A Thorough Evaluation of RAMBAM

Daniel Lammers*
Ruhr University Bochum
Horst Görtz Institute for IT Security
Bochum, Germany
daniel.lammers@rub.de

Amir Moradi*
Ruhr University Bochum
Horst Görtz Institute for IT Security
Bochum, Germany
amir.moradi@rub.de

Nicolai Müller*
Ruhr University Bochum
Horst Görtz Institute for IT Security
Bochum, Germany
nicolai.mueller@rub.de

Aein Rezaei Shahmirzadi*
Ruhr University Bochum
Horst Görtz Institute for IT Security
Bochum, Germany
aein.rezaeishahmirzadi@rub.de

ABSTRACT

The application of masking, widely regarded as the most robust and reliable countermeasure against Side-Channel Analysis (SCA) attacks, has been the subject of extensive research across a range of cryptographic algorithms, especially AES. However, the implementation cost associated with applying such a countermeasure can be significant and even in some scenarios infeasible due to considerations such as area and latency overheads, as well as the need for fresh randomness to ensure the security properties of the resulting design. Most of these overheads originate from the ability to maintain security in the presence of physical defaults such as glitches and transitions. Among several schemes with a trade-off between such overheads, RAMBAM, presented at CHES 2022, offers an ultra-low latency in terms of the number of clock cycles. It is dedicated to the AES and utilizes redundant representations of the finite field elements to enhance protection against both passive and active physical attacks.

In this paper, we have a deeper look at this technique and provide a comprehensive analysis. The original authors reported that the number of required traces to mount a successful attack increases exponentially with the size of the redundant representation. We however examine their scheme from theoretical point of view. More specifically, we investigate the relationship between RAMBAM and the well-established Boolean masking and, based on this, prove the insecurity of RAMBAM. Through the examples and use cases, we assess the leakage of the scheme in practice and use verification tools to demonstrate that RAMBAM does not necessarily offer adequate protection against SCA attacks neither in theory nor in practice. Confirmed by real-world experiments, we additionally highlight that – if no dedicated facility is incorporated – the RAMBAM designs are susceptible to fault-injection attacks despite providing some degree of protection against a sophisticated attack vector, i.e., SIFA.

KEYWORDS

RAMBAM; Power Analysis Attack; Hardware; Masking

*Authors list in alphabetical order; see https://www.ams.org/profession/leaders/CultureStatement04.pdf

1 INTRODUCTION

Since the pioneering work of Kocher et al. in the late 1990s [21, 22], Side-Channel Analysis (SCA) attacks have emerged as a serious threat to the confidentiality of sensitive information processed by cryptographic hardware circuits. SCA attacks exploit unintended information leakage through physical characteristics, so-called side channels, such as power consumption [22], electromagnetic radiation [15], or timing [21]. Even small variations in these physical characteristics can be analyzed to deduce sensitive information such as cryptographic keys. These threats are not just theoretical but have been demonstrated through real-world attacks documented in the literature over the past twenty years of research [26, 29]. As a result, the reliable protection of cryptographic hardware circuits against SCA attacks has become an essential aspect of ensuring confidentiality.

To do this, two main concepts, called hiding and masking, have been developed. The goal of hiding is to obscure any dependency between sensitive information and the measured physical characteristic below the noise level, making it more challenging for an adversary to obtain any useful information. This is accomplished by decreasing the Signal-to-Noise Ratio (SNR) through, e.g., the addition of random noise to the measured signal [23]. While hiding can make the accomplishment of attacks more difficult, in the sense that more measurements are required for a successful attack, there is no security guarantee or formal proof of its effectiveness. The efficiency of hiding countermeasures depends heavily on the measurement setup and the specific device being used. Therefore, these techniques are not generalizable or transferable, and their efficiency must be evaluated for each individual device separately.

In contrast, the masking countermeasure provides security proofs under formal adversary models and aims to remove the dependency of the circuit's power consumption on the cipher intermediates by randomizing them based on secret sharing [38]. Despite its simple security assumptions, realizing masking on hardware is a complicated task due to physical defaults, such as glitches. This results in overhead in terms of circuit size, latency, and the demand for randomness to correctly implement a masked circuit. Consequently, researchers have focused on minimizing these particular overheads while maintaining others at an acceptable level. This has led to various research branches focusing on low-area [19, 31], low-latency [34, 42], or low-randomness [9, 37].

Low-latency masking is of particular interest because masking naturally increases the latency of designs by requiring additional register stages to synchronize intermediate signals and prevent glitch propagation. To address this issue, multiple masking schemes, such as Generic Low-Latency Masking (GLM) [18], LUT-based Masked Dual-Rail with Pre-charge Logic (LMDPL) [34], and Redundancy AES Masking Basis for Attack Mitigation (RAMBAM) [1], reduce the number of consecutive register stages, resulting in a significant latency improvement.

Generic Low-Latency Masking. GLM [18] is a method to remove register stages from Domain-Oriented Masking (DOM)-protected designs [19], but at the cost of substantially higher area and randomness requirements. Unlike DOM that requires a share compression after each non-linear operation, GLM offers greater flexibility by allowing the designer to selectively choose which share compression steps to perform while skipping the others. However, when a share compression step is skipped, the circuit temporarily processes a higher number of shares and requires duplication of certain parts of the circuit what greatly increases the circuit size.

LUT-based Masked Dual-Rail with Pre-charge Logic. LMDPL combines both Dual-Rail Pre-charge (DRP) and Boolean masking to create glitch-free composable sub-circuits, so-called gadgets, that can be composed without increasing the latency [34]. This means that regardless of the circuit depth, the circuit only encompasses a single register stage. The DRP technique ensures that the circuit is glitch-free [39], and other delay-based effects do not leak sensitive information.

The current state-of-the-art techniques in low-latency masking suffer from increased area overhead or a higher demand for fresh randomness in exchange for reduced latency. This renders the implementation of provably secure, low-latency masking schemes in commercial devices a costly proposition. Recently, a new technique called RAMBAM was introduced at CHES 2022, which claims to provide a solution to this long-standing problem. Compared to the existing techniques such as GLM and LMDPL, RAMBAM offers protection against SCA and Statistical Ineffective Fault Attack (SIFA) [11], a particular class of Fault Injection (FI) attacks, while maintaining low latency and a relatively low gate count. Notably, RAMBAM achieves a latency of only one clock cycle per round for the Advanced Encryption Standard (AES), which is unattainable by GLM and LMDPL due to their impractical circuit complexity, while its compact version exhibits a circuit size of 12.075 kGE [1]. Additionally, RAMBAM offers fault protection, whereas redundant computation must be retrofitted to a GLM or LMDPL-protected circuit, further increasing the gate count. In summary, RAMBAM appears to offer a viable solution to this long-standing problem, provided that its security guarantees remain valid. Note that the original authors did not claim the provable security of their scheme. Instead, they claimed that the number of traces required to successfully attack their scheme increases exponentially with the redundancy size, as explained in Section 3.1.

1.1 Our Contributions

In this work, we carefully evaluate the underlying scheme of RAMBAM in both theory and practice. First, we present the main concept behind the scheme with concrete examples, which should help the reproducibility of the technique and further evaluations. Moreover, focusing on concrete design choices (suggested by the original authors) towards a protected implementation of AES, we show theoretical as well as experimental analysis results for each step indicating its failure in providing security against SCA and FI attacks.

2 PRELIMINARIES

In this section, we provide the background necessary to understand and follow the rest of the paper. We also present the details of the setup and evaluation schemes which we have employed to conduct our analyses.

2.1 Notation

We denote single-bit Boolean variables as well as integers (e.g., $x \in \mathbb{F}_2$ and $p \in \mathbb{Z}$) by lower case letters while we denote n-bit vectors of Boolean variables by upper case letters, e.g., $X \in \mathbb{F}_2^n$. We use subscripts to show certain elements of a vector, e.g., $\langle X_{n-1}, \ldots, X_0 \rangle$ or $\langle x_{n-1}, \ldots, x_0 \rangle$ depending on the context, and a sans serif font for functions, e.g., $\mathsf{T}(.)$. We also use superscripts to refer to specific shares of a sensitive variable. For example, the share with index i of a sensitive variable $x \in \mathbb{F}_2$ is denoted as $x^i \in \mathbb{F}_2$. Further, we denote the set of all shares of a sensitive variable x as $Sh(x) = \{x^{|Sh(x)|-1}, \ldots, x^0\}$.

2.2 Galois Fields

A Galois field $\mathbb{F}_q^n[X]\backslash P = GF(p^n)[X]\backslash P$, also known as finite field, with characteristic p and order p^n defines a set encompassing a finite number of p^n elements. In this work, we focus on Galois fields with characteristic 2, i.e., $\mathbb{F}_2^n[X]\backslash P$. Every element $A \in \mathbb{F}_2^n\backslash P$ represents a polynomial with coefficients $\{a_{n-1}, \ldots, a_0\} \in (\mathbb{F}_2)^n$ with $A \equiv \sum_{i=0}^{n-1} a_i x^i$. Computing sum of $A, B \in \mathbb{F}_2^n\backslash P$ is done by applying XOR on their corresponding coefficients with equal indices.

$$C = A + B = \sum_{i=0}^{n-1} c_i x^i, \quad \forall i, c_i = a_i \oplus b_i$$

Multiplication of $A, B \in \mathbb{F}_2^n\backslash P$ requires an irreducible polynomial P of degree n with coefficients in $\mathbb{F}_2$, i.e., $A \cdot B \mod P$. However, P is not the only irreducible polynomial to construct $\mathbb{F}_2^n$, but all such Galois fields are isomorphic.

Definition 2.1 (Irreducible). *A polynomial P is irreducible if it cannot be factored into two non-constant polynomials of lower degree over its field of coefficients.*

Definition 2.2 (Isomorphic). *Two Galois fields $\mathbb{F}'$ and $\mathbb{F}''$ are isomorphic (denoted as $\mathbb{F}' \cong \mathbb{F}''$) iff there exists a linear transformation $\phi : \mathbb{F}' \to \mathbb{F}''$ called isomorphism that preserves all arithmetic operations of $\mathbb{F}'$ and $\mathbb{F}''$. In particular, ϕ satisfies the following conditions:*

- *For any $A, B \in \mathbb{F}'$ it holds that $\phi(A + B) = \phi(A) + \phi(B)$ and $\phi(A \cdot B) = \phi(A) \cdot \phi(B)$ while $+$ and $\cdot$ denote addition and multiplication in the fields, respectively, as defined above.*
- *$\phi(1) = 1$ where 1 denotes the multiplicative identity in $\mathbb{F}'$ and $\mathbb{F}''$ respectively.*

Example 2.1. *Every S-box of the AES computes an inversion in* $\mathbb{F}_2^8[X]\backslash P_0$ *with the fixed irreducible polynomial* $P_0 = x^8+x^4+x^3+x+1$. *Nevertheless, there exist* 30 *irreducible polynomials* $P_0, \ldots, P_{29}$ *of degree* 8 *with coefficients in* $\mathbb{F}_2$ *allowing to create* 30 *isomorphic Galois fields* $\mathbb{F}_2^8[X]\backslash P_0, \ldots, \mathbb{F}_2^8[X]\backslash P_{29}$. *Further, every isomorphism* ϕ *between two fields* $\mathbb{F}_2^8[X]\backslash P_i$ *and* $\mathbb{F}_2^8[X]\backslash P_j$ *is a linear function over* $\mathbb{F}_2$, *hence can be represented by an* 8×8 *matrix multiplication.*

2.3 Boolean Masking

Boolean masking [3], a well-established and predominant approach to achieve provable t-order security (cf. Section 2.4) against SCA, randomizes every sensitive variable $X \in \mathbb{F}_2^n$ by splitting it into a sharing encompassing $s > t$ independent and uniformly distributed shares $Sh(X) = \{X^{s-1}, \ldots, X^0\} \in (\mathbb{F}_2^n)^s$. To generate $Sh(X)$ it holds that $\{X^{s-2}, \ldots, X^0\} \in_R (\mathbb{F}_2^n)^{s-1}$ while $X^{s-1} = X \oplus \left(\bigoplus_{i=0}^{s-2} X^i\right)$. This implies that $X = \bigoplus_{i=0}^{s-1} X^i$ enables to compute the unshared X from $Sh(X)$. Due to the fact that only the combination of all shares in $Sh(X)$ reveals information about X, all cipher operations must process only a restricted subset of $Sh(X)$, i.e., an incomplete set of shares giving no information about X itself.

2.4 Robust Probing Security Model

Ishai et al. proposed the t-probing model [20] to formally abstract the security of masked circuits against adversaries. The t-probing model limits the adversary to observe the distribution over a maximum of t wires in a given circuit by placing probes on the respective wires. Each probe records one stable and noise-free intermediate during a specific point in time. Thus, a circuit is considered t-probing secure if it does not reveal any information to a potential t-probing adversary.

Definition 2.3 (t-probing security). *A masked circuit achieves* t*-probing security iff every joint distribution over up to* t *wires observed by a* t*-probing adversary is statistically independent of any sensitive variable.*

Definition 2.3 serves as a cornerstone for evaluating the security of masked circuits. However, the t-probing model falls short in accounting for the behavior of physical defaults, which is a well-known issue when practically realizing a masked hardware circuit. Therefore, masked hardware circuits, which are shown secure under t-probing model, are not necessarily secure in practice. To address this limitation, the robust t-probing model was introduced in [13], which extends the t-probing model by considering all possible occurrences of glitches [24], transitions [7], and couplings [5]. This is achieved by enhancing the adversary's capabilities to place a maximum of t extended-probes on the circuit that capture not only the stable signals but also the transient signals that may be induced on the wire due to physical defaults. This extended definition of the adversary aligns with Definition 2.3 to form the so-called robust t-probing adversary.

Definition 2.4 (Robust t-probing security). *A masked circuit achieves robust* t*-probing security iff every joint distribution over up to* t *wires observed by a robust* t*-probing adversary is statistically independent of any sensitive variable.*

In this work, our focus is on glitches, which are temporary signal transitions that occur in combinational circuits due to delay imbalances of signals arriving at a logic gate. To account for glitches within the robust probing model, glitch-extended probes not only record the stable signal of a probed wire, but also all the stable signals that contribute to the probed wire. For example, a probe placed on the output of a gate in a combinational circuit propagates backward to all primary inputs and register outputs that contribute to the value of the probed signal.

2.5 Leakage Assessment

To evaluate the effectiveness of protection mechanisms applied on a circuit, it is necessary to practically test the resistance of a fabricated prototype of the Device Under Test (DUT) against real-world adversaries who exploit the device's physical characteristics. Unfortunately, due to the large number of possible combinations of attacks, power models, and intermediate values, exhaustive testing of the prototype's resistance is infeasible. However, leakage assessment methodologies, such as Welch's t-test [6, 35], are statistical hypothesis tests applied to detect any possible leakage within a certain security order t. In contrast to performing attacks, Welch's t-test evaluates whether SCA measurements have a dependency with the associated data, regardless of the exploitability of the leakage. For instance, the most common fixed-versus-random t-test [6] (also known as non-specific t-test) compares the statistical properties of two sets of measurements. One set corresponds to the SCA measurement associated with a fixed plaintext, and the other to random plaintexts, while the key is kept the same in both sets. If there is a significant difference in the statistical properties between these two sets (e.g., through student t-test), its reported that the SCA leakage is detectable, i.e., there might be an attack exploiting such a leakage.

2.6 PROLEAD

Since the evaluation of masked implementations via experimental analysis requires a suitable measurement setup, a high level of expertise, and prototyping the DUT, some efforts have been put in developing schemes and tools to enable assessing the design prior to prototyping. PROLEAD [28] is a novel tool, which combines formal verification and leakage simulation, to evaluate the robust probing security of masked hardware circuits presented by a gate-level netlist. By simulating the netlist, PROLEAD obtains the distributions for every possible robust probing adversary under different input settings. Subsequently, the tool checks the independence of these distributions using a statistical hypothesis test. In the event that a significant dependency is detected for at least one probing set, PROLEAD reports the detectability of the leakage and provides details of the most leaking probing sets. It is highly efficient and can handle first-order masked full cipher hardware cores in a matter of minutes to hours. If enough simulations are used, PROLEAD provides reliable evaluation results with respect to robustness/vulnerability of the given masked circuit. We make use of PROLEAD in some of our evaluations explained in detail in Section 3.

2.7 SCA Measurement Setup

In order to conduct experimental SCA evaluations, we employed an Field Programmable Gate Array (FPGA)-based platform, and synthesized and mounted all investigated designs on a customized platform, SAKURA-G board [33], which is suitable for SCA evaluations containing a target Spartan-6 FPGA. We captured the power consumption using a 1 Ω shunt resistor inserted on the target FPGA's VDD path, and recorded the traces using a digital oscilloscope at a sampling rate of 500 MS/s, while all target designs were supplied by a stable 3 MHz clock. In order to facilitate and accelerate the measurement process for fixed vs. random t-test, we employed the techniques and procedures suggested in [35].

3 RAMBAM AND EVALUATIONS

3.1 Concept

As mentioned in Section 2.2, there are 30 different polynomials of degree 8 over $GF(2)$ that can be used to construct the field $GF(2^8)$. As a matter of fact, all of these polynomials are isomorphic to each other, meaning that a linear transformation $\phi(.)$ can be used to switch between different representations of the field.

In their work on protecting AES against SCA, the authors of RAMBAM [1] employed a larger algebraic structure to randomize the representation. Specifically, they used a redundant representation to construct a protected construction after transforming the key and the input data bytes from the representation in $\mathbb{F}_{P_0}[X] = GF(2)[X]\backslash P_0 = GF(2^8)[X]$ to any other representation in $\mathbb{F}_P[X] = GF(2)[X]\backslash P$. An element X of $\mathbb{F}_P$ can be expressed equivalently as $X + CP$, where C is an arbitrary polynomial over $GF(2)$ of degree d. The authors utilized a ring homomorphism $H : \mathbb{R}_{P.Q} \rightarrow \mathbb{F}_P$ defined as $H(X) = X \mod P$ to map any redundant representation $B \in \mathbb{R}_{P.Q}$ to $\mathbb{F}_P$, where Q is a polynomial of degree d over $GF(2)$. All calculations are performed modulo $Z = P \cdot Q$, and the elements of $\mathbb{R}_{P.Q}$ are represented by $(8 + d)$-bit words. The parameters P and Q are specific to each variant of RAMBAM, and the degree d of the polynomial Q represents the level of *redundancy* in the representation, as it reflects the number of additional bits in the representation of each byte.

It is worth noting that any given value X can be represented using 2^d different interchangeable redundant representations. Therefore, the redundant representation of each byte can be seen as a form of masking and the authors of [1] also utilized this property to re-randomize intermediate values in their masked S-box realization, as it is possible to replace any one of these representations with another one at any time. Moreover, a redundant representation can be uniquely defined using the two independent parameters $\phi(.)$ and Q, where the former is used to transform the key and data bytes from the representation in $\mathbb{F}_{P_0}$ to any other representation in $\mathbb{F}_P$.

The masking entropy is naturally lower than 8 bits when d is smaller than 8, which is known as Low Entropy Masking Schemes (LEMS). It can be beneficial under certain adversarial and implementation scenarios. However, the limitations of such schemes have been discussed in [41], as they often lack sufficient entropy and fail to meet the requirements for provable security. On the other hand, when $d \geq 8$, the RAMBAM scheme can be interpreted as a form of Boolean masking. As previously mentioned, each byte X in the AES standard is represented as $X + CP$, where C is a random d-bit number. This representation can be seen as two shares $\langle R, X + \mathsf{T}(R)\rangle$, where R is a random d-bit number, and $\mathsf{T} : \{0,1\}^d \rightarrow \{0,1\}^8$ is a linear function. In other words,

$$X + CP = Rx^8 + X + \mathsf{T}(R) \Rightarrow CP = Rx^8 + \mathsf{T}(R). \quad (1)$$

CP is a linear function over C, since P is a fixed irreducible polynomial. Notably, if $d \geq 8$, $\mathsf{T}(.)$ becomes surjective, and for $d = 8$ bijective. Therefore, $\mathsf{T}(R)$ covers all elements of $\{0,1\}^8$, and particularly for $d = 8$ has a uniform distribution if R is taken from a uniform distribution at random. Therefore, one can potentially transform Boolean masking into RAMBAM by following a specific procedure. Consider a byte in the AES standard, denoted as X and represented by two shares $\langle R, X + R\rangle$, where R represents an 8-bit random number. Evidently, the redundancy size is $d = 8$. To obtain the RAMBAM representation of the shares, the inverse of the transformation function $\mathsf{T}(.)$, represented as $\mathsf{T}^{-1}(.)$, must be applied to R. The resulting values are then represented as a pair in the form of $\left\langle \mathsf{T}^{-1}(R), X + R\right\rangle$. We will use this property of the equivalent representation of the design in Boolean masking to check the security of the RAMBAM with verification tools.

The authors of RAMBAM considered several criteria when selecting the polynomials P and Q for a redundancy size of d. They asserted that Q should be an irreducible polynomial and not divisible by P. Additionally, they examined the product $Z = P \cdot Q$ based on two further criteria. Firstly, they searched for a product with a minimum Hamming weight since doubling in the ring $\mathbb{R}_{P.Q}$ would be more cost-effective in hardware. Secondly, the product should possess specific properties that enhance protection against SIFA-1 [32]. Further details and explanations on each of these criteria are provided in the original publication [1]. It is noteworthy that SIFA-1 is a type of SIFA [11] where the attacker can only inject faults into the registers. Neither RAMBAM nor any other form of pure Boolean masking provides complete protection against SIFA in general (e.g., when the faults are injected into the combinational logic) or against other forms of fault attacks (e.g., Differential Fault Analysis (DFA)). Based on these criteria, the authors of RAMBAM proposed the following polynomials for a redundancy size of $d = 8$:

$$P(x) = x^8+x^6+x^5+x^3+1, \quad Q(x) = x^8+x^6+x^5+x^4+x^3+x+1. \quad (2)$$

3.2 Components

To implement the protected version of the AES encryption using the redundant representation, we initially present each byte of both plaintext and key with Boolean masking. Specifically, each byte X is represented with two shares $\langle R, X + R\rangle$, where R is an 8-bit random number. Subsequently, we convert the bytes from the representation in $\mathbb{F}_{P_0}$ to the representation in $\mathbb{F}_P$, where the polynomial P is provided in Equation (2). This conversion can be achieved with the linear transformation $\phi(.)$, which is defined below. It is worth noting that since these operations are linear, we express them as matrix multiplication with the corresponding matrices presented below.

$$\Phi = \begin{bmatrix} 1 & 0 & 1 & 0 & 0 & 1 & 1 & 1 \\ 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 1 & 1 & 1 \\ 0 & 0 & 1 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 \\ 0 & 1 & 0 & 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 1 & 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 1 & 0 & 1 & 0 & 1 \end{bmatrix}, \quad \Phi^{-1} = \begin{bmatrix} 1 & 0 & 1 & 1 & 0 & 0 & 1 & 1 \\ 0 & 1 & 1 & 0 & 1 & 0 & 1 & 1 \\ 0 & 0 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 & 0 & 1 & 1 \\ 0 & 1 & 1 & 1 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 & 1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 \end{bmatrix} \tag{3}$$

To convert the representation from Boolean masking to RAMBAM, we utilize the function $\mathsf{T}^{-1}(.)$ on one share of each byte. Therefore, the representation becomes as follows.

$$\left\langle \mathsf{T}^{-1}(\Phi \cdot R), \Phi \cdot (X + R) \right\rangle$$

The corresponding matrices of $\mathsf{T}(.)$ and $\mathsf{T}^{-1}(.)$ are defined below.

$$T = \begin{bmatrix} 1 & 0 & 1 & 1 & 1 & 1 & 0 & 1 \\ 0 & 1 & 0 & 1 & 1 & 1 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 & 1 & 1 & 1 \\ 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 \\ 1 & 0 & 0 & 1 & 0 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 0 & 1 & 1 & 0 \\ 0 & 1 & 1 & 1 & 1 & 0 & 1 & 1 \end{bmatrix}, \quad T^{-1} = \begin{bmatrix} 1 & 0 & 0 & 1 & 0 & 1 & 0 & 0 \\ 1 & 1 & 0 & 0 & 1 & 0 & 1 & 0 \\ 1 & 1 & 1 & 0 & 0 & 1 & 0 & 1 \\ 0 & 1 & 1 & 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 \\ 1 & 0 & 0 & 0 & 1 & 1 & 0 & 1 \\ 0 & 1 & 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 & 0 & 0 & 1 \end{bmatrix} \tag{4}$$

At this stage, every byte in the internal state belongs to the $\mathbb{R}_{P.Q}$ ring and is represented by 16 bits. As opposed to operating on individual bytes, the standard operations of `AddRoundKey` and `ShiftRows` can now be applied to each redundant representation.

The `SubBytes` operation in the protected version of the AES is slightly different from the standard one, and we will explain it in more details in the next paragraphs. The `KeyExpansion` function in the protected AES also requires replacing the `SubBytes` operation with the protected version. The other operations in `KeyExpansion`, such as `RotWord`, `SubWord`, and `XOR` with the round constant can be applied to each 16-bit redundant representation as usual. However, instead of XORing with the round constant, $\Phi \cdot r_{\mathsf{con}}$ should be added to the state, where r_{con} is the round constant in the standard AES.

In the ordinary AES, the S-box operation involves the inversion of X over $\mathbb{F}_{P_0}$ followed by an affine function $\mathsf{A}(.)$ while the inversion can be expressed as X^{254}. As raising to any power of two in a ring of characteristic 2 is a linear transformation, it has a lower hardware implementation cost compared to ring multiplication. To calculate the inversion function as a sequence of multiplications, an optimal search is conducted by the authors of the RAMBAM to determine the sequence that minimizes the number of multiplications and raising to the power of n to result in a lower area overhead while considering low circuit depth to increase the maximal frequency of the circuit. It has been demonstrated in the original publication that the number of multiplications in such a sequence cannot be less than 4. By performing a brute-force search, two distinct versions of the `ProtectedS-box` were proposed, where one is optimized for the maximal frequency and the other one for the area overhead. For simplicity without losing generality, we focus on the version that is optimized for area, and reiterate the general algorithm in Algorithm 1, which is borrowed from the original paper [1].

Algorithm 1 Protected S-box (optimized for the area) [1]

Input: $Sbox\_in$ ▷ The input of the S-box which is $(8+d)$-bit value
Input: $R_0, \ldots, R_6$ ▷ 7 random d-bit values
Output: $Sbox\_out$ ▷ $(8+d)$-bit values after S-box operation

function $ProtectedS\text{-}box_Z(Sbox\_in, R_0, \ldots, R_6)$
  $V \leftarrow Sbox\_in$
  $V_2 \leftarrow \Psi_Z^{(2)} \cdot V + \langle R_0, T \cdot R_0 \rangle$ ▷ Raising to the power of 2
  $V_3 \leftarrow Mul_Z(V, V_2) + \langle R_1, T \cdot R_1 \rangle$
  $V_{12} \leftarrow \Psi_Z^{(4)}(V_3) + \langle R_2, T \cdot R_2 \rangle$ ▷ Raising to the power of 4
  $V_{14} \leftarrow Mul_Z(V_2, V_{12}) + \langle R_3, T \cdot R_3 \rangle$
  $V_{15} \leftarrow Mul_Z(V_3, V_{12}) + \langle R_4, T \cdot R_4 \rangle$
  $V_{240} \leftarrow \Psi_Z^{(16)}(V_{15}) + \langle R_5, T \cdot R_5 \rangle$ ▷ Raising to the power of 16
  $V_{254} \leftarrow Mul_Z(V_{14}, V_{240}) + \langle R_6, T \cdot R_6 \rangle$
  $Sbox\_out \leftarrow \mathsf{A}_Z(V_{254})$
end function

As stated, raising to the power of 2, 4, and 16 are all linear functions in $\mathbb{F}_P$ and hence can be expressed as matrix multiplication. The matrices that should be multiplied to $X \in \mathbb{F}_P$ are defined below.

$$\Psi^{(2)} = \begin{bmatrix} 1 & 0 & 0 & 0 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 \\ 0 & 1 & 0 & 0 & 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 1 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 \end{bmatrix}, \quad \Psi^{(4)} = \begin{bmatrix} 1 & 0 & 1 & 1 & 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 \\ 0 & 1 & 0 & 0 & 0 & 1 & 1 & 1 \\ 0 & 0 & 1 & 0 & 0 & 1 & 1 & 1 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 & 0 & 1 & 1 \end{bmatrix},$$

$$\Psi^{(16)} = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} \tag{5}$$

In fact, when the redundancy $d \geq 8$, a matrix multiplication can be formulated in a new, larger domain by performing two matrix multiplications, each on one side of the variables: one on the 8-bit part and another one on the redundancy part. In other words, suppose that $\left\langle T^{-1} \cdot R, X + R \right\rangle$ should be squared. It is possible to realize such an operation by two matrix multiplications, one on $\mathsf{T}^{-1} \cdot R$ and another one on $X + R$. Since each of these parts is seen as a share in a masked implementation, this separation indeed would allow us to perform the operations on each share individually. For instance, the matrices for raising to the power of $i \in \{2, 4, 16\}$ can be constructed as

$$\Psi_Z^{(i)} = \begin{bmatrix} T^{-1} \cdot \Psi^{(i)} \cdot T & 0 \\ 0 & \Psi^{(i)} \end{bmatrix}, \tag{6}$$

with matrices T and T^{-1} provided in Equation (4). However, when the redundancy $d < 8$, $\mathsf{T}(.)$ is not surjective, and hence $\mathsf{T}^{-1}(.)$ does not necessarily exist. Therefore, it is not possible to write $\Psi_Z^{(i)}$ as shown in Equation (6), i.e., the operations cannot be individually performed on each share. This means that $\Psi_Z^{(i)}$ can still be realized

Algorithm 2 Multiplication

Input: X, Y ▷ Two $(8+d)$-bit values
Output: Mul_out ▷ An $(8+d)$-bit value
1: **function** $Mul_Z(X, Y)$
2: $Mul\_out \leftarrow 0$
3: $Y\_Shifted \leftarrow Y$
4: **for** $i = 0$ to $7 + d$ **do**
5: **if** $X_i = 1$ **then**
6: $Mul\_out \leftarrow Mul\_out + Y\_Shifted$
7: **end if**
8: $Y\_Shifted \leftarrow ModularShiftLeft_Z(Y\_Shifted)$
9: **end for**
10: **end function**

Algorithm 3 Modular Shift Left

Input: X ▷ An $(8+d)$-bit value
Output: Y ▷ An $(8+d)$-bit value
1: **function** $ModularShiftLeft_Z(X)$
2: $Y \leftarrow X << 1$ ▷ binary left shift
3: **if** $Y_{8+d} = 1$ **then**
4: $Y \leftarrow Y + Z$ ▷ Reduction modulo $Z = P \cdot Q$
5: **end if**
6: **end function**

by a matrix multiplication, but some operations will include both the masked data and the mask, which may result in leakage. More precisely, if the redundant value is represented by $\langle R, X + T \cdot R\rangle$, one way of application of squaring is to write

$$\left\langle R, \Psi^{(2)} \cdot (X + T \cdot R) + \Psi^{(2)} \cdot T \cdot R + T \cdot R \right\rangle,$$

which means that the following matrix is multiplied by $\langle R, X + T \cdot R\rangle$.

$$\Psi_Z^{(2)} = \begin{bmatrix} I & 0 \\ \Psi^{(2)} \cdot T + T & \Psi^{(2)} \end{bmatrix}, \tag{7}$$

where I denotes the identity matrix of size d. It can be seen that the lower rows of such a matrix contain elements multiplied to both shares, i.e., both parts of $\langle R, X + T \cdot R\rangle$. We should highlight that this problem is not identified in the original RAMBAM paper, and the authors reported hardware implementations with redundancy smaller than 8.

The authors of RAMBAM utilized the schoolbook multiplication algorithm, shown in Algorithm 2, which employs reduction modulo $Z = P \cdot Q$, to implement the multiplication. The function $ModularShiftLeft_Z$ encapsulates the modular reduction, as can be seen in Algorithm 3. It is important to note that this multiplication operates on the entire redundant representation as a whole, and cannot be split into two parts like linear operations for $d \geq 8$. Consequently, due to applying the operation to both masked data and the mask, this operation is susceptible to leakage in practice.

As mentioned earlier, an affine function $\mathsf{A}(.)$ needs to be applied to each byte after the inversion to construct the AES S-box. $\mathsf{A}(.)$ is written as

$$\mathsf{A}(X) = A \cdot X + \mathtt{63},$$

where $X \in \mathbb{F}_{P_0}$ and A is an 8×8 matrix and constant is in hexadecimal format. In order to exchange $\mathbb{F}_{P_0}$ with $\mathbb{F}_P$, we can write

$$\mathsf{A}'(X) = A' \cdot X + \Phi \cdot \mathtt{63},$$

where $X \in \mathbb{F}_P$ and $A' = \Phi \cdot A \cdot \Phi^{-1}$. Then, in the redundant domain Z, the multiplication with A' can be performed similar to what we have explained in Equation (6) for $d \geq 8$ and in Equation (7) for $d < 8$. The addition with the constant can also be straightforwardly done by adding with $\langle 0, \Phi \cdot \mathtt{63}\rangle$. Note that it is also possible to generate A' for any field based on the representation of $\mathsf{A}(.)$ in $\mathrm{GF}(2^8)$ [8][1].

To refresh the masked elements in $\mathbb{R}_{P \cdot Q}$, one can add CP to each element, where C is a d-bit random value. Alternatively, we can use Equation (1) for refreshing. This involves adding $\langle R, T \cdot R\rangle$ to each element, where R is a d-bit random value. Note that the authors of RAMBAM claimed that to avoid the leakage observed in the state of the art [17], it is necessary to add fresh masks, i.e., refresh the sharing at the output of each Square and each multiply as demonstrated in Algorithm 1. In our implementations, we have employed this approach for share refreshing.

The original `MixColumns` operation involves multiplication by both 2 and 3 in $\mathbb{F}_{P_0}$, which needs to be replaced by multiplication in $\mathbb{R}_{P \cdot Q}$. Given that both values are constant, they can be implemented as binary linear functions, i.e., matrix multiplication. More specifically, we can write

$$2 * X = \Gamma^{(2)} \cdot X, \quad 3 * X = \Gamma^{(3)} \cdot X$$

with $*$ being the modular multiplication in $\mathbb{F}_{P_0}$ and $\Gamma^{(2)}/\Gamma^{(3)}$ 8×8 binary matrices. One can also write $\Gamma^{(3)} = \Gamma^{(2)} + I$ with I being identity.

Without loss of generality, suppose that $\langle R, X + T \cdot R\rangle$ should be multiplied by 2. Similar to squaring, it is possible to apply the operation using two distinct matrix multiplications on each part individually, if $d \geq 8$. Following Equation (6), this allows us to construct the matrix $\Gamma_Z^{(2)}$ as follows. The same can be written for $\Gamma_Z^{(3)}$. Note that if $d < 8$, one solution is the procedure explained for Equation (7).

$$\Gamma_Z^{(2)} = \begin{bmatrix} T^{-1} \cdot \Phi \cdot \Gamma^{(2)} \cdot T & 0 \\ 0 & \Phi \cdot \Gamma^{(2)} \end{bmatrix} \tag{8}$$

Herein, we present the matrices utilized for the implementation of the `MixColumns` operation.

[1] $\mathsf{A}(X) = \mathtt{05} \cdot X + \mathtt{09} \cdot X^2 + \mathtt{F9} \cdot X^{2^2} + \mathtt{25} \cdot X^{2^3} + \mathtt{F4} \cdot X^{2^4} + \mathtt{01} \cdot X^{2^5} + \mathtt{B5} \cdot X^{2^6} + \mathtt{8F} \cdot X^{2^7} + \mathtt{63}$

$$\Gamma_Z^{(2)} = \begin{bmatrix}
0&0&0&1&0&1&1&0 & 0&0&0&0&0&0&0&0\\
1&0&0&0&1&0&1&1 & 0&0&0&0&0&0&0&0\\
0&1&0&0&0&1&0&1 & 0&0&0&0&0&0&0&0\\
0&0&1&1&0&1&0&0 & 0&0&0&0&0&0&0&0\\
0&0&0&1&1&0&1&0 & 0&0&0&0&0&0&0&0\\
1&0&0&1&1&0&1&1 & 0&0&0&0&0&0&0&0\\
0&1&0&1&1&0&1&1 & 0&0&0&0&0&0&0&0\\
0&0&1&0&1&1&0&1 & 0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0 & 0&0&0&1&0&1&1&0\\
0&0&0&0&0&0&0&0 & 1&0&0&0&1&0&1&1\\
0&0&0&0&0&0&0&0 & 0&1&0&0&0&1&0&1\\
0&0&0&0&0&0&0&0 & 0&0&1&1&0&1&0&0\\
0&0&0&0&0&0&0&0 & 0&0&0&1&1&0&1&0\\
0&0&0&0&0&0&0&0 & 1&0&0&1&1&0&1&1\\
0&0&0&0&0&0&0&0 & 0&1&0&1&1&0&1&1\\
0&0&0&0&0&0&0&0 & 0&0&1&0&1&1&0&1
\end{bmatrix},$$

$$\Gamma_Z^{(3)} = \begin{bmatrix}
1&0&0&1&0&1&1&0 & 0&0&0&0&0&0&0&0\\
1&1&0&0&1&0&1&1 & 0&0&0&0&0&0&0&0\\
0&1&1&0&0&1&0&1 & 0&0&0&0&0&0&0&0\\
0&0&1&0&0&1&0&0 & 0&0&0&0&0&0&0&0\\
0&0&0&1&0&0&1&0 & 0&0&0&0&0&0&0&0\\
1&0&0&1&1&1&1&1 & 0&0&0&0&0&0&0&0\\
0&1&0&1&1&0&0&1 & 0&0&0&0&0&0&0&0\\
0&0&1&0&1&1&0&0 & 0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0 & 1&0&0&1&0&1&1&0\\
0&0&0&0&0&0&0&0 & 1&1&0&0&1&0&1&1\\
0&0&0&0&0&0&0&0 & 0&1&1&0&0&1&0&1\\
0&0&0&0&0&0&0&0 & 0&0&1&0&0&1&0&0\\
0&0&0&0&0&0&0&0 & 0&0&0&1&0&0&1&0\\
0&0&0&0&0&0&0&0 & 1&0&0&1&1&1&1&1\\
0&0&0&0&0&0&0&0 & 0&1&0&1&1&0&0&1\\
0&0&0&0&0&0&0&0 & 0&0&1&0&1&1&0&0
\end{bmatrix} \quad (9)$$

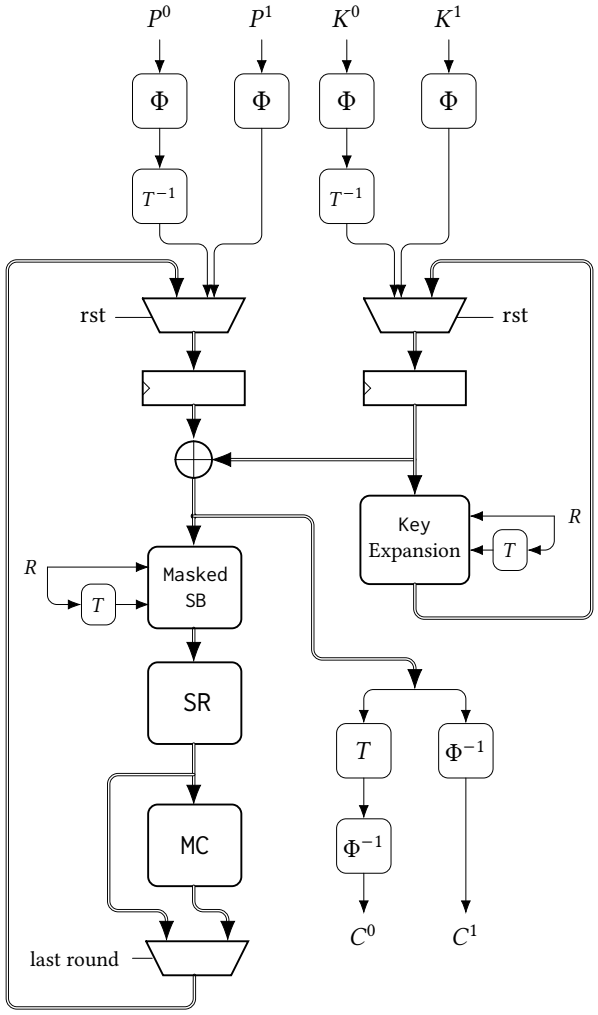


Figure 1: Round-based AES design.

3.3 Hardware Implementation

We have implemented the AES encryption using the above-given components in two distinct architectures: round-based and byte-serial. The overall structure of our round-based AES design is illustrated in Figure 1, outlining the various steps involved. Firstly, the plaintext and the key bytes are given with two shares using Boolean masking. Subsequently, the linear function $\phi(.)$ as described in Equation (3) is applied to each byte of both shares to transform the representation to $\mathbb{F}_P$. Next, the function $\mathsf{T}^{-1}(.)$ is applied to one share of each byte to convert the representation from Boolean masking to the redundant RAMBAM representation in the ring $\mathbb{R}_{P.Q}$. All the operations are then performed in $\mathbb{R}_{P.Q}$, and at the end, the reverse steps are applied. Namely, once the AES encryption in the redundant representation is accomplished, the function $\mathsf{T}(.)$ is applied to the redundant part of each ciphertext byte to convert it back to the Boolean masking form in $\mathbb{F}_P$. Finally, $\phi^{-1}(.)$ is applied to transform the representation to the standard field $\mathbb{F}_{P_0}$. At this stage, the ciphertext is trivially obtained by XORing the corresponding two shares C^0 and C^1.

The round-based implementation naturally needs 16 masked S-boxes for the data path and 4 such instances for the `KeyExpansion`. In a byte-serial implementation, this can be reduced to only one masked S-box. Due to the absence of registers in the S-box, we seamlessly integrated it into the AES byte-serial implementation design of [25], resulting in 226 clock cycles per encryption. During the first 16 clock cycles, the circuit loads the Boolean-masked key and plaintext serially while transforming them into the redundant representation by applying $\mathsf{T}^{-1}(.)$ and $\phi(.)$ to the corresponding shares. Simultaneously, the `AddRoundKey` operation is performed and the output is fed into the `ProtectedS-box`. The next four cycles are dedicated to the S-box operation for the `KeyExpansion`. Once the last state byte is obtained from the S-box module, the `ShifRows` operation is applied. Finally, the `MixColumns` operation is executed in parallel with the `KeyExpansion` in 4 clock cycles. In summary, the circuit requires 21 clock cycles to accomplish every cipher round.

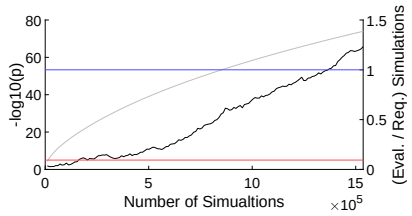
3.4 PROLEAD Evaluation

To evaluate the security of these designs, we first used PROLEAD [28], which is introduced in Section 2.6. As stated in Section 3.2, the linear operations can be realized by individual matrix multiplications on each share when $d \geq 8$, which is also the case in our implementations. However, the non-linear operation, i.e., multiplication, needs to be realized by a function dealing with both shares, which is susceptible to leak information. In order to examine this, we first evaluated a single multiplication module implemented following Algorithm 2. However, PROLEAD is designed to handle Boolean masked circuits only. To enable its application to the multiplication module, we took advantage of the property of conversion between the design in the redundant RAMBAM representation and Boolean masking, and vice versa, as also applied for the start and end of the AES encryption in Figure 1. In other words, we have supposed that two 8-bit Boolean masked inputs $\langle X^0, X^1 \rangle$ and $\langle Y^0, Y^1 \rangle$ are given while $X = X^0 \oplus X^1$ and $Y = Y^0 \oplus Y^1$. After applying $\phi(.)$ and $\mathsf{T}^{-1}(.)$ we achieved $X' : \langle T^{-1} \cdot \Phi \cdot X^0, \Phi \cdot X^1 \rangle$ and $Y' : \langle T^{-1} \cdot \Phi \cdot Y^0, \Phi \cdot Y^1 \rangle$ which are now masked in the redundant domain Z. Now we can apply the multiplication of Algorithm 2 on X' and Y'. This allows us to assess the security of such an implementation using PROLEAD while considering the specific characteristics of RAMBAM. It is worth noting that such a circuit including the multiplication by Φ and T^{-1} and the field multiplication is fully combinational without any register, which is inline with the implementations reported in the original work [1].

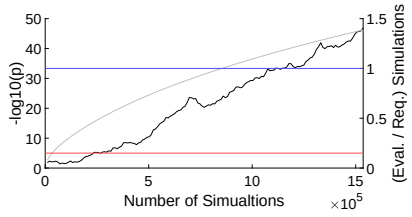
The PROLEAD results are depicted in Figure 2, where we present the progression of p-values for two separate experiments. For both experiments, we considered fixed versus random test; with fixed we gave X and Y a constant value (still uniformly shared to represent X^0 and X^1 (resp. Y^0 and Y^1), and with random we selected all inputs X^0, X^1, Y^0 and Y^1 randomly for each simulation.

In the first experiment, we set both inputs to zero, i.e., $X = Y = 0$, while in the second experiment we arbitrary selected a non-zero value for each input, i.e., $X \neq 0$ and $Y \neq 0$. The corresponding results are shown in Figure 2. As a side note, we applied the default statistical parameters proposed in the original PROLEAD publication [28], i.e., $\beta = 10^{-5}$ and $\varphi = 0.1$. Notably, in these experiments we considered only glitch-extended probing security and ignored transitions, since – as stated above – the circuit contains no registers.

The result of PROLEAD is the probability p to reject the null hypothesis, i.e., assuming that the leakages associated to two groups (fixed input versus random input) are taken from the same population and hence not distinguishable. In each plot, the minimum p-values are represented by $-\log_{10}(p)$ in black, while the horizontal red line indicates the 10^{-5} threshold. It means that if the black curve exceeds the red line, the null hypothesis is rejected. Additionally, the grey line shows the quotient between processed simulations and required simulations to achieve $\beta < 10^{-5}$ for a predefined φ. We have set another threshold at 1.0 (drawn in blue) which must be surpassed by the grey line in order to imply that PROLEAD has considered enough simulations to detect all effects with an effect size larger or equal to φ with a false-negative probability of $\beta = 10^{-5}$. Both figures show significant first-order leakage,

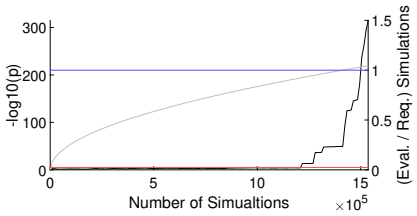


(a) Zero input vector

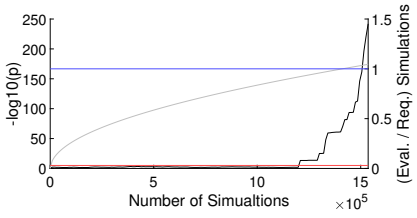


(b) Non-zero input vector

Figure 2: PROLEAD results for the RAMBAM matrix multiplication.



(a) Zero input vector



(b) Non-zero input vector

Figure 3: PROLEAD results for the RAMBAM S-Box.

which highlights the insecurity of RAMBAM multiplier with respect to the glitch-extended probing model. It is noteworthy that we observed more leakage when the fixed input was tied to zero, i.e., the first experiment. Specifically, for the zero input vector, we observed a 40% increase in the value of $-\log_{10}(p)$ compared to the non-zero input. This indicates that relying solely on a single input vector when evaluating a design might be not sufficient, and it is always suggested to evaluate a circuit with multiple fixed input vectors to make the overall results more reliable.

We would like to highlight that the observed leakage is due to the glitch-extended probes. In such a model, a probe can propagate backwards up to the last synchronization point, which in this case

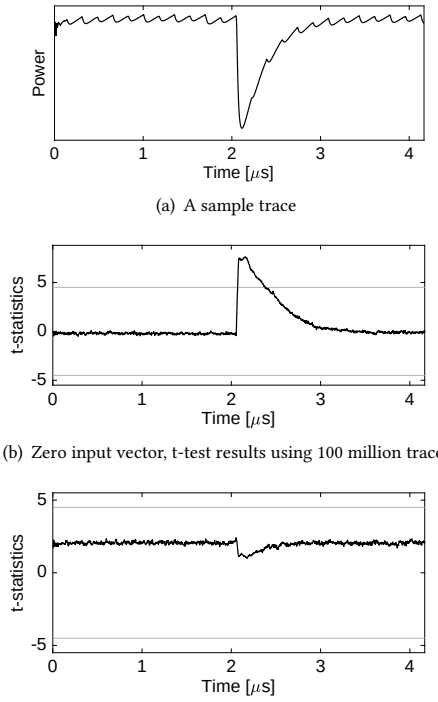
is the primary inputs of the circuit, as it does not contain any registers. We repeated both evaluations on the RAMBAM S-Box as well which is again fully combinational, i.e., without any registers. In contrast to the multiplier, the S-Box module requires $7 \times 8 = 56$ fresh masks which PROLEAD sets randomly in every clock cycle. However, the high demand for fresh masks in a fully combinational circuit results in large glitch-extended probing sets that primarily record random data. As a result, if PROLEAD evaluates a potentially leaking probing set, the probes on random data act as noise and may potentially hide the leakage from PROLEAD. Nevertheless, the leakage persists and PROLEAD reveals the leakage, as shown in Figure 3. We observed substantial leakage, in particular, the evaluation shows that the leakage can still be clearly detected by conducting around 1.5 million simulations. Again, we observed a remarkable increase in the value of $-\log_{10}(p)$ for a zero input vector compared to the non-zero case. However, due to the large number of fresh masks, this increase, about 30%, is smaller compared to Figure 2. In fact, this is consistent with our recommendation to examine the design using multiple input vectors.

3.5 SCA Experimental Evaluations

Multiplier. In order to examine our findings in real world settings, we conducted the corresponding experiments using the setup introduced in Section 2.7. In the first set of experiments, we evaluated the multiplier which we have examined by PROLEAD in Section 3.4. To this end, as explained in Section 2.5 we measured the traces suitable for a fixed vs. random t-test and conducted the corresponding analyses using two fixed values: (1) fully zero and (2) non-zero. The corresponding results using 100 million traces are shown in Figure 4 clearly indicating first-order leakage of the design when the fixed input is fully zero. We should highlight that the leakage which was reported by PROLEAD for a non-zero input vector is not detected in practice which can be due to the noise of the measurement setup and the fact that the multiplier is a tiny circuit with a low power consumption. This already highlights the importance of the selected fixed input vector in such experimental leakage assessments.

Round-based AES Encryption. In the next set of experiments, we implemented the round-based AES encryption engine depicted in Figure 1 on our measurement setup. It is noteworthy to mention that, following the RAMBAM design, we placed no registers between the square and multiply chain in the inversion module of Algorithm 1. This results in accomplishing the entire AES-128 encryption in 11 clock cycles while employing 20 instances of the masked S-box.

As explained in Section 3.2, fresh masks are added to the output of every square as well as every multiplier. In order to generate the fresh masks, we employed a 31-bit Linear Feedback Shift Register (LFSR) with a feedback polynomial of $x^{31} + x^{28} + 1$ for each required single fresh mask bit, while the LFSRs are updated at every clock cycle. While the authors of the original work proposed a scheme to minimize the number of required fresh masks by reusing some of them following a specific fashion, we did not incorporate this optimization in our implementation. Instead, we followed the square and multiply chain along with the affine transformation as given in Algorithm 1, and employed individual fresh masks at each position in the S-box. Specifically, we utilized seven 8-bit fresh masks per S-box, without reusing any of them, to avoid any probable leakage originating from the mask reuse, i.e., in total 140 fresh mask bits per clock cycle generated by 140 individual LFSRs seeded randomly at the power-up cycle. In fact, the target FPGA receives a two-share masked input, i.e., plaintext along with a masked key, and produces the ciphertext also in a two-share masked form while the fresh masks are generated internally.



(a) A sample trace

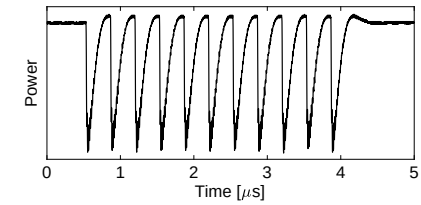
(b) Zero input vector, t-test results using 100 million traces

(c) Non-zero input vector, t-test results using 100 million traces

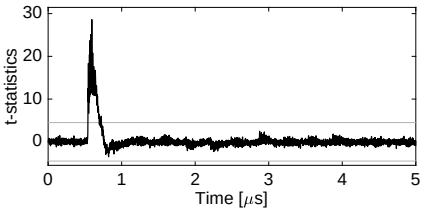
Figure 4: FPGA-based experiments on RAMBAM multiplier.

Similar to our findings with PROLEAD, the fixed-versus-random t-test experiments revealed that more leakage is detectable when the fixed input vector leads to zero at the input of the S-boxes. To emulate this, we set the fixed plaintext the same as the key, where the input of all S-boxes in the first round is set to zero. This is reflected in Figure 5 clearly indicating the presence of detectable leakage using only 1 million traces. It is noteworthy that since the fixed plaintext=key, the leakage is prominently visible in the first cipher round, but it significantly decreases in the subsequent rounds, since in the later rounds the input of the S-boxes is not zero anymore.

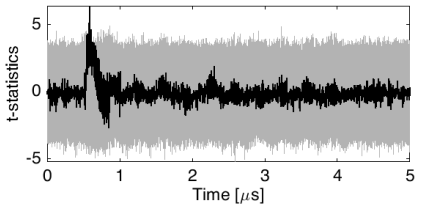
The mask refreshes in RAMBAM S-box add noise to the circuit and reduce leakage to some extent, but they do not fully prevent the leakage as the building block, i.e., the multiplier, is not secure under the glitch-extended probing model and the design contains no registers to avoid the propagation of glitches. Namely, to address the high demand of fresh randomness many LFSRs should be instantiated which would add a considerable amount of noise to the power traces.



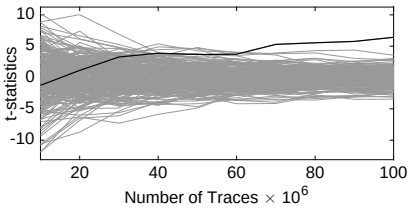
(a) A sample trace



(b) t-test results using 1 million traces



(c) DPA results using 100 million traces



(d) DPA results over number of traces

Figure 5: FPGA-based experiments on round-based RAMBAM AES.

Nevertheless, we also conducted a key-recovery attack to further evaluate the exploitability of the detected leakages. We collected 100 million traces from the round-based design when the plaintext is always selected randomly and performed classical Differential Power Analysis (DPA) attack using the well-known zero-value model, i.e.,

$$ZV(x) = \begin{cases} 0, & x = 0 \\ 1, & x \neq 0 \end{cases}.$$

In this model, the traces are divided into two groups for each key guess. One group corresponds to operations where the predicted input to the cryptographic primitive (here the S-box) is set to zero, while the other group corresponds to non-zero values. By comparing the power consumption patterns associated to these two groups for every key guess by means of student t-test, an intuition about the correct key byte may be achieved. Using 100 million traces, we could recover 4 key bytes, one of which is shown in Figure 5. In order to recover all key bytes, one can collect more traces. Alternatively, a Correlation Power Analysis (CPA) attack using a more sophisticated power model can be used to consider the leakage of more S-boxes than only one. Suppose that the key bytes k_0 and k_1 are confidently found using the classical DPA as explained above. The third key byte k_2 might be recovered using by a CPA utilizing the hypothetical power model $ZV(p_0 \oplus k_0) + ZV(p_1 \oplus k_1) + ZV(p_2 \oplus k_2)$. Considering more known key bytes in such a power model would help estimating the power consumption leading to more confident results.

Byte-serial AES Encryption. We further evaluated the byte-serial design as explained in Section 3.3. In general, the noise level of the byte-serial design is typically lower compared to the round-based design due to two main reasons. Firstly, the demand for fresh masks per clock cycle is reduced as only one S-box is instantiated. This results in a smaller circuit required for generating the necessary fresh masks, leading to a lower noise level. Secondly, in key-recovery attacks naturally, one key byte is targeted at a time, i.e., a divide-and-conquer scenario. Therefore, in the round-based design, the effect of other S-boxes can be considered as noise, making it more challenging to reveal the key as more traces are typically needed.

By repeating the same experiment as before using 1 million traces when the fixed plaintext=key, the results depicted in Figure 6 have been obtained. Note that in this experiment, we covered only the 1.5 cipher rounds. Similar to the former results, the leakage is more easily detectable in the first round.

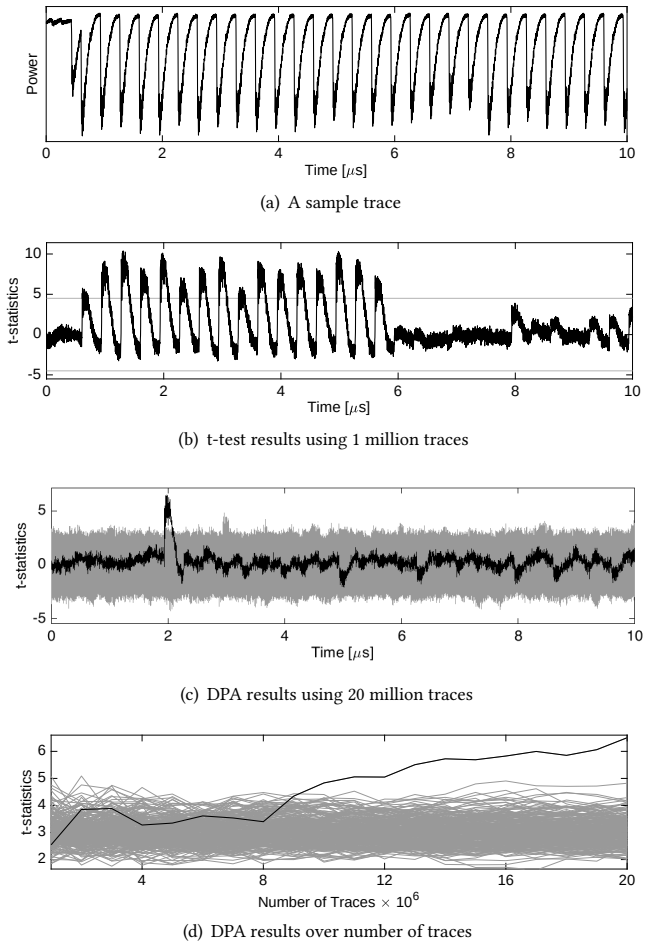
Similar to the round-based implementation, we conducted key-recovery attacks on the byte-serial design. To this end, we collected 20 million traces when the plaintext is fully selected randomly and performed similar DPA attacks using the aforementioned zero-value model. Remarkably, we can confirm the possibility of recovering all key bytes from the byte-serial implementation using around 10 million traces. One of such results is also shown in Figure 6.

3.6 Fault Injection Experimental Evaluations

The authors of RAMBAM also presented arguments regarding the resilience of their design against SIFA [11]. SIFA combines the advantageous aspect of Ineffective Fault Attack (IFA) [4] and Statistical Fault Attack (SFA) [14] principles while relaxing many of their requirements for key recovery. More precisely, SIFA exploits the dependency of ineffective fault injection and the targeted intermediate value, similar to IFA, but relies only on the statistical bias of the intermediate value for correct key hypothesis, akin to SFA. Unlike IFA, which requires a specific fault model, SIFA does not have such a requirement and can use fault-free ciphertexts to retrieve the key, which is not the case for SFA.

To initiate the attack, the adversary collects fault-free ciphertexts despite injecting a fault during encryption. Using a guessed key, the adversary partially decrypts the ciphertexts to obtain the targeted (faulted) intermediate value. Subsequently, the distribution of the intermediate value is evaluated, and the Squared Euclidean Imbalance (SEI) score is calculated for each key candidate. The key candidate with the highest SEI score is identified as the most probable correct key candidate.

In the study by Saha et al. [32], SIFA is divided into two versions: SIFA-1 and SIFA-2. In the first one, only the state variable, namely



(a) A sample trace

(b) t-test results using 1 million traces

(c) DPA results using 20 million traces

(d) DPA results over number of traces

Figure 6: FPGA-based experiments on byte-serial RAMBAM AES.

the registers, can be targeted for fault injection. On the other hand, in SIFA-2, faults are injected into the sub-operations of the cipher, such as the S-box or MixColumns operations of the AES.

According to the authors of [32], Boolean masking can provide protection against SIFA-1, as long as the fault does not corrupt all shares of a variable. Therefore, a masked implementation with s shares can provide security against up to $s-1$ faulty bits in the SIFA-1 model. The authors of RAMBAM have improved the security margin against SIFA-1, achieving at most 4 bits security with a redundancy size of $d = 8$, which is equivalent to using Boolean masking with two shares. Security of their design against SIFA-2 is not clearly specified in the paper, although the authors claimed practical security against SIFA-2 in their presentation at CHES 2022[2]. However, from a theoretical point of view, the design is insecure with faults being injected into the combinational logic of the cipher.

Apart from that, the design is vulnerable to other kinds of fault attacks including DFA, first introduced by Biham and Shamir [2]

[2]https://iacr.org/submit/files/slides/2022/tches/ches2022/2_10/slides.pptx

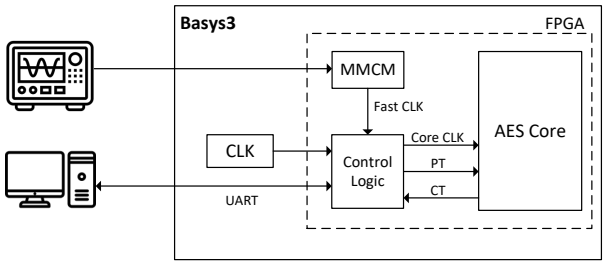


Figure 7: Fault Injection experimental setup [36].

inspired from the principles of differential cryptanalysis on round-reduced ciphers. In DFA, the adversary injects faults at specific points in the computation and then compares the faulty ciphertext with the correct one. By analyzing the difference between the two ciphertexts, some information about the key can be inferred. DFA is indeed effective against implementations like RAMBAM that use no error-detecting or error-correcting facilities. We should highlight that the authors of RAMBAM did not claim any security against DFA, but our argument is that sole protection against SIFA is not helpful. To protect against DFA, fault-detection facilities should be added to the design. Such facilities either are independent of data being processed (e.g., by detecting clock glitch, laser light, or EM pulses), or are based on concurrent error detection schemes utilizing error-detection codes. The first category can potentially protect against SIFA as well since the fault becomes effective or ineffective independent of the processed data. The second category however may give a new opportunity to successfully mount SIFA.

To demonstrate the effectiveness of the attack, we conducted DFA of the round-based AES design. Our experimental setup is similar to the one used in [36], where the fault is injected by means of the clock glitch. It utilizes a Basys3 FPGA development board from Digilent, which is equipped with a Xilinx Artix-7 (XC7A35T-1CPG236C) FPGA [10]. Further, we employed an Agilent Function/Arbitrary Waveform Generator 33521A to generate an external precise clock frequency to control the duration of the clock glitch.

In order to overcome the limitations of the function generator, which has a maximum frequency of 30 MHz, and the low-bandwidth of the I/O ports, we utilized a Mixed-Mode Clock Manager (MMCM) module on the FPGA. This allowed us to increase the frequency of the externally provided clock signal, enabling us to inject a fault into the combinational circuit. Specifically, we needed to generate a glitchy cycles in the range of 35 MHz to induce faulty behavior in the circuit.

We devised a comprehensive framework for the evaluation board that facilitates communication with the PC. The experimental setup, depicted in Figure 7, entails the reception of a plaintext and an index c by the board. Subsequently, the encryption function is executed on the given plaintext and the embedded key after being masked and transferred to the RAMBAM domain as given in Figure 1. To inject the clock glitch, the ordinary clock is replaced with a fast clock at the clock cycle c after the start of the encryption. The resulting ciphertext is then transmitted back to the PC.

Two consecutive encryptions with the same plaintext are performed on the target device: one without any fault to obtain the

correct ciphertext, and another one with a clock glitch at the 8-th encryption round. The induced faults on each byte propagate through the S-box and `ShiftRows` in the 9-th round up to the input of the `MixColumns`. This allowed us to mount state-of-the-art DFA attacks which commonly target the input of the `MixColumns` in the 9-th cipher round and model the injected fault as a linear difference between the faulty and fault-free values [12, 16, 27, 30]. In short, similar to the results reported in the literature, we successfully recovered every four bytes of the 10-th roundkey with only two faulty/fault-free ciphertext pairs. Notably, one can apply the technique presented in [40] to reveal the full key using only one fault injection. All these known attacks are possible since the design does not employ any fault-detection facility.

4 CONCLUSIONS

In this paper, we conducted an in-depth analysis of the RAMBAM technique and its relation to the commonly used Boolean masking approach. Specifically, we demonstrated that RAMBAM with a redundancy size of $d = 8$ can be seen as an alternative representation of Boolean masking with two shares, each containing full 8-bit entropy for every byte. Using the verification tool PROLEAD, we showed that the design suffers from SCA leakage in the multiplier despite the presence of 8-bit entropy. We also showed that independent of the leakage of the multiplier, the protected S-box construction is insecure due to lack of register layers to stop propagating the probes in glitch-extended probing model. We performed experimental SCA evaluations on an FPGA-based platform and illustrated that the leakage is not only detectable but also exploitable using the school-book zero-value model. In our byte-serial RAMBAM implementation, the secret key was fully recovered using around 10 million traces, while we even did not re-use any fresh randomness suggested by the original authors. In our round-based implementation of RAMBAM, where 20 S-box instances are active in parallel, we revealed 4 key bytes using 100 million traces. We should highlight that recently, i.e., after being informed about our investigations, the original authors made some of their implementations publicly available[3]. Hence, our results with respect to the number of required traces to successfully mount the attacks hold only for our implementations of RAMBAM not necessarily for the designs recently made public. However, our analyses and claims related to the insecurity of the scheme under robust probing model stay valid. As a side note, the original authors confirmed the existence of a first-order leakage in their design, but claim a very high level of effort to exploit it.

Furthermore, we evaluated the security of our implementation against fault-injection attacks and presented that avoiding SIFA-1 is insufficient, as no fault detection or correction mechanism is employed in the design, hence enabling several other attack vectors, e.g., DFA, although RAMBAM original authors did not provide any claims about the security of their design against DFA. In order to provide security against DFA, certain facilities should be integrated. However, pure fault-detection techniques may or may not turn the design susceptible to SIFA.

This work emphasizes the importance of providing security proofs alongside any countermeasures against implementation attacks. There is a considerable body of work on how masking works and how physical defaults can compromise them. Glitches are a well-known phenomenon in hardware platforms, and their potential impact was not adequately addressed during the design of RAMBAM. Actually, the original authors of RAMBAM did not claim the security of their scheme under glitch-extended probing model. They, in fact, claimed that the number of traces required to successfully attack their scheme increases exponentially with the redundancy size d. This is not what we evaluated in this work, as our main goal was to examine the scheme under common and well-known hardware masking concepts and tools as RAMBAM with $d = 8$ is equivalent to first-order Boolean masking. Indeed, one can also find such an equivalence between RAMBAM with $d = 16$ and second-order masking.

It is important to note that experimental analyses alone are insufficient to establish the security of a cryptographic design since the measurement setup can significantly impact the result. The authors of RAMBAM did not provide any proofs for the theoretical security of their scheme, and some details of their conducted evaluations are not fully given, e.g., which fixed value was used in fixed-versus-random t-tests. The authors reported that the power analysis evaluation involved collecting only four samples per clock cycle, raising questions about the reliability and accuracy of their findings. That motivated us to conduct further investigation and more rigorous testing to establish the effectiveness of the RAMBAM scheme in protecting against SCA attacks. Our analysis revealed that the scheme is indeed insecure and might be compromised by exploiting the leakage.

On the other hand, we reported that the selection of the fixed value in the fixed-versus-random t-test can significantly affect the detectability of SCA leakage of the RAMBAM design. Apparently, this was not considered by the authors of RAMBAM. This highlights the importance of using verification tools like PROLEAD to verify the security of any masked hardware design prior to experimental analysis, as such tools evaluate the circuits in the worst-case scenario.

Our designs, as well as the evaluation scripts and results (PROLEAD), are publicly available via https://github.com/ChairImpSec/RAMBAM.

[3] https://github.com/fortify-iq/fiq-openaes-128e

ACKNOWLEDGMENTS

The work described in this paper has been supported in part by the German Research Foundation (DFG) under Germany's Excellence Strategy - EXC 2092 CASA - 390781972, and through the projects 406956718 (SuCCESS), 435264177 (SAUBER), and 456967092 (SecF-Share).

REFERENCES

[1] Yaacov Belenky, Vadim Bugaenko, Leonid Azriel, Hennadii Chernyshchyk, Ira Dushar, Oleg Karavaev, Oleh Maksimenko, Yulia Ruda, Valery Teper, and Yury Kreimer. 2022. Redundancy AES Masking Basis for Attack Mitigation (RAMBAM). *IACR Trans. Cryptogr. Hardw. Embed. Syst.* 2022, 2 (2022), 69–91.

[2] Eli Biham and Adi Shamir. 1997. Differential Fault Analysis of Secret Key Cryptosystems. In *CRYPTO '97 (LNCS)*, Vol. 1294. Springer, 513–525.

[3] Suresh Chari, Charanjit S. Jutla, Josyula R. Rao, and Pankaj Rohatgi. 1999. Towards Sound Approaches to Counteract Power-Analysis Attacks. In *CRYPTO 1999 (LNCS)*, Vol. 1666. Springer, 398–412.

[4] Christophe Clavier. 2007. Secret External Encodings Do Not Prevent Transient Fault Analysis. In *CHES 2007 (LNCS)*, Vol. 4727. Springer, 181–194.

[5] Thomas De Cnudde, Begül Bilgin, Benedikt Gierlichs, Ventzislav Nikov, Svetla Nikova, and Vincent Rijmen. 2017. Does Coupling Affect the Security of Masked Implementations?. In *COSADE 2017 (LNCS)*, Vol. 10348. Springer, 1–18.

[6] Jeremy Cooper, Elke DeMulder, Gilbert Goodwill, Joshua Jaffe, Gary Kenworthy, Pankaj Rohatgi, et al. 2013. Test vector leakage assessment (TVLA) methodology in practice. In *International Cryptographic Module Conference*, Vol. 20.

[7] Jean-Sébastien Coron, Christophe Giraud, Emmanuel Prouff, Soline Renner, Matthieu Rivain, and Praveen Kumar Vadnala. 2012. Conversion of Security Proofs from One Leakage Model to Another: A New Issue. In *COSADE 2012 (LNCS)*, Vol. 7275. Springer, 69–81.

[8] Joan Daemen and Vincent Rijmen. 2002. *The Design of Rijndael: AES - The Advanced Encryption Standard.* Springer.

[9] Siemen Dhooghe, Aein Rezaei Shahmirzadi, and Amir Moradi. 2022. Second-Order Low-Randomness d + 1 Hardware Sharing of the AES. In *CCS 2022*. ACM, 815–828.

[10] Digilent. 2019. Basys3. (2019). https://reference.digilentinc.com/reference/programmable-logic/basys-3/.

[11] Christoph Dobraunig, Maria Eichlseder, Thomas Korak, Stefan Mangard, Florian Mendel, and Robert Primas. 2018. SIFA: Exploiting Ineffective Fault Inductions on Symmetric Cryptography. *IACR Trans. Cryptogr. Hardw. Embed. Syst.* 2018, 3 (2018), 547–572.

[12] Pierre Dusart, Gilles Letourneux, and Olivier Vivolo. 2003. Differential Fault Analysis on A.E.S. In *ACNS 2003 (LNCS)*, Vol. 2846. Springer, 293–306.

[13] Sebastian Faust, Vincent Grosso, Santos Merino Del Pozo, Clara Paglialonga, and François-Xavier Standaert. 2018. Composable Masking Schemes in the Presence of Physical Defaults & the Robust Probing Model. *IACR Trans. Cryptogr. Hardw. Embed. Syst.* 2018, 3 (2018), 89–120.

[14] Thomas Fuhr, Éliane Jaulmes, Victor Lomné, and Adrian Thillard. 2013. Fault Attacks on AES with Faulty Ciphertexts Only. In *FDTC 2013*. 108–118.

[15] Karine Gandolfi, Christophe Mourtel, and Francis Olivier. 2001. Electromagnetic Analysis: Concrete Results. In *CHES 2001 (LNCS)*, Vol. 2162. Springer, 251–261.

[16] Christophe Giraud. 2004. DFA on AES. In *AES 2004 (LNCS)*, Vol. 3373. Springer, 27–41.

[17] Jovan Dj. Golic and Christophe Tymen. 2002. Multiplicative Masking and Power Analysis of AES. In *CHES 2002 (LNCS)*, Vol. 2523. Springer, 198–212.

[18] Hannes Groß, Rinat Iusupov, and Roderick Bloem. 2018. Generic Low-Latency Masking in Hardware. *IACR Trans. Cryptogr. Hardw. Embed. Syst.* 2018, 2 (2018), 1–21.

[19] Hannes Groß, Stefan Mangard, and Thomas Korak. 2016. Domain-Oriented Masking: Compact Masked Hardware Implementations with Arbitrary Protection Order. In *TIS @ CCS 2016*. ACM, 3.

[20] Yuval Ishai, Amit Sahai, and David A. Wagner. 2003. Private Circuits: Securing Hardware against Probing Attacks. In *CRYPTO 2003 (LNCS)*, Vol. 2729. Springer, 463–481.

[21] Paul C. Kocher. 1996. Timing Attacks on Implementations of Diffie-Hellman, RSA, DSS, and Other Systems. In *CRYPTO 1996 (LNCS)*, Vol. 1109. Springer, 104–113.

[22] Paul C. Kocher, Joshua Jaffe, and Benjamin Jun. 1999. Differential Power Analysis. In *CRYPTO 1999 (LNCS)*, Vol. 1666. Springer, 388–397.

[23] Stefan Mangard, Elisabeth Oswald, and Thomas Popp. 2007. *Power analysis attacks - revealing the secrets of smart cards.* Springer.

[24] Stefan Mangard, Norbert Pramstaller, and Elisabeth Oswald. 2005. Successfully Attacking Masked AES Hardware Implementations. In *CHES 2005 (LNCS)*, Vol. 3659. Springer, 157–171.

[25] Amir Moradi, Axel Poschmann, San Ling, Christof Paar, and Huaxiong Wang. 2011. Pushing the Limits: A Very Compact and a Threshold Implementation of AES. In *EUROCRYPT 2011 (LNCS)*, Vol. 6632. Springer, 69–88.

[26] Amir Moradi and Tobias Schneider. 2016. Improved Side-Channel Analysis Attacks on Xilinx Bitstream Encryption of 5, 6, and 7 Series. In *COSADE 2016 (LNCS)*, Vol. 9689. Springer, 71–87.

[27] Amir Moradi, Mohammad T. Manzuri Shalmani, and Mahmoud Salmasizadeh. 2006. A Generalized Method of Differential Fault Attack Against AES Cryptosystem. In *CHES 2006 (LNCS)*, Vol. 4249. Springer, 91–100.

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

[29] Christof Paar, Thomas Eisenbarth, Markus Kasper, Timo Kasper, and Amir Moradi. 2009. KeeLoq and Side-Channel Analysis-Evolution of an Attack. In *FDTC 2009*. IEEE Computer Society, 65–69.

[30] Gilles Piret and Jean-Jacques Quisquater. 2003. A Differential Fault Attack Technique against SPN Structures, with Application to the AES and KHAZAD. In *CHES 2003 (LNCS)*, Vol. 2779. Springer, 77–88.

[31] Axel Poschmann, Amir Moradi, Khoongming Khoo, Chu-Wee Lim, Huaxiong Wang, and San Ling. 2011. Side-Channel Resistant Crypto for Less than 2, 300 GE. *J. Cryptol.* 24, 2 (2011), 322–345.

[32] Sayandeep Saha, Dirmanto Jap, Debapriya Basu Roy, Avik Chakraborty, Shivam Bhasin, and Debdeep Mukhopadhyay. 2020. A Framework to Counter Statistical Ineffective Fault Analysis of Block Ciphers Using Domain Transformation and Error Correction. *IEEE Trans. Inf. Forensics Secur.* 15 (2020), 1905–1919.

[33] SAKURA. 2016. Side-channel Attack User Reference Architecture. http://satoh.cs.uec.ac.jp/SAKURA/index.html. (2016).

[34] Pascal Sasdrich, Begül Bilgin, Michael Hutter, and Mark E. Marson. 2020. Low-Latency Hardware Masking with Application to AES. *IACR Trans. Cryptogr. Hardw. Embed. Syst.* 2020, 2 (2020), 300–326.

[35] Tobias Schneider and Amir Moradi. 2015. Leakage Assessment Methodology - A Clear Roadmap for Side-Channel Evaluations. In *CHES (LNCS)*, Vol. 9293. Springer, 495–513.

[36] Aein Rezaei Shahmirzadi and Amir Moradi. 2020. Clock Glitch versus SIFA. In *DFT 2020*. IEEE, 1–6.

[37] Aein Rezaei Shahmirzadi and Amir Moradi. 2021. Re-Consolidating First-Order Masking Schemes Nullifying Fresh Randomness. *IACR Trans. Cryptogr. Hardw. Embed. Syst.* 2021, 1 (2021), 305–342.

[38] Adi Shamir. 1979. How to Share a Secret. *Commun. ACM* 22, 11 (1979), 612–613.

[39] Kris Tiri and Ingrid Verbauwhede. 2004. A Logic Level Design Methodology for a Secure DPA Resistant ASIC or FPGA Implementation. In *DATE 2004*. IEEE Computer Society, 246–251.

[40] Michael Tunstall, Debdeep Mukhopadhyay, and Subidh Ali. 2011. Differential Fault Analysis of the Advanced Encryption Standard Using a Single Fault. In *WISTP 2011 (LNCS)*, Vol. 6633. Springer, 224–233.

[41] Xin Ye and Thomas Eisenbarth. 2013. On the Vulnerability of Low Entropy Masking Schemes. In *CARDIS 2013 (LNCS)*, Vol. 8419. Springer, 44–60.

[42] Sara Zarei, Aein Rezaei Shahmirzadi, Hadi Soleimany, Raziyeh Salarifard, and Amir Moradi. 2021. Low-Latency Keccak at any Arbitrary Order. *IACR Trans. Cryptogr. Hardw. Embed. Syst.* 2021, 4 (2021), 388–411.