Implementing Trojan-Resilient Hardware from (Mostly) Untrusted Components Designed by Colluding Manufacturers

Olivier Bronchain
UC Louvain
olivier.bronchain@uclouvain.be

Louis Dassy
ARM Ltd.
louis.dassy@gmail.com

Sebastian Faust
TU Darmstadt
sebastian.faust@gmail.com

François-Xavier Standaert
UC Louvain
fstandae@uclouvain.be

Abstract

At CCS 2016, Dziembowski et al. proved the security of a generic compiler able to transform any circuit into a Trojan-resilient one based on a (necessary) number of trusted gates. Informally, it exploits techniques from the Multi-Party Computation (MPC) literature in order to exponentially reduce the probability of a successful Trojan attack. As a result, its concrete relevance depends on (*i*) the possibility to reach good performances with affordable hardware, and (*ii*) the actual number of trusted gates the solution requires. In this paper, we assess the practicality of the CCS 2016 Trojan-resilient compiler based on a block cipher case study, and optimize its performances in different directions. From the algorithmic viewpoint, we use a recent MPC protocol by Araki et al. (CCS 2016) in order to increase the throughput of our implementations, and we investigate various block ciphers and S-box representations to reduce their communication complexity. From a design viewpoint, we develop an architecture that balances the computation and communication cost of our Trojan-resilient circuits. From an implementation viewpoint, we describe a prototype hardware combining several commercial FPGAs on a dedicated printed circuit board. Thanks to these advances, we exhibit realistic performances for a Trojan-resilient circuit purposed for high-security applications, and confirm that the amount of trusted gates required by the CCS 2016 compiler is well minimized.

Keywords

Hardware Trojans, Trojan-Resilience, Multi-Party Computation.

ACM Reference Format:
Olivier Bronchain, Louis Dassy, Sebastian Faust, and François-Xavier Standaert. 2018. Implementing Trojan-Resilient Hardware from (Mostly) Untrusted Components Designed by Colluding Manufacturers. In *The Second Workshop on Attacks and Solutions in Hardware Security (ASHES'18), October 19, 2018, Toronto, ON, Canada.* ACM, New York, NY, USA, 11 pages. https://doi.org/10.1145/3266444.3266447

1 Introduction

Context. While technology scaling has been the enabling factor for producing increasingly powerful micro- and nano-electronic devices, it has also made the manufacturing process of these devices a more difficult and expensive task. As a result, selling Intellectual Property (IP) cores and outsourcing the fabrication of Integrated Circuits (ICs) have become common in the semiconductor industry. For example, a recent study by Semico Research estimates that the IP market should exceed 8 Billion USD by 2020 [29]. Yet, despite being well motivated from a cost viewpoint, such a model raises important issues regarding the trust one can have in the manufactured ICs [16]. The latter are particularly critical for computing devices embedding cryptographic engines carrying out sensitive tasks. In this context, hardware Trojan attacks (where an adversary modifies an implementation during its manufacturing and hides a backdoor that may be used after deployment) have gained relevance over the last years: various "hard-to-detect" hardware Trojans have been described in the academic literature (e.g., [7, 12, 26]), real-world examples have been publicized (e.g., [2, 18, 30]), and several surveys have confirmed the difficulty to capture malicious manufacturers, due to the diversity of the attack vectors, activation mechanisms and payloads they exploit [8, 31, 36].

Taxonomy and countermeasures. Hardware Trojans can be classified in roughly two main classes. First, digital hardware Trojans for which the activation mechanism and the payload are communicated via the infected implementation's standard inputs and outputs. Typical examples include *cheat codes* (which trigger the malicious behavior when a specific input is provided to the device) and *time bombs* (which activate the Trojan after the device is executed a certain number of times). Second, physical hardware Trojans for which the activation mechanism and payload are communicated via physical side-channels. In the following, we will only consider digital hardware Trojans, which already cover a broad range of published threats (e.g., [7, 12]) and are very damaging since easily activated and exploited remotely.

As for countermeasures, we can also distinguish two main classes. First, reactive countermeasures which aim at detecting the hardware Trojans thanks to physical inspection. Typical examples include [1, 3, 25]. While conceptually able to detect any type of hardware Trojan, they are limited by their empirical nature and become less effective as the size of the infected circuit increases (or the size of the Trojan circuit decreases).[1] Second, preventive countermeasures which aim at making the insertion or exploitation of a Trojan more difficult for the adversary. Seed approaches for this purpose include *split manufacturing* [17] and *input scrambling* [34], which also come with limitations. For example, the sole use of split manufacturing has been shown to be circumvented by new threats [35], and input scrambling was originally limited to the protection of specialized cryptographic primitives with input homomorphisms.

As a result of this state-of-the-art, two recent (and independent) works initiated a more systematic study of distributed cryptographic protocols in order to better exploit these intuitively appealing principles. First, the work by Ateniese et al. takes advantage of Multi-Party Computation (MPC) in its classical setting and provides *security* as long as $m - 1$ among $m \geq 2$ outsourcing manufacturers are honest [5]. Concurrently, the work by Dziembowski et al. (on which we focus next) exploits MPC protocols in order to design a generic compiler that ensures a stronger property of *robustness* [10]. Informally, robustness is modeled by a game with two phases. First, in the testing phase, a tester checks whether some (untrusted) devices implement their correct specification. If the testing is passed, the adversary can then interact with the untrusted devices via a (small) *master* which is the only trusted part of the circuit. Robustness guarantees that for fresh inputs, the outputs produced in the second phase are identical to the outputs produced by the honest specification. It is parameterized by two quantities t and n, where t denotes the number of tests carried out and n is the number of online executions for which the output has to be correct. The CCS 2016 compiler requires $t \gg n$ and bounds the probability of an incorrect output by $\left(\frac{4n}{t}\right)^{\lceil \frac{\lambda}{2} \rceil}$, with λ the number of concurrently executed passively secure three-party protocols of the functionality to implement.

Our contribution. While the previous compiler is theoretically convincing, one important question left open by Dziembowski et al. is whether it is also practically-relevant? Answering this question boils down to the analysis of two main issues. First, is it possible to reach acceptable cost and performances for a Trojan-resilient circuit implementing a useful primitive, with high robustness guarantees? Second, is the amount of trusted gates required for the Trojan-resilient circuit sufficiently reduced so that ensuring their trust via detection-based approaches is indeed more realistic? (Typically, a natural requirement in this respect is that this amount of trusted gates of the master is at least significantly lower than the one needed to implement the target primitive).

We answer the first question positively by investigating a prototype Trojan-resilient block cipher circuit. For this purpose, we implement a passively secure three-party protocol that is the core of the CCS 2016 compiler on a Printed Circuit Board (PCB) combining four commercially available Field Programmable Gate Arrays (FPGAs), and optimize its performances in different directions. From the algorithmic viewpoint, we take advantage of a recent MPC protocol by Araki et al. proposed at CCS 2016 which allows improved throughput [4]. We also compare two different block ciphers (i.e., the standard AES Rijndael [9] and the bitslice cipher Mysterion [19]) and different representations for the AES S-box, in order to reduce the communication complexity of our Trojan-resilient circuits and gain understanding on the impact of such optimizations. We then develop a hardware architecture that balances the computation and communication cost of our Trojan-resilient circuits, and investigate the impact of the communication channel's bit size and frequency for this purpose. The total cost of the components used to implement a three-party protocol amounts to 120 USD. Based on our setup, we are able to execute a Trojan-resilient AES at a throughput of 2.3 [Mbps]. Thanks to our modeling of the communication interface, we extrapolate that this throughput could be improved by a factor 20 by using state-of-the-art tools for higher-frequency communications between the FPGAs (e.g., using so-called RocketIO's [24]).[2] An additional factor 2 is gained when implementing the bitslice cipher Mysterion rather than the standard AES. Overall, this implies that one could AES-encrypt "on-the-fly" 1 Gbit of data after an offline testing phase of 7 days using a Trojan-resilient circuit with $\lambda = 13$ three-party protocols, for a total cost of 640 USD, limiting the probability of incorrect outputs to 2^{-89}.

We answer the second question positively by minimizing and carefully analyzing the number of trusted gates in the master FPGA. For this purpose, we take advantage of the proposal made in [10] for the design of a secure (not robust) Pseudo-Random number Generator (PRG). Indeed, since the latter is only used for generating the shares of our three-party protocol (which are recombined by the master before the output is transmitted to a potential adversary), robustness is not required for this part of the Trojan-resilient circuits. The latter is naturally combined with the protocol of Araki et al. [4]. It allows limiting the master to a couple of XOR gates for the generation of correlated randomness from uniform randomness and for the shares' recombination, together with the logic needed for computing a majority over λ bits. Concretely, we then evaluate the number of Gate Equivalents (GEs) in the master. For this purpose, we first describe how to implement a majority function on the master with minimum area requirements. We then compare the GE count of our master with the number of GEs needed to implement an AES core. We observe that even in the (pessimistic) case where a single primitive (i.e., a block cipher) is implemented, we already reach significant gains. Namely, the master can be implemented in a few hundreds of GEs which is one order of magnitude smaller than the smallest AES cores published in the literature [11, 14, 23]. This allows improving the relevance of detection-based approaches to prevent hardware Trojans for this (trusted) part of the circuit. The latter factor naturally increases as the complexity of the system to implement increases. For example, one could design a Trojan-resilient circuit mixing several cryptographic primitives (of which the combined cost would increase) with the same minimum master.

2 Related works

We first mention a separate line of papers which considers a different setting, where an untrusted circuit proves that its execution is correct each time it performs a computation [5, 6, 33]. These papers exploit techniques from the Verifiable Computation (VC) literature and correspond to a different tradeoff between security and

[1] The latter limitation provides strong motivation for minimizing the size of the trusted part in the Trojan-resilient circuits investigated next, for which confidence has to be gained thanks to detection.

[2] The latter improvement would mostly requires a PCB design able to deal with higher communication frequencies, which is out of our scope here since it does not imply any specific research challenge.

trust. Namely, they cover an even broader class of hardware Trojans and achieve security for an arbitrary number of executions (unlike us who restrict the number of executions a-priori), at the cost of a more complex (trusted) master circuit and the impossibility to prevent "denial-of-service attacks". By these attacks, we mean that the hardware Trojan aims to stop the system (which is easily achieved in the VC framework by sending incorrect proofs). The latter are avoided by the compiler of Dziembowski et al. thanks to the testing phase and the robustness guarantees it brings (see Section 3.1.3).

More recently, a publication of Mavroudis et al. detailed the design of a Trojan-resilient system based on specialized cryptographic primitives [21]. We next explain the differences between this work and ours:

(a) Security vs. robustness. The work in [21] guarantees security while we focus on robustness, which we believe is often a more meaningful notion for protection against hardware Trojans. First, robustness is a stronger guarantee than security, because robustness ensures that the implementation satisfies its intended functionality. Hence, if a cryptographic implementation achieves robustness, then it also achieves security as long as the underlying cryptographic algorithm is secure. As discussed above robustness also implies resilience against denial-of-service attacks, while the security notion of [21] does not protect against them. Second, robustness is more general because it makes sense for non-cryptographic/security related functionalities. For instance, consider a hardware Trojan in a navigation system, which maliciously directs the system's owner to a wrong location. Finally, for many cryptographic primitives defining a meaningful security notion in the presence of hardware Trojans is hard. For example, consider an implementation of a public key digital signature scheme. If the implementation of the signing algorithm is corrupted by a hardware Trojan, then one plausible security guarantee may be that after interacting with the corrupted signing algorithm, it is not possible for the attacker to forge signatures with respect to the public key. However, such a guarantee does not say much about the usefulness of a trojan-infected signing implementation because a malicious implementation may output false signatures that do not offer any authenticity guarantee.

(b) Need of multiple facilities. The security level in the paper by Mavroudis et al. is based on the equation:

$$\Pr[\text{secure}] = 1 - \Pr[\text{error}]^k,$$

with $\Pr[\text{error}]$ being an estimation of the probability of a backdoored component and k a number of independent foundries building the system's components. So in order to (exponentially) increase the security, one requires multiple independent facilities (which is hard to ensure for large k's and in general inconvenient). The testing amplification in the compiler of Dziembowski et al. that we exploit allows exponential robustness increases with all circuits designed by a single manufacturer.[3]

(c) Qualitative vs. quantitative guarantees. The $\Pr[\text{error}]$ parameter in [21] depends on the design under investigation, and there is no commonly accepted way of evaluating it. By contrast, testing amplification provides easy-to-estimate quantitative guarantees for a number of correct executions of the functionality to implement.

[3] Two manufacturers if the master is designed separately.

(d) Trust assumptions. The security guarantees in [21] require that at least one implementation of the secure distributed computations to perform is honest. The latter (carrying out public key protocols) is a significantly larger circuit than the small trusted master we require.

So overall, the work of Mavroudis et al. rather corresponds to a different tradeoff between (better) performances and (weaker) security guarantees. In this respect, the following implementations provide an interesting comparison point in order to gauge the performance overheads that the stronger (robustness) guarantees of the generic compiler of Dziembowski et al. allows.

3 Background

The next descriptions use elements of a finite field $\mathbb{F}$, denoted with lowercase letters x. Field additions and multiplications are respectively denoted as $\oplus$ and $\odot$. When these field elements represent correlated randomness, we denote them with Greek letters (e.g., $\sum_{i=1}^{d} \alpha_i = 0$, with all α_i's but one picked up uniformly at random). When a value x is secret-shared, we denote its shares as $\tilde{x}_i$.

3.1 The CCS 2016 Trojan-resilient circuits

In this section, we recall the basics of the CCS 2016 Trojan-resilient circuits, starting with the threat model, following with the compiler used to build them and concluding with a short description of testing amplification and the robustness guarantees it provides.

3.1.1 Threat model The following solutions prevent any digitally-triggered hardware Trojan produced by a malicious manufacturer (or adversary) $\mathcal{A}$. In this setting, $\mathcal{A}$ receives the circuit specifications Γ and produces a device D supposedly implementing the specified functionality. No assumptions are made on the malicious circuitry inside D excepted that it is produced by a polynomially-bounded manufacturer.[4] This allows that the size of the Trojan circuitry can be larger than the one of circuit needed to implement the specifications. This threat model typically corresponds to the situation of a small country (or big company) with no IC facilities and sensitive information to manipulate / tasks to perform with the untrusted ICs it can assemble.

3.1.2 Generic compiler The generic circuit compiler TR (here TR stands for transform) applies to any deterministic circuit specification Γ a mapping from the initial functionality to a Hardware-Trojan resilient architecture based on three different components shown in Figure 1. Each of them has a different purpose and trust requirements.

The master, next denoted as M, is the only trusted piece of hardware required in the architecture to achieve Trojan-resilience. Therefore, it has to be built by an honest manufacturer or to be easily verifiable with (reactive) Trojan detection tools. The latter implies that M should contain a minimum number of gates. Concretely, it is the only component directly interacting with the adversary and is in charge of the secret sharing / reconstruction and the majority vote on the sub-circuits outputs.

The sub-circuits, next denoted as Γ_i, correspond to λ independent instances running on input x a semi-honest three-party computation protocol for $\Gamma(x)$. This implies that the shares of the three-party computation protocol run by Γ_i should be independent of the others. Note that the use of a (more efficient) semi-honest protocol

[4] The latter is needed to prevent that $\mathcal{A}$ can break the underlying cryptographic functionalities thanks to classical cryptanalysis.

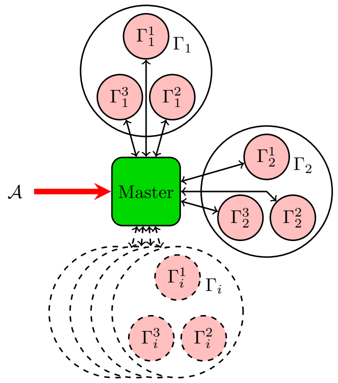


Figure 1: Trojan-robust circuit architecture: the trusted (resp., untrusted) components are in green (resp., in red).

(where corrupted parties can gather information out of the protocol, but do not deviate from its specifications) is in contrast with [5] which requires security against active attackers. Passive security is sufficient in our case thanks to the testing phase (which ensures the correct protocol execution up to some probability).

The mini-circuits are the three components of the sub-circuits Γ_i and are denoted $(\Gamma_i^1, \Gamma_i^2, \Gamma_i^3)$. They compute the target functionality based on a (semi-honest) three-party computation protocol. This requires that each (untrusted) Γ_i^j must have uniformly distributed inputs, which are shared by M. Each of the mini-circuits inside a sub-circuit is connected to other mini-circuits through the master. In practice, the production of all the Γ_i^j can be outsourced to a single polynomially-bounded manufacturer $\mathcal{A}$ that returns physical devices D_i^j supposedly corresponding to the mini-circuits specifications. So all the D_i^j are unstrusted hardware components.

3.1.3 Testing amplification and robustness Robustness against hardware Trojans ensures that a device tested up to t times will output correct values during the following n times of online executions. So it essentially corresponds to correctness for such a number of executions. More precisely, the ROB game (formally defined in [10]) proceeds as follows: the device D is produced by a polynomially-bounded adversary $\mathcal{A}$ and after a testing phase $\mathsf{T}(D)$ including up to t tests, $\mathcal{A}$ fails (i.e., ROB = 0) if D follows the specification Γ for n inputs selected by $\mathcal{A}$. Qualitatively, D is Trojan-robust if for adversary-controlled inputs x_i, $D(x_i) \neq \Gamma(x_i)$ (i.e., ROB = 1) only happens with negligible probability. As already mentioned, an interesting feature of this definition is that the untrusted devices are (with high probability) guaranteed to output correct values during the n usages, hence preventing denial-of-service attacks.

During the testing phase T, each device (i.e., mini-circuit) is tested independently a random number of times $t' \overset{r}{\leftarrow} \{1, \dots, t\}$, where t is a security parameter such that $t \gg n$. The same t' can be used for the mini-circuits of the same sub-circuits, but different sub-circuits need different t' values. For this purpose and for each test input, one simply verifies that the outputs of the devices under test correspond to their specification Γ_i^j. Note that the devices' execution under test or actual inputs should be indistinguishable in order to avoid trivial attacks detecting the testing procedure, which is exactly what the compiler working on uniformly random share guarantees. Besides, the testing must be performed by a trusted party. As a result, one obtains a simple robustness bound for a single device as given by $\Pr[\text{ROB} = 1] \leq \left(\frac{n}{t}\right)$.

Next, by exploiting redundancy at the sub-circuit level and letting M performing a majority vote on the λ outputs of the different Γ_i's, $\mathcal{A}$ can only win the ROB game if it is able to produce $\lceil \lambda/2 \rceil$ devices that trigger simultaneously during the n online runs. Hence, a robustness bound for the entire architecture is given by Equation 1:

$$\Pr[\text{ROB} = 1] \leq \left(\frac{4n}{t}\right)^{\lceil \frac{\lambda}{2} \rceil}. \tag{1}$$

It requires to test and combine $3 \cdot \lambda$ untrusted devices. Note again that the master performs a majority vote on the sub-circuits' outputs to amplify the robustness bound, but none of the mini-circuits (nor majority of mini-circuits) must be honest for this bound to hold.

3.1.4 The PRG case and security bounds The previous (generic, Trojan-robust) architecture can be easily turned into a Trojan-secure PRG [10]. For this purpose, just note that a value $y = x_1 \oplus \ldots \oplus x_l$ is pseudorandom if a single x_i out of l is pseudorandom. As a result, in contexts where one is only interested in the security of the PRG (which is a significantly weaker notion than robustness), it is possible to implement l PRGs (with different keys) in devices D_i and let the master sum their outputs. The resulting construction is a Trojan-secure PRG and the probability of an attack against the PRG security can be bounded by $(n/t)^l$. The latter construction is particularly interesting for the generation of the randomness needed for secret sharing the inputs of a Trojan-robust circuit (since there shares will never be output by the circuit and therefore are only required to be secure).

3.2 Target block ciphers

We next take block ciphers as a running example of circuits that we want to implement in a Trojan-robust manner. For this purpose, we will consider two case studies. First the AES Rijndael [9] which is a natural candidate to evaluate in view of its standard nature. Second, the bitslice cipher Mysterion [19] which we use to evaluate the extent to which having ciphers working in binary fields with a reduced number of multiplications is beneficial to the implementation of our three-party secure protocol. Both are Substitution Permutation Networks (SPN) and follow the 3-layer structure with a non-linear substitution made of S-boxes, a linear diffusion layer and a state permutation.

As usual in MPC, the key mixing and the permutation layers (which are linear operations) nearly come for free and most of the implementation efforts are spent on the (non-linear) S-box optimizations. In the AES case, this S-box is a multiplicative inverse in GF(2^8) (followed by an affine transformation). For Mysterion, the 4-bit S-box [32] is based on a simple bitslice description with only four AND gates.

4 Algorithmic improvements

The main goal of this paper is to discuss (and ideally confirm) the concrete relevance of the abstract specifications in the previous section, based on a meaningful case study. As mentioned in the introduction, this boils down to analyzing the performance level that can be reached using commercially available hardware (we will consider low-grade FPGAs) and evaluating the complexity of the trusted master circuit. A first natural step in this direction is to investigate algorithmic optimizations that can be used in order to

Algorithm 1 Generation of correlated randomness.

```
INIT: Trusted setup generates $(k_1, k_2, k_3, id)$
    • $\Gamma^1$ receives $(k_1, k_2)$ and $id$
    • $\Gamma^2$ receives $(k_2, k_3)$ and $id$
    • $\Gamma^3$ receives $(k_3, k_1)$ and $id$
GetCorrRandom:
    • $\Gamma^1$ computes $\alpha_1 = F_{k_1}(id) \oplus F_{k_2}(id)$
    • $\Gamma^2$ computes $\alpha_2 = F_{k_2}(id) \oplus F_{k_3}(id)$
    • $\Gamma^3$ computes $\alpha_3 = F_{k_3}(id) \oplus F_{k_1}(id)$
    • $\Gamma^1, \Gamma^2, \Gamma^3$ computes $id := id + 1$
```

Algorithm 2 Addition of secret shared values.

```
SecretShare: $v$ secret sharing
    • run GetCorrRandom
    • $\mathsf{M}$ sends $x_3 = v \oplus \alpha_3$ to $\Gamma^1$ that holds $\tilde{v}_1 = (\alpha_1, x_3)$
    • $\mathsf{M}$ sends $x_1 = v \oplus \alpha_1$ to $\Gamma^2$ that holds $\tilde{v}_2 = (\alpha_2, x_1)$
    • $\mathsf{M}$ sends $x_2 = v \oplus \alpha_2$ to $\Gamma^3$ that holds $\tilde{v}_3 = (\alpha_3, x_2)$
SharedAdd: shared addition $c = a \oplus b$
    • run SecretShare on $a$ and $b$
    • $\Gamma^1$ computes $\tilde{c}_1 = (\alpha_1 \oplus \beta_1, x_3 \oplus y_3)$
    • $\Gamma^2$ computes $\tilde{c}_2 = (\alpha_2 \oplus \beta_2, x_1 \oplus y_1)$
    • $\Gamma^3$ computes $\tilde{c}_3 = (\alpha_3 \oplus \beta_3, x_2 \oplus y_2)$
```

increase performances. In this section, we consider two solutions for this purpose. Namely, we first describe a recent MPC protocol by Araki et al. [4] which improves over the one initially proposed by Dziembowski et al. [10]. Second, we discuss implementation tricks used in the block cipher literature to speed up hardware / masked implementations [20, 22].

4.1 Three-party computation protocol

The three-party computation protocol with honest majority of Araki et al. allows reduced data transfers, a lower latency and a minimum amount of operations to be performed by the trusted master M (essentially the secrets' sharing and opening). Security and correctness proofs are available in [4]. We next describe its most important operations, namely the sharing with correlated randomness and the field multiplications. Note that for simplicity, the indexes of the sub-circuits are omitted in the rest of the paper whenever clear from the context, (since each sub-circuit carries out the same operations).

4.1.1 Correlated randomness generation The three-party computation protocol of Araki et al. exploits correlated randomness. The latter can be generated by implementing a pseudorandom function F_k (e.g., the AES Rijndael) in the mini-circuits and following Algorithm 1. First, an initialization phase is performed (e.g., at the beginning of the testing procedure) during which three secret keys k_i and a public seed id are generated by a trusted party. (3 different keys are required for each sub-circuit, which means that 3λ secret keys are needed for the entire architecture). Each of the mini-circuits Γ^i receives only two of those three secrets and finally holds (k_i, k_{i+1}, id), where $i + 1 = 1$ if $i = 3$. Then, the sub-circuits can generate an unlimited amount of correlated randomness by running **GetCorrRandom** and letting each mini-circuit Γ^i compute independently $\alpha_i = \mathsf{F}_{k_i}(id) \oplus \mathsf{F}_{k_{i+1}}(id)$, leading to values correlated such that $\alpha_1 \oplus \alpha_2 \oplus \alpha_3 = 0$.

No communication is involved during this online randomness generation, since the id can be updated locally, e.g., with a counter. Compared to the secure PRG exposed in subsubsection 3.1.4, the main difference is that the randomness generation is embedded inside the mini-circuits that will be used to perform the MPC operations of our target block ciphers (rather than requiring dedicated ones). This is possible because the robustness of our Trojan-resilient circuits will anyway be bounded by Equation 1, so having a better bound only for the randomness generation is an overkill in this case.

4.1.2 Secret sharing & reconstruction The secret sharing takes as input a value v and outputs three shares $\tilde{v}_i$ respectively held by Γ^i (Algorithm 2). In order to fulfill the requirements of the robustness bounds in subsubsection 3.1.3, each of those shares must individually reveal no information about the secret value v. For this purpose, all the operations which imply direct manipulations of v are performed by the master M.

Informally, the architecture first needs to generate random correlated values (as described in Algorithm 1). Each of the mini-circuits sends its random correlated value α_i to M. Once the master circuit receives all the α_i's, it performs the secret sharing locally. For this purpose, it first computes intermediate values $x_i = v \oplus \alpha_i$ which are distributed uniformly at random thanks to the α_i's. It then sends each x_i to the corresponding Γ^{i+1}. Note that no mini-circuit is able to retrieve the secret value v since they only hold shares $\tilde{v}_i = (\alpha_i, x_{i-1} = v \oplus \alpha_{i-1})$ with secret α_{i-1}. Finally a shared value v can be reconstructed by the trusted master by computing $v = \alpha_i \oplus x_i$ by asking only two mini-circuits for their shares (i.e., the protocol uses a 2-out-of-3 secret sharing).

4.1.3 Field operations Since using an additive secret sharing, a field addition $c = a \oplus b$ over $\mathbb{F}$ can be performed on the shares based on Algorithm 2. Informally, the values a and b are first secret shared. Next, each mini-circuit Γ^i independently computes a share $\tilde{c}_i$ of c based on its shares $\tilde{a}_i$ and $\tilde{b}_i$. Such an addition requires only two field additions and no data transfer (i.e., its cost is essentially negligible compared to multiplications).

A field multiplication $c = a \odot b$ involves three steps as described in Algorithm 3 and illustrated in Figure 2. Informally, the values a and b are first secret shared and three random correlated field elements γ_i are generated. Next the shares $\tilde{c}'_i$ are computed locally by each mini-circuit and correspond to a 3-out-of-3 sharing where the secret value is reconstructed by performing $c = \tilde{c}'_1 \oplus \tilde{c}'_2 \oplus \tilde{c}'_3$. To remain consistent with the secret sharing exposed in Algorithm 2, the value c should finally be shared in a 2-out-of-3 fashion. To this end, each of the mini-circuit Γ^i sends its $\tilde{c}'_i$ to Γ^{i+1}. Those transfers can be performed simultaneously since one mini-circuit does not require to receive data from another one to compute $\tilde{c}'_i$. Overall, once the operands a and b are shared, this field multiplication only requires a single set of correlated random values and a single field element transfer. This is a significant improvement compared to the three-party computation protocol proposed in [10].

4.2 Block ciphers optimizations for MPC

A consequence of the previous descriptions is that non-linear field multiplications dominate the performance overheads of our MPC protocol. Hence, a natural optimization is to minimize their number. As will be clear in the next sections, this allows reducing the amount of data transfers, which is the most expensive part of our implementations. In this section, we therefore evaluate the impact

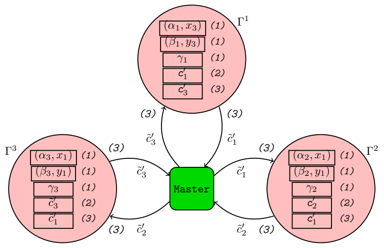


Figure 2: Secret multiplication

Algorithm 3 Multiplication of secret shared values.

```
STEP 1: shared multiplication $c = a \odot b$ 3-out-of-3
    • run GetCorrRandom and SecretShare on $a$ and $b$
    • $\Gamma^1$ computes $\tilde{c}'_1 = (\alpha_1 \odot \beta_1) \oplus (x_3 \odot y_3) \oplus \gamma_1$
    • $\Gamma^2$ computes $\tilde{c}'_2 = (\alpha_2 \odot \beta_2) \oplus (x_1 \odot y_1) \oplus \gamma_2$
    • $\Gamma^3$ computes $\tilde{c}'_3 = (\alpha_3 \odot \beta_3) \oplus (x_2 \odot y_2) \oplus \gamma_3$
STEP 2: 2-out-of-3 from 3-out-of-3 secret sharing
    • $\Gamma^1$ sends $\tilde{c}'_1$ to $\Gamma^2$      $\Gamma^1$ holds $\tilde{c}_1 = (\tilde{c}'_1 \oplus \tilde{c}'_3, \tilde{c}'_1)$
    • $\Gamma^2$ sends $\tilde{c}'_2$ to $\Gamma^3$      $\Gamma^1$ holds $\tilde{c}_1 = (\tilde{c}'_1 \oplus \tilde{c}'_3, \tilde{c}'_1)$
    • $\Gamma^3$ sends $\tilde{c}'_3$ to $\Gamma^1$      $\Gamma^3$ holds $\tilde{c}_3 = (\tilde{c}'_3 \oplus \tilde{c}'_2, \tilde{c}'_3)$
```

of the S-box representation in the AES Rijndael in this respect, and the additional gains that can be obtained by using a bitslice cipher such as Mysterion.

For the **AES Rijndael**, the only non-linear part is the S-box which is made of a multiplicative inverse in GF(2^8) and an affine transformation. One direct way to implement it is to perform the multiplicative inverse by computing 2^{254} with a minimal number of multiplications. As discussed in [28], this can be done with four GF(2^8) multiplications and 7 squaring (which are linear operations). This allows executing the S-box with four GF(2^8) elements (32 bits) to transfer between every consecutive mini-circuits per S-box. It leads to a total communication complexity of 512 bits for a single AES round (16 S-boxes) and 5, 120 bits for an entire encryption. The latter complexity can be easily improved thanks to composite field arithmetic, as usually exploited both for hardware and masked implementations of the AES Rijndael [20, 22]. In view of the similarity between the masking countermeasure against side-channel attacks and MPC, it is indeed natural to exploit those optimizations in our context [13]. Concretely, we perform the S-box thanks to composite field GF($(2^4)^2$) requiring 5 multiplication. This corresponds to 20 bit transfers per S-box and a total of 3, 200 for a full AES.

As for **Mysterion**, the S-box has an efficient bitslice representation with a minimum amount of multiplications (i.e., AND gates) since designed for masked implementations. It only requires to transfer 4 bits, leading to a communication complexity of 128 bits per round, and a total of 1,536 bits for the full cipher.

These optimizations are summarized in Table 1 which allows gauging the respective gains obtained thanks to S-box and cipher optimizations. Roughly, a factor two is gained by moving from a GF(2^8) S-box representation to a GF($(2^4)^2$) one for the AES, and another factor two is gained when moving from the AES to Mysterion. These figures omit the secret sharing and reconstruction which require two 128-bit transfers for both block ciphers.

Cipher	# of rounds	**bits per round**	**bits per enc.**
AES (GF(2^8) **S-box**)	10	512	5,120
AES (GF($(2^4)^2$) **S-box**)	10	320	3,200
Mysterion	12	128	1,536

Table 1: Block cipher's communication complexities.

5 Design choices

We now describe our design choices for the implementation of a Trojan-resilient block cipher. We start by recalling our optimization goals. Then, we expose how to implement the operations to be performed by the trusted master with minimum area requirements (which is our primary optimization goal). Eventually, we detail how to efficiently implement the operations to be performed by the mini-circuits and argue that this second optimization goal is dominated by the communication delays.

5.1 Optimization goals

In summary, our design choices are driven by security and performances. For security, we minimize the number of trusted gates required by the entire system. This point is crucial since it corresponds to a core hypothesis of the generic compiler of Dziembowski et al. As will be clear next, special attention has to be paid for this purpose, in particular regarding the majority function implemented in the master. As for performances, we aim to implement the mini-circuits in low-cost FPGAs (Xilinx Spartan family) and to obtain the best data throughput given this constraint. The latter essentially requires balancing the amount of logic resources used for computation within the mini-circuits in order to feed the communication interface with a sufficient throughput. Concretely, the latter will be the dominating factor in our performance evaluations (as usual in MPC) and therefore, our mini-circuit implementations will be primarily optimized for small size. In this context, an important parameter of our designs is the communication bus size, that we denote as N, and its frequency that we denote as f_{bus}. This means that the communication bus can send and receive N bits during a single bus clock cycle in a full duplex fashion resulting in a data throughput of $N f_{bus}$[bps]. The performances of the field operations will be evaluated according to data throughput.

5.2 Minimal trusted components

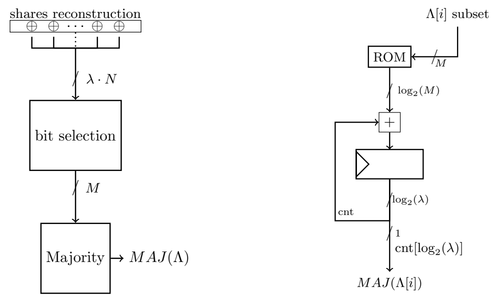
We next describe the components to be implemented on the trusted master, and how to minimize their area. For this purpose, a natural strategy is to process the bits received by the sub-circuits sequentially by the bus of size N. While this comes directly for most operations, we argue why it is also acceptable when computing a majority function on the master. We then evaluate the gate count of the operations exposed in section 4.

5.2.1 On bit-wise majority The construction of Dziembowski et al. [10] relies on a trusted majority gate, where majority here means that the most common value among the λ output values is returned by the master. In general the latter cannot simply be implemented by taking the bit-wise majority on the bits representing the output values. Instead we would need to implement a rather costly algorithm that first stores the outputs of the λ sub-circuits and then outputs the most common value among these λ outputs. Such an implementation would result into a large circuitry for the master. Yet, while indeed in general taking majority on bits does not yield the same value as taking the majority on (possibly large) output values, we argue that in the construction of [10] we can nevertheless rely on this simple approach. To understand why this is the case, we first recall that the security analysis of [10] guarantees that after a testing phase the output of all the sub-circuits is equal to the output of the trusted specification. Clearly, this means that at this point taking bit-wise majority is the same as returning the most common output. During the real execution some of the sub-circuits may change their outputs (when a hardware Trojan gets active), but according to the security analysis of [10] this happens for each of the λ sub-circuits independently. The security analysis then "counts" for each real execution the number of sub-circuits that change their outputs (i.e., switch from the correct output to a value that differs from the output of the specification). It is easy to see that as long as more than $\lambda/2$ of these sub-circuits do not switch their output, bit-wise majority is the same as taking the majority on the outputs of the sub-circuits. Since Lemma 4 in [10] guarantees that this happens with negligible probability, it means that robustness remains intact when returning bit-wise majority instead of the most common value among the λ outputs of the sub-circuits.

5.2.2 Secret sharing The secret sharing proposed in Algorithm 2 can easily be implemented in hardware. As a reminder, the master performs the masking of a value $v \in \mathbb{F}$ thanks to a bit-wise XOR with random correlated values α_i. Those last values are generated by the corresponding Γ^i that first must send its α_i to M. The hardware architecture can directly performs that masking operation by directly XORing a correlated value α_i with v only using combinatorial logic. The three different shares retrieved in that way are computed thanks to 3 XOR gates operating on N bits. Those XOR gates are of size N since the mini-circuits are only able to send N bits of v at the same time. The obtained shares are then forwarded to the mini-circuit it belongs to (still only using combinatorial logic). As a result, this sharing has a throughput of Nf_{bus}[bps], equal to the bus throughput. Performances can be improved by increasing N, at the cost of linearly increasing the size of the master M (i.e., the number of XOR gates in the master is proportional to $3 \cdot N$).

5.2.3 Field operations Field operations are straightforward to implement. A field addition does not require any data transfer (nor imply additional circuitry inside the master M). During a field multiplication, M simply needs to forward a field element between the mini-circuits, involving no gates. Hence, the throughput of a field element multiplication is directly linked to the bus ability to transfer data and the ability of a mini-circuits to provide fresh data. As discussed in subsection 5.3, the multiplication throughput is therefore bounded by the bus throughput Nf_{bus}[bps].

5.2.4 Secret opening Similarly to the secret sharing, the secret reconstruction can efficiently be performed in hardware. The master M has to unmasked shared values by XORing two shares hold by different mini-circuit since the secret sharing is performed in a two-out-of-three fashion. The share opening is performed directly by the master thanks to a XOR gate of size N. Similarly to the case of secret sharing, secret opening can be performed at the throughput of the communication interface. Note that the operations described above do not require any register in M. The trusted master simply operates as a routing module between the mini-circuits and only performs combinatorial operations. Hence, the interface can be



(a) Generic circuit. **(b) Serial majority gate.**

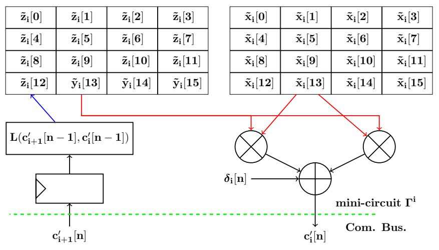
Figure 3: Majority vote inside the trusted master M.

simply implemented as a wire entering M with no other logic. (We discuss in Section 6.3.2 how to maintain a good f_{bus} in this case).

5.2.5 Majority vote The last operation that the trusted master must perform is the majority vote between all the reconstructed values. Since the secrets are opened "on-the-fly", we propose to do the same for the majority vote (as justified in subsubsection 5.2.1). The bit-wise majority can be performed as illustrated in Figure 3a based on a bit selection and a serial majority vote module. Intuitively, the bit selection feeds the majority gate serially with the required bits. More precisely, the bit selection module serially outputs subsets $\Lambda[i] := \{v_1[i], \ldots, v_\lambda[i]\}$ with $v_j[i]$ being the ith bit of the value v_j outputted by a sub-circuit Γ_j. The main parameter of this majority vote is the value M which is the number of bits processed at the same time.

From an implementation point-of-view, the bit selection module must select subsets of size M in the $\lambda \cdot N$ bits outputted by all the sub-circuits. Those reconstructed bits are directly obtained thanks to combinatorial XOR gates. This can be achieved with M independent $(\lambda \cdot N/M)$-to-1 multiplexers, the latter being synthetized from $M \cdot (\lambda \cdot \frac{N}{M} - 1)$ 2-to-1 multiplexers. The control signal for those multiplexers is generated thanks to a counter on $\log_2(\lambda \cdot N/M)$ bits involving an adder and registers. The serial majority voter is based on a counter that accumulates how many ones are contained in $\Lambda[i]$. To do so, the accumulation is performed by observing the number of ones contained in each subsets of $\Lambda[i]$ it receives, which is achieved thanks to a table look up into a ROM. Once all the subsets have been received, the MSB of the counter can be output, since it is high if the number of ones is higher or equal to $\lambda/2$. Note that the accumulator must be of size $\log_2(\lambda)$ allowing to count up to λ. The ROM involved in the table look ups stores the hamming weight for any subset of size M requiring a memory of $2^M \cdot \log_2(M)$ bits.

In term of performances, the majority voter requires λ/M cycles internal to the chip to perform the majority on a set $\Lambda[i]$ containing λ bits. This observation leads to a throughput for the majority vote of $\frac{M}{\lambda} f_i$[bps] with f_i being the internal frequency of the majority vote gate. It appears that the time for reconstruction depends on λ. For large λ's, the majority vote throughput may be the bottleneck of the reconstruction process. For example, it is the case if this throughput is smaller than the one of the secret opening such that $\frac{M}{\lambda} f_i \leq Nf_{bus}$. In that case, the majority gate cannot perform the vote as fast as the reconstruction module provides inputs. Note that this throughput limitation just impacts the reconstruction phase and not the entire encryption process. Note also that solutions exist



$\tilde{z}_i[0]$ $\tilde{z}_i[1]$ $\tilde{z}_i[2]$ $\tilde{z}_i[3]$
$\tilde{z}_i[4]$ $\tilde{z}_i[5]$ $\tilde{z}_i[6]$ $\tilde{z}_i[7]$
$\tilde{z}_i[8]$ $\tilde{z}_i[9]$ $\tilde{z}_i[10]$ $\tilde{z}_i[11]$
$\tilde{z}_i[12]$ $\tilde{y}_i[13]$ $\tilde{y}_i[14]$ $\tilde{y}_i[15]$
$\tilde{x}_i[0]$ $\tilde{x}_i[1]$ $\tilde{x}_i[2]$ $\tilde{x}_i[3]$
$\tilde{x}_i[4]$ $\tilde{x}_i[5]$ $\tilde{x}_i[6]$ $\tilde{x}_i[7]$
$\tilde{x}_i[8]$ $\tilde{x}_i[9]$ $\tilde{x}_i[10]$ $\tilde{x}_i[11]$
$\tilde{x}_i[12]$ $\tilde{x}_i[13]$ $\tilde{x}_i[14]$ $\tilde{x}_i[15]$
$L(c'_{i+1}[n-1], c'_i[n-1])$
$\delta_i[n]$
mini-circuit Γ^i
Com. Bus.
$c'_{i+1}[n]$
$c'_i[n]$

Figure 4: Multiplication followed by linear layer.

to perform those operations in a single cycle (either by using monotone Boolean equations [15] or by implementing fixed threshold Hamming weight comparisons [27]), but these propositions result in a larger area for the λ's used in section 6.

5.2.6 *Control logic outsourcing*

Since the logic involved in the computations inside M is relatively small, the control modules may have a significant influence on the final size of M. Therefore, we outsource the control logic dedicated to a sub-circuit in its mini-circuit. This is feasible since the operations that a sub-circuit perform are part of its functionality (and therefore tested in order to obtained the bounds of Equation 1). The later is in fact the concrete counterpart of the proposal in [10] where each mini-circuits sends commands to the others, implying that the operations performed are controlled by the untrusted parties.

5.3 Mini-circuit design

The hardware involved in mini-circuit Γ^i for multiplications on a block cipher state is presented Figure 4 with $\tilde{x}_i[n]$ and $\tilde{y}_i[n]$ respectively denoting the shares of x and y it holds. The architecture processes sequentially each of the 16 bytes of the state. During a cycle n, shared bytes at index n are accessed in two memories. The value $c'_i[n]$ from Algorithm 3 is then computed on those shares using two multipliers in parallel. The two output values are then XORed together with an additional correlated random value $\delta_i[n]$ to retrieve $c'_i[n]$. This byte is directly sent to the communication bus. During the same cycle, this mini-circuit receives the corresponding share $c'_{i+1}[n]$ output by another mini-circuit. This is done without latency since the master only forwards messages from Γ^i to Γ^{i+1}. This received byte is stored in a register. During the same cycle, the linear operation is performed on the previously multiplied shares by storing $L(c'_{i+1}[n-1], c'_i[n-1])$ in memory at index $n-1$. In Figure 4, this is done by erasing previously stored $y_i[n-1]$ values if those are not useful anymore, hence allowing to save memory.

In terms of performances, since this architecture is able to provide a byte at each of its internal clock cycles, its throughput is $8f_i$[bps]. The latter naturally becomes suboptimal if the throughput for computing $c'_i[n]$ is lower than the throughput of the communication interface. In this case, we can simply duplicate the hardware presented in Figure 4 B times in order to fulfill Equation 2, ensuring that the internal throughput is higher than the one of the bus. More precisely, the latter requires to multiply by a factor B the circuit dedicated to the computation of the $c'_i[n]$'s. This represents $2 \cdot B$ multipliers and B additional XOR gates. The obtained architecture can be viewed as a parallel version of Figure 4 with:

$$Nf_{bus} \leq B \cdot 8f_i. \tag{2}$$

In addition, the correlated randomness generator must provide fresh randomness for each multiplication. For this purpose, we rely on a loop implementation of the AES producing 128 bits of randomness in 11 cycles, which has sufficient throughput to fulfill Equation 2.

6 Implementation

In order to assess the system practicality of the Trojan-resilient compiler of Dziembowski et al., we now propose a dedicated board which includes four Spartan-6 Xilinx FPGAs. We first describe such a board and the security considerations of the design. The small area required by the trusted M is then evaluated by comparing its implementation to a small AES design [23] (in terms of GEs). We finally discuss the performances of the dedicated board as well as an extrapolation to more efficient communication interfaces. Typical values for the area and security of such an extrapolated design are proposed to highlight the concrete feasibility of a complete Trojan-resilient system.

6.1 Board design

Our demonstration board includes four Xilinx Spartan-6 LX9 FPGAs with one of them being the master and the three other ones corresponding to mini-circuits. Those are a low-cost FPGAs with an unitary price of around 17USD, including 1430 slices and 9152 logic cells. The board has a total cost of 120 USD including additional peripheral components. The communication interface of each mini-circuit is an SPI interface with size 8 using I/O standard LVCMOS33. The master is interfaced to a computer thanks to an UART dedicated module. This board has been designed according to the guidelines of [10], and each mini-circuit can only communicate with the master. They have their own power supply chain and the JTAG chain used to program the FPGAs has to be disconnected after programming thanks to jumpers' inserted between the mini-circuits. This guarantees that no physical link exists between the mini-circuits.

6.2 Area (and trust) requirements

In this section we discuss the trusted master's area requirements according to the architectural parameters N, λ and the majority vote parameter M. We first discuss the area required to support the three-party computations and then evaluate the complexity of the majority vote. The area results are be compared with [23] which implements a very compact AES hardware module in a 0.18μm technology, with an area of 2,400 GEs. Hereunder we show that the trusted master M can be implemented in a few hundreds GEs with architectural parameters leading to good robustness guarantees.

6.2.1 *MPC implementation*

Since each sub-circuit runs independently and simultaneously an instance of three-party computation protocol, their hardware requirements can also be evaluated independently. For this reason, we next focus on the area required by a single sub-circuit: the area for λ instances is obtained by duplicating the circuitry λ times. The number of trusted gates per sub-circuit depends on N according to section 5. Roughly, by increasing the bus width, additional trusted gates are required to process signals simultaneously. This implies that the size of the master linearly grows with N and λ. More precisely a single sub-circuit with a bus width $N = 1$ requires 3 XOR gates for secret sharing, 1 XOR gate for secret reconstruction and 3 multiplexers to select the output values (between masked ones and forwarded ones). This results in an area of 16 GEs, leading to a total area for the three-party computation protocol inside M equal to $N \cdot \lambda \cdot 16$ GEs.

λ	N	M	**Bit select.** [GEs]	**Serial Maj.** [GEs]	**Total** [GEs]
8	1	1	61.3	45	106.3
		2	44	52	96
		4	24.3	114.9	139.2
	2	2	77.6	52	129.6
	4	4	110.2	114.9	225.1
16	1	1	95	60	155
		2	77.6	67	144.6
		4	58	130	188
	2	2	130	67	197
	4	4	199.8	130	329.8

Table 2: Area required for a trusted majority vote.

6.2.2 Majority vote area The majority vote structure exposed in Figure 3a is the most area-consuming building block of the master M. Its cost depends on the architectural parameters λ, N and M (the latter being the number of bits that the serial majority voter can process simultaneously), leading to the areas reported in Table 2. Based on these results, a first observation is that the M parameter sets a tradeoff between the area cost of the bit selection module and the serial majority voter. That is, by increasing M the area required for bit selection is reduced while the ROM size increases. For example, by doubling M from 1 to 2 the area of the bit selection is reduced more than the majority vote grows, hence resulting in a smaller area. However, by doubling M from 2 to 4 the area of the serial voter becomes larger due its ROM size growing according to $2^M \cdot \log_2(M)$. A second observation is that increasing λ and N always increases the number of gates required, since either the majority vote operates on large data or the bit selection is performed on increased amount of bits.
This table is particularly important to confirm the minimized size of the trusted master. It illustrates that for a range of relevant parameters, the master can be implement in a few hundreds of GEs. It is an interesting scope for further research to investigate the extent to which relevant hardware Trojans can be inserted in such a small master, and whether existing (e.g., side-channel based) detection techniques could (not) easily spot them.

6.3 Performances and security evaluations

We conclude the paper by evaluating the performances of our system. First we describe the data throughput obtained on the demonstration board. Then we extrapolate our results based on a better communication interface.

6.3.1 Demonstration board evaluation As mentioned in section 5, the performances of our Trojan-resilient hardware are defined by the throughput of its different components. According to Equation 2, the throughput achieved by the mini-circuits must be higher than the communication interface one. In the case of the demonstration board, the SPI bus only allows frequencies up to 10[Mhz] giving a throughput of $N \cdot 10$[Mbps]. The design of the mini-circuit proposed in section 5 can run up to 80[MHz] resulting in an internal throughput of 640[Mbps] for a parallelism factor of $B = 1$. In this case no parallelism is required since the mini-circuits' internal throughput remains higher than the communication interface one. Given those physical parameters, the performances shown in subsubsection 6.3.1 are obtained. Depending on the bus width, the throughput changes from 354 to 2, 392[kbps] for the AES, and from 625 to 3, 471[kbps] for Mysterion. The gain obtained by moving from the AES to Mysterion highlights that the ciphers' non-linear parts and the corresponding data transfers are the main explicative factor the final throughputs in the table.

Bus		**Mini-circuit**		**AES**		**Mysterion**	
Throug. [Mbps]	**N** [bit]	**Throug.** [Mbps]	**B**	**Cycles** [cycle]	**Through.** [kbps]	**Cycles** [cycle]	**Through.** [kbps]
10	1	640	1	14,430	354	8,195	625
20	2	640	1	7,516	681	4,355	1,175
40	4	640	1	3,932	1,302	2,435	2,102
80	8	640	1	2,140	2,392	1,475	3,471

Table 3: Performance evaluations ($\lambda = 1$).

Bus		**Mini-circuit**		**AES**		**Mysterion**	
Throug. [Gbps]	**N** [bit]	**Throug.** [Mbps]	**B**	**Cycles** [cycle]	**Through.** [Mbps]	**Cycles** [cycle]	**Through.** [Mbps]
1.5	1	640	1	432	23.7	224	457
1.5	1	1,920	3	180	55	96	107
3	2	3,200	5	92	111	48	214
6	4	6,400	10	46	222	24	428

Table 4: Extrapolated performance evaluations ($\lambda = 1$).

6.3.2 Extrapolation Concretely, the main limitation of our demonstration board is its handmade communication interface. In order to evaluate the throughput that could be obtained with better (though still standard) facilities (without increasing the bus size N which is detrimental to security), we next propose an extrapolation to state-of-the-art communication interfaces. Namely, we propose to increase the communication interface throughput not by increasing the bus width but by increasing the bus frequency. Therefore, we rely on an high-frequency bus of width 1. Such an interface can for example be implemented thanks to the RocketIO's that are Xilinx's standard interfaces enabling up to 3.2[Gbps] serial communication. In this extrapolation, we assume that the maximum frequency obtained is 1.5[Gbps] due to doubled path from one mini-circuit to the other. For this extrapolation to remain optimal, we need to adapt the mini-circuits so that their degree of parallelism is sufficient to feed this improved communication interface (with the previously introduced parallelism factor B). The extrapolated results are exposed in Table 4 for different bus throughputs and mini-circuit parallel implementations. The first line represents the situation of a mini-circuit using no parallelism. In this case, the internal throughput is limiting the performances of the entire system (i.e., equation Equation 2 is not fulfilled) leading to an encryption throughput of 23.7[Mbps]. By implementing parallelism with a factor 3 a throughput of 55[Mbps] is obtained with the minimal number of trusted gates inside the master. Similarly to subsubsection 6.3.1, increasing the bus width leads to higher throughput (with the right degree of parallelism). [5]

Note that as mentioned in section 5, these performances are significantly influenced by the majority vote. This phenomenon is observed in Table 5 that represents the data throughput of the extrapolated board with a bus throughput of 1.5[Gbps] and variable number of sub-circuits. More precisely, the secret opening time depends on the throughput of the majority gate. Similarly

[5] Note that these rocket IO's have a non-negligible hardware cost. Yet, we can consider them as untrusted (i.e., formally view them as part of the mini-circuits) if we can guarantee that mini-circuits act independently. The latter requires to verify that there is no direct communication path between the untrusted Rocket IO's in the master, which can be achieved by constraining the place and route so that they are well separated physically, hence allowing visual inspection. So overall, the relevance of this optimization/extrapolation boils down to the assumption that such a visual inspection is a simpler task than the verification that the master anyway has to undergo for its trusted gates.

λ	1	2	4	8	16
Throug. [Mbps]	55	51.4	46.3	38.7	29.1

Table 5: Influence of λ on the AES encryption throughput with extrapolated bus throughput of 1.5[Gbps].

testing [days]	**online** [bits]	**AES**			**Mysterion**		
		λ	**ROB.**	**Area** [GEs]	λ	**ROB.**	**Area** [GEs]
1	10^3	5	2^{-92}	152	5	2^{-94}	152
	10^6	7	2^{-82}	198	7	2^{-84}	198
	10^9	17	2^{-87}	430	15	2^{-81}	383
7	10^3	5	2^{-100}	152	5	2^{-102}	152
	10^6	7	2^{-93}	198	7	2^{-95}	198
	10^9	13	2^{-89}	337	11	2^{-80}	291

Table 6: Robustness and trusted area for Trojan-Resilient implementations of the AES and Mysterion.

to Equation 2, if its throughput is too low it will limit the overall throughput. For example, the throughput of the majority gate operating on sets of size λ that we use is equal to $940/\lambda$[Mbps] thanks to a synthesis at 470[MHz] and its factor $M = 2$. Even with that throughput, the performances are reduced by an approximate factor two if the number of sub-circuits is equal to 16.

6.3.3 Practical Robustness bounds Ultimately, the robustness bounds depend on the parameters (n, t, λ) according to equation $(4n/t)^{\lceil \lambda/2 \rceil}$. The number of tests t performed in a fixed amount of time depends on the encryption throughput achieved by the system. When this throughput increases, more tests can be performed during the same period of time. Hence, for a fixed number of encryptions n, improved performances result in an smaller ratio $(4n/t)$. This directly leads to better robustness bounds for fixed λ. It also implies that for a given layout and trusted area, Mysterion achieves better robustness than the AES. Table 6 shows the robustness bounds and the trusted area required for encryptions performed by the extrapolated mini-circuit for the AES and Mysterion.[6] This table targets robustness bounds of at least 2^{-80} for a given testing time and amount of bits to encrypt ranging from 1kb to 1Gb. For example, after a single day of test and for 1kbit to encrypt using the AES, the architecture proposed reaches robustness bounds of 2^{-92} with only 152 trusted GEs. If the amount of data to encrypt is extended to 1Gbit, 17 sub-circuits must be used, increasing the master area to 430 GEs. By spending one week in testing, the number of sub-circuits decreases to 13 and the trusted area decreases to 337 GEs. And thanks to its improved communication complexity and throughput, Mysterion can achieve either better area or robustness.

7 Conclusions

Our investigations put forward acceptable performances for the encryption of non-negligible amounts of data in a Trojan-resilient manner, with a well minimized master. In particular, and as already mentioned, the size of the trusted master is roughly one order of magnitude smaller than the unprotected AES (state-of-the-art) implementation of [23] that requires 2,400 GEs, confirming the global relevance of the proposed approach. This ratio would be further improved if moving from our simple case study implementing a single cryptographic primitive to a more complex system mixing several primitives to be implemented in a Trojan-resilient manner. The latter could for example be needed in situations where sensitive data has to be manipulated with cipher suites. Admittedly, these positive results come at the cost of quite large redundancy levels (i.e., the λ parameter) and long testing phases. But they highlight that in sensitive contexts where high robustness levels are required, such a strong flavor of Trojan-resilience is achievable.

Acknowledgments. François-Xavier Standaert is a senior associate researcher of the Belgian Fund for Scientific Research (FNRS-F.R.S.). This work has been funded in parts by the ERC project 724725 (acronym SWORD).

References

[1] J. Aarestad, D. Acharyya, R. M. Rad, and J. Plusquellic. 2010. Detecting Trojans Through Leakage Current Analysis Using Multiple Supply Pad I_{DDQ} s. *IEEE Trans. Information Forensics and Security* (2010).
[2] S. O. Adee. 2008. The Hunt For The Kill Switch. *IEEE Spectrum* (2008).
[3] D. Agrawal, S. Baktir, D. Karakoyunlu, P. Rohatgi, and B. Sunar. 2007. Trojan Detection using IC Fingerprinting. In *IEEE Symposium on Security and Privacy.*
[4] T. Araki, J. Furukawa, Y. Lindell, A. Nof, and K. Ohara. 2016. High-Throughput Semi-Honest Secure Three-Party Computation with an Honest Majority. In *ACM Conference on Computer and Communications Security.*
[5] G. Ateniese, A. Kiayias, B. Magri, Y. Tselekounis, and D. Venturi. 2016. Secure Outsourcing of Circuit Manufacturing. *IACR Cryptology ePrint Archive* (2016).
[6] C. Bayer and J.-P. Seifert. 2013. Trojan-resilient circuits. In *PROOFS.*
[7] G. T. Becker, F. Regazzoni, C. Paar, and W. P. Burleson. 2014. Stealthy dopant-level hardware Trojans: extended version. *J. Cryptographic Engineering* (2014).
[8] S. Bhunia, M. S. Hsiao, M. Banga, and S. Narasimhan. 2014. Hardware Trojan Attacks: Threat Analysis and Countermeasures. *Proc. IEEE* (2014).
[9] J. Daemen and V. Rijmen. 2002. *The Design of Rijndael: AES - The Advanced Encryption Standard.*
[10] Stefan Dziembowski, Sebastian Faust, and François-Xavier Standaert. 2016. Private Circuits III: Hardware Trojan-Resilience via Testing Amplification. *IACR Cryptol. ePrint Arch.* 2016 (2016), 1004.
[11] M. Feldhofer. [n. d.]. AES implementation on a grain of sand. ([n. d.]), 13–20(7).
[12] S. Ghandali, G. T. Becker, D. Holcomb, and C. Paar. 2016. A Design Methodology for Stealthy Parametric Trojans and Its Application to Bug Attacks. In *CHES.*
[13] V. Grosso, F.-X. Standaert, and S. Faust. 2015. Masking vs. Multiparty Computation: How Large is the Gap for AES? *IACR Cryptology ePrint Archive* (2015).
[14] C. Hocquet, D. Kamel, F. Regazzoni, J.-D. Legat, D. Flandre, D. Bol, and F-X. Standaert. 2011. Harvesting the potential of nano-CMOS for lightweight cryptography: an ultra-low-voltage 65 nm AES coprocessor for passive RFID tags. *J. Cryptographic Engineering* (2011).
[15] S. Hoory, A. Magen, and T. Pitassi. 2006. Monotone Circuits for the Majority Function. In *APPROX-RANDOM.*
[16] IEEE International Symposium on Hardware Oriented Security And Trust. [n. d.]. http://www.hostsymposium.org/. Retrieved on October 15, 2017.
[17] F. Imeson, A. Emtenan, S. Garg, and M. V. Tripunitara. 2013. Securing Computer Hardware Using 3D Integrated Circuit (IC) Technology and Split Manufacturing for Obfuscation. In *USENIX Security Symposium.*
[18] Y. Jin and Y. Makris. 2010. Hardware Trojans in Wireless Cryptographic ICs. *IEEE Design & Test of Computers* (2010).
[19] A. Journault, F.-X. Standaert, and Kerem Varici. 2017. Improving the security and efficiency of block ciphers based on LS-designs. *Des. Codes Cryptography* (2017).
[20] H. Kim, S. Hong, and J. Lim. 2011. A Fast and Provably Secure Higher-Order Masking of AES S-Box. In *CHES.* 95–107.
[21] V. Mavroudis, A. Cerulli, P. Svenda, D. Cvrcek, D. Klinec, and G. Danezis. 2017. A Touch of Evil: High-Assurance Cryptographic Hardware from Untrusted Components. In *CCS.*
[22] N. Mentens, L. Batina, B. Preneel, and I. Verbauwhede. 2005. A Systematic Evaluation of Compact Hardware Implementations for the Rijndael S-Box. In *CT-RSA.*
[23] A. Moradi, A. Poschmann, S. Ling, C. Paar, and H. Wang. 2011. Pushing the Limits: A Very Compact and a Threshold Implementation of AES. In *EUROCRYPT.*
[24] Multi-Gigabit Tranceiver. [n. d.]. https://en.wikipedia.org/wiki/Multi-gigabit_transceiver. Retrieved on October 15, 2017.
[25] S. Narasimhan, D. Du, R. Subhra Chakraborty, S. Paul, F. G. Wolff, C. A. Papachristou, K. Roy, and S. Bhunia. 2013. Hardware Trojan Detection by Multiple-Parameter Side-Channel Analysis. *IEEE Trans. Computers* (2013).
[26] C. Paar. 2017. Hardware Trojans and Other Threats against Embedded Systems. In *AsiaCCS.*
[27] B. Parhami. 2009. Efficient Hamming Weight Comparators for Binary Vectors Based on Accumulative and Up/Down Parallel Counters. *IEEE Trans. on Circuits and Systems* (2009).
[28] M. Rivain and E. Prouff. 2010. Provably Secure Higher-Order Masking of AES. In *CHES.*

[6] We do not use the bound of Equation 1 and rather compute $\Pr[\text{ROB} = 1] = \sum_{i=\lceil \lambda/2 \rceil}^{\lambda} \binom{\lambda}{i} \cdot (\frac{n}{t})^i \cdot (1 - \frac{n}{t})^{\lambda - i}$ directly to be tight [10].

[29] Semico Research. [n. d.]. https://semico.com/. Retrieved on October 15, 2017.
[30] S. Skorobogatov and C. Woods. 2012. Breakthrough Silicon Scanning Discovers Backdoor in Military Chip. In *CHES*.
[31] M. Tehranipoor and F. Koushanfar. 2010. A Survey of Hardware Trojan Taxonomy and Detection. *IEEE Design & Test of Computers* (2010).
[32] Markus Ullrich, Christophe De Canniere, Sebastiaan Indesteege, Özgül Küçük, Nicky Mouha, and Bart Preneel. 2011. Finding optimal bitsliced implementations of 4× 4-bit s-boxes. In *SKEW 2011 Symmetric Key Encryption Workshop, Copenhagen, Denmark*. 16–17.
[33] R. S. Wahby, M. Howald, S. Garg, A. Shelat, and M. Walfish. 2016. Verifiable ASICs. In *IEEE Symposium on Security and Privacy*.
[34] A. Waksman and S. Sethumadhavan. 2011. Silencing Hardware Backdoors. In *IEEE Symposium on Security and Privacy*.
[35] Y. Wang, P. Chen, J. Hu, and J. Rajendran. 2016. The cat and mouse in split manufacturing. In *DAC*.
[36] K. Xiao, D. Forte, Y. Jin, R. Karri, S. Bhunia, and M. M. Tehranipoor. 2016. Hardware Trojans: Lessons Learned after One Decade of Research. *ACM Trans. Design Autom. Electr. Syst.* (2016).