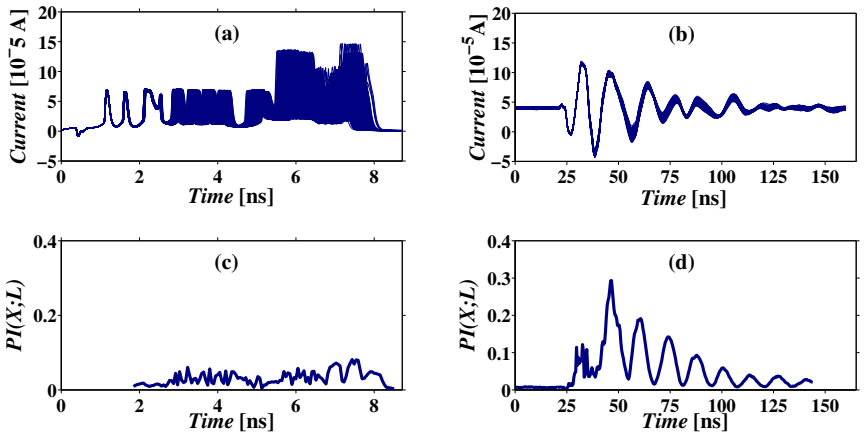


Fig. 2. (a) Simulated and (b) measured current traces of the DDSLL S-box during the evaluation phase for different inputs (inputs transition from 0 to an arbitrary state from 0 to 255). (c) Perceived information using a linear stochastic model in function of time for simulated traces without any noise and (d) measured traces with real noise.

preliminary simulations, the traces are perfectly aligned with minimal amplitude and time shifts at the beginning of the clock cycle, and start to misalign as the computations progress. Such a phenomenon is mainly due to the presence of unavoidable imbalances between the capacitances of the differential signals and circuitry which lead to some dependencies on the processed data. Due to the larger span of the measured traces over time, this misalignment appears to be smaller in actual traces. Furthermore, it is also reshaped by the various components present in the measurement setup forming oscillations

As a natural complement to this informal analysis, we investigated the informativeness of all the time samples in the simulated and measured traces, as illustrated in Figure 2 (c) and (d), respectively. The leakage function in Figure 2 (c) corresponds to simulations (defined as $L_t^1(X)$ in Section 2.1) where the noise parameter is set to 0. The leakage function in Figure 2 (d) corresponds to the actual measurements $L_t^2(X, N)$. Clearly, the linear stochastic model is unable to efficiently extract information from the simulated traces (for all time samples) as the PI < 0.1 for basic simulation, whereas it is quite successful in the actual measurement case where the PI is $\simeq 0.3$ for measurement noise in the order of 6.10^{-6}. So we can indeed conclude that there is a significant difference of linearity between these two contexts. In view of the previously mentioned filtering effects, a natural explanation would be that these deviations can be explained by the measurement setup. As a result, we try in the next section to incorporate measurement artifacts in our simulation environment.

3.2 Measurement artifact models

A measurement environment includes different components such as equipment, cables, PCB, socket, chip's package and pads, ... The precise modeling of these components is a tedious task and goes with a risk to significantly increase the simulation time (which is of course undesirable). As a result, our main goal is to have a flexible model that adequately captures the most significant physical phenomenon contributing to the filtering of the traces seen in measurements (thus removing all the nonlinearities, as will be shown in the following section), within reasonable simulation time. One way to do this is to gradually add components that we believe important to the model, until the simulations are "good enough". For example, we easily found out (as will be confirmed in the rest of the section) that the resistor in the supply path, package [4], QFP socket [18], cables and differential oscilloscope probe are important components in the model[4]. By contrast, we choose to neglect other components to keep the model simple, such as the terminations of the testing equipment (e.g. the supply sources and the oscilloscope), the bonding wires, the pads and the PCB connecting paths.

Note that a previous work [5] developed a method to simulate cryptographic systems using an equivalent circuit model (the linear equivalent circuit and current source model [6,11]), based on real measurements obtained from a prototype chip. The authors built their model for an FPGA implementing the AES algorithm and accurately emulated the peripheral circuitry, such that the simulation results are in agreement with measurements. Our approach is rather different because we focus on building an admittedly simpler peripheral model that does not need prior measurements (which typically corresponds to the situation during ASIC developments). So both approaches can be viewed as complementary: ours for preliminary investigations based on simulation models only, and the one in [5] whenever measurements are available to refine the model.

Figure 3 illustrates the equivalent circuit model of the measurement artifacts used in our improved simulation environment, and Table 2 details the description and value of each element in the model. Based upon the length of the cables used, the self inductance is calculated for the supply and input/output signals. However, since the ground occupies the entire lower plane of the PCB and is at the same time connected to the supplying equipment via several cables (the grounds of all equipment are connected together), we assumed the ground cable inductance to be less than the supply cable inductance. As for the mutual inductances and capacitances between adjacent pins in the socket and package, only those to the immediate pins are shown in Figure 3. Nevertheless, mutual inductances to the next-of-immediate pins in the package model are also included (not shown in the Figure 3 for simplicity). The relationship between pins can be seen from the test chip in Figure 1 in order to effectively model the mutual inductances and capacitances. The value of the resistor in the S-box supply path

[4] Models for the package [4] and QFP socket [18] do not exactly correspond to our setup (e.g. they differ in pin count) - but were the only publicly available ones.

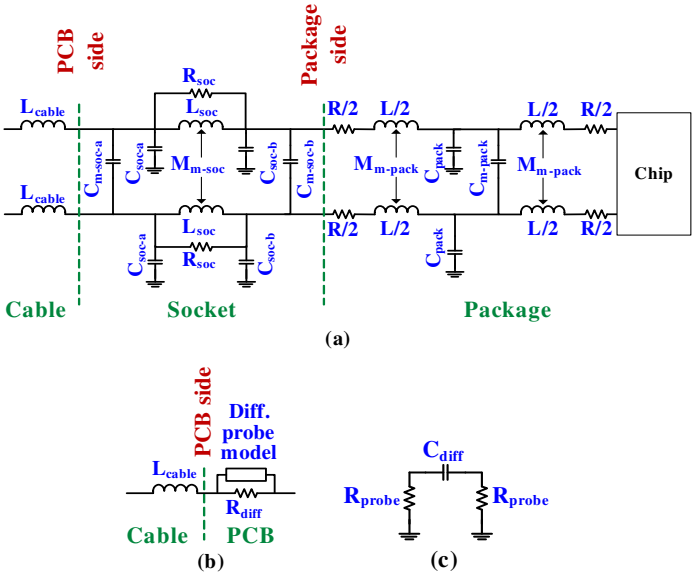


Fig. 3. Equivalent circuit model of (a) two arbitrary signals illustrating the cable, socket and package elements featuring self and mutual inductances/capacitances and (b) extra components that are included in the S-box supply path (namely, R_{diff} and differential probe model which is further detailed in part (c) of the figure).

depends on the amount of instantaneous current to be monitored and the resolution of the oscilloscope. The probe model can be simplified by a small differential capacitance and two resistors connected on each side to the ground.

Of course, neither our measurement artifacts model nor the one in [5] can be considered as completely generic, in the sense that they both need "some prior knowledge" about the target circuit (FPGA / ASIC, type of package to be used, ... etc.) and measurement setup (socket / no socket, PCB, types of probes, ... etc.). Quite naturally, a circuit designer is always advised to incorporate the most precise information in his peripheral model. Yet, we belive the most important shortcomings in security evaluations happen when measurement setups are simply ignored from the simulation environment. In this respect, the quality of our model gradually improves with the accuracy of its component specifications, but even approximated specifications already allow increasing the relevance of simulations significantly, which is the main contribution of this work.

3.3 Improved analysis

We now employ the model illustrated in the previous section in order to improve our analysis and reflect our measurement environment in simulations. First, we show in Figure 4 (a) how close the shape of the simulated traces gets to the measured ones in Figure 2 (b). In particular, the figure now features an expansion

Table 2. List of the elements in the measurement environment model of Figure 3.

Element	Symbol	Description	Value
Cable		supply inductance	688 nH
	L_{cable}	input/output inductance	300 nH
		GND inductance	200 nH
Socket [18]	L_{soc}	lead inductance	1.35 nH
	R_{soc}	parallel lead resistance	600 Ω
	C_{soc-a}	cap. to ground (PCB side)	0.3 pF
	C_{soc-b}	cap. to ground (pack. side)	0.45 pF
	L_{m-soc}	mutual inductance	0.3 nH
	$C_{m-soc-a}$	mutual cap. (PCB side)	0.09 pF
	$C_{m-soc-b}$	mutual cap. (pack. side)	0.09 pF
Pack. [4]	L	inductance	1.2 nH
	R	series resistance	0.28 Ω
	C_{pack}	cap. to ground	0.1 pF
	L_{m-pack}	mutual inductance	1.3 nH
	C_{m-pack}	mutual capacitance	0.2 pF
Diff. Probe	C_{diff}	capacitance	0.7 pF
	R_{probe}	resistance	25 kΩ
	R_{diff}	resistor in S-box VDD path	1 kΩ

in the time span (with respect to preliminary simulations), filtering off the high frequency components and oscillations at a changing frequency. Although neither the time span nor the frequency of oscillations of the improved simulated traces perfectly fit the measured results, the model proves to adequately emulate the measurement specificities to a much better extent. In particular, it directly shows the impact of different components on the linearization of the leakage samples. As can be observed in Figure 4 (b), most of the time samples now appear to be sufficiently linear (i.e. the leakage function is well modeled by the linear stochastic approach based upon the output bits of the S-box).

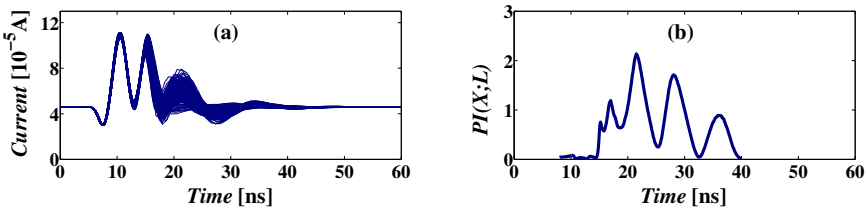


Fig. 4. (a) Simulated current traces for the the DDSLL S-box with measurement artifacts and (b) corresponding perceived information using a linear stochastic model.

As a complement, we investigated the impact of some key elements in the measurement artifacts model on linearizing the leakage function of the simulated traces. For this purpose, we computed the PI in function of the simulated noise

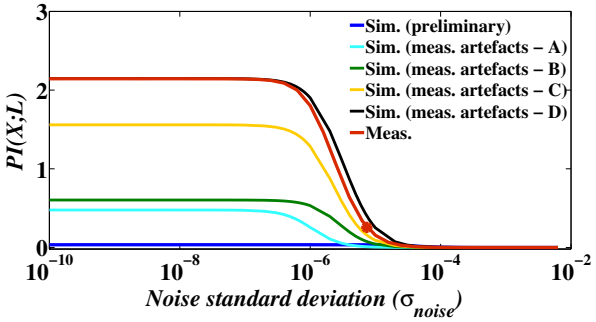


Fig. 5. Perceived information using the linear stochastic approach in function of the noise for preliminary simulated traces, simulated traces with measurement artifacts (accumulated effect of different elements in the model - A, B, C and D) and measured traces of the DDSLL S-box (The PI with real noise is marked with a star (*)).

for leakage variables $L_t^1(X,N)$ and $L_t^1(X,N)$ (represented by solid lines) as well as the PI for the actual noise in our measurements for leakage variable $L_t^2(X,N)$ (represented by a dot). The results of this experiment are reported in Figure 5, in which the curves are given for the single time sample that maximizes the perceived information using the linear stochastic model in all the investigated cases (denoted with letters A to D, depending on the simulation model). The description of these different models used is provided in Table 3.

Table 3. Description of the different models used in Figure 5.

Model	Description
A	1 kΩ + diff. probe
B	1 kΩ + diff. probe + package and socket
C	1 kΩ + diff. probe + package and socket + V_{DD} cable
D	1 kΩ + diff. probe + package and socket + V_{DD} cable + GND cable

As previously observed, it is clear that the leakage function given by preliminary simulations is highly non-linear. Next for case A, only the 1 kΩ resistor in the S-box supply path is modeled without any cable, socket or package models, leading to a first (slight) improvement of the linearity in simulated traces. Adding the socket and package models in case B (including all the mutual effects) without the cable inductances to the 1 kΩ resistor further increases the linearity at low noise levels, and shifts the curve towards the measurement one at real noise. Although not shown here, the mutual effects in the socket and package models did not have a significant impact on linearizing the simulated leakage function. Nevertheless, the use of the mutual inductances and capacitances in the model helped forming the oscillations in the simulated traces. Furthermore, including cable inductances in the supply paths (S-box and buffers) without those of the GND in case C significantly raises the perceived information to 1.6 bits at low

noise levels, and pushed the curve even closer to the measurement one at real noise. This is mainly attributed to the fact that the effect of the cable inductance for the S-box supply is taken into account (with negligible impact of the cable inductance of the buffers supply). Finally, adding the GND cable inductances to the model in case D helped the simulated perceived information to coincide with the measurements with a slight overestimation at noise levels higher than 10^{-6}. Here, the effect of ground bounce could be seen both in the improved simulated traces and linearization of the leakage function to match the measurements.

Clearly, the cable inductances of both the S-box supply and GND are the dominating contributors to the linear dependency between the actual traces and the S-box output bits. They helped forming the low-pass filter that leads to smoothing the shape of the traces by removing the high frequency components, which eventually emphasizes the amplitude shifts and increase the linearity of the leakage function. Overall, the careful choice of the contributing elements in the measurement artifacts model rendered the simulation time overhead negligible since the DSSLL S-box itself (on-chip) has many orders of magnitude more elements than the simple off-chip model (*the simulation time overhead due to our model was less than 1% of the original S-box simulation time*). Also, it is important to note that the Gaussian noise hypothesis is reasonably accurate, as the hybrid leakages with simulated noise correspond well to the actual measurements with real noise (which is confirmed by the position of the dot in Figure 5).

Consequently, our study highlights the importance of having (even approximate) prior knowledge about the measurement specificities of a cryptographic implementation during its early design cycle. This way, the designer can incorporate these artifacts in the simulation environment and make any design changes (if necessary) before having to actually manufacture the final chip.

4 Conclusions

In a previous work by Renauld et al. at CHES 2011, a difference in the linearity of the leakage functions was observed between actual and simulated traces, raising doubts about the relevance of simulations and the possibility of designing circuits with non-linear leakages. This result suggested the improvement of simulation-based side-channel security evaluations as an important open problem, and a preliminary step for the integration of such tools in standard design flows. In this paper, we consequently aimed to improve this situation, both regarding the linearity of simulated traces (that matters for the application of non-profiled side-channel attacks) and regarding the amount of information they provide (that determines the data complexity of profiled side-channel attacks). For this purpose, we first modeled the measurement specificities associated to side-channel attacks and integrated it into SPICE simulations. Then we validated the model by gradually accumulating the different components that we believe important such as the resistor in the supply path, package, socket, and cables, until the simulated and measured traces of the target implementation

provided the same level of linearity and similar perceived information. By doing so, preliminary simulated traces were transformed from highly non-linear to fairly linear traces. Our investigations confirmed the need to incorporate the key physical artifacts in the simulation environment to obtain an accurate assessment of countermeasures against side-channel attacks at the design stage. Besides, our study also raises the question whether it is possible to design an implementation with highly non-linear leakages in light of the linearization effect that the physical artifacts inevitably have on the final outcome.

Acknowledgements. This work has been funded in parts by the European Commission through the ERC project 280141 (acronym CRASH), the European ISEC action grant HOME/2010/ISEC/AG/INT-011 B-CCENTRE project, and the Walloon region WIST program project MIPSs. F.-X. Standaert is an associate researcher of the Belgian Fund for Scientific Research (FNRS-F.R.S.).

References

1. Eric Brier, Christophe Clavier, and Francis Olivier. Correlation power analysis with a leakage model. In *Cryptographic Hardware and Embedded Systems - CHES 2004: 6th International Workshop Cambridge, MA, USA, August 11-13, 2004. Proceedings*, volume 3156 of *Lecture Notes in Computer Science*, pages 16–29. Springer, 2004.
2. Suresh Chari, Josyula R. Rao, and Pankaj Rohatgi. Template attacks. In *Proceedings of Cryptographic Hardware and Embedded Systems, CHES*, pages 13–28, 2002.
3. Ilham Hassoune, François Macé, Denis Flandre, and Jean-Didier Legat. Dynamic differential self-timed logic families for robust and low-power security ICs. *Integration*, 40(3):355–364, 2007.
4. Texas instruments. AN-1205 electrical performance of packages. Application report, May 2004.
5. K. Iokibe, T. Amano, K. Okamoto, and Y. Toyota. Equivalent circuit modeling of cryptographic integrated circuit for information security design. *Electromagnetic Compatibility, IEEE Transactions on*, 55(3):581–588, 2013.
6. K. Iokibe, R. Higashi, T. Tsuda, K. Ichikawa, K. Nakamura, Y. Toyota, and R. Koga. Modeling of microcontroller with multiple power supply pins for conducted emi simulations. In *Advanced Packaging and Systems Symposium, 2008. EDAPS 2008. Electrical Design of*, pages 135–138, 2008.
7. Dina Kamel, Mathieu Renauld, David Bol, François-Xavier Standaert, and Denis Flandre. Analysis of dynamic differential swing limited logic for low-power secure applications. *Journal of Low Power Electronics and Applications*, 1(2):98–126, 2012.
8. François Macé, François-Xavier Standaert, and Jean-Jacques Quisquater. Information theoretic evaluation of side-channel resistant logic styles. In Pascal Paillier and Ingrid Verbauwhede, editors, *CHES*, volume 4727 of *Lecture Notes in Computer Science*, pages 427–442. Springer, 2007.
9. Stefan Mangard, Elisabeth Oswald, and Thomas Popp. *Power analysis attacks - revealing the secrets of smart cards*. Springer, 2007.

10. Stefan Mangard, Thomas Popp, and Berndt M. Gammel. Side-channel leakage of masked cmos gates. In Alfred Menezes, editor, *CT-RSA*, volume 3376 of *Lecture Notes in Computer Science*, pages 351–365. Springer, 2005.
11. K. NAKAMURA. Emc macro-model (leccs-core) for multiple power-supply pin lsi. *Proc. EMC'04, Sendai, June*, 2004.
12. Francesco Regazzoni, Alessandro Cevrero, François-Xavier Standaert, Stéphane Badel, Theo Kluter, Philip Brisk, Yusuf Leblebici, and Paolo Ienne. A design flow and evaluation framework for dpa-resistant instruction set extensions. In Christophe Clavier and Kris Gaj, editors, *CHES*, volume 5747 of *Lecture Notes in Computer Science*, pages 205–219. Springer, 2009.
13. Francesco Regazzoni, Thomas Eisenbarth, Axel Poschmann, Johann Großschädl, Frank K. Gürkaynak, Marco Macchetti, Zeynep Toprak Deniz, Laura Pozzi, Christof Paar, Yusuf Leblebici, and Paolo Ienne. Evaluating resistance of mcml technology to power analysis attacks using a simulation-based methodology. *Transactions on Computational Science*, 4:230–243, 2009.
14. Mathieu Renauld, Dina Kamel, François-Xavier Standaert, and Denis Flandre. Information theoretic and security analysis of a 65-nanometer DDSLL AES S-Box. In *Proceedings of Cryptographic Hardware and Embedded Systems, CHES*, pages 223–239, 2011.
15. Mathieu Renauld, François-Xavier Standaert, Nicolas Veyrat-Charvillon, Dina Kamel, and Denis Flandre. A formal study of power variability issues and side-channel attacks for nanoscale devices. In *EUROCRYPT*, pages 109–128, 2011.
16. Werner Schindler, Kerstin Lemke, and Christof Paar. A stochastic model for differential side channel cryptanalysis. In *Proceedings of Cryptographic Hardware and Embedded Systems, CHES, Springer, LNCS 3659*, pages 30–46. Springer, 2005.
17. François-Xavier Standaert, Tal G. Malkin, and Moti Yung. A unified framework for the analysis of side-channel key recovery attacks. In *Proceedings of the 28th Annual International Conference on Advances in Cryptology: the Theory and Applications of Cryptographic Techniques*, EUROCRYPT, pages 443–461, Berlin, Heidelberg, 2009. Springer-Verlag.
18. Giga test labs. ARIES electronics 64 pin QFP (0.55mm) test socket, Electrical characterisation 0.05 - 3.05 GHz. Characterisation report, Oct. 1997.
19. Kris Tiri and Ingrid Verbauwhede. Simulation models for side-channel information leaks. In William H. Joyner Jr., Grant Martin, and Andrew B. Kahng, editors, *DAC*, pages 228–233. ACM, 2005.
20. Kris Tiri and Ingrid Verbauwhede. A digital design flow for secure integrated circuits. *IEEE Trans. on CAD of Integrated Circuits and Systems*, 25(7):1197–1208, 2006.
21. Nicolas Veyrat-Charvillon and Francçois Xavier Standaert. Generic side-channel distinguishers: Improvements and limitations. In *Advances in Cryptology - CRYPTO 2011 - 31st Annual Cryptology Conference*, volume 6841 of *Lecture Notes in Computer Science*, page 348. Springer, 2011.
22. Carolyn Whitnall, Elisabeth Oswald, and François-Xavier Standaert. The myth of generic DPA... and the magic of learning. Cryptology ePrint Archive, Report 2012/256, 2012. `http://eprint.iacr.org/`.

Better Provability through Computer Architecture

Timothy Sherwood
UCSB, ArchLab, http://www.cs.ucsb.edu/~arch/

Invited keynote at PROOFS — August 24, 2013.

Abstract

Computer performance has doubled many times over during the past 40 years, but the very techniques used to achieve these performance gains have made it increasingly difficult to build hardware/software systems with cross-cutting properties such as determinism, real-time guarantees, and (perhaps most critically) security. As we move towards increasingly complex chips, with more and more hidden state (e.g., predictors, caches, modes), these properties are only becoming harder to realize. This trend significantly impedes progress in the development of our most safety-critical embedded systems such as those found in medical, avionic, and automotive systems. What if we started from scratch? What if we built processors from the ground up with an eye towards both the new complex reality in which we live and the radical improvements in automated formal methods we now have access to? How much better could we do? Dr. Sherwood will discuss his and his collaborators efforts to explore this question through the creation of a complete prototype system including: a new instruction set architecture, a synthesizable hardware designs, a new programming language and compiler, a custom micro-kernel generator, interfaces to existing I/O devices, and ultimately non-trivial trustworthy applications.

Formal Design of Composite Physically Unclonable Function

Durga Prasad Sahoo, Debdeep Mukhopadhyay, and Rajat Subhra Chakraborty

Computer Science and Engineering
Indian Institute of Technology, Kharagpur, India
{dpsahoo,debdeep,rschakraborty}@cse.iitkgp.ernet.in

Abstract. Physically Unclonable Function (PUF) in silicon is the embodiment of an instance-specific challenge-response mapping, that exploits random manufacturing process variation in an integrated circuit (IC) to determine the mapping. PUF designs proposed in the recent literature vary widely in diverse characteristics such as hardware resource requirement, *reliability*, *entropy*, and robustness against mathematical cloning. Most of the standalone PUF designs suffer from either poor performance profile or huge resource-overhead. This work presents a PUF design paradigm, termed as *PUF Synthesis*, that exploits the smaller PUFs (both *strong* and *weak*) as design building blocks to define a composite PUF having large challenge-space and good performance profile. A formal framework for PUF synthesis has also been developed to guide the composition in systematic fashion. The notion of PUF synthesis has been validated by the implementation of a composite PUF that combines two widely studied PUFs, Arbiter PUF (APUF) and Ring Oscillator PUF (ROPUF), and inherits the desirable characteristics of both. Resource requirement of this target design with 60-bit challenge is lesser than a standalone 10-bit ROPUF, while its robustness against model building attack is much superior compared to APUF. Implementation of the proposed design on Altera Cyclone-III Field Programmable Gate Array (FPGA) shows 47% uniqueness and 91% reliability on average.

Keywords: Physically Unclonable Function, PUF synthesis, composite PUF, optimized composition.

1 Introduction

Physically Unclonable Function [1] is a well-defined, instance-specific and (ideally) consistent mapping from a set of input values (*challenges*) to a set of output values (*responses*). Although every logical instance of a PUF is identical, its unclonability is manifested when it is implemented in some physical form. For example, oscillation frequencies of two identical ring oscillators mapped on two different FPGAs of the same family and from the same manufacturer are non-identical, and the difference in these frequencies can be encoded to provide the response for a PUF. Apart from its physical unclonability, a PUF should be

hard to simulate, emulate, or predict, but easy to evaluate. Generally, the unclonability is the result of unique and uncontrollable variations in every physical system. Silicon PUFs exploit variation in manufacturing across different dies, wafers, and process to generate (ideally) unique *challenge-response* mapping for each IC (Integrated Circuit). Over the years, versatile applications of PUFs have been proposed, such as: device authentication and identification [2–4], random number generation [5] and intellectual property protection [6, 7]. Research in this field is important, as PUFs introduce one more root of trust for designing secure hardware.

The silicon manufacturing process variation is the major source of static noise in silicon device and that is exploited in silicon PUF design. From a circuit design perspective process variations can be divided into two major groups: *die-to-die* variations and *within-die* variations [8]. The die-to-die variations have a variation radius larger than the die size including within wafer, wafer-to-wafer, lot-to-lot and fab-to-fab variations. These variations affect all the circuits within the die equally. Within-die variations, on the other hand, refer to the variations that occur between various circuit elements of the same die. They can be grouped into *systematic* and *random* variations. The radius of systematic variation is in the order of few millimeters. The spatially correlated systematic variations happen due to variation in design layout or manufacturing equipment. Finally, random variations are non-systematic and unpredictable in nature and including random variation in device length, discrete doping fluctuations and oxide thickness variations. The radius of this variation is comparable to the sizes of individual devices, so each device can vary independently.

Research on PUFs started with the seminal work of Lofstrom et al. [9] that proposed the exploitation of mismatch in silicon devices for identification of ICs. In 2001, Pappu et al. [10] presented the concept of *physical one-way function*, which subsequently led to the idea of PUF [11]. In the last decade, several PUF designs have been proposed by researchers. Each of these designs come with a different way of measuring the effect of device-level process variation, and its quantization to binary response. A common strategy is to exploit delay variation in CMOS logic components, in the so–called *Delay PUFs*. The most well-known delay PUFs are Arbiter [2], Ring Oscillator [12], Butterfly [6], Bistable Ring [13], Loop [14] and FPGA specific Anderson [15] PUF. Memory based PUFs (e.g. SRAM [7] and Flip-Flop [16]), use variation in power–up values of memory elements, and this type of PUFs are also known as *challengeless PUFs*. ClockPUF [17] and ScanPUF [18] are two recently proposed interesting PUF designs. ClockPUF uses pairwise difference in sink latencies in on-chip clock network (*clock skews*), and ScanPUF uses the path delay variation between scan flip-flops in *scan chain*, a common on-chip structure used for improving design testability, to generate random bit strings.

A very important question about the usability and security of PUFs is currently under intense research scrutiny: can PUFs be mathematically cloned? Several works [19–21] on PUF modeling using statistical and machine learning [22] techniques like *Support Vector Machine* (SVM), *Artificial Neural Net-*

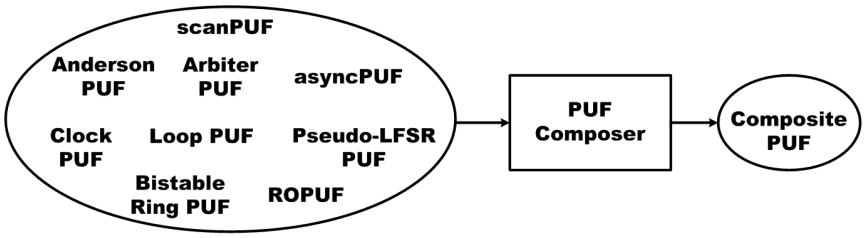


Fig. 1: PUF Synthesis overview: Composer selects subset of PUF blocks and generate a new design obeying composition rules that defines the topology of composite PUF.

work (ANN), and *Logistic Regression* (LR), have been recently proposed. PUFs exhibit different levels of robustness to model building attacks, for example, arbiter PUF and its variants [2, 23] have been shown to be the most vulnerable candidates for this attack. However, there is no conclusive answer to the question of which among the existing PUF designs is the best. This is because of the fact that they vary widely with respect to diverse metrics of suitability: some take less hardware resource to implement, some exhibit more randomness, while some others provide more stability to temperature and voltage fluctuations.

It could be an obvious conclusion form existing literature of silicon PUF is that most of the standalone PUFs are not suitable to be used as root-of-trust in hardware security application due to either their poor performance quality or significant resource-overhead. This observation motivates us to design PUFs that exploit smaller version of existing PUFs, both weak and strong [24], as design building blocks. We term this PUF design paradigm as *PUF Synthesis* (see Fig. 1) and resultant design as *composite PUF*.

The main contribution of our work is: Design of a formal framework to guide the *PUF Synthesis* process that use smaller PUFs as building blocks to build a new PUF. Advantage of smaller PUF blocks is that they are confined in a small chip area and eliminate the negative effects of systematic process variation in PUF design. If the PUF circuit needs large area of IC for placement, then it includes many regions with uncorrelated systematic process variation that dominates useful random variation. Thus, the PUF synthesis could be very useful for large PUF design.

The rest of the paper is organized as follows: Section 2 introduces the notion of PUF synthesis and explains a formal framework for designing new PUF using $PUFSynthesis$ approach. Section 3 discusses usefulness of the notion of PUF synthesis with a motivating example and its implementation results. Finally, Section 4 concludes the paper and explains the future work directions.

2 PUF Synthesis

The *PUF synthesis* is a PUF design paradigm that exploits smaller PUFs (both *weak* and *strong* PUFs) blocks of diverse types as a building blocks in the design

of new PUF with large challenge space and good performance profile. New PUF which emerges as result of the synthesis process could be termed as *Composite PUF*, and hence synthesis process might also be called as *PUF Composition*. We use these terms interchangeably.

Certain classes of PUFs, for example Ring Oscillator PUF, have an acceptable performance profile, but not feasible for large design, a design with large challenge-space, due to huge resource demand. Similarly, too many light-weight PUF designs, e.g. Arbiter PUF, have poor performance profile with respect to uniqueness, randomness, reliability, and/or unpredictability. The PUF composition is a new design paradigm as trade-off of above two design problems. In addition, we would see that a composite PUF inherits the desirable qualities of its component PUFs to a great extent.

To exploit a PUF in composition, it is necessary to have information about its physical implementation like, resource requirement, performance quality. Otherwise, it is difficult to evaluate the composition quality. For the clarity of explanation, we use term $PUFType_i$ as defined follows.

Definition 1 *$PUFType_i =< ID = i, Plat, I_i, O_i, R_i, U_i, Q_i, P_i, A_i, B_i >$ is the profile of physically implemented PUF having index i in the library of PUFs where,*

- *ID is an integer used to retrieve details of a PUFType.*
- *$Plat$ is the specification of platform used for the implementation, e.g. Altera Cyclone-III. This is important to consider because performance of same PUF design greatly depends on platform used for its physical embodiment.*
- *I_i and O_i are the challenge and response size of PUF in bits.*
- *R_i specifies the amount of resource used in design. For instance, in FPGA platform this is the number of LUTs and FFs.*
- *U_i, Q_i, P_i, A_i, and B_i are the values of performance metrics uniqueness, uniformity, reliability, auto-correlation, and bit-aliasing in percentage, respectively.*

We use following mathematical definition of PUF as a basis in formal definition of PUF composition.

Definition 2 *A silicon Physically Unclonable Function is a mapping*

$$\gamma : \{0,1\}^n \longrightarrow \{0,1\}^k$$

, where the output k-bit words are unambiguously identified by both the n challenge bits and the unclonable, unpredictable (but repeatable) instance specific system behavior.

Each node in composite PUF circuit is a standalone PUF and connection among PUF nodes define the topology of composition. For a given set of PUF nodes, quality of composition is subject to topology. Following formal definition details a set of constructive rules to guide the composition in a systematic way.

Definition 3 *A composite PUF (ζ) over set of PUFs $\Gamma = \{\gamma_1, \gamma_2, ..., \gamma_m\}$ is a PUF circuit that is defined by recursively applying following rules:*

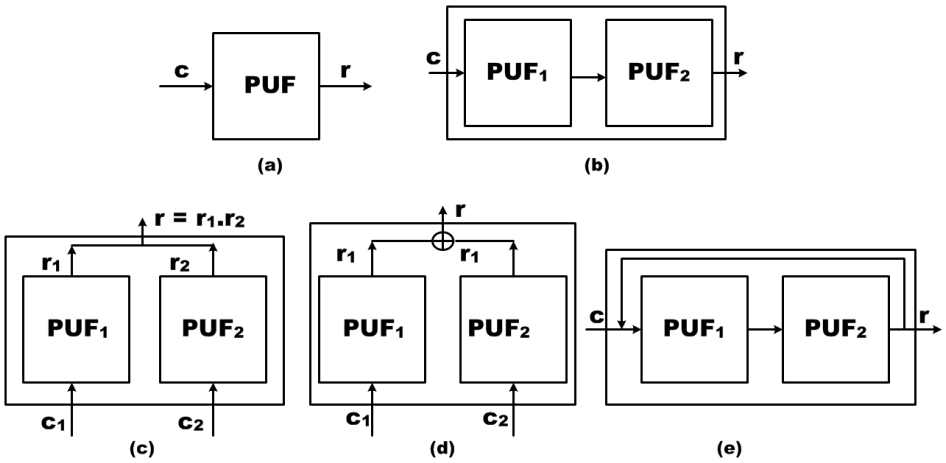


Fig. 2: Overview of Composition Rules: (a) Basic PUF Block. (b) $r = (PUF_2 \lhd PUF_1)(c)$. (c) $r = (PUF_1 \parallel PUF_2)(c_1, c_2)$. (d) $r = (PUF_1 \oplus PUF_2)(c_1, c_2)$ (e) $r = (PUF_2 \bowtie PUF_1)(c)$

a. $\gamma_i : C_i \longrightarrow R_i$, *where* $C_i, R_i \subseteq \{0,1\}^+$ *and* $\gamma_i \in \Gamma$.
b. $(\gamma_i \lhd \gamma_j)(x) = \gamma_i(\gamma_j(x))$, *where* $x \in C_j$.
c. $(\gamma_i \parallel \gamma_j)(x, y) = \gamma_i(x) \cdot \gamma_j(y))$, *where* $x \in C_i$, $y \in C_j$, *and* $'.'$ *is binary strings concatenation operator.*
d. $(\gamma_i \oplus \gamma_j)(x, y) = \gamma_i(x) \oplus \gamma_j(y)$, *where* $x \in C_i$, $y \in C_j$, $\oplus$ *is bit-wise exclusive-OR operator.*
e. $(\gamma_i \bowtie \gamma_j)(x) = \gamma_j(\gamma_i(\gamma_j(x)))$, *where* $x \in C_j$
f. $\gamma_i(perm(x))$ *and* $perm(\gamma_i(x))$ *are PUFs with input and output permutation network* $perm(y)$ *respectively, and* $y \in \{0,1\}^*$ *and* $x \in C_i$.

Now, it is important to explain why these composition operators have been selected to design a composite PUF. Let X and Y be two random variables regarding the output distributions of PUFs γ_i and γ_j.

Lemma 1 ([25]) *Let* X *and* Y *be two independent random variables with entropy* $H(X)$ *and* $H(Y)$, *respectively. Then,* $H(X, Y) = H(X) + H(Y)$.

From Lemma 1, it is evident that entropy of $(\gamma_i \parallel \gamma_j)$ is the sum of their individual entropies. This operation also enhances both uniqueness and uniformity when one component PUF has greater value than ideal value (50%) and other has lesser than ideal one. It is clear that this operation is most effective when it combines two weak PUFs with above mentioned features.

Lemma 2 *Let* X *and* Y *be two Bernoulli random variables with probability* p *and* q, *respectively. Then, random variable* $Z = X \oplus Y$ *also follows Bernoulli distribution with probability* $p + q - 2pq$. *It implies that if any of the component distributions is uniform, then* Z *is also uniform.*

The claim of Lemma 2, is well-known. Let X be uniform and $p = 0.5$, then $Pr(Z = 1) = 0.5 + q - 2 \times 0.5 \times q = 0.5$. In general, if the distribution of at least one input is uniform, then the output of the XOR gate is also uniform. This operation is used in cryptography to a great extent. So, this is another way to enhance the entropy of composite PUF. Now, it seems that one PUF with uniform distribution is equivalent to its XORing with multiple PUFs when only output with uniform distribution is concerned. So, why does it need to XORing them when one PUF is enough? Like other compositions, the XOR function between the outputs of two PUF instances enhances the challenge space at less hardware overhead, and improves unpredictability.

Lemma 3 ([25]) *Let X and Y be two random variables. If $Y = f(X)$ is a deterministic function of X, then $H(Y) \leq H(X)$ with equality if and only if $f(.)$ is one-to-one.*

Proof. Joint entropy of X and Y is $H(X, f(X)) = H(X) + H(f(X) \mid X) = H(X)$ since, $H(f(X) \mid X) = 0$ because f is deterministic function of X and X is known. Again, by chain rule we have: $H(X, f(X)) = H(f(X)) + H(X \mid f(X)) \geq H(f(X))$ since, $H(X \mid f(X)) \geq 0$ with equality if and only if f is one-to-one. Thus, $H(Y) \leq H(X)$.

Lemma 3 is applicable to composition $(\gamma_j \lhd \gamma_i)$ when random variable X and Y represent output distributions of γ_i and γ_j, respectively, and $Y = \gamma_j(X)$. Furthermore, every physical instance of a PUF is (semi-) deterministic, but not one-to-one while each instance is different from others. The Lemma 3 implies that the composition $(\gamma_j \lhd \gamma_i)$ does not improve the entropy, but propagates entropy of γ_i to the output of composite PUF. Nevertheless, this composition operator is used to enhance the unpredictability. The adversary now needs to learn value of $m + n$ parameters combinedly to reverse engineer the composite PUF where m and n are number of parameters of γ_i and γ_j that determine their mappings, respectively.

It is worth mentioning that some circuits obtained by applying Definition 3 of composite PUF are ill-formed. For instance, this can happen if number of outputs of γ_i is different from number of inputs of γ_j in the sequential composition $\gamma_j \lhd \gamma_i$. Similarly, this is true for $\gamma_i \oplus \gamma_j$ when number of outputs of γ_i and γ_j are different. To eliminate all such ill-formed compositions, we define a type system $\tau : \Gamma \to \mathbb{N} \times \mathbb{N}$ i.e., $\forall \gamma \in \Gamma, \tau(\gamma) = (n, m)$ where n and m are number of inputs and outputs of γ, respectively. Formal definition of well-formed composite PUF is given below.

Definition 4 (Well-formed composite PUF) *Let ζ be a composite PUF having n-input and m-output – written as $\zeta : n \otimes m$– and defined over Γ. The PUF ζ is said to be well-formed if and only if each of its sub-circuit obeys the rules of type system given below. Otherwise, ζ is said to be ill-formed.*

i) $\frac{\tau(\gamma)=(n,m)}{\gamma:n\otimes m}\gamma \in \Gamma$ *ii)* $\frac{\gamma_i:n_i\otimes m_i,\gamma_j:n_j\otimes m_j}{\gamma_i \| \gamma_j:n_i+n_j\otimes m_i+m_j}$ *iii)* $\frac{\gamma_i:n_i\otimes m_i,\gamma_j:n_j\otimes m_j,n_i=m_j}{\gamma_i \lhd \gamma_j:n_j\otimes m_i}$

iv) $\frac{\gamma_i:n_i\otimes m_i,\gamma_j:n_j\otimes m_j,m_i=m_j}{\gamma_i \oplus \gamma_j:n_i+n_j\otimes m_i}$ *v)* $\frac{\gamma_i:n_i\otimes m_i,\gamma_j:n_j\otimes m_j,n_i=m_j,n_j=m_i}{\gamma_i \bowtie \gamma_j:n_j\otimes m_i}$

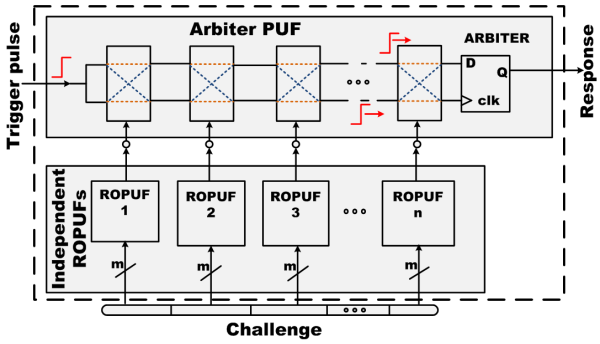


Fig. 3: Motivational Example: A composite PUF exploiting ROPUF and APUF as design building blocks.

3 Motivating Example

The objective of this section is to show the usefulness of composite PUF design with an example.

3.1 Composite PUF Instance

The architectural description of the example composite PUF is shown in Fig. 3, using Ring Oscillator and Arbiter PUFs. It consists of one arbiter PUF and set of independent ROPUFs. Let, this PUF is denoted by $\chi_{n,m}$. According to Definition 3,

$$\begin{aligned}\chi_{n,m} &= \gamma_{n+1}((\gamma_1 \parallel \gamma_2 \parallel \gamma_3 \parallel \cdots \parallel \gamma_{n-1} \parallel \gamma_n)(c_1, c_2, c_3, \ldots, c_{n-1}, c_n)) \\ &= \gamma_{n+1}((\gamma_1(c_1) \cdot \gamma_2(c_2) \cdot \gamma_3(c_3) \cdot \cdots \cdot \gamma_{n-1}(c_{n-1}) \cdot \gamma_n(c_n))\end{aligned}$$

where γ_{n+1} is an n-bit Arbiter PUF, and γ_i, $1 \leq i \leq n$, are m-bit ROPUFs. Following formal definition illustrates how it works. The $\chi_{n,m}$ is a 5-tuple $(\Delta, \Sigma, \Lambda, \mathbb{B}, \Psi)$, where

- $\Delta = \gamma_{n+1}$ is a n-bit arbiter PUF
- $\Sigma = \{\gamma_1, \gamma_2, \gamma_3, ..., \gamma_n\}$ is a set of independent m-bit ring oscillator PUF
- Λ is a set of nm-bit challenges and $|\Lambda| = 2^{nm}$
- $c \in \Lambda$ consists of n m-bit components i.e. $c = (c_1, c_2, ..., c_n)$ and $|c_i| = m,\ 1 \leq i \leq n$
- $\mathbb{B} = \{0, 1\}$
- $\Psi : \Lambda \rightarrow \mathbb{B}$ is the characteristic mapping of a composite physically unclonable function, and,
- $\forall c \in \Lambda, \Psi(c) = \Delta(\gamma_1(c_1) \cdot \gamma_2(c_2) \cdot \gamma_3(c_3) \cdot \cdots \cdot \gamma_n(c_n)) \in \mathbb{B}$ and $\gamma_i(c_i) \in \mathbb{B}$.

The applied challenge is divided into n equal size sub-challenge, each of size m. Each part of challenge is applied to n independent ROPUFs. Responses of the

ROPUFs together form the (internal) challenge for the APUF that eventually generates final response. Responses of the ROPUFs control the path switching activities of APUF.

This design is superior than APUF and ROPUF in three aspects: i) shows better modeling robustness than APUF, ii) consumes less resource than ROPUF, and iii) has better performance profile than both ROPUF and APUF.

Note that apart from the composite behavior of $\chi_{n,m}$, it reduces to an APUF or an ROPUF, for special values of n and m. For example, if $n = 1$, then it works almost similar as a ROPUF, while for $m = 1$ it works as an APUF. Also, note that the components c_i might or might not be in order in the total challenge c. If they are in order, $c = c_1 \parallel c_2 \parallel \cdots \parallel c_n$. Otherwise, an optional bit permutation network might be placed at the input of the composite PUF to permute the applied external challenge and then apply the permuted challenge to the input of the ROPUFs. This helps to increase the amount of randomness.

3.2 Implementation Results

Experimental Setup. The implementation of 60-bit target composite PUF (see Fig. 3) was made using 11 Altera Cyclone-III EP3C80F780I7 FPGAs. The 60-bit target composite PUF instance consists of one 15-bit arbiter PUF (n=15) and 15 4-bit ROPUFs (m=4). The 60-bit input challenge was segmented into 15 segments, each of size 4-bit that was applied to 15 ROPUFs as challenge. Standard Altera CAD tool was used for designing and generating programming file.

Altera Cyclone FPGA consists of LABs (Logic Array Block) that are arranged in 2D matrix, and few columns are reserved for BRAM (Block RAM) blocks. Each LAB is a collection of 16 LEs (Logic Elements) that composes of one 4-input LUT (Look Up Table) and one programmable register. One LAB was used for designing a ring oscillator that contains one NAND gate and 15 delay elements in a loop. Altera's *lcell* HDL primitive was used for each delay element that requires one LUT in LE. In addition, NIOS-II soft processor, NIOS-II IDE, and JTAG interface were employed for CRP (Challenge Response Pair) exchange between FPGA and host computer.

Table 1 depicts the resource requirement for three PUF design: 60-bit APUF, 10-bit ROPUF, and 60-bit composite PUF. Resource requirement of the target

Table 1: Comparison of Hardware Resource Requirements of Proposed PUF with ROPUF and APUF

PUF Type	**#Comb. Logic(4LUT)**	**#Register**
60-bit APUF	121	2
10-bit ROPUF*	17828	32
60-bit Proposed PUF	5192	497

*Design of 60-bit ROPUF was infeasible on the FPGA we worked with.

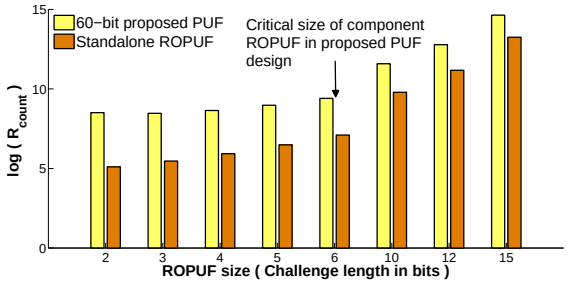


Fig. 4: Resource required for 60–bit composite PUF and its critical dependency on component ROPUF size (m), compared with standalone m-bit ROPUFs. R_{count} is resource count that includes 4-input LUT(s) and register(s) on an Altera Cyclone-III FPGA.

composite design depends on the size of the component ROPUFs. Small-sized ROPUFs, that are confined in a relatively small chip area with spatially correlated process parameters, could avoid the negative effects of systematic process variation. It allows designer to select any two distinct ROs for calculation of response for a challenge without thinking whether they have been placed closed to each other. In addition, size of ROPUF depends on size of MUXes, counters, comparator, and number of ROs and RO size. For small m value, composite PUF design requires relative more component ROPUFs and that results huge resource overhead due to MUXes, counters and comparator of each component ROPUFs. Fig. 4 depicts how the total resource requirement of target composite PUF (60-bit challenge) varies as a function of component ROPUF size, when implemented on an Altera Cyclone-III FPGA. It also reveals that if the challenge size of the component ROPUFs is less/equal than six bits, then amount of resource required for 60-bit proposed PUF is lesser than that required for a standalone 10-bit ROPUF. From our experiments, component ROPUF challenge size not exceeding 6-bit (i.e. $m \leq 6$) seems a reasonable choice.

Results. Performance quality of target composite PUF design have been evaluated based on standard metrics [26]: *Uniqueness*, *Reliability*, *Uniformity*, *Bit Aliasing*, and *Autocorrelation*. Uniqueness is a measure of ability to identify PUF instances uniquely and it is estimated in term of inter *Hamming Distance* (HD). PUF instance should have ability to repeatedly generate same response for given challenge – called as Reliable PUF. Reliability is expressed in term of intra Hamming Distance. Uniformity measures the 0's and 1's distribution in PUF's binary response and it should be uniform for ideal PUF. Bit-aliasing happens when different chips produce nearly identical PUF responses, which is undesirable. In [26], bit-aliasing of l-th bit of a n-bit PUF response over k chips is defined by (1),

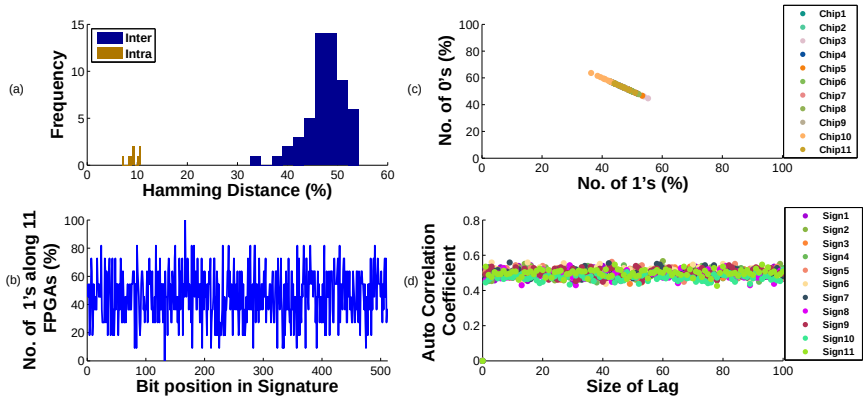


Fig. 5: PUF Analysis. (a) Uniqueness and Reliability in terms of inter and intra HD, respectively. (b) Bit Aliasing along different FPGAs. (c) Uniformity of 0's and 1's. It's a measure of randomness. (d) Autocorrelation Test to measure dependency among bits of a signature.

$$\beta_l = \frac{1}{k}\sum_{i=1}^{k} r_{i,l} \times 100\% \tag{1}$$

where $r_{i,l}$ is the l-th binary bit of a response from a chip i. Ideal value for β_l should be 50%. Here, we define minimum, maximum, and average bit-aliasing of n-bit PUF signature by (4), (3), and (2), respectively.

$$\beta_{avg} = \frac{1}{n}\sum_{l=1}^{n} |\beta_l - 50| \tag{2}$$

$$\beta_{max} = \max_{1\le l\le n} |\beta_l - 50| \tag{3}$$

$$\beta_{min} = \min_{1\le l\le n} |\beta_l - 50| \tag{4}$$

The value of β_{avg} lies in $[0, 50]$. Value of $\beta_{avg} = 50$ implies all bits of signature are fully aliased, whereas $\beta = 0$ presents no aliasing. Autocorrelation test is used to estimate dependency between the bits of a given signature and to determine the existent of periodic behavior. If bits are highly correlated then prediction of response for an unknown challenge become easy. It can be defined by (5), where x is the n-bit signature being observed:

$$\rho_{xx}(j) = \frac{1}{n}\sum_{i=1}^{n} x_i \oplus x_{i-j} \tag{5}$$

and $\rho_{xx}(j)$ is auto correlation coefficient with lag j. This value tends towards 0.5 for uncorrelated bit-strings and toward 0 or 1 for correlated bit-strings. Fig. 5(d)

Table 2: Comparison of Performance Metrics of Composite PUF with ROPUF and APUF[†]

Metrics	Ideal Value	Composite PUF				APUF	ROPUF
		Min.	Max.	Avg.	Std. Div.	Avg.	Avg.
Uniqueness(%)	50	32.42	54.30	47.57	4.06	37.40	31.34
Reliability(%)	100	89.26	92.97	90.70	1.12	100	99.85
Uniformity(%)	50	36.33	55.27	47	3.27	70.63	51.56
Bit-aliasing[0,50]	0	4.55	50	14.95	10.26	30.90	28.20
Autocorrelation Coefficient[0,1]	0.5	0.43	0.57	0.50	0.23	0.42	0.49

[†]Challenge size of composite PUF, APUF, and ROPUF are 60, 60, and 10 bits, respectively.

shows the scattering of correlation coefficient values of 11 different 512-bit signatures and gives a clear indication of the mutual inter-independence of the bits in a signature.

Table 2 provides every details of quality metrics and compares the performance of composite design with standalone APUF and ROPUF. Fig. 5 is the visual interpretation of quantities mentioned in tabular form. Reliability of composite design is not good, but satisfactory. One reason could be the use of NIOS II soft processor core that is a heavy resource-consume component. It is obvious that this reported reliability would further be degraded in consideration of temperature and voltage variation. In addition, reliability could be improved by switching off the components no longer in use. We did not perform the reliability test with temperature and voltage variation as a part of this work.

In the next subsection, we describe the robustness of the target composite PUF design to machine learning based modeling attacks, as compared to standalone ROPUFs and APUFs.

3.3 Robustness against Modeling Attacks

Objective of modeling attack is to build a numerical approximator for a PUF instance based on known set of its CRPs. Machine Learning based modeling attacks on PUF was discussed in [20]. Success in modeling attacks does not imply its physical clonability, rather it is an implication of response prediction for unknown challenge without knowing circuit details. Here, we discuss modeling robustness of composite PUF with respect to SVM and ANN machine learning approaches. Our implementations of SVM and ANN are based on the standard Python package *scikit-learn* [27] and the MATLAB *Neural Network Toolbox* [28], respectively. To build machine learning models, we used 11 different datasets, each of size $25,000$ CRPs and obtained from 11 PUF instances. The derived models were tested on 5000 unseen challenges for the proposed composite PUF and APUFs, while for ROPUF the testing set consisted of 400 CRPs. For a given training set, we built large number of models by tuning meta-parameters of learning algorithm, for example kernel type for SVM and type of activation function for ANN. The box-plot in Fig. 6 shows the prediction accuracy of all

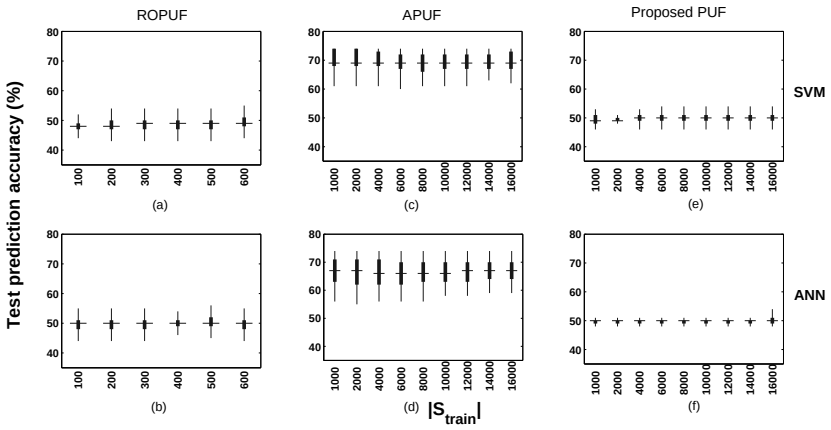


Fig. 6: Test prediction accuracy of PUF models. It shows the testing accuracy of models built from known set of CRPs of ROPUF, APUF and composite PUF using SVM and ANN. The X-axis presents number of CRPs, $|S_{train}|$, used in training phase. The Y-axis presents testing accuracy of the derived model. Box-plot is used to show the accuracy of a population of models.

models for different size of training set ($|S_{train}|$). The X-axis shows the different value of $|S_{train}|$ and the Y-axis shows the prediction accuracy of models that were trained with training set of size $|S_{train}|$. From Fig. 6, it is inferred that prediction accuracy of target composite PUF design is close to 50% (equivalent to random prediction) and it does not improve with increase in training set size. We verified this behavior of target composite PUF upto training set size $90,000$ CRPs. One reason of this modeling robustness could be the use of sequential composition ($\lhd$) rule. So, apart from its good performance metrics (as reported in the previous subsection), the composite PUF design also has good modeling robustness.

4 Conclusion

This work introduces a concept of PUF synthesis – also called as PUF composition – that shows how smaller PUFs of diverse types could be used to design a qualitative large PUF. One important problem associated with this work is the design of optimal composition. It needs to explore large space of composite PUFs with diverse kinds of topology, and variety of PUF blocks. Our future work would be directed towards the implementation of local search method and genetic programming for designing optimal composite PUF, and development of a CAD flow to automate the PUF synthesis methodology.

References

1. Maes, R., and Verbauwhede, I. Physically Unclonable Functions: A Study on the State of the Art and Future Research Directions. In *Towards Hardware-Intrinsic Security*, A.-R. Sadeghi and D. Naccache, Eds., Information Security and Cryptography. Springer, Berlin Heidelberg, 2010, pp. 3–37.
2. Gassend, B., Lim, D., Clarke, D., van Dijk, M., and Devadas, S. Identification and authentication of integrated circuits: Research Articles. *Concurrency and Computation: Practice & Experience 16*, 11 (2004), 1077–1098.
3. Yu, H., Leong, P. H. W., and Xu, Q. An FPGA Chip Identification Generator Using Configurable Ring Oscillators. *IEEE Trans. VLSI Syst. 20* (2012), 2198–2207.
4. Majzoobi, M., and Koushanfar, F. Time-Bounded Authentication of FPGAs. In *IEEE Transactions on Information Forensics and Security* (2011), vol. 6, pp. 1123–1135.
5. Maiti, A., Nagesh, R., Reddy, A., and Schaumont, P. Physical unclonable function and true random number generator: a compact and scalable implementation. In *Proc. of ACM Great Lakes Symposium on VLSI* (2009), pp. 425–428.
6. Kumar, S., Guajardo, J., Maes, R., Schrijen, G.-J., and Tuyls, P. Extended abstract: The butterfly PUF protecting IP on every FPGA. In *Proc. of IEEE International Workshop on Hardware-Oriented Security and Trust(HOST)* (June 2008), pp. 67–70.
7. Guajardo, J., Kumar, S. S., Schrijen, G. J., and Tuyls, P. FPGA intrinsic PUFs and their use for IP protection. In *Proc. of Cryptographic Hardware and Embedded Systems Workshop* (September 2007), vol. 4727 of *LNCS*, pp. 63–80.
8. Zuchowski, P. S., Habitz, P. A., Hayes, J. D., and Oppold, J. H. Process and environmental variation impacts on ASIC timing. In *Proc. of IEEE/ACM International conference on Computer-aided design (ICCAD)* (2004), pp. 336–342.
9. Lofstrom, K., Daasch, W. R., and Taylor, D. IC Identification Circuit Using Device Mismatch. In *Proc. of ISSCC* (2000), pp. 372–373.
10. Pappu, R. S. *Physical one-way functions*. PhD thesis, Massachusetts Institute of Technology, March 2001.
11. Gassend, B., Clarke, D., van Dijk, M., and Devadas, S. Silicon physical random functions. In *Proc. of ACM Conference on Computer and Communications Security* (New York, NY, USA, 2002), ACM Press, pp. 148–160.
12. Xin, X., Kaps, J., and Gaj, K. A Configurable Ring-Oscillator-Based PUF for Xilinx FPGAs. In *Proc. of 14th Euromicro Conference on Digital System Design (DSD)* (2011), pp. 651–657.
13. Chen, Q., Csaba, G., Lugli, P., Schlichtmann, U., and ührmair, U. R. The Bistable Ring PUF: A new architecture for strong Physical Unclonable Functions. In *Proc. of IEEE International Symposium on Hardware-Oriented Security and Trust (HOST)* (june 2011), pp. 134 –141.
14. Cherif, Z., Danger, J.-L., Guilley, S., and Bossuet, L. An Easy-to-Design PUF Based on a Single Oscillator: The Loop PUF. In *Proc. of 15th Euromicro Conference on Digital System Design (DSD)* (2012), pp. 156–162.
15. Anderson, J. A PUF design for secure FPGA-based embedded systems. In *15th Asia and South Pacific Design Automation Conference (ASP-DAC)* (jan. 2010), pp. 1 –6.
16. Maes, R., Tuyls, P., and Verbauwhede, I. Intrinsic PUFs from Flip-flops on Reconfigurable Devices. In *Proc. of 3rd Benelux Workshop on Information and System Security (WISSec)* (Eindhoven,NL, 2008), p. 17.

17. Yao, Y., Kim, M.-B., Li, J., Markov, I. L., and Koushanfar, F. ClockPUF: Physical Unclonable Functions based on Clock Networks. In *Design, Automation & Test in Europe (DATE)* (Grenoble, France, 2013).
18. Zheng, Y., Krishna, A. R., and Bhunia, S. ScanPUF: Robust Ultralow Overhead PUF Using Scan Chain. In *Proc of IEEE/ACM Asia and South Pacific Design Automation Conference (ASP-DAC)* (2013).
19. Lim, D. Extracting Secret Keys from Integrated Circuits. Master's thesis, MIT, USA, 2004.
20. Ührmair, U. R., Sehnke, F., "olter, J. S., Dror, G., Devadas, S., and "urgen Schmidhuber, J. Modeling attacks on physical unclonable functions. In *Proc. of 17th ACM conference on Computer and communications security(CCS)* (New York, NY, USA, 2010), ACM, pp. 237–249.
21. Hospodar, G., Maes, R., and Verbauwhede, I. Machine learning attacks on 65nm Arbiter PUFs: Accurate modeling poses strict bounds on usability. In *In Proc. of the 4th IEEE International Workshop on Information Forensics and Security (WIFS)* (2012), pp. 37–42.
22. Bishop, C. M. *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2006.
23. Majzoobi, M., Koushanfar, F., and Potkonjak, M. Lightweight secure PUFs. In *Proc. of the 2008 IEEE/ACM International Conference on Computer-Aided Design(ICCAD)* (Piscataway, NJ, USA, 2008), IEEE Press, pp. 670–673.
24. Rührmair, U., Devadas, S., and Koushanfar, F. *Security based on Physical Unclonability and Disorder*. Springer, 2011, ch. Book Chapter in Introduction to Hardware Security and Trust.
25. Ash, R. B. *Information Theory*. Dover Publication, Inc., 1990.
26. Maiti, A., Gunreddy, V., and Schaumont, P. A Systematic Method to Evaluate and Compare the Performance of Physical Unclonable Functions. *IACR Cryptology ePrint Archive 2011* (2011), 657.
27. Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., and Duchesn, E. Scikit-learn: Machine Learning in Python . *Journal of Machine Learning Research 12* (2011), 2825–2830.
28. Demuth, H., Beale, M., Demuth, H., and Beale, M. *Neural Network Toolbox For Use with Matlab*, 1993.
29. Aarts, E., and Lenstra, J. K., Eds. *Local Search in Combinatorial Optimization*. Princeton University Press, 2003.
30. Koza, J. R. *Genetic Programming: On the Programming of Computers by Means of Natural Selection (Complex Adaptive Systems)*, 1 ed. A Bradford Book, 1992.

A hierarchical graph-based approach to generating formally-proofed Galois-field multipliers

Okamoto Kotaro, Naofumi Homma, Takafumi Aoki

Graduate School of Information Sciences, Tohoku University
6–6–05, Aramaki Aza Aoba, Aoba-ku, Sendai-shi 980-8579, Japan
E-mail: okamoto@aoki.ecei.tohoku.ac.jp

Abstract. This paper presents a formal design of Galois-field multipliers based on a graph representation called *Galois-Field Arithmetic Circuit Graph* (*GF-ACG*). We focus on one of the optimal $GF(2^m)$ parallel multipliers, called Mastrovito Multiplier, and derive the hierarchical description from the original flatten description. The hierarchical description can be verified formally within a practical time even if the data width is more than 128. This paper also applies the hierarchical design to an automatic generation of formally-proofed GF parallel multipliers for any irreducible polynomial.

Keywords: arithmetic circuits, formal verification, hierarchical description, computer algebra

1 Introduction

Applications of arithmetic operations over Galois fields (GFs) have been rapidly increasing owing to the high demands of ECC (error correction code) and cryptographic processors for reliable/secure communications and transactions [1]. These operations are often implemented on hardware in recent embedded systems, such as smart cards and RFIDs, and the performance and security of arithmetic circuits have a significant impact on the entire systems. Currently, many hardware algorithms on GF arithmetic have been devised and some of such algorithms are implemented in actual systems.

On the other hand, most of such arithmetic circuits are designed at the logic level by researchers who had trained in a particular way to understand GF arithmetic. The conventional Hardware Description Languages (HDLs) do not have high-level arithmetic data structures, arithmetic operations and formulae over Galois fields. This sometimes requires us to describe structural details of arithmetic circuits by hand at the lowest level of abstraction (i.e., the level of AND-XOR expressions). In addition, the functional verification using the conventional logic simulation is quite time-consuming since these operations are usually performed with more-than 64-bit operands. The test pattern generation

is also complicate since it varies with the irreducible polynomial even for the same operation (e.g., multiplication). There is a decision diagram for Galois fields based on the decomposition of multiple-valued functions [2], but it is difficult to handle practical fields such as $GF(2^{16})$ and $GF(2^{32})$ and apply it to the formal verification.

Addressing the above problems, a formal description and verification method of arithmetic circuits over GFs were proposed in [3],[4]. The proposed idea is to use a high-level mathematical graph based on variables and arithmetic formulae over GFs, which is called *Galois-field Arithmetic Circuit Graph*: *GF-ACG*. Using GF-ACGs, we can describe any GF arithmetic circuit in a hierarchical manner as a combination of arithmetic sub-circuits (sub-graphs). Such description is formally verified by checking for every sub-circuit whether the function is obtained from the internal structure. The equivalence checking can be performed by formula manipulations based on a polynomial reduction algorithm using Gröbner Basis [5], which makes it possible to verify practical arithmetic circuits in a short time.

In this paper, we present an automatic generation system producing formally-proofed GF arithmetic circuits based on GF-ACG. Given a design specification, the system generates $GF(2^m)$ parallel multipliers for any irreducible polynomial in HDL format. The system first generates a GF-ACG code according to the design specification, verifies it by the formal verification, and translates the verified code into the corresponding HDL code. We can generate Mastrovito and Massey-Omura multiplication algorithms which are considered as typical ones by polynomial basis (PB) and normal basis (NB), respectively [6],[7]. In this paper, we first derive the hierarchical description of Mastrovito multiplier from the original flatten description in order to exploit the GF-ACG approach. We then demonstrate the performance of the proposed system through the experimental generations of multipliers with some typical degrees. We confirm that the proposed system successfully generates practical multipliers with more-than 128-bit inputs.

2 Galois-field arithmetic circuit graph

This section briefly describes the graph-based representation of GF arithmetic circuits, where the graphs are referred to as GF Arithmetic Circuit Graphs (GF-ACGs).

Figure 1 shows an overview of a GF-ACG. A GF-ACG G is defined as $(\boldsymbol{N}, \boldsymbol{E})$, where $\boldsymbol{N}$ is a set of nodes, and $\boldsymbol{E}$ is a set of directed edges. The node represents an arithmetic circuit by its functional assertion and internal structure. The directed edge represents the flow of data between nodes, and defines the data dependency. We assume that every node has at least one edge connection.

A node n ($\in \boldsymbol{N}$) is defined by $(\boldsymbol{F}, G')$, where $\boldsymbol{F}$ is the functional assertion given as a set of equations over GFs (*GF equations*) and G' is the internal structure given as a smaller GF-ACG. A node at the lowest level of abstraction, which does not have its internal structure, is described as $(\boldsymbol{F}, nil)$. A functional asser-

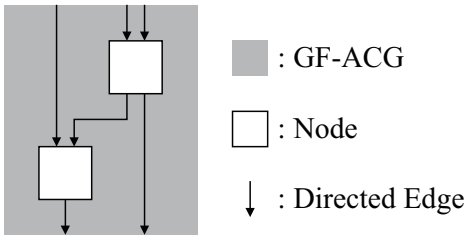


Fig. 1. Galois-field arithmetic circuit graph.

tion is represented as a relation $E_l = E_r$, where E_l and E_r are the output and input expressions, respectively, and each expression is given by variables, constants or combinations of the two or more expressions connected by arithmetic operations $+$, $-$, $\times$, and $/$.

A directed edge e ($\in \boldsymbol{E}$) is defined as $(src, dest, x)$, where src and $dest$ represent the start and end node, respectively, and x represents the variable indicating an element of GF. If either src or $dest$ is nil, its directed edge represents an external input or output for the given GF-ACG. Each variable is associated with a Galois field. A Galois field GF based on polynomial basis (PB) is defined as $(\boldsymbol{B}, \boldsymbol{C}, IP)$, where $\boldsymbol{B}$ is the basis, $\boldsymbol{C}$ is the coefficient vector, and IP is the irreducible polynomial. More precisely, $\boldsymbol{B}$, $\boldsymbol{C}$, and IP are given as

$$\boldsymbol{B} = \left(\beta^{m-1}, \beta^{m-2}, \cdots, \beta^0\right), \tag{1}$$

$$\boldsymbol{C} = \left(C_{m-1}, C_{m-2}, \cdots, C_0\right), \tag{2}$$

$$IP = \beta^m + c_{m-1}\beta^{m-1} + \cdots + c_0\beta^0, \tag{3}$$

where β is the indeterminate element, C_i is the coefficient set of degree i, m is the degree of field extension, and c_i is the element of the coefficient set C_i. $IP = nil$ if the GF is a prime field. Thus, the above description can handle both prime and extension fields. Let h ($0 \leq h \leq m-1$) and l ($0 \leq l \leq h$) be the most and least significant degrees, respectively. A variable is represented as $x = (GF, (h, l))$, where the tuple (h, l) is called the degree range. Using the above notation, we can handle a specific variable x_i of degree i.

A variable can be decomposed to an expression with sub-variables at a lower level of abstraction. Let x be a variable and x_i ($l \leq i \leq h$) be a lower-level variable. We have two types of decomposition nodes whose functions are given as

$$x_h^{(e)} + x_{h-1}^{(e)} + \cdots + x_l^{(e)} = x, \tag{4}$$

$$x_h^{(p)}\beta^h + x_{h-1}^{(p)}\beta^{h-1} + \cdots + x_l^{(p)}\beta^l = x. \tag{5}$$

Eq. (4) indicates that $x \in GF(p^m)$ is divided into a number of variables of degree i (i.e., $x_i^{(e)}(l \leq i \leq h) \in GF(p^m)$). On the other hand, Eq. (5) indicates that $x \in GF(p^m)$ is divided into a number of variables over the prime field

(i.e., $x_i^{(p)}(l \leq i \leq h) \in GF(p)$). We also have two types of composition nodes given as inverse relations between the above inputs and outputs. Using the decomposition/composition nodes, we can change the level of abstraction in edge representation. Note here that these nodes are implemented by wiring and have no internal structures.

The above GF-ACG can be used also for representing any logic circuit. A logic variable is considered as a variable over the GF whose coefficient set is limited to the zero element "0" and the unit element "1". Any logical operation can be represented with pseudo logic equations. For example, the functions of AND and XOR circuits are given as

$$and(a, b) = ab, \tag{6}$$

$$xor(a, b) = a + b - 2ab, \tag{7}$$

respectively. Note that the idempotent law is considered as one of functional assertions in the corresponding node (i.e., $a = a^2$ and $b = b^2$).

Thus, GF-ACG can represent any arithmetic circuit over GF represented by PB and any logic circuit. The arithmetic circuits given by GF-ACGs are verified by a formal verification method using Gröbner basis and a polynomial reduction technique [3].

3 Hierarchical design of Mastrovito multiplier

The Mastrovito multiplier [6] is a 2-input 1-output parallel multiplier over $GF(2^m)$ represented by Polynomial Basis (PB), which is known as one of the smallest multiplier structures by PB. The Mastrovito multiplier is originally composed of matrix generation and matrix operation parts.

Let a and $b \in GF(2^m)$ be the inputs and let $c \in GF(2^m)$ be the output. The matrix generation part generates an $m \times m$ matrix $\boldsymbol{Z}$ from the input $a(= \sum_{i=0}^{m-1} a_i^{(p)} \cdot \beta^i)$. For example, a matrix $\boldsymbol{Z}$ of the $GF(2^4)$ Mastrovito multiplier by an irreducible polynomial $\beta^4 + \beta + 1$ is given as

$$\boldsymbol{Z} = \begin{bmatrix} Z_{3,3}^{(p)} & Z_{2,3}^{(p)} & Z_{1,3}^{(p)} & Z_{0,3}^{(p)} \\ Z_{3,2}^{(p)} & Z_{2,2}^{(p)} & Z_{1,2}^{(p)} & Z_{0,2}^{(p)} \\ Z_{3,1}^{(p)} & Z_{2,1}^{(p)} & Z_{1,1}^{(p)} & Z_{0,1}^{(p)} \\ Z_{3,0}^{(p)} & Z_{2,0}^{(p)} & Z_{1,0}^{(p)} & Z_{0,0}^{(p)} \end{bmatrix} = \begin{bmatrix} a_0^{(p)} + a_3^{(p)} & a_1^{(p)} & a_2^{(p)} & a_3^{(p)} \\ a_3^{(p)} + a_2^{(p)} & a_0^{(p)} + a_3^{(p)} & a_1^{(p)} & a_2^{(p)} \\ a_2^{(p)} + a_1^{(p)} & a_3^{(p)} + a_2^{(p)} & a_0^{(p)} + a_3^{(p)} & a_1^{(p)} \\ a_1^{(p)} & a_2^{(p)} & a_3^{(p)} & a_0^{(p)} \end{bmatrix}, \tag{8}$$

where $Z_{i,j}^p \in GF(2)$. The original description is not suitable for the design and verification of GF-ACG since the complexity of the description is exponentially increased by the degree (i.e., input width).

For the hierarchical design, we represent the matrix $\boldsymbol{Z}$ by column vectors Z_i as follows.

$$\boldsymbol{Z} = \begin{bmatrix} Z_{m-1} & Z_{m-2} & \cdots & Z_0 \end{bmatrix}. \tag{9}$$

In the case of the above example (i.e., Eq. (8)), Z_1 is given as $[a_2^{(p)}\ a_1^{(p)}\ a_0^{(p)} + a_3^{(p)}\ a_3^{(p)}]^{\mathrm{T}}$. Note here that the column vector is represented by the following polynomial.

$$Z_i = a \cdot \beta^i,\ 0 \leq i \leq m-1. \tag{10}$$

The column vector is also given as the following recurrence formula.

$$Z_i = Z_{i-1} \cdot \beta,\ 1 \leq i \leq m-1. \tag{11}$$

Thus, the function and internal structure of the matrix generation are given by Eqs. (10) and (11), respectively. The internal structure of Eq. (11) is given by 2-input 1-output adders over $GF(2)$.

The matrix operation part calculates the inner product of $\boldsymbol{Z}$ and the other input $b(= \sum_{i=0}^{m-1} b_{i,i}^{(p)} \cdot \beta^i)$, where $b_{i,i}^{(p)}$ is the i-th element given by the decomposition of Eq. (5) after the decomposition of Eq. (4). The operation is given by

$$c = \left[\ Z_{m-1}\ \ Z_{m-2}\ \ \cdots\ \ Z_0\ \right] \times \begin{bmatrix} b_{m-1,m-1}^{(p)} \\ b_{m-2,m-2}^{(p)} \\ \vdots \\ b_{0,0}^{(p)} \end{bmatrix} = \sum_{i=0}^{m-1} Z_i \times b_{i,i}^{(p)}. \tag{12}$$

Let $b_i^{(e)}$ be $b_{i,i}^{(p)} \cdot \beta^i$ ($b_i^{(e)} \in GF(2^m)$). The function of Eq. (12) is also given as

$$c = \sum_{i=0}^{m-1} Z_i \times (b_i^{(e)} \cdot \beta^{-i}). \tag{13}$$

For the hierarchical design, the function of Eq. (13) is represented by the following equations.

$$w_i = Z_i \times (b_i^{(e)} \cdot \beta^{-i}),\ 0 \leq i < m, \tag{14}$$

$$c = \sum_{i=0}^{m-1} w_i. \tag{15}$$

Thus, the matrix operation part is represented by two operations whose function are given by Eqs (14) and (15). Here, the function of Eq. (15) is implemented by m-1 2-input 1-output adders over $GF(2^m)$. In the lower level of abstraction, the internal structures of Eqs. (14) and (15) are given by 2-input 1-output multipliers and adders over $GF(2)$, respectively.

Figure 2 shows the GF-ACGs for the Mastrovito multiplier over $GF(2^4)$, where the GF-ACGs are represented by four levels of abstraction. The nodes in Fig. 2 (a), (b) and (c) correspond to the shaded parts in Fig. 2 (b), (c) and (d), respectively. Tables 1 and 2 show the functional assertions and GF variables,

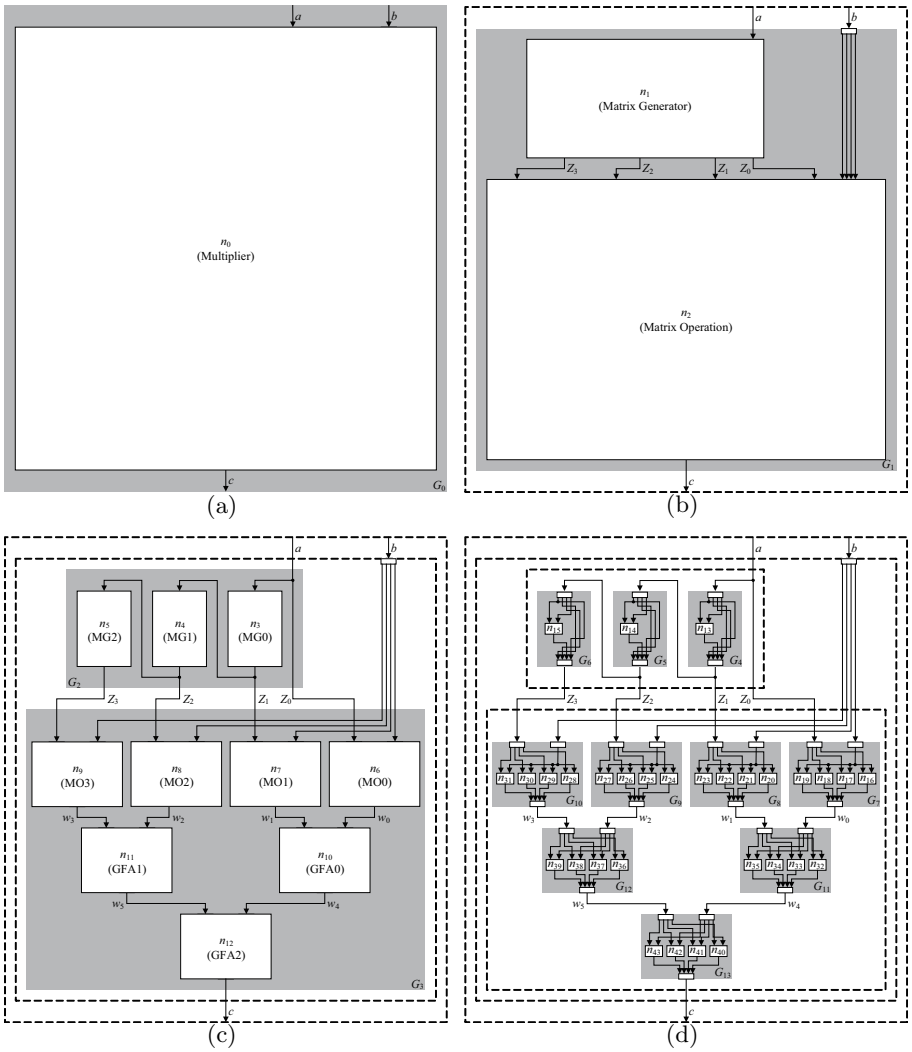


Fig. 2. GF-ACGs for $GF(2^4)$ Mastrovito multiplier from (a) the highest level to (d) the lowest level.

respectively. Note that the decomposition/composition nodes are not shown in Tab. 1.

The 2^{nd}-level nodes "Matrix Generator" and "Matrix Operation" in Fig. 2 (b) have functional assertions corresponding to Eqs. (10) and (13), respectively. The 3^{rd}-level nodes "MGi" and "MOi" in Fig. 2 (c) have functional assertions corresponding to Eqs. (11) and (14), respectively. And, The 3^{rd}-level nodes "GFAi" in Fig. 2 (c) indicate 2-input 1-output adders over $GF(2^4)$. It is important to

Table 1. NODES IN FIG. 2

[Multiplier] $n_0 = (\{c = a \times b\}, G_1)$
[Matrix Generator] $n_1 = (\{Z_0 = a \cdot \beta^0, Z_1 = a \cdot \beta^1, Z_2 = a \cdot \beta^2, Z_3 = a \cdot \beta^3\}, G_2)$
[MG0] $n_3 = (\{Z_1 = Z_0 \cdot \beta\}, G_4)$
$n_{13} = (\{Z_{1,1}^{(p)} = Z_{0,0}^{(p)} + Z_{0,3}^{(p)}\}, nil)$
[MG1] $n_4 = (\{Z_2 = Z_1 \cdot \beta\}, G_5)$
$n_{14} = (\{Z_{2,1}^{(p)} = Z_{1,0}^{(p)} + Z_{1,3}^{(p)}\}, nil)$
[MG2] $n_5 = (\{Z_3 = Z_2 \cdot \beta\}, G_6)$
$n_{15} = (\{Z_{3,1}^{(p)} = Z_{2,0}^{(p)} + Z_{2,3}^{(p)}\}, nil)$
[Matrix Operation] $n_2 = (\{c = \sum_{i=0}^{3} Z_i \times (b_i^{(e)} \cdot \beta^{-i}), G_3)$
[MO0] $n_6 = (\{w_0 = Z_0 \times (b_0^{(e)} \cdot \beta^{-0})\}, G_7)$
$n_{16} = (\{w_{0,0}^{(p)} = Z_{0,0}^{(p)} \times b_{0,0}^{(p)}\}, nil),\ n_{17} = (\{w_{0,1}^{(p)} = Z_{0,1}^{(p)} \times b_{0,0}^{(p)}\}, nil)$
$n_{18} = (\{w_{0,2}^{(p)} = Z_{0,2}^{(p)} \times b_{0,0}^{(p)}\}, nil),\ n_{19} = (\{w_{0,3}^{(p)} = Z_{0,3}^{(p)} \times b_{0,0}^{(p)}\}, nil)$
[MO1] $n_7 = (\{w_1 = Z_1 \times (b_1^{(e)} \cdot \beta^{-1})\}, G_8)$
$n_{20} = (\{w_{1,0}^{(p)} = Z_{1,0}^{(p)} \times b_{1,1}^{(p)}\}, nil),\ n_{21} = (\{w_{1,1}^{(p)} = Z_{1,1}^{(p)} \times b_{1,1}^{(p)}\}, nil)$
$n_{22} = (\{w_{1,2}^{(p)} = Z_{1,2}^{(p)} \times b_{1,1}^{(p)}\}, nil),\ n_{23} = (\{w_{1,3}^{(p)} = Z_{1,3}^{(p)} \times b_{1,1}^{(p)}\}, nil)$
[MO2] $n_8 = (\{w_2 = Z_2 \times (b_2^{(e)} \cdot \beta^{-2})\}, G_9)$
$n_{24} = (\{w_{2,0}^{(p)} = Z_{2,0}^{(p)} \times b_{2,2}^{(p)}\}, nil),\ n_{25} = (\{w_{2,1}^{(p)} = Z_{2,1}^{(p)} \times b_{2,2}^{(p)}\}, nil)$
$n_{26} = (\{w_{2,2}^{(p)} = Z_{2,2}^{(p)} \times b_{2,2}^{(p)}\}, nil),\ n_{27} = (\{w_{2,3}^{(p)} = Z_{2,3}^{(p)} \times b_{2,2}^{(p)}\}, nil)$
[MO3] $n_9 = (\{w_3 = Z_3 \times (b_3^{(e)} \cdot \beta^{-3})\}, G_{10})$
$n_{28} = (\{w_{3,0}^{(p)} = Z_{3,0}^{(p)} \times b_{3,3}^{(p)}\}, nil),\ n_{29} = (\{w_{3,1}^{(p)} = Z_{3,1}^{(p)} \times b_{3,3}^{(p)}\}, nil)$
$n_{30} = (\{w_{3,2}^{(p)} = Z_{3,2}^{(p)} \times b_{3,3}^{(p)}\}, nil),\ n_{31} = (\{w_{3,3}^{(p)} = Z_{3,3}^{(p)} \times b_{3,3}^{(p)}\}, nil)$
[GFA0] $n_{10} = (\{w_4 = w_0 + w_1\}, G_{11})$
$n_{32} = (\{w_{4,0}^{(p)} = w_{0,0}^{(p)} + w_{1,0}^{(p)}\}, nil),\ n_{33} = (\{w_{4,1}^{(p)} = w_{0,1}^{(p)} + w_{1,1}^{(p)}\}, nil)$
$n_{34} = (\{w_{4,2}^{(p)} = w_{0,2}^{(p)} + w_{1,2}^{(p)}\}, nil),\ n_{35} = (\{w_{4,3}^{(p)} = w_{0,3}^{(p)} + w_{1,3}^{(p)}\}, nil)$
[GFA1] $n_{11} = (\{w_5 = w_2 + w_3\}, G_{12})$
$n_{36} = (\{w_{5,0}^{(p)} = w_{2,0}^{(p)} + w_{3,0}^{(p)}\}, nil),\ n_{37} = (\{w_{5,1}^{(p)} = w_{2,1}^{(p)} + w_{3,1}^{(p)}\}, nil)$
$n_{38} = (\{w_{5,2}^{(p)} = w_{2,2}^{(p)} + w_{3,2}^{(p)}\}, nil),\ n_{39} = (\{w_{5,3}^{(p)} = w_{2,3}^{(p)} + w_{3,3}^{(p)}\}, nil)$
[GFA2] $n_{12} = (\{c = w_4 + w_5\}, G_{13})$
$n_{40} = (\{c_0^{(p)} = w_{4,0}^{(p)} + w_{5,0}^{(p)}\}, nil),\ n_{41} = (\{c_1^{(p)} = w_{4,1}^{(p)} + w_{5,1}^{(p)}\}, nil)$
$n_{42} = (\{c_2^{(p)} = w_{4,2}^{(p)} + w_{5,2}^{(p)}\}, nil),\ n_{43} = (\{c_3^{(p)} = w_{4,3}^{(p)} + w_{5,3}^{(p)}\}, nil)$

note that we can simply extend the above GF-ACG description to describe any Mastrovito multiplier over $GF(2^m)$ $(2 \leq m)$.

4 Galois-Field Arithmetic Module Generator

This section presents an application of our approach to an automatic generation system (GF-AMG: Galois-Field Arithmetic Module Generator) producing GF multipliers. We employ GF-ACG for describing Mastrovito and Massey-Omura parallel multiplication algorithms as shown in the above section and [4]. The use of GF-ACG makes it possible to produce multiplier modules whose functions are formally verified at the algorithm level.

Table 2. GALOIS FIELDS AND VARIABLES IN FIG. 2

Galois field
$GF(2^4) = (\{\beta^3, \beta^2, \beta^1, \beta^0\}, (\{0,1\}, \{0,1\}, \{0,1\}, \{0,1\}), \beta^4 + \beta^1 + \beta^0)$
$GF(2) = (\{\beta^0\}, (\{0,1\}), nil)$
Galois field variables
$a = (GF(2^4), (3,0))$
$b = (GF(2^4), (3,0))$
$b_i^{(e)} = (GF(2^4), (i,i)),\ (0 \le i \le 3)$
$b_{i,i}^{(p)} = (GF(2), (0,0)),\ (0 \le i \le 3)$
$c = (GF(2^4), (3,0))$
$c_i^{(p)} = (GF(2), (0,0)),\ (0 \le i \le 3)$
$Z_i = (GF(2^4), (3,0)),\ (0 \le i \le 3)$
$Z_{i,j}^{(p)} = (GF(2), (0,0)),\ (0 \le i \le 3,\ 0 \le j \le 3)$
$w_i = (GF(2^4), (3,0)),\ (0 \le i \le 5)$
$w_{i,j}^{(p)} = (GF(2), (0,0)),\ (0 \le i \le 5,\ 0 \le j \le 3)$

Figure 3 is a block diagram of GF-AMG, which consists of (i) GF-ACG Code Synthesizer, (ii) GF-ACG Verifier, and (iii) ACG-to-HDL Translator. First, the GF-ACG Code Synthesizer generates a GF-ACG code according to the design specification. Either Mastrovito algorithm or Massey-Omura algorithm with any irreducible polynomial can be selected for the specification. The degree for irreducible polynomial is acceptable from 2 to 256/64 for the selected algorithm. The GF-ACG verifier then verifies the generated GF-ACG code by computer algebra in a formal manner. The ACG-to-HDL Translator finally translates the verified GF-ACG code into the equivalent Verilog-HDL code. This can be done simply by the one-to-one mapping.

Figure 4 shows an overview of the verification procedure for GF-ACGs. Given a GF-ACG, all the nodes having functional assertions and internal structures are verified by the formula evaluation ($FormulaEvaluation$). If GF equations of the internal structure are equivalent to the functional assertion(s), $FormulaEvaluation$ returns *true*. Here, we assume that the lowest-level nodes, where the functional assertions are given by logical functions such as XOR and AND, are predetermined and reliable. The major feature of GF-ACGs is that the formula evaluation can be performed by computer algebra, which utilizes polynomial reduction and Gröbner basis techniques.

For any polynomial p, we obtain a unique element, called a normal form, from repeated reductions with respect to a set of polynomials $\boldsymbol{Q} = \{q_1, q_2, \cdots, q_m\}$. The normal form is denoted by $\mathrm{NF}_{\boldsymbol{Q}}(p)$. If the set is a Gröbner Basis, $\mathrm{NF}_{\boldsymbol{Q}}(p) = 0$ for any polynomial p in an ideal $\boldsymbol{I}$ (i.e., a set of polynomials genereted by $\boldsymbol{Q}$).

Figure 5 illustrates the formula evaluation procedure using Gröbner Basis, where $GroebnerBasis(\boldsymbol{P})$ indicates Buchberger's algorithm to obtain a Gröbner Basis $\boldsymbol{GB}$ from a set of polynomials $\boldsymbol{P}$. Given a functional assertion f and internal structure G, $\boldsymbol{P}$ is generated from functional assertions in the internal

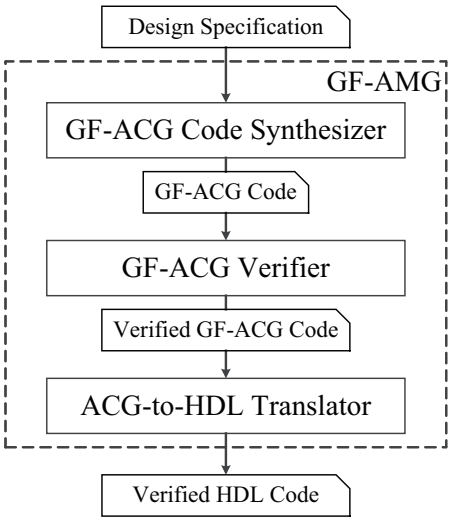


Fig. 3. Block diagram of GF-AMG.

Input:	Galois-field arithmetic circuit graph $G = (\boldsymbol{N}, \boldsymbol{E})$
Output:	Verification result $r \in \{true, false\}$

```
1:  Function Verify((N, E))
2:    r := true
3:    for each (F, G) ∈ N
4:      if G ≠ nil
5:        r := r & Verify(G)
6:        for each f ∈ F
7:          r := r & FormulaEvaluation(f, G)
8:        end for
9:      end if
10:   end for
11:   return r
12: end
```

Fig. 4. Verification algorithm for GF-ACGs.

structure. $\boldsymbol{GB}$ is then obtained from $GroebnerBasis(\boldsymbol{P})$. If the normal form of f with respect to $\boldsymbol{GB}$ is equal to zero, f is a member of the ideal from $\boldsymbol{P}$. This means that the functional assertion can be realized with the internal structure. Therefore, $FormulaEvaluation(f, G)$ returns $true$. To complete the verification of a GF-ACG, the above formula evaluation is applied to all the nodes in the GF-ACG.

Example 1. Consider a formula evaluation for the highest-level node n_0 of the Mastrovito multiplier over $GF(2^4)$ (as shown in Fig. 2), where the functional

Input:	Functional assertion f Internal structure $G = (\boldsymbol{N}, \boldsymbol{E})$
Output:	Verification result $r \in \{true, false\}$

```
1:  Function FormulaEvaluation(f, G)
2:    P := ∅
3:    for each (F, G') ∈ N
4:      P := P ∪ F
5:    end for
6:    GB := GroebnerBasis(P)
7:    if NF_GB(f) = 0
8:      r := true
9:    else
10:     r := false
11:   end if
12:   return r
13: end
```

Fig. 5. Formula evaluation algorithm.

assertion is $c = a \times b$, the internal structure is composed of ten nodes including decomposition/composition nodes, and the irreducible polynomial IP is $\beta^4 + \beta^1 + \beta^0$. The goal of this evaluation is to prove the correctness of the functional assertion of n_0 ($c = a \times b$) under the condition that the lower-level nodes are correctly implemented, in other words, that the functional assertions of lower nodes $(n_1, n_2, \cdots)$ are correct.

We first obtain a set of polynomials $\boldsymbol{P}$ from the internal structure.

$$\begin{aligned} P = \{&Z_0 - a \cdot \beta^0, Z_1 - a \cdot \beta^1, Z_2 - a \cdot \beta^2, Z_3 - a \cdot \beta^3, \\ &b_0^{(e)} + b_1^{(e)} + b_2^{(e)} + b_3^{(e)} - b, \\ &c - (Z_0 \times b_0^{(e)} \cdot \beta^{-0} + Z_1 \times b_1^{(e)} \cdot \beta^{-1} + Z_2 \times b_2^{(e)} \cdot \beta^{-2} + Z_3 \times b_3^{(e)} \cdot \beta^{-3}), \\ &\beta^4 + \beta^1 + \beta^0\}. \end{aligned} \tag{16}$$

Note here that the irreducible polynomial of $GF(2^4)$ is also included in $\boldsymbol{P}$. We then derive the Gröbner basis $\boldsymbol{GB}$ from $\boldsymbol{P}$ according to Buchberger's algorithm.

$$\begin{aligned} GB = \{&c + a \times b, \\ &b + b_0^{(e)} + b_1^{(e)} + b_2^{(e)} + b_3^{(e)}, \\ &a \cdot \beta^3 + Z_3, a \cdot \beta^2 + Z_2, a \cdot \beta^1 + Z_1, a \cdot \beta^0 + Z_0, \\ &\beta^4 + \beta^1 + \beta^0\}. \end{aligned} \tag{17}$$

Note here that the operator "+" is identical to the operator "−" since the variables are defined as $GF(2)$ variables.

Finally, the normal form of the function (i.e., $c - a \times b$) with respect to $\boldsymbol{GB}$ is given by $\mathrm{NF}_{\boldsymbol{GB}}(c - a \times b) = 0$. Therefore, the function is derived from the internal functions and the formula evaluation returns true.

Table 3. Generation time of Mastrivito multipliers (sec)

	$GF(2^4)$	$GF(2^8)$	$GF(2^{16})$	$GF(2^{32})$	$GF(2^{64})$	$GF(2^{128})$
Verilog-XL simulation	0.137	0.279	9330	N/A	N/A	N/A
Formal verification	2.499	3.374	5.188	9.487	19.55	52.61

Table 4. Generation time of Massey-Omura parallel multipliers (sec)

	$GF(2^4)$	$GF(2^8)$	$GF(2^{16})$	$GF(2^{32})$	$GF(2^{64})$	$GF(2^{128})$
Verilog-XL simulation	0.309	0.460	N/A	N/A	N/A	N/A
Formal verification	2.334	3.618	5.482	16.24	372.5	34,263

If the set of polynomials consists of linear polynomials, the Gröbner Basis calculation is equivalent to Gaussian Elimination [5]. In this case, the computation cost of the proposed method becomes $O(k^3)$, where k is the number of variables. For many arithmetic circuits, word-level structures are commonly represented by linear equations, and thus the proposed verification method can be effective for verifying such word-level functions.

To evaluate the performance of our system, we generated the two types of multipliers with some typical degrees. Tables 3 and 4 show the generation times for Mastrovito and Massey-Omura multipliers, respectively. The evaluation was done on Linux PC with Intel Core2 Due E4600 2.40GHz and 7GB memory. The formal verification was implemented with an open-source general computer algebra system called Risa/Asir. For comparison, we also performed the Verilog-XL simulation using the corresponding HDL descriptions. We were not able to succeed the complete simulation of $GF(2^{32})$ and larger multipliers in this experiment because the verification time increases exponentially as the signal length increases. On the other hand, using our system, we were able to succeed the complete verification even for the 128-bit multiplier over $GF(2^{128})$. Most of the generation times were used by the GF-ACG verifier. The time consuming related to the GF-ACG Synthesizer and the ACG-to-HDL Translator was almost constant independent of the multiplier size.

5 Conclusion

This paper has presented the hierarchical designs of Galois-field parallel multipliers based on GF-ACG, and shown an application to an automatic generation system producing two types of multipliers (i.e., Mastrovito and Massey-Omura multipliers) for any irreducible polynomial. The generated HDL codes are completely verified by the formal verification method in the system. The proposed system is available from our website [8]. Further investigations are being conducted to develop advanced module generators for cryptographic datapaths with GF arithmetic circuits.

References

1. E. Savas and C.K. Koc.: Finite Field Arithmetic for Cryptography. IEEE Circuits and Systems Magazine, vol. 10, no. 2, pp. 40–56 (2010).
2. R. Stankovic and R. Drechsler.: Circuit design from kronecker Galois field decision diagrams for multiple-valued functions. Proc. 27nd IEEE Int. Symp. Multiple-Valued Logic.pp. 275–280 (1997).
3. N. Homma, K. Saito, and T. Aoki.: A formal approach to designing cryptographic processors based on $GF(2^m)$ arithmetic circuits. IEEE Trans. Information Forensics and Security, Vol. 7, No. 1, pp. 3–13 (2012).
4. K. Okamoto, N. Homma, and T Aoki.: A graph-based approach to designing parallel multipliers over Galois fields based on normal basis representations. Proc. 43rd IEEE Int. Symp. Multiple-Valued Logic, pp. 158–163 (2013).
5. D. A. Cox, J. B. Little, and D. O'Shea.: Ideals, Varieties, and Algorithms. NY: Springer-Verlag, 2nd ed., pp. 536 (1996).
6. A. Halbutogullari and C.K. Koc.: Mastrovito multiplier for general irreducible polynomials. IEEE Trans. Computers, vol. 49, no. 5, pp. 503–518 (2000).
7. A. Reyhani-Masoleh and M.A. Hasan.: A New Construction of Massey-Omura Parallel Multiplier over $GF(2^m)$. IEEE Trans. Computers, vol. 51, no. 5, pp. 511–520 (2001).
8. Galois-field Arithmetic module generator based on GF-ACG, `http://www.aoki.ecei.tohoku.ac.jp/arith/gfamg/index.html`.

Trojan-Resilient Circuits

Christoph Bayer and Jean-Pierre Seifert

Security in Telecommunications
Technische Universität Berlin and Deutsche Telekom Laboratories
{christoph,jpseifert}@sec.t-labs.tu-berlin.de

Abstract. Integrated circuits (ICs) can contain malicious logic or backdoors, known as hardware trojans, that may impede them from functioning properly. These hardware trojans include the recent bug attacks due to Biham et al. In order to protect ICs against such hardware trojans and the bug attacks, we present an effective method to efficiently construct "trojan-resilient" circuits. Therefore, we revisit the fundamental work on fault-tolerant circuits by Gál and Szegedy. We extend their attack model; and from this extended adversary scenario we derive a mathematical definition of "IC resilience" against well-defined hardware trojans. In our model we allow an all-powerful adversary to modify a constant fraction of the gates and wires at each level of the resilient circuit in an arbitrary way. We prove that every Boolean circuit can be transformed into another Boolean circuit with the same functionality as the original circuit even in the presence of an adversary tampering with the "resilient" circuit. The transformation is polynomial time computable and yields a circuit, which has a logarithmic depth in the size of the original circuit. To the best of our knowledge this is the first work to counteract hardware trojans with a rigorous mathematical security proof — backed up by a practical and meaningful model.

Keywords: bug attacks; error-correcting codes; hardware backdoors and trojans; probabilistically checkable proofs; reliable and secure circuits; resilient circuit design; security and trust.

1 Introduction

The recently presented *bug attacks* of Biham et al. [11] make use of deterministic errors in the hardware implementation of cryptosystems (such as Pohlig-Hellman[29] and RSA[32]). By a single chosen ciphertext or a larger number of ciphertexts (in the case of RSA OAEP [8]) the full secret key can be computed. Even if there is only one error in the multiplication of a single pair of numbers, this can be exploited by an adversary. The errors, which are utilized by attackers, can occur due to accidental bugs like the Intel division bug. More alarming is the problem of maliciously tampered hardware. Until recently, integrated circuits (ICs) were assumed to be secure against malicious activities. However, it is questionable if this assumption is still valid. ICs are becoming more and more vulnerable to malicious alterations, known as hardware trojans, as many in-house steps of an IC fabrication process are being outsourced to third-parties.

This risk is presented in [1, 13, 25]. In particular, hardware trojans have drawn attention to the potential threats to governmental and military systems as well as financial infrastructure and transportation security. For all mission critical areas and in order to prevent bug attacks, it is essential to use computer systems that ensure their correct functionality in all circumstances. Thus, a reasonable and challenging question is whether ICs can be designed in such a way that the functionality intended by the designers is guaranteed — even if a malicious adversary is allowed to tamper with the ICs during certain production steps. That means, we consider that scenario, in which the design process is trusted, but the subsequent production steps are not trustworthy. Due to the so-called "fabless" trend, this scenario is the most relevant one as outsourcing to silicon foundries is by virtue untrusted. This challenge was explicitly presented in [12] as the first of three key topics and it was mentioned in [13] as well: "Given an IC corresponding to a known design, does the IC that is delivered do what it is supposed to do and nothing more? This is the case when the fabrication facility is not trusted, but the design process is."

Our paper answers the central question affirmatively by presenting a resilient IC design that ensures equivalent IC functionality even if a malicious adversary physically alters the IC in the fabrication phase. In a real-world scenario an attacker will only slightly alter a chip since large-scale alterations would later be detected during testing phase. This scenario goes along with the model of bug attacks, in which only a single multiplication needs to be incorrect in order for the attack to work. Thus, our concept for a resilient IC design fulfills the requirement that small changes to the IC do not affect its correct functionality, whereas more serious changes should be detected otherwise. Such a design could,e.g., prevent the danger of so-called kill switches [1] and bug attacks [11].

Since the topic of hardware trojans is rather new, recent research has followed other approaches. In [23, 26, 36] hardware trojans have been constructed and practically investigated. There are various approaches to a hardware trojan taxonomy [22, 37, 43]. This is necessary as adversaries have many different objectives, and hence, the developed hardware trojans are vastly different as well. Several contributions to the topic of hardware trojans focus exclusively on trojan detection [2, 18, 24, 37, 41, 43, 46]. However, little research has gone into design techniques that detect hardware trojans at runtime or counteract them at runtime [18, 41, 42]. The proposed directions are rather heuristic and do not cover all hardware backdoors. Ishai et al. [20, 21] examine the problem of privacy in circuits, which is motivated by side channel attacks. They define a formal threat model, and suggest provably secure methods to counteract probing attacks on circuits.

Compared to all other existing research concerning proper IC functionality, this paper develops a trojan-resilient IC model, which is provably secure. To achieve this, we first revisit the model of fault-tolerant Boolean circuits by Gál and Szegedy [17]. Their work deals with deliberate non-random faults in Boolean circuits. We extend their work and introduce a formal attack scenario that targets the malicious altering of ICs. This hardware trojan model is mapped into a real-

world taxonomy [22] as well. By leveraging Gál and Szegedy's techniques of fault-tolerant computations, we prove that it is indeed possible to design ICs that maintain their correct functionality even if they are infected by hardware trojans. To the best of our knowledge there is no prior work relating the area of fault-tolerant circuit computation with hardware trojans.

Research in the area of fault-tolerant circuits has mainly been based on the model of random faults by von Neumann [40]. In this model it is assumed that all gates of a Boolean circuit fail, i.e., produce a faulty value, independently and with probability bounded by some small constant. From this early research it is known that any function can be reliably computed by another circuit of size $L \log L$, where L is the size of the error-free circuit that computes the given function [15, 28, 40]. However, hardware trojans can not be described by this classical fault model since maliciously altered hardware closely resembles deliberately inserted faults. Gál and Szegedy deal with this more difficult case when the faults are not random [17]. In their model an adversary may arbitrarily choose a small constant fraction of the gates at each level of a Boolean circuit to be faulty. They introduce a constructive way to efficiently build fault-tolerant Boolean circuits with small redundancy even in the presence of non-random faulty gates. Their construction makes use of very elaborated techniques from the field of probabilistically checkable proofs (PCPs) amongst many others, cf. [6]. The PCP theorem guarantees an efficient proof system with "robust" proofs for every set of **NP**. In the notion of PCPs, proofs can be efficiently encoded and provide enough redundancy such that a proof of a false statement will result in many errors. Additionally, a verifier has to read only a constant number of proof bits, but will still catch faulty proofs with high probability. These PCP encodings are used to construct fault-tolerant circuits for arbitrary Boolean functions.

The paper is structured as follows. The next section introduces background knowledge on Boolean circuits, ICs and PCPs. Furthermore, it revisits the adversary model of fault-tolerant circuits that was presented in [17]. Our attack model is explained in Sect. III, and consequently the trojan is defined. In Sect. IV the model of [17] is extended and the presented techniques are applied in order to counteract the trojan. Finally, in Sect. V our work is concluded; open questions and future research directions are also presented.

2 Definitions and Preliminaries

This section briefly introduces Boolean circuits, PCPs and basic fault-tolerant circuit constructions due to Gál and Szegedy [16, 17]. We consider combinational, synchronous circuits with a single output bit; for a thorough treatment we refer to [4, 16, 17, 44].

2.1 Boolean Circuits

A *Boolean circuit* C with N_I input bits and one output bit is defined as an acyclic directed graph, in which every vertex (called *gate*) corresponds to either

a Boolean function from the set $B = \{AND, OR, NOT\}$ or an input gate (of in-degree 0) labeled by one of the N_I input bits. Every edge represents a *wire*. One gate is labeled as output gate. A Boolean circuit C computes a *Boolean function* $f : \{0,1\}^{N_I} \to \{0,1\}$. We can assume that AND and OR gates have in-degree 2, NOT gates have in-degree 1, and all gates have a maximum out-degree of 2. A *combinational* (or *combinatorial*) *circuit* with N_I inputs and one output is a Boolean circuit C with N_I input bits $x_1, \ldots, x_{N_I}$ and one output bit z. A circuit is called *synchronous* (or *synchronized*) if for any gate g all paths from the inputs to g have the same length. Let C be a circuit and let g be any gate in C. The *depth* of g is the length of the longest path from any input to g and the *circuit depth* $D(C)$ is defined as the maximal depth of an output gate. The *size* $S(C)$ of a circuit C is defined as the total number of gates it consists of. The *i-th level* of a circuit consists of all gates with depth equal to i. A circuit C is synchronous if and only if every wire in C is between adjacent levels.

2.2 Integrated Circuits

An *integrated circuit (IC)* is composed of combinational logic having some inputs, which are output derived, cf. [31]. There may also be independent inputs. All or some of the outputs are fed back into the input. We denote by C the combinational logic, by $x_1, \ldots, x_I$ the independent inputs to C and by Y the output of C that is also fed back to C as input y. The output value Y, called *state vector*, may be *clocked* (*delayed*) and becomes an input y to C. Because of these possible delays the output Y and the corresponding input y may differ, and therefore, we denote them differently. W.l.o.g. we can assume that C is a synchronous circuit [44]. For simplicity, only ICs with one output bit are considered. This single output bit is fed back into the input.

2.3 Probabilistically Checkable Proofs (PCPs)

Conventional proofs have to be checked by a verifier step by step since a false theorem could be "proven" by a proof that contains only one incorrect clause. PCPs are more robust and a verifier can decide whether a proof is valid or not much more efficiently.

Let $L \in \mathbf{NP}$ be a set. A PCP verifier V receives an input x and gets access to a proof π. It is allowed to read some $\mathcal{O}(r)$ random bits, but at most $\mathcal{O}(q)$ bits from the proof. Denote by $V^\pi(x, \rho)$ the output of V on proof π, input x and randomness ρ. Then the class $PCP[r, q]$ is the set of all languages L, for which there exists such a verifier V with the following properties:

- (*completeness*) If $x \in L$, then there exists a proof π such that $\text{Prob}_\rho[V^\pi(x, \rho)\text{accepts}] = 1$.
- (*soundness*) If $x \notin L$, then it holds true for all proofs π that $\text{Prob}_\rho[V^\pi(x, \rho)\text{accepts}] \leq 1/2$.

The PCP theorem states that $\mathbf{NP} = PCP[\log n, 1]$ and is proved in [5, 14]. Improvements concerning proof length, query complexity and fault-tolerance

were achieved in [5, 9, 10, 14, 30]. In [9] PCPs of length $n \exp(\text{poly}(\log \log n))^2$ and query complexity $\text{poly}(\log \log n)$ are presented. Compared with this, in [10] the PCPs have length n $\text{poly}(\log n)$ and can be verified by $\text{poly}(\log n)$ queries. PCPs are closely related to so-called *locally testable codes*, on which we will not elaborate in this paper. However, they may be of interest in order to practically implement PCPs. In each case [9, 10] constructions for more efficient locally testable codes are presented.

2.4 Gál and Szegedy's Model of Fault Tolerant Boolean Circuits

The conventional scenario of fault-tolerant Boolean circuits deals with natural random errors. In order to address the more difficult scenario of non-random faults, Gál and Szegedy define a model, in which an adversary is allowed to maliciously choose at most a constant fraction of the gates at each level of the circuit to be faulty [17]. This means that gates may be destroyed or gate types may even change. The wires of the circuit are assumed to work correctly. This model is equivalent to the idea that a constant number of absolutely reliable gates is used for the last few levels of a circuit. Such an assumption is reasonable because more expensive and reliable hardware may be used for certain parts of a circuit. As tiny alterations to the IC should not affect its functionality, the circuit is required to compute a given function only "loose". However, if the IC is substantially altered, the loose version of the function is undefined and thus, the malicious modification should be detected. Gál and Szegedy define and prove the following fundamental insights towards their model.

Loose Computation Let $f : \{0,1\}^n \to \{0,1\}$ be a Boolean function and M be any computational device. We say that M *δ-loosely computes* f if the following holds:

1. If $f(x) = 1$, then $M(x) = 1$.
2. If $f(y) = 0$ for every y with $d(x, y) \leq \delta n$, then $M(x) = 0$,

where $d(x, y)$ denotes the Hamming distance between x and y. If an input x does not belong to one of the two categories, M can output an arbitrary value or no value at all. Combined with an appropriate error-correcting code, the loose computation behaves like the usual evaluation of a given function. For an error-correcting code E_n with codewords of length q_n and a function $f : \{0,1\}^n \to \{0,1\}$, we define the function $f \circ E_n : \{0,1\}^{q_n} \to \{0,1\}$ as follows:

1. $(f \circ E_n)(z) = 0$ for all z where z is not a codeword of E_n.
2. If $z = E_n(x)$, then $(f \circ E_n)(z) = f(x)$.

Lemma 1. *If the Hamming distance of any two codewords in an error-correcting code E_n with codewords of length q_n is at least δq_n and M is a computational device that computes $f \circ E_n$ δ-loosely, then $M(E_n(x)) = f(x)$ on any input x.*

It is known from coding theory, cf. [27], that there exist linear binary codes E_n with the required properties: The matrix of E_n can be polynomially computed in n. (This also means that the length of the codeword q_n is polynomial in n.) And the Hamming distance of any two codewords in E_n is at least δq_n for some small constant δ.

Faulty Gates and Circuits Formal definitions for the intuitive notions of errors, faulty gates and faulty circuits are now given. Denote by $g(x_1, x_2)$ the function, which gate g is supposed to compute on an inputs x_1 and x_2. A gate g is called *faulty* if its output is different from $g(x_1, x_2)$. Let C be a circuit with no faulty gates and let $\tilde{C}$ be a copy of the same circuit with possibly faulty gates. The output of a gate of $\tilde{C}$ is *incorrect* if it is different from the value computed by the same gate in C. Note that the output of a gate may be incorrect because the gate is faulty or because the inputs of the gates are incorrect. Let $\tilde{C}$ be a circuit with possibly faulty gates, but correctly working wires. If a gate g receives an incorrect input, at least one of the previous adjacent gates that provide the inputs to g computed an incorrect output. We say that a circuit C is γ-*faulty* if at most a γ fraction of the gates on each level is faulty. A circuit C for a function f is called *fault-tolerant* if it computes f δ-loosely even if it is γ-faulty.

ϵ-Halvers By the techniques presented by Gál and Szegedy it is possible to construct a synchronous fault-tolerant circuit for every symmetric function [17]. The construction for symmetric functions makes use of ϵ-*halvers* from Ajtai et al. [3]. An ϵ-*halver* is a bounded depth comparator network with the following property: For any set of the l smallest (largest) inputs, where $l \leq n/2$, at most ϵl elements will be among the last (first) $n/2$ outputs. Equivalently, an ϵ-*halver* can be defined as a halver (a network that separates the $n/2$ largest and the $n/2$ smallest inputs into two disjoint sets) that may misplace at most an ϵ fraction of the elements. For 0-1 inputs ϵ-halvers can be implemented by monotone Boolean circuits. Each comparator can be realized by an $AND-OR$ gate pair, where the AND gate computes the minimum and the OR gate computes the maximum of the two common input bits. We will consider ϵ-halvers with faulty gates and examine the number of faults and their propagation. At level d of an ϵ-halver with faulty gates the number of incorrect outputs is at most the number of incorrect outputs at level $d-1$ plus the number of faulty gates at level d. For an ϵ-halver we denote by L_1 (L_0) the number of 1's (0's) in the lower part of the output, and by U_1 (U_0) the number of 1's (0's) in the upper part of the output. The following two lemmas from [17] deal with the propagation of faults in ϵ-halvers.

Lemma 2. *Consider a γ-faulty ϵ-halver of depth c with m inputs and let the number of 0's in the input be a. If $a \geq m/2$, then $L_1 \leq \epsilon(m-a) + c\gamma m$ and $(a-m/2)-c\gamma m \leq U_0 \leq (a-m/2)+\epsilon m/2+c\gamma m$. If $a \leq m/2$, then $U_0 \leq \epsilon a + c\gamma m$ and $(m/2-a) - c\gamma m \leq L_1 \leq (m/2-a) + \epsilon m/2 + c\gamma m$.*

Lemma 3. *Consider a 0-1 string of length m, such that there are z 0's followed by $m - z$ 1's. Denote by z_L (z_U) the number of 0's in the lower (upper) part of this ordered string. Consider a γ-faulty ϵ-halver of depth c with m inputs and let the number of 0's in the input be a, such that $|a - z| \leq r$. Then $|U_0 - z_U| \leq r + (\epsilon + 2c\gamma)m/2$ and $|L_0 - z_L| \leq r + (\epsilon + 2c\gamma)m/2$. A similar statement holds for the number of 1's in the output.*

Construction for Symmetric Functions ϵ-halvers are building blocks of so-called *overwhelming majority functions* and *threshold functions*. These can be computed correctly on all defined inputs by γ-faulty circuits, and they are used to construct fault-tolerant circuits for symmetric functions [17].

The *overwhelming majority function* Maj_k^m has value 1 (respectively 0) if the number of 1's (0's) in the input is at least k, and it is undefined otherwise. Let $k > 3/4m$. Then for some $\gamma > 0$ there is a γ-faulty circuit of size $\mathcal{O}(m)$ and depth $\mathcal{O}(\log m)$ computing Maj_k^m correctly on every input belonging to its domain, cf. [17]. The *threshold function* Th_k^n has value 1 if and only if at least k of the n input bits have value 1. For any $\delta > 0$ there is a $\gamma > 0$ such that for any threshold function Th_k^n there is a synchronous circuit such that, cf. [17]: If an adversary destroys a γ fraction of the gates on every level (including the input level), the circuits still computes Th_k^n in a δ-loose manner. The size of the circuit is $\mathcal{O}(n)$. The depth of the circuit is $\mathcal{O}(\log n)$.

The above results constitute a central result of [17] showing that it is indeed possible to construct a synchronous fault-tolerant circuit for every symmetric function.

Theorem 1. *For any $\delta > 0$ there is a $\gamma > 0$ such that for any symmetric function f with n inputs there is a synchronous circuit with the following properties. If an adversary destroys a γ fraction of the gates on every level (including the input level), the circuit still computes f in a δ-loose manner. The size of the circuit is $\mathcal{O}(n)$ and the depth of the circuit is $\mathcal{O}(\log n)$.*

Arbitrary Boolean Functions Furthermore, synchronous fault-tolerant circuits can be obtained for any Boolean function in this model if certain error-correcting codes are applied. In addition to ϵ-halvers the fault-tolerant circuits for arbitrary Boolean functions are also based on techniques from the area of PCPs as defined by Arora et al. [5]. As we extend this construction from [17] for arbitrary Boolean functions to our new hardware trojan model, we will thoroughly present the adapted and extended proof later as our main result.

3 Hardware Trojans

The risk of using maliciously altered hardware has risen over the years and poses a real threat to critical computer systems. Since hardware is the lowest layer of a computer system, it controls everything running on it. Not surprisingly, is is

also very hard to detect and prevent such attacks on this lowest level of control, cf. [23, 26, 36]. Additionally, there is no convenient hardware solution in order to repair altered hardware afterwards. This means, there is no equivalent to the usual software updates. Due to the plethora of different hardware trojans, there is also a huge taxonomy [22, 37, 43] of different hardware trojans. Table 1 shows the taxonomy presented by Karri et al. [22] based on five attributes.

In order to focus on feasible problem solutions and to counteract bug attacks, this paper considers hardware trojans that might alter the proper functionality of an IC. The central idea is that small changes to the IC should not affect the functionality of a "resilient" IC, while more serious changes could be detected otherwise. It is obvious that massive design changes are clearly visible by a simple mask comparison with the IC die under investigation. So, even if an adversary knew of this resilient design method and the thresholds defining small and massive changes, he cannot destroy the IC arbitrarily too much. Hence, although the proposed solution seems to be static, it is virtually impossible for an adversary to circumvent it.

Let us consider the development cycle of an IC [12, 45] and the trustworthiness of its stages. In the specification phase the characteristics of the system, e.g., its expected function, are defined. Next, the specification is transformed into a chip design taking account of logical and physical requirements. These two steps take place in trusted research laboratories. During the fabrication phase the physical design layout is transformed into a set of mask layouts. A set of masks is then produced from these layouts and the wafers can be processed by the masks. We will consider the case when the specification and design processes are trustworthy, but the mask and fabrication phase are not. Since the mask and fabrication process are often outsourced to untrusted factories, our approach is well-founded [12, 13] in practice. At these or some later stages an adversary may have the opportunity to change the mask or even change the IC during production. It is likely that an adversary subtly tampers with the mask or the complex IC production process itself. This is due to the fact that major changes could be detected during the trusted testing phase, and will thus force the adversary to make only tiny deliberate alterations.

In this paper we consider adversaries that change the existing gates and wires of the combinational logic of an IC. Since an adversary who can tamper with a mask will not only change gates, but also wires, this is a very natural assumption and a logical extension of Gál and Szegedy's model [17]. Gates may be destroyed or gate types may be changed, and wires may be cut or swapped. This work does not address adversarial additions of logic to an IC directly as it is presented, e.g., in [23]. We hereby follow one of the presented scenarios from [12], in which only "destructive" alterations are taken into account. However, in current ICs there usually is a large surplus of logic elements anyway that could be exploited by an attacker [33]. These logic elements are not essential for the proper functionality of the IC. Instead they are used to implement built-in self-tests for debugging facilities like JTAG [33, 35] and they provide the last possibility to patch non-functional electronic devices after an erroneous fabrication. Suppose there was a

Table 1. Hardware trojan taxonomy according to [22]. Properties of our hardware trojan are highlighted in green.

Hardware Trojans

Insertion phase	Abstraction level	Activation mechanism	Effects	Location
specification	system level	always on	change the functionality	processor
design	development environment	triggered	downgrade performance	memory
fabrication	register-transfer level	▷ internally	leak information	I/O
testing	gate level	• time-based	denial of service	power supply
assembly and package	transistor level	• physical-condition-based		clock grid
	physical level	▷ externally		
		• user input		
		• component output		

secure IC design against added malicious logic like the hardware trojan in [23]. When it comes to practice, even in such a design model an attacker could take advantage of the above-mentioned additional test-components, which are integral features of the device by design. Our model is not limited to special components of a system. However, it might be reasonable to examine components with certain attributes, e.g., a trusted input, separately.

Formal Attack Scenario Consider an integrated circuit IC with synchronous and combinational logic C. An adversary is allowed to choose at most a small constant fraction $\alpha < 1$ of the gates at each level of C to be faulty. Additionally, he is allowed to choose at most a fraction $\beta < 1$ of the wires between all levels of C to be faulty. That means the adversary may choose at most a $\gamma := \alpha + \beta$ fraction of the gates at each level to produce *incorrect outputs*. Observe that the last gate and the outgoing wire that reconnects the single output of C with the input of C via the clock must work correctly since the adversary is only allowed to destroy a $0 \leq \beta < 1$ fraction of this connection. The gates and wires of the last few levels are very likely to remain unchanged since alterations would be too obvious and detected during testing phase.

Definition 1. *We call an IC* infected by a hardware trojan *if an adversary has tampered with it for some $\gamma > 0$ according to our model. After the adversary tampered with an IC the way it is described for our hardware trojan, its combinational circuit C is called* γ-faulty*.*

Definition 2. *A circuit C' for a function f is said to be* γ-resilient *if C' computes the function f even if C' has been tampered by a hardware trojan for some $\gamma > 0$. Additionally, an IC that contains C' is also called* γ-resilient*.*

The previously defined hardware trojan covers a large and important variety of hardware trojans. It can be classified according to the taxonomy of [22] in the following way (see Table 1):

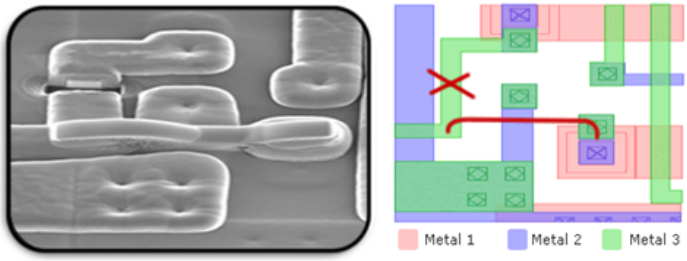


Fig. 1. An infected IC. The right side shows the intended layout with 3 metal layers for the correct design where malicious design alterations are highlighted by red lines in the correct layout. Left, an X-ray of the modified silicon is shown where a wire in metal 1 was cut (red cross in layout image) and was instead connected to another via (red bridge in layout). In terms of Table 1 this trojan was realized via FIB (Focused Ion Beam) post fabrication and testing, but before assembly and package, on the physical level, will be always on, changes the functionality and might affect all elements of the IC (processor, memory, I/O, power supply, and the clock grid).

- Insertion phase during development cycle: after design phase.
- Hardware abstraction level: gate level or below.
- Activation mechanism: always on. However, depending on the gates and wires, which are destroyed, the trojan might only make an impact if triggered by some event.
- Effects on the target device: change of functionality, denial of service.
- Location in a system: not limited to a special component.

Since this attack scenario is similar in its consequences to the one of the previous section, namely a γ-faulty circuit, we can apply similar techniques to the combinational logic of an IC. In the next section we will extend the adversary model from [17] to faulty wires in order to cover our model of hardware trojans. Thus, the combinational circuit is provided with more robustness and redundancy and is able to reliably compute the intended function. Having transformed a circuit C according to these techniques into a fault-tolerant circuit C', C' clearly is γ-resilient. If C' is the combinational component of an IC, this IC is γ-resilient as well. These γ-resilient ICs of course counteract the predefined hardware trojan, and hence, prevent computer systems from adversarial misuse.

4 Trojan-Resilient Circuit Construction

The model of Gál and Szegedy is limited to the alteration of gates [17]. As seen above, an adversary is interested in tampering with the wires as well. Hence, in our extended model an adversary is allowed to maliciously choose at most a constant fraction of the gates at each level and as well wires between adjacent levels to be faulty. All in all, we assume that at most a constant fraction of the gates on each level newly produces incorrect outputs because of faulty gates or

wires. The corresponding definitions of Sect. II are extended and stronger results are proven.

Faulty Wires and Circuits Let g_1 and g_2 be two adjacent gates connected by a wire w in a circuit without faults where g_1 provides an input to g_2. In a copy of the same circuit, which may have been tampered with by an adversary the wire w is called *faulty* if the output of g_1 is different from the input to g_2. Let C be a circuit with no faulty gates and let $\tilde{C}$ be a copy of the same circuit with possibly faulty gates and faulty wires. The output of a gate of $\tilde{C}$ is *incorrect* if it is different from the value computed by the same gate in C. Note that the output of a gate may be incorrect because the gate is faulty or because the inputs of the gates are incorrect. The inputs of the gate may be incorrect because the input wires are faulty or because the outputs of the previous gates are incorrect. At most a γd fraction of the outputs at level d of a γ-faulty circuit is incorrect. Note that this modified definition of a *γ-faulty* circuit does not contradict the definition from Sect. II, since in that model all wires work correctly. Hence, $\beta = 0$ and $\gamma = \alpha$ is the maximum fraction of faulty gates on each level.

ϵ-Halvers Like in Sect. II we will use ϵ-halvers. However, the ϵ-halvers may now contain faulty gates and faulty wires. Gál and Szegedy's lemmas 2 and 3 easily translate to ϵ-halvers with faulty gates and wires. This is implied by the proposition stated and proved in appendix A.

4.1 Construction for Symmetric Functions

We will construct γ-faulty circuits, which can correctly compute overwhelming majority functions and threshold functions for all defined inputs and base our techniques on [17].

Lemma 4. *Let $k > 3/4m$. Then for some $\alpha, \beta > 0$ and $\gamma := \alpha + \beta$ there is a γ-faulty circuit of size $\mathcal{O}(m)$ and depth $\mathcal{O}(\log m)$ computing Maj_k^m correctly on every input belonging to its domain. At most an α fraction of the gates on each level and a β fraction of the wires between all levels are faulty.*

Proof. The construction is based on *majority preservers*. In [7] this component is defined as a comparator network with m inputs and $m/2$ outputs that guarantees that if at least a given constant fraction greater than $1/2$ of the m inputs have the same value, this value appears in at least the same given constant fraction of the $m/2$ outputs. A majority preserver, which tolerates a small constant fraction of errors at each of its levels can be constructed by triplets of ϵ-halvers. An m-input ϵ-halver, denoted by *M-halver*, is applied to the m inputs. Next, one $m/2$-input ϵ-halver is applied to the upper part and another one is applied to the lower part of the output of the M-halver. They are called *U-halver* and *L-halver*. The output of this majority preserver consists of the upper part of the output of the L-halver and of the lower part of the output of the U-halver. A family of

ϵ-halvers of depth c with the same parameters ϵ and c for all input lengths exists for an appropriate choice of the constants ϵ and c.

Now let a be the number of 0's in the input of the majority preserver. Suppose $a \geq (3/4+c\gamma)m$. By $L_0(M)$ (respectively $U_0(M)$) the number of 0's in the lower (upper) part of the output of the M-halver is denoted. A similar notation is used for the U-halver and the L-halver. From lemma 2 it follows that $L_0(M) \geq m/2 - (\epsilon + 2c\gamma)m/2$ and $U_0(M) \geq m/4$. Denote by $I_0(L)$ (respectively $I_0(U)$) the number of 0's in the input of the L-halver (U-halver). As the M-halver is connected to the L-halver (U-halver) by $m/2$ wires, at most $\beta m/2$ wires may be faulty. Hence, $I_0(L) \geq m/2 - (\epsilon + 2c\gamma)m/2 - \beta m/2$ and $I_0(U) \geq m/4 - \beta m/2$. Applying lemma 2 to the L-halver and the U-halver yields $U_0(L) + L_0(U) \geq (1-2(\epsilon+2c\gamma+\beta))m/2$. If the number of 0's in the input of a majority preserver is at least $(3/4 + c\gamma)$ fraction of the input, then at least a $(1 - 2(\epsilon + 2c\gamma + \beta))$ fraction of its output is 0 as well.

Since several majority preservers will be combined by feeding the outputs of one as inputs to the next, we have to consider the faulty wires of these connections. Connecting an $m/2$-output majority preserver with an $m/2$-input majority preserver, it holds for the number of 0's $I_0(MP)$ in the input of the second majority preserver that $I_0(MP) \geq (1 - 2(\epsilon + 2c\gamma + \beta))m/2 - \beta m/2$. A similar statement can be shown for the number of 1's. Choose α, β and $\gamma = \alpha + \beta$ small enough so that $k > 3/4 + c\gamma$ and $-3\beta + 1 - 2(\epsilon + 2c\gamma) \geq 3/4 + c\gamma$. By feeding the outputs of one majority preserver as inputs to the next, we combine j majority preservers. This construction with $n/2^j$ outputs has the property that its overwhelming majority is the same as the overwhelming majority of the input. Once $n/2^j < \min\{1/\alpha, 1/\beta\}$, the computation can be finished by a small circuit that has fewer than $1/\alpha$ gates at each of its levels, fewer than $1/\beta$ wires between all levels and that computes the usual majority of its inputs. In this small circuit the adversary can destroy neither gates nor wires — according to our model. □

Theorem 2. *For any $\delta > 0$ there are $\alpha, \beta > 0$ and $\gamma := \alpha + \beta$ such that for any threshold function Th_k^n there is a synchronous circuit such that the following holds. If an adversary destroys an α fraction of the gates on each level (including the input level) and a β fraction of the wires between all levels, the circuit still computes Th_k^n in a δ-loose manner. The size of the circuit is $\mathcal{O}(n)$ and the depth of the circuit is $\mathcal{O}(\log n)$.*

Proof. As the circuit is supposed to compute the threshold function in a δ-loose way, the circuit has to output 1 if the number of 0's is $\leq n - k$, 0 if the number of 0's is $\geq n - k + \delta n$ and an arbitrary value otherwise. Assume that the input is correctly ordered and consider the set of bits at positions $n - k + 1, \ldots, n - k + \delta n$. Whenever the circuit has to output 1 (respectively 0) all these bits are 1 (respectively 0). Parameters t and l are chosen such that $ln/2^t \geq n - k$ and $(l + 1)n/2^t \leq n - k + \delta n$. t depends on n and δ, but not on k. Let T be the set of elements at positions $ln/2^t + 1, \ldots, (l + 1)n/2^t$ of the correctly ordered input, and let z_T be the number of 0's in T.

The construction consists of two components. First, a circuit, denoted by C_1, is constructed, which has n inputs and $n/2^t$ outputs. The number v of 0's

in its output should satisfy $|v - z_T| \leq (\epsilon + \alpha + 2\beta + 2c\gamma)n$, where ϵ and c are parameters of the ϵ-halvers that are used.

The second component is the construction from lemma 4, which computes the overwhelming majority function Maj_s^m of the outputs of C_1, where $m = n/2^t$ and $s = (1-2^t(\epsilon+\alpha+2\beta+2c\gamma))m$. In order to apply this lemma, it is necessary that $1 - 2^t(\epsilon + \alpha + 2\beta + 2c\gamma) \geq 3/4 + c\gamma$. Parameters ϵ, α, β and $\gamma = \alpha + \beta$ are determined by this inequality. Depending on the kind of ϵ-halvers that are used and depending on ϵ, c can be computed.

Circuit C_1 is a comparator network consisting of t γ-faulty ϵ-halvers each of depth c. The i-th ϵ-halver has $n/2^{i-1}$ inputs and $n/2^i$ outputs. Denote by $l_1, \ldots, l_t$ the binary representation of l.

- If $l_i = 0$, the input to the $i+1$-st ϵ-halver is the lower part of the output of the i-th ϵ-halver.
- If $l_i = 1$, the upper part is fed to the next ϵ-halver.

Denoted by a_0 is the number of 0's in the correct input. However, the adversary may destroy an α-fraction of the input gates of C_1. So, let a be the number of 0-valued input gates. Then $a_0 - \alpha n \leq a \leq a_0 + \alpha n$. Lemma 3 can be repeatedly applied with $r = \alpha n$. Since at most a β fraction of the wires between the i-th and $i+1$-st ϵ-halver may be faulty, these errors have to be considered by introducing the summand depending on β. It follows that $|v - z_T| \leq \alpha n + \sum_{i=1}^{t}(\epsilon + 2\beta + 2c\gamma)n/2^i \leq (\epsilon + \alpha + 2\beta + 2c\gamma)n$. □

Theorem 3. *For any $\delta > 0$ there is a $\gamma > 0$ such that for any symmetric function f there is a synchronous circuit with the following properties. If an adversary destroys an α fraction of the gates on each level (including the input level) and a β fraction of the wires between all levels and $\gamma := \alpha + \beta$, the circuit still computes f in a δ-loose manner. The size of the circuit is $\mathcal{O}(n)$ and the depth of the circuit is $\mathcal{O}(\log n)$.*

Proof. Every symmetric function f can be described by a binary string of length $n+1$. The i-th element of the string is 1 if and only if the value of f is 1 on inputs containing exactly i 1's. Suppose every set of consecutive 0's in this string has length $< \delta n$. In this case the δ-loose computation of f becomes trivial since the value 1 can be output for every input. Now consider the case that the string contains $h \geq 1$ sets of consecutive 0's of length $\geq \delta n$. Denote by (l_i, u_i) the i-th set. In order to compute f in a δ-loose way, it is sufficient to compute each $\neg\mathrm{Th}_{l_i}^n$ and each $\mathrm{Th}_{u_i}^n$. These $2h$ functions can be computed in parallel. They can all be computed in a δ-loose way by constructions from theorem 2 because the negated threshold functions may be computed analogously to presented techniques.

Set $\alpha := \tilde{\alpha}/2h$, $\beta := \tilde{\beta}/2h$ and $\gamma := \tilde{\gamma}/2h$, and choose $\tilde{\alpha}$, $\tilde{\beta}$ and $\tilde{\gamma}$ according to theorem 2. The parameters $\tilde{\alpha}, \tilde{\beta}, \tilde{\gamma}, \delta, \epsilon$ and c can be chosen in such a way that they have the same value for all $2h$ circuits, cf. [17]. By this choice the number of gates and wires at corresponding levels of all circuits is the same. If an attacker destroys at most an α fraction of the gates at each level, at most an $\tilde{\alpha}$ fraction of the gates at each level of any one of the $2h$ circuits will be

destroyed. There is a similar correlation between β and $\tilde{\beta}$ concerning the wires of the circuits. As for the parameters it is true that $h < 1/\delta < 1/2\gamma$ (from [17]) and $1/2\gamma \leq \min\{1/2\alpha, 1/2\beta\}$, the $2h$ values can be combined without having to consider any faulty gates or faulty wires. □

4.2 Arbitrary Boolean Functions

Like in the original model it is possible to design fault-tolerant circuits if the input is specifically encoded. The circuit constructions for symmetric Boolean functions are used as well as the aforementioned PCP techniques [5]. First, the Boolean circuit computing a given function is transformed into a PCP verifier consisting of several circuits. Then the resulting circuits are structured, and finally the techniques of fault-tolerant circuits for symmetric functions are used to recombine them. We now state the main theorem of our paper, which shows how to efficiently construct provable secure trojan-resilient circuits.

Theorem 4. *Let C be a Boolean circuit and let $f : \{0,1\}^n \to \{0,1\}$ be the corresponding Boolean function computed by C. There exists a code $E = E_C$ and a circuit C' such that C' computes $f \circ E$ in a δ-loose manner for every $\delta > 0$ even if an adversary destroys an α fraction of the gates at each level of C' and as well a β fraction of the wires between all levels of C'. Moreover, E and C' have the following properties:*

1. *$|E(x)| \leq q(|x|)$ for some polynomial q independent of C.*
2. *The Hamming distance $d(x, y)$ between any two codewords x and y of E is at least $\delta_0 |E|$ for some $0 < \delta_0 < 1$ independent of C (δ_0 is a function of δ).*
3. *$D(C') \leq O(\log S(C))$. This implies that $S(C')$ is polynomial in $S(C)$.*
4. *C' can be computed from C in probabilistic polynomial time and $E(x)$ can be computed from C and x in polynomial time.*

Before we present the proof of the above main theorem, we will simplify the PCP properties 1 (completeness) and 2 (soundness) into a single property. Consider the subset C^1 of tuples from $\{0,1\}^n$, for which C evaluates to 1 and the analogously defined subset C^0. Arora et al. presented a procedure to transform a circuit C into a family $\{C_i\}_{i \in I}$ of constant size circuits with input $(G(x), Y)$ [5]. Here $G(x)$ is an adequate error-correcting encoding of the input x to C, and Y is an advise string (both have length polynomial in $S(C)$). For this family $\{C_i\}_i$ we have:

1. For every $x \in C^1$ there is a Y such that for all $i \in I$: $C_i(G(x), Y) = 1$.
2. For every $x \in C^0$ and for every Y it is true that $\text{Prob}_{i \in I}[C_i(G(x), Y) = 0] \geq \epsilon$, for some ϵ.

Now, by applying techniques due to Valiant and Vazirani, see lemma 6 of the appendix, we can modify the circuit family C_i such that Y of property 1 is unique for every x and property 2 holds for every Y. Thus, properties 1 and 2 can be combined into a single property, where $|\cdot|$ denotes the binary length of the argument and $d(\cdot, \cdot)$ the Hamming distance between its arguments.

Lemma 5. *For every $x \in C^1$ there is, with high probability, a unique Y_x such that for all $i \in I : C_i(G(x), Y_x) = 1$. Also, for every $\delta > 0$ there exist $\epsilon > 0$ such that $Prob_{i \in I}[C_i(H, Y) = 0] < \epsilon$ implies that $d((H, Y), (G(x), Y_x)) \leq \delta \, |(H, Y)|$ for some $x \in C^1$.*

Remark 1. Originally, Gál and Szegedy [17] cited an unknown construction due to C. Lund and D. Spielman to guarantee a unique witness Y_x for the PCP completeness. However, their construction was neither clear nor described in [17]. Due to this lack we developed our own Y_x uniqueness construction. Details can be found in the appendices.

Proof (Proof of the theorem). The circuit C' will consist of several parts. Circuit C' should contain all members of the modified family $\{C_i\}_{i \in I}$ with the property that:

> For every $x \in C$ there is a unique Y_x such that for all $i \in I$: $C_i(G(x), Y_x) = 1$. Also, for every $\delta > 0$ there exists $\epsilon > 0$ such that $\text{Prob}_{i \in I}[C_i(H, Y) = 0] \leq \epsilon$ implies that $d((H, Y), (G(x), Y_x)) \leq \delta \, |(H, Y)|$ for some $x \in C^1$.

The members of the family should have disjoint inputs, which can be achieved by transforming $G(x)$ into $G'(x)$ and Y_x into Y'_x. $G'(x)$ is formed by repeating the bits of $G(x)$ and Y'_x is formed by repeating Y_x as many times as the number of times, for which they appear as input to some $\{C_i\}$, $i \in I$. By this construction each bit of $G(x)$ (respectively Y_x) will be repeated for the same number of times [17]. Next, the error-correcting code E is defined. If $x \in C^1$, $E(x) := (G'(x), Y'_x)$, and if $x \in C^0$, $E(x) := (G'(x), 0, \ldots, 0)$. Finally, C' is constructed in the following way. The family $\{C_i\}_{i \in I}$ is transformed into a family of synchronous circuits such that all of its members have disjoint inputs and all outputs are at the same level. In order to ensure that the groups of repeated bits consist of identical bits extra tests are added. Therefore, a bounded degree expander is introduced over each group of input bits that have to be identical, cf. [4, 19]. For every edge of these expanders equality is checked. This family is referred to as $\{D_j\}_{j \in J}$ and the circuits have to be synchronized with the members of $\{C_i\}_{i \in I}$ so that all outputs are at the same level. Furthermore, the sizes of I and J have to be the same within a constant factor. Clearly, $\wedge_{i \in I} C_i(H, Y) \wedge \wedge_{j \in J} D_j(H, Y) = f \circ E(H, Y)$. Since the construction for threshold functions can be used to compute the Boolean AND function, this technique is applied to combine the output bits of the circuits $C_i, i \in I$, and $D_j, j \in J$. Theorem 2 and the above Lemma 5 conclude the proof that C' computes $f \circ E$ in a δ-loose way even if an α fraction of the gates at each level of C' and a β fraction of the wires between all levels are destroyed for adequate α and β. □

5 Conclusions

This paper connects the fundamental work on fault-tolerance for Boolean circuits by Gál and Szegedy [17] with the very vibrant research area of hardware trojans.

Their work was enhanced in a natural way and extended to fully cover faulty wires as well. Developing their approach further, it was shown how to efficiently design fault-tolerant Boolean circuits in this extended model of deliberate faults on gates *and* wires. A corresponding and natural trojan model affecting the functionality of an IC, practically backed up by DARPA's problem challenge [12], was also defined. This hardware trojan fits very well into the above model of deliberate faults. Thus, in theory, we can design ICs that reliably compute their intended functions even if they are infected by a hardware trojan from our model. Our finding is highly relevant to industries, in which trust in computer systems is crucial, cf. [13]. As bug attacks require only very small errors, adversaries launching such attacks will only slightly alter the affected hardware as well. The overhead of additional circuits arising from our constructions might therefore be kept within reasonable limits. The presented design for trojan-resilient ICs is worthwhile to be developed further; especially from a practical point of view when it comes to the very elaborated PCP construction. First, the theoretical techniques of PCP constructions may be improved, as there have been many advancements in the fields of PCPs and related codes, cf. [9, 10, 14, 30]. Second, it is important to examine the practical PCP implementation, its complexity and costs. Just recently another approach was presented, which uses PCPs to solve a very practical problem [34] Indeed, this recent work put forward the thesis, that PCP usage can be put into practice. Thus, we are optimistic, that further achievements in Moore's law will enable us to spend many transistors on their sole usage of Boolean circuit resilience.

Bibliography

[1] S. Adee. The Hunt for the Kill Switch. *IEEE Spectrum*, 45(5):34–39, 2008.

[2] D. Agrawal, S. Baktir, D. Karakoyunlu, P. Rohatgi, and B. Sunar. Trojan detection using IC fingerprinting. In *Proceedings of the 2007 IEEE Symposium on Security and Privacy*, pages 296–310, 2007.

[3] M. Ajtai, J. Komlós, and E. Szemerédi. An O(n log n) sorting network. In *Proceedings of the fifteenth annual ACM Symposium on Theory of Computing*, volume 129, pages 1–9. ACM, 1983.

[4] S. Arora and B. Barak. *Computational Complexity: A Modern Approach*. Cambridge University Press, 2009.

[5] S. Arora, C. Lund, R. Motwani, and M. Sudan. Proof verification and hardness of approximation problems. In *Proceedings., 33rd Annual Symposium on Foundations of Computer Science*, pages 14–23. IEEE, 1992.

[6] S. Arora and S. Safra. Probabilistic Checking of Proofs. In *Proceedings of the 33rd IEEE Symposium on Foundations of Computer Science*, pages 2–13, 1992.

[7] S. Assaf and E. Upfal. Fault tolerant sorting network. In *Proceedings of 31st IEEE Symposium on Foundations of Computer Science*, pages 275–284. IEEE Comput. Soc. Press, 1990.

[8] M. Bellare and P. Rogaway. Optimal Asymmetric Encryption - How to encrypt with RSA (Extended Abstract). *EUROCRYPT 1994*, pages 92–111, 1995.

[9] E. Ben-Sasson, O. Goldreich, P. Harsha, M. Sudan, and S. Vadhan. Robust PCPs of Proximity, Shorter PCPs, and Applications to Coding. *SIAM Journal on Computing*, 36(4):889–974, 2006.

[10] E. Ben-Sasson and M. Sudan. Short PCPs with Polylog Query Complexity. *SIAM Journal on Computing*, 38(2):551–607, 2008.

[11] E. Biham, Y. Carmeli, and A. Shamir. Bug Attacks. In *CRYPTO'08*, pages 221–240, 2008.

[12] D. Collins. DARPA "TRUST in IC's" Effort. In *DARPA Microsystems Technology Symposium*, 2007.

[13] Defense Science Board Task Force. On High Performance Microchip Supply, 2005.

[14] I. Dinur. The PCP theorem by gap amplification. *Journal of the ACM*, 54(3):12, June 2007.

[15] R. Dobrushin and S. Ortyukov. Upper bound on the redundancy of self-correcting arrangements of unreliable functional elements. *Problems of Information Transmission*, 13(3):203–218, 1977.

[16] A. Gál. *Combinatorial methods in Boolean function complexity*. PhD thesis, University of Chicago, 1995.

[17] A. Gál and M. Szegedy. Fault tolerant circuits and probabilistically checkable proofs. *Proceedings of Structure in Complexity Theory. 10th Annual IEEE Conference*, pages 65–73, 1995.

[18] M. Hicks, M. Finnicum, S. King, M. Martin, and J. Smith. Overcoming an untrusted computing base: Detecting and removing malicious hardware automatically. In *Proceedings of the 2010 IEEE Symposium on Security and Privacy*, pages 159–172, 2010.

[19] S. Hoory, N. Linial, and A. Wigderson. Expander graphs and their applications. *Bulletin of the American Mathematical Society*, 43(04):439–562, Aug. 2006.

[20] Y. Ishai, A. Sahai, and D. Wagner. Private Circuits: Protecting Hardware against Probing Attacks. In *CRYPTO'03*, pages 462–479, 2003.

[21] Y. Ishai, M. Prabhakaran, A. Sahai, and D. Wagner. Private Circuits II: Keeping Secrets in Tamperable Circuits. In *EUROCRYPT 2006*, pages 308–327, 2006.

[22] R. Karri, J. Rajendran, K. Rosenfeld, and M. Tehranipoor. Trustworthy Hardware: Identifying and Classifying Hardware Trojans. *IEEE Computer Magazine*, 43(10):39–46, 2010.

[23] S. King, J. Tucek, A. Cozzie, C. Grier, W. Jiang, and Y. Zhou. Designing and implementing malicious hardware. In *Proceedings of the 1st USENIX Workshop on Large-scale Exploits and Emergent Threats*, pages 1–8, 2008.

[24] F. Koushanfar and A. Mirhoseini. A Unified Framework for Multimodal Submodular Integrated Circuits Trojan Detection. *IEEE Transactions on Information Forensics and Security*, 6(1):162–174, 2011.

[25] J. I. Lieberman. National Security Aspects of the Global Migration of the U . S . Semiconductor Industry, white paper, 2003.

[26] L. Lin, M. Kasper, T. Güneysu, C. Paar, and W. Burleson. Trojan side-channels: lightweight hardware Trojans through side-channel engineering. *Cryptographic Hardware and Embedded Systems-CHES 2009*, pages 382–395, 2009.

[27] F. J. MacWilliams and N. J. A. Sloane. *The Theory of Error-Correcting Codes*. North-Holland, 1977.

[28] N. Pippenger. On networks of noisy gates. In *Proceedings of 26th IEEE Symposium on the Foundations of Computer Science*, pages 30–36, 1985.

[29] S.C. Pohlig and M.E. Hellman. An Improved Algorithm for Computing Logarithms Over GF(p) and Its Cryptographic Significance. *IEEE Transactions on Information Theory*, 24(1):106–111, 1978.

[30] A. Polishchuk and D. Spielman. Nearly-linear size holographic proofs. In *Proceedings of the twenty-sixth annual ACM symposium on Theory of computing*, pages 194–203, 1994.

[31] D. A. Pucknell. *Fundamentals of Digital Logic Design: With VLSI Circuit Applications*. Prentice Hall, 1990.

[32] R.L. Rivest, A. Shamir, and L. Adleman. A Method for Obtaining Digital Signatures and Public-Key Cryptosystems. *Communications of the ACM*, 21(2): 120–126, 1978.

[33] K. Rosenfeld and R. Karri. Attacks and Defenses for JTAG. *IEEE Design & Test of Computers*, 27(1):2–13, 2010.

[34] S. Setty, R. McPherson, A. Blumberg, and M. Walfish. Olive: Making argument systems for outsourced computation practical (sometimes). To appear

in *Proceedings of the Network & Distributed System Security Symposium (NDSS) 2012*.
[35] C. Stroud, M. Ding, S. Seshadri, I. Kim, S. Roy, S. Wu, and R. Karri. A Parameterized VHDL Library for On-Line Testing. In *Proceedings of the 1997 IEEE International Test Conference*, pages 479–488, 1997.
[36] C. Sturton, M. Hicks, D. Wagner, and S. King. Defeating UCI: Building Stealthy and Malicious Hardware. In *Proceedings of the 2011 IEEE Symposium on Security and Privacy*, pages 64–77, 2011.
[37] M. Tehranipoor and F. Koushanfar. A Survey of Hardware Trojan Taxonomy and Detection. *IEEE Design & Test of Computers*, 27(1):10–25, Jan. 2010.
[38] L. Trevisan. *Lecture Notes on Computational Complexity.* Computer Science Division, UC Berkeley, 2004.
[39] L. Valiant and V. Vazirani. NP is as easy as detecting unique solutions. *Theoretical Computer Science*, 47(3):85–93, 1986.
[40] J. Von Neumann. Probabilistic logics and the synthesis of reliable organisms from unreliable components. In C. E. Shannon and J. McCarthy, editors, *Automata Studies*, volume 34, pages 329–378, Princeton University Press, 1956.
[41] A. Waksman and S. Sethumadhavan. Tamper evident microprocessors. In *Proceedings of the 2010 IEEE Symposium on Security and Privacy*, pages 173–188, 2010.
[42] A. Waksman and S. Sethumadhavan. Silencing Hardware Backdoors. In *Proceedings of the 2011 IEEE Symposium on Security and Privacy*, pages 49–63, 2011.
[43] X. Wang, M. Tehranipoor, and J. Plusquellic. Detecting malicious inclusions in secure hardware: Challenges and solutions. In *Proceedings of IEEE Int'l Workshop Hardware-Oriented Security and Trust (Host 08)*, pages 15–19. IEEE CS Press, 2008.
[44] I. Wegener. *The complexity of Boolean functions.* Wiley-Teubner, 1987.
[45] N. H. E. Weste and K. Eshraghian. *Principles of CMOS VLSI Design: A Systems Perspective.* Addison-Wesley, second edition, 1994.
[46] F. Wolff, C. Papachristou, S. Bhunia, and R. S. Chakraborty. Towards Trojan-Free Trusted ICs: Problem Analysis and Detection Scheme. *Design, Automation and Test in Europe (DATE) 2008*, pages 1362–1365, Mar. 2008.

A ϵ-Halvers with faulty gates and faulty wires

In Sect. IV we consider ϵ-halvers with faulty gates and faulty wires. We therefore examine the possible malicious alterations of wires between comparators of level $d-1$ and level d. Assume there are no faulty gates. An adversary may cut a wire near the output of a gate so that two input bits are affected. Formally, two wires are faulty. He may also cut a wire so that only one wire is faulty, and hence, one input bit is affected. In both cases at most one of the two output bits

is incorrect. Finally, an adversary could swap two wires, which means that two wires are faulty. This leads to lemmas similar to the ones from Gál & Szegedy as presented within Sect. II concerning faulty ϵ-halvers. Their lemmas 2 and 3 easily translate to (extended) ϵ-halvers with faulty gates and wires. This is implied by the following proposition.

Proposition 1. *1. In a comparator with correctly working gates, but faulty wires, the number of incorrect output bits is at most the number of faulty wires.*

2. At level d of an ϵ-halver with faulty gates and faulty wires the number of incorrect outputs is at most the number of incorrect inputs at level d plus the number of faulty gates at level d. The number of incorrect inputs at level d is at most the number of incorrect outputs at level $d-1$ plus the number of faulty wires between levels $d-1$ and d.

Proof. We prove the second statement. Level d of an ϵ-halver consists of disjoint comparators. A comparator in such an ϵ-halver is an AND-OR gate pair with common inputs. It is sufficient to show that the number of incorrect output bits of a comparator at level d is at most the number of its faulty gates plus the number of faulty wires feeding the comparator plus the number of incorrect output bits of the adjacent gates at level $d-1$. Hence, it is enough to deal with a single comparator with two inputs and two outputs at a time. From the previous observations and the first statement the proposition follows. □

B Valiant-Vazirani Uniqueness Reduction

We briefly describe the fundamental Valiant-Vazirani uniqueness reduction, cf. [39]. This is a probabilistic method that can be applied in order to reduce the number of satisfying assignments of a satisfiable formula to a single one. We will follow the notation of [38].

Lemma 6. *There is a probabilistic polynomial time algorithm that on input ϕ (a CNF formula) and an integer k outputs a formula ψ such that*

- *If ϕ is unsatisfiable then ψ is unsatisfiable.*
- *If ϕ has at least 2^k and less than 2^{k+1} satisfying assignments, then there is a probability of at least 1/8 that the formula ψ has exactly one satisfying assignment.*

The proof of the above lemma will easily follow from certain other definitions and results, which will be presented first.

Let $\mathcal{H}$ be a family of functions of the form $h : \{0,1\}^n \to \{0,1\}^m$. We say that $\mathcal{H}$ is a *family of pairwise independent hash functions* if for every two different inputs $x, y \in \{0,1\}^n$ and for every two possible outputs $a, b \in \{0,1\}^m$ we have

$$\mathrm{Prob}_{h\in\mathcal{H}}[h(x) = a \wedge h(y) = b] = 2^{-2m}.$$

For m vectors $a_1, \ldots, a_m \in \{0,1\}^m$ and m bits $b_1, \ldots, b_m$, define $h_{a_1,\ldots,a_m,b_1,\ldots,b_m} : \{0,1\}^n \to \{0,1\}^m$ as $h_{\mathbf{a},\mathbf{b}}(x) = (a_1 \cdot x + b_1, \ldots, a_m \cdot x + b_m)$, and let $\mathcal{H}'$ be the family of functions defined this way. Then $\mathcal{H}'$ is a family of pairwise independent hash functions. This construction implies the following simple result.

Lemma 7. *Let $T \subseteq \{0,1\}^n$ be a set such that $2^k \leq |T| < 2^{k+1}$ and let $\mathcal{H}$ be a family of pairwise independent hash functions of the form $h : \{0,1\}^n \to \{0,1\}^{k+2}$. Then if we pick h at random from $\mathcal{H}$, there is a constant probability that there is a unique element $x \in T$ such that $h(x) = 0$. Precisely,*

$$\mathrm{Prob}_{h \in \mathcal{H}}[\,|\,\{x \in T : h(x) = 0\}\,| = 1] \geq 1/8.$$

Now we can prove the famous uniqueness reduction due to Valiant and Vazirani.

Proof. In order to prove the above lemma, the following algorithm is presented. The algorithm randomly chooses vectors $a_1, \ldots, a_{k+2} \in \{0,1\}^n$ and bits $b_1, \ldots, b_{k+2}$ and produces a formula ψ that is equivalent to the expression $\phi(x) \wedge (a_1 \cdot x + b_1 = 0) \wedge \ldots \wedge (a_{k+2} \cdot x + b_{k+2} = 0)$. By construction, the number of satisfying assignments of ψ is equal to the number of satisfying assignments x of ϕ such that $h_{a_1,\ldots,a_{k+2},b_1,\ldots,b_{k+2}}(x) = 0$. If ϕ is unsatisfiable, then, for every possible choice of the a_i, ψ is also unsatisfiable. If ϕ has between 2^k and 2^{k+1} assignments, then the previous lemma implies that with probability of at least $1/8$ there is exactly one satisfying assignment for ψ. □

Consider the subset C^1 of tuples from $\{0,1\}^n$, for which a Boolean circuit C evaluates to 1 and the analogously defined subset C^0. Arora et al. presented a procedure to transform C into a family $\{C_i\}_{i \in I}$ of constant size circuits with input $(G(x), Y)$ [5] where $G(x)$ is an adequate error-correcting encoding of the input x to C, and Y is an advise string (both have length polynomial in $S(C)$). For this family $\{C_i\}_i$ we have:

1. For every $x \in C^1$ there is a Y such that for all $i \in I$: $C_i(G(x), Y) = 1$.
2. For every $x \in C^0$ and for every Y it is true that $\mathrm{Prob}_{i \in I}[C_i(G(x), Y) = 0] \geq \epsilon$, for some ϵ.

The techniques by Valiant and Vazirani described above guarantee the uniqueness of Y in property 1. Properties 1 and 2 can now be combined. $|\cdot|$ denotes the binary length of the argument and $d(\cdot,\cdot)$ the Hamming distance between its arguments.

Lemma 8. *For every $x \in C^1$ there is, with high probability, a unique Y_x such that for all $i \in I : C_i(G(x), Y_x) = 1$. Also, for every $\delta > 0$ there exist $\epsilon > 0$ such that $Prob_{i \in I}[C_i(H, Y) = 0] < \epsilon$ implies that $d((H, Y), (G(x), Y_x)) \leq \delta\,|(H, Y)|$ for some $x \in C^1$.*

Proof. The equivalence of property 1 and the first part of the lemma is obvious. Hence, we focus on property 2 and the second part. Assume for the

contrary that the second part is not true. Then there does not exist an $x \in C^1 : G^{-1}(H) = x$. Hence, there is an $x_0 \in C^0 : G^{-1}(H) = x_0$ and $\text{Prob}_{i \in I}[C_i(H, Y) = 0] = \text{Prob}_{i \in I}[C_i(G(x_0), Y) = 0] < \epsilon$. For the other direction assume that property 2 does not hold true. Then there exist $x \in C^0$ and Y such that $\text{Prob}_{i \in I}[C_i(H, Y) = 0] < \epsilon$. Since $x \in C^0$ and $H = G(x)$ it holds for all $x_1 \in C^1$ that $\text{d}((G(x), Y), G(x_1, Y_{x_1})) \geq \delta_0 |E| \geq \delta |(G(x), Y)|$. □

Committees

▷ **Steering committee**:

- Sylvain Guilley, *TELECOM-ParisTech, France.*
- Çetin Kaya Koç, *UCSB, USA.*
- David Naccache, *ENS, France.*
- Akashi Satoh, *AIST, Japan.*
- Werner Schindler, *BSI, Germany.*

▷ **Local committee**:

- Prof. Jean-Luc Danger, *TELECOM-ParisTech and Secure-IC, France.*
- Çetin Kaya Koç (General chair), *UCSB, USA.*

▷ **Programme Committee**:

- Alessandro Barenghi, *Politecnico di Milano, Italy.*
- Loïc Correnson, *CEA LIST, France.*
- Emmanuelle Encrenaz, *LIP6 / UPMC and CNRS, France.*
- Naofumi Homma, *Tohoku U., Japan.*
- Éliane Jaulmes, *SGDSN / ANSSI, France.*
- Gerwin Klein, *NICTA and UNSW, Australia.*
- Debdeep Mukhopadhyay, *IIT Kharagpur, India.*
- Svetla Nikova, *K.U.Leuven, Belgium* and *UTwente, Netherlands.*

- Bruno Robisson, *CEA LETA and ENSMSE, France.*
- Graham Steel, *LSV, France.*
- Mehdi Tibouchi, *NTT Secure Platform Laboratories, Japan.*

▷ **Sub-reviewers**:

- Rajat Subhra Chakraborty, *IIT Kharagpur, India.*
- Emmanuel Prouff, *SGDSN / ANSSI, France.*
- Thomas Roche, *SGDSN / ANSSI, France.*