Consolidated Linear Masking (CLM)

Generalized Randomized Isomorphic Representations, Powerful Degrees of Freedom and Low(er)-cost...

Itamar Levi[1] and Osnat Keren[1]

Bar-Ilan University, Faculty of Engineering, Computer-Engineering, Ramat-Gan 5290002, Israel.
Email: {first-name}.{last-name}@biu.ac.il

Abstract. Masking is a widely adopted countermeasure against side-channel analysis (SCA) that protects cryptographic implementations from information leakage. However, current masking schemes often incur significant overhead in terms of electronic cost. RAMBAM, a recently proposed masking technique that fits elegantly with the AES algorithm, offers ultra-low latency/area by utilizing redundant representations of finite field elements. This paper presents a comprehensive generalization of RAMBAM and various other masking schemes within a unified framework and a mathematical representation known as Consolidated Linear Masking (CLM), where masking schemes are formalized by their encoding. We establish a theoretical foundation for CLM linking randomized isomorphic (code) representations and the entropy provided by the redundancy to a revised notion of masking order. Our analysis reveals that RAMBAM is a specific instance of CLM as well as other masking constructions, thus paving the way for significant enhancements. For example, a 1^{st}-order secure design can be achieved almost without increasing the size of the representation of the variables. This property scales up to any order and is versatile. We demonstrate how CLM enables: (1) randomized selection of the isomorphic field for improved security; (2) flexible choice of the randomization polynomial; (3) embedded mask-refreshing via the randomized isomorphic representation that reduces randomness requirements significantly as well as improves performance; (4) a wider range of isomorphic randomized mappings that significantly increases the available randomization space compared to RAMBAM; (5) considerable improvement in securing fault-injection attacks and inherent security against probing adversaries, i.e., more required probes. In addition, our framework addresses ways to improve the brute-force parameter choices in the original RAMBAM. By offering a unifying theoretical perspective for masking and practical enhancements, this work advances the design of efficient and secure masking countermeasures against SCA threats.

Keywords: Algebraic representation · Consolidated Linear Masking · CLM · Isomorphic Fields · Side Channel Analysis · SCA · Masking · Masking Order · RAMBAM · Randomization · Redundancy Entropy · Rings

1 Introduction

Masking is a widely acknowledged countermeasure against side-channel analysis (SCA) attacks that exploit physical information leakage such as power consumption or timing to extract secrets. It provides robust protection for various cryptographic algorithms such as the Advanced Encryption Standard (AES) from such hardware deficits. However, masking techniques often introduce substantial implementation overhead, including increased area, latency, or both, and the need for quite large fresh randomness. Among others, most notably for hardware designs, concrete steps are needed to mitigate physical vulnerabilities such as glitches, memory transitions and compositional issues.

Several masking schemes have been proposed to protect against side-channel attacks, each of which has distinct characteristics. The masking literature is vast, and includes several watershed constructions.The Ishai-Sahai-Wagner (ISW) masking scheme [ISW03], a foundational approach, provides provable security against a limited number of probes but often incurs significant overhead due to its reliance on expensive gadgets for non-linear operations, and encapsulates a formulation-oriented software implementation. Generic Low-Latency Masking (GLM) [GIB18] aims to reduce this overhead by minimizing the number of non-linear operations, albeit by sacrificing some provable security guarantees (which indeed result in attacks and sensitivities, e.g., [DCEM18, MMSS19]). The Consolidated Masking Scheme (CMS) [RBN+15] focuses on optimizing randomness in masking to reduce the amount of fresh randomness required (and later attacks and improvements [SM21, DCEM18, MMSS19]). In turn, Domain-Oriented Masking (DOM) [GMK18] tailors masking to specific algebraic structures and the splitting of shared interactions with randomness and refreshing, thus offering efficiency gains for certain algorithms (with some fixed vulnerabilities in some scenarios and compositions and more advanced variants [MMSS19]). More recently, Hardware Private Circuits (HPC) [CGLS20] were put forward, which provide a hardware-based solution by transforming the circuit into a provably secure form in the hardware scenario which is composable and glitch- resistant. Along with these advances, several polynomial masking variants, including higher-order masking of field representations and inner-product masking have generalized the concept of masking to higher-degree polynomials. These provide stronger security against higher-order attacks but at the cost of increased complexity and overhead [BFG+17]. Each of these schemes thus presents a unique trade-off between security under different adversarial scenarios for different hardware or software contexts, performance needs, and implementation complexity. The choice of masking scheme hence depends on the specific requirements of the application, the target platform, and the desired level of security assurance. Implementation efficiency is also cardinal and generally require high expertise in both hardware [CGLS20, GMK18] and software [SL23].

Of the various masking schemes that aim to trade off security and electronic cost factors and overheads, RAMBAM, which was presented in 2022 at CHES [BBA+22], is in a league of its own in terms of its ultra-low latency design. RAMBAM, specifically tailored for AES, leverages redundant representations of finite field elements to resist both passive (observing side-channel information without affecting the device) and active (intentionally manipulating the device's operation to induce side-channel leakage) physical attacks.

In this paper, we first delve deeper into RAMBAM's underpinnings. We then present a comprehensive generalization showing that RAMBAM is one instance of a consolidated form for which we provide a mathematical formulation and analysis. Our theoretical investigation reveals that the general framework, denoted Consolidated Linear Masking (CLM), is ultra-intuitive and links randomized representation entropy to the revised associated masking-order. This masking framework encompasses various linear transformations and randomization techniques. The RAMBAM approach involves transforming elements to an isomorphic field, randomizing them, and mapping them back to the original field.

Our analysis demonstrates that this isomorphic mapping can be viewed as a form of CLM, thus enabling us to generalize and enhance RAMBAM's technique. This generalization opens up opportunities for: (1) Randomizing the choice of the isomorphic field for added security. (2) Flexible selection of a randomization polynomial Q, subject to specific requirements. (3) Implementing embedded mask refreshing with each atomic operation, potentially reducing the need for dedicated refresh cycles. (4) Expanding the range of possible mappings beyond RAMBAM's fixed set, by increasing the randomness offered by the masking and reducing the cost and amount of randomness from the protocol. We also address some of the potential weaknesses arising from the brute-force choices of parameters in the original RAMBAM proposal.

This work provides a theoretical foundation for CLM by associating it with our consolidated notion of masking and masking-*order*. In so doing, we show that RAMBAM is one (rather weak) offspring of CLM, which should help contribute to the development of more secure and efficient masking techniques against SCA threats. Note that in this short note, we only consider SCA and leave fault-injection (and Statistical Ineffective Fault Attacks, SIFA) for future investigation. However, we generalize and list the properties provided by the CLM scheme against fault-injection (FI). Crucially, we highlight why CLM is far more secure against a probing-adversary than conventional masking schemes.

Paper organization. The paper starts with a short background reiterating the basic definitions in [BBA+22].Then, in Section 3 we provide the formal definitions of the notations used in this paper and the basic formulations for representing masking in a general coding-sense form that links the (generalized) notion of masking order and entropy (Sub-section 3.2), including developing isomorphic representations, the code generator matrix and how to compute it in the masked domain efficiently in Sub-sections 3.3, 3.4, and 3.5, respectively. For completeness we then elaborate on how to transform to and from the mask domain in Sub-section 3.6. Finally, in Sub-section 3.7 we elaborate on FI and probing strength. In Section 4 we provide simulation and modeling results supporting various aspects of the construction, including validation of d^{th} order leakage with the CML multiplier and fixing the multiplier proposed by [BBA+22], thus illustrating the weaknesses in previous constructions and the significant added (security) value of CLM.

2 Brief Background

In [BBA+22] the authors presented RAMBAM, an isomorphic representation of variables which in addition enables additional randomized redundancy in the representation of the variables. The AES algorithm was adapted to fit the proposed representation. Their main idea was choose two good (in terms of information leakage) irreducible polynomials: $P(x)$ of degree m and $Q(x)$ of degree d'. Since the original AES polynomial, $P_0(x)$ may not be the optimal polynomial in terms of information leakage, the SBOX's input, $v(x)$, which is an element of the original finite field defined by $P_0(x)$ is first mapped to the element $y(x)$ in the isomorphic finite field defined by $P(x)$ Then $y(x)$ is masked by randomly mapping it to an element $u(x)$ in the polynomial ring $\mathbb{R}_2[x]/P(x)Q(x)$ where the SBOX function is calculated. That is,

$$u(x) = y(x) + r(x)P(x), \quad \deg(u(x)) < m + d'. \tag{1}$$

The SBOX's output is mapped back to the finite field $\mathbb{F}_2[x]/P(x)$ by applying modulo $P(x)$ and then is brought back to the original field. Since P and Q are carefully selected, the randomization is reflected in the choice of $r(x)$. Based on experimental results, the authors recommended using $P(x) = x^8 + x^6 + x^5 + x^3 + 1$ which turned out to be better than the original AES polynomial $P_0(X) = x^8 + x^4 + x^3 + x + 1$.

In this paper we show that Eq.1 is only one case (and equivalent) to what we dub consolidated linear masking (CLM), and hence can also benefit from:

1. Formulating a consolidated notion of linear masking, which subsumes classical linear masking schemes. RAMBAM is only one instance of this class. We also provide a security order notion that corresponds in some cases to the classical masking security order, along with its relationship to the entropy.

2. Choosing the isomorphic field at random that is kept constant throughout the encryption cycle. In contrast to [BBA+22], we define the isomorphism using all the roots of the irreducible polynomials of degree m. Thus we have m times more randomness. This significantly increases the potential randomization of previous constructions.

3. Showing that choosing a polynomial Q at random is possible and that it provides inherent embedded mask-refresh for any atomic operation using a random Q[1].

3 Consolidated linear masking

3.1 Notations

Denote $\mathbb{F}_2^k$ a vector space of dimension k over $\mathbb{F}_2 = GF(2)$. An element $v = (v_0, v_1, \ldots, v_{m-1}) \in \mathbb{F}_2^k$ can be referred to as the coefficient vector of a binary polynomial $v(x) = \sum_{i=0}^{k-1} v_i x^i$ of degree less than k. We call the bijective mapping between a vector space and a group of polynomials of the same cardinality the *natural mapping*.

In this paper we use three algebraic structures:

- $\mathbb{F}_{2^m}$ denotes a finite field over $\mathbb{F}_2$ defined by an irreducible polynomial $\pi(x)$ of degree m. That is, $\mathbb{F}_{2^m} \triangleq \mathbb{F}_2[x]/\pi(x)$. An element in $\mathbb{F}_{2^m}$ can also be represented as a linear combination of m linearly independent elements in the field. For example, let $\alpha \in \mathbb{F}_{2^m}$ be a root of $\pi(x)$, then $v = \sum_{i=0}^{m-1} v_i \alpha^i$. There are many sets of m linearly independent elements in $\mathbb{F}_{2^m}$ that form a basis. We call the set of the first m powers of α the natural basis.
- $\mathbb{R}_{2^n}$ denotes a polynomial ring over $\mathbb{F}_2$ where multiplication is defined modulo a polynomial $h(x)$ of degree n. That is, $\mathbb{R}_{2^n} \triangleq \mathbb{R}_2[x]/h(x)$.
- $\mathbb{G}_{2^\ell}$ denotes a group of binary polynomials of degree less than ℓ over $\mathbb{F}_2$.

When it is clear from the context which irreducible polynomial defines the algebric structure, we use simpler notation (e.g., $\mathbb{F}_{2^m}$), otherwise it is written out explicitly (e.g. $\mathbb{F}_2[x]/\pi(x)$). In addition, when it is clear from the context, we do not explicitly state whether a variable is a vector of length m, n or ℓ or an element in $\mathbb{F}_{2^m}, \mathbb{R}_{2^n}$ or $\mathbb{G}_{2^\ell}$, respectively.

Let $n \geq m$. A surjective function f from the polynomial ring $\mathbb{R}_{2^n}$ to the polynomial ring (or field) $\hat{\mathbb{R}}_{2^m}$ induces a bijection defined on a quotient of its domain. Specifically, the elements of $\mathbb{R}_{2^n}$ are divided into 2^m equivalence classes under f. That is, two elements are in the same equivalence class, $u_1 \sim u_2$, if $f(u_1) = f(u_2)$. The function f sends an element $u \in \mathbb{R}_{2^n}$ to its equivalence class $[u]$ and then bijects it to $v \in \hat{\mathbb{R}}_{2^m}$.

A ring homomorphism is a structure-preserving mapping function f between two rings. It preserves the unit elements as well as addition and multiplication. If $\pi_2(x)$ that defines $\hat{\mathbb{R}}_{2^m}$ is a factor of $\pi_1(x)$ that defines $\mathbb{R}_{2^n}$ then we have a ring homomorphism between the two rings where: $v(x) = f(u(x)) = u(x) \mod \pi_2(x)$. Any two finite fields of the same order are isomorphic, i.e., the homomorphism is bijective.

3.2 General notions of masking and the masking order

Conceptually, masking deals with a redundant representation of an m-bit information vector $v \in \mathbb{F}_2^m$ at the SBOX's input as an n-bit tuple u. Thus, it involves partitioning a vector space $\mathbb{F}_2^n$ into 2^m equivalence classes. In linear masking, the elements in each equivalence class form a coset that contains 2^{n-m} elements. Each information vector corresponds to a polynomial $v(x) \in \mathbb{F}_{2^m}$ which is also a member of a different coset of $\mathbb{R}_{2^n}$. The key idea in masking is to represent v by one of the vectors (polynomials) picked at random from the corresponding coset of $\mathbb{R}_{2^n}$. Formally,

$$u = v \cdot (L, 0) + r \cdot (G_l, G_m) = (v, r) \begin{pmatrix} L & 0 \\ G_l & G_m \end{pmatrix} = (v, r)M \tag{2}$$

[1]in previous constructions Q was chosen by brute-force and was deterministic for each instance; here we relax this limitation

where $r \in \mathbb{F}_2^{n-m}$ is a random vector, $G_l \in \mathbb{F}_2^{(n-m)\times m}$, $G_m \in \mathbb{F}_2^{(n-m)\times(n-m)}$ and $L \in \mathbb{F}_2^{m\times m}$. Matrix L must be non singular (otherwise the information v cannot be recovered from u), and matrix $G = (G_l, G_m)$ is a full row rank matrix.

Let $H = (I, H_2)$ be the matrix of row rank m for which $HG^T = 0$. The information v is recovered from a given tuple u by calculating

$$v = u \cdot H^T L^{-1}. \tag{3}$$

The order of the masking, denoted by d, is the ratio of the number of random bits to the [number of?] information bits.

Definition 1 (Order of linear masking)**.** Denote by $\mathcal{H}(v)$ the entropy of a random variable v and by $\mathcal{H}(u|v)$ the conditional entropy of a random variable u given v, then

$$d = \frac{\mathcal{H}(u|v)}{\mathcal{H}(v)} = \frac{\mathcal{H}(r|v) + \mathcal{H}(vL|v)}{\mathcal{H}(v)}. \tag{4}$$

Remark 1. If the non-singular matrix L is fixed, then $\mathcal{H}(vL|v) = 0$.

Remark 2. In [BBA+22], L is fixed and $n = m + d'$, thus RAMBAM provides masking of order $d = 1 + d'/m$.

Remark 3. In linear masking, $L = I$, $G = (B|I)$ and $n = (d+1)m$. Therefore, $\mathcal{H}(u|v) = dm$ and hence Def. 1 matches the conventional definition of d'th order masking. That is, u is considered a tuple that consists of $d + 1$ shares.

Remark 4. For simplicity (ease of implementation and comparatively low performance overhead) in Section 3.1 we define the algebraic structures over $\mathbb{F}_2$. Namely, we deal with Boolean masking. However, the masking described in Eq. 2 may be performed (with proper adjustments) over any $\mathbb{F}_q$ for which $q = 2^s$ and $s|m$ or $m|s$. Note that masking techniques with higher algebraic complexity, i.e., defined over $\mathbb{F}_q$, provide more security than Boolean masking at the cost of higher overhead; for example, the software implementation of the inner product masking in [BFG15] was found to be four times slower than Boolean masking.

Remark 5. When considering conventional masking, it is convenient to treat v, r and the matrix B as structures over $\mathbb{F}_{2^m}$:

$$\hat{u} = (\hat{v}, \hat{r}) \begin{pmatrix} \hat{l} & 0_{1\times d} \\ \hat{B}_{d\times 1} & I_{d\times d} \end{pmatrix}. \tag{5}$$

Here, we used the superscript 'hat' to emphasize the fact that the elements are from $\mathbb{F}_{2^m}$ and not binary vectors in $\mathbb{F}_2^m$. In what follows, when it is clear from the context we omit the hat. In general, the multiplication of a variable v by a constant $l \in \mathbb{F}_{2^m}$ is equivalent (under natural one-to-one mapping) to the multiplication of the corresponding binary vector in $\mathbb{F}_2^m$ by the binary $L_{m\times m}$ matrix that represents that constant. Therefore, Eq. 2 and Eq. 5 are equivalent. In this work, we prefer the binary representation because it provides more latitude.This is because we can use various L's, but more importantly, n does not have to be a multiple of m. Consequently, d can be a fraction (and even less than 1) and we work over (relatively small) polynomial rings rather than over fields.

It follows from Eq. 4 that drawing a matrix L at random increases the order of the masking scheme. Here, we are interested in isomorphic rings (see rationale below). Thus, we consider a subset $\mathcal{L}$ of matrices that function as isomorphisms between two finite fields of the same order. These matrices change the basis vector in a way that preserves the unit elements $(0, 1 \in \mathbb{F}_{2^m})$. Thus, this choice of $\mathcal{L}$ slightly reduces the entropy of vL given v. That is,

$$d = \frac{\mathcal{H}(r) + \mathcal{H}(vL|v)}{\mathcal{H}(v)} = \frac{(n-m) + (2^{m-1}-1)/2^{m-1}\mathcal{H}(L)}{m}.$$

By applying the Gauss formula, the number of monic irreducible binary polynomials of degree m is given by

$$\frac{1}{m}\sum_{a|m}\mu(m/a)2^a$$

where a runs over the set of all positive divisors of m including 1 and m. and $\mu(r)$ is the Moebius function. That is, $\mu(1) = 1$ and $\mu(r) = 1$ (or -1) when the number of distinct prime factors of r is even (or odd), otherwise, it equals zero. Thus,

$$\mathcal{H}(L) = \log_2\left(\sum_{a|m}\mu(m/a)2^a\right) < m.$$

For example, for $m = 8$ we have

$$\begin{aligned}\mathcal{H}(L) &= \log_2(\mu(8/1)2^1 + \mu(8/2)2^2 + \mu(8/4)2^4 + \mu(8/8)2^8)\\ &= \log_2(0\cdot 2^1 + 0\cdot 2^2 - 1\cdot 2^4 + 1\cdot 2^8) = \log_2(240)\\ &= 7.9069 \end{aligned} \tag{6}$$

and

$$d = \frac{n}{8} - 0.0194.$$

That is, (an almost) 1^{st}-order security can be achieved with no added bits ($n - m = 0$) in the representation of the variables. This is illustrated in Subsection 4.3. *Clearly choosing/computing once at random L (and L^{-1}) may be far cheaper than increasing register/variable sizes for the entire algorithm, and reflects a property which may be desirable.*

3.3 The need for isomorphism - restrictions on the L matrix

An SBOX output is usually calculated over the finite field $\mathbb{F}_2[x]/P_0(x)$ where P_0 is the AES polynomial. In masking-based implementations, the computation is performed in a ring $\mathbb{R}_{2^n}$. The result is only projected back to the original field at the end of the encryption (decryption). Thus, it is essential to have ring homomorphism between $\mathbb{R}_{2^n}$ and $\mathbb{F}_{2^m}$. This implies that:

1. L should exhibit isomorphism between the original field $\mathbb{F}_2[x]/P_0(x)$ and another field $\mathbb{F}_2[x]/P(x)$ of the same cardinality, and

2. There should be homomorphism between $\mathbb{R}_2[x]/h(x)$ and $\mathbb{F}_2[x]/P(x)$. In other words, $P(x)$ must divide $h(x)$.

Specifically, let $P_0(x)$ and $P(x)$ be two irreducible polynomials of degree m. Let α and $\beta \in \mathbb{F}_2[x]/P_0$ be roots of $P_0(x)$ and $P(x)$ respectively; $P_0(\alpha) = P(\beta) = 0$ and $\beta^j = \sum_{i=0}^{m-1} a_{j,i}\alpha^i$. Then, the isomorphism is defined by the matrix L:

$$\begin{pmatrix}\beta^0\\ \beta^1\\ \beta^2\\ \vdots\\ \beta^{m-1}\end{pmatrix} = \underbrace{\begin{pmatrix}1 & 0 & \cdots & 0\\ a_{1,0} & a_{1,1} & \cdots & a_{1,m-1}\\ a_{2,0} & a_{2,1} & \cdots & a_{2,m-1}\\ \vdots & \vdots & \cdots & \vdots\\ a_{m-1,0} & a_{m-1,1} & \cdots & a_{m-1,m-1}\end{pmatrix}}_{L^{-1}}\begin{pmatrix}\alpha^0\\ \alpha^1\\ \alpha^2\\ \vdots\\ \alpha^{m-1}\end{pmatrix} \tag{7}$$

That is, an element $v \in \mathbb{F}_2[x]/P_0(x)$ is mapped to $vL \in \mathbb{F}_2[x]/P(x)$.

For $m = 8$, there are 30 irreducible polynomials of degree 8. Two polynomials have roots of order 17, four have roots of order 51, eight of order 85, and 16 polynomials are primitive (i.e., their roots are of order 255). In other words, $|\mathcal{L}| = 240$ and each $L \in \mathcal{L}$ defines an isomorphic mapping between the original finite field defined by $P_0(X)$ to a finite field of order 256. *One of the main added values is that as compared to [BBA$^+$22], we define the isomorphism using all the roots of the irreducible polynomials of degree m, so that we have m times more randomness.*

It is worth noting that isomorphism inherently weakens security. As we show in Subsection 4.2, the t-test fails (i.e., it indicates that there is an information leakage) when $n = m$ and the input to the SBOX is $\mathbf{0}^m$ or $(\mathbf{0}^{m-1}1)$, where here $\mathbf{a}^s$ stands for a vector of a's of length s.

3.4 The structure of the G matrix

From a coding theory point of view, a polynomial $P(x)$ of degree m can serve as a generator polynomial of a (shortened) cyclic code $\mathcal{C}$ of length n and dimension $n - m$. If n is a factor of $2^m - 1$ the code is cyclic; otherwise, it is a shortened cyclic code.

The encoder of a cyclic code maps an $(n - m)$-bit information word, $r \in \mathbb{F}_2^m$, into an n bit codeword in $\mathcal{C} \subset \mathbb{F}_2^n$. There are several types of encoders:

- A convolution encoder performs a convolution between the coefficients of $r(x)$ and $P(x)$, that is $c(x) = r(x)P(x)$. Equivalently, $c = rG$ where the generator matrix G is of the form

$$G_{(n-m)\times n} = \begin{pmatrix} p_0 & p_1 & p_2 & \cdots & p_m & & \\ & p_0 & p_1 & \cdots & p_{m-1} & p_m & \\ & & \cdots & \cdots & \cdots & \cdots & \\ & & \underbrace{p_0 \quad \cdots \quad \cdots \quad \cdots \quad p_m}_{(m+1)} & & & & \end{pmatrix} \tag{8}$$

- A systematic encoder maps the information word $r(x)$ to a codeword $c(x) = x^m r(x) - (x^m r(x) \mod P(x))$. In matrix notation, $c = rG^{sys}$ where the *generator matrix* of the systematic code is

$$G^{sys}_{(n-m)\times n} = (B_{(n-m)\times m} | I_{(n-m)\times(n-m)})$$

  and the rows of B are the m-bit binary representation of the polynomials $-x^{m+i} \mod P(x)$ for $i = 0, 1, \ldots, n - m - 1$. That is,

$$B_{k\times m} = \begin{pmatrix} -x^m \mod P(x) \\ -x^{m+1} \mod P(x) \\ -x^{m+2} \mod P(x) \\ \vdots \\ -x^{m+k-1} \mod P(x) \end{pmatrix}. \tag{9}$$

- A matrix G whose rows are $n - m$ linearly-independent codewords of $\mathcal{C}$ picked at random.

The code $\mathcal{C} \in \mathbb{R}_{2^n}$ consists of 2^{n-r} binary vectors (codewords). Consequently, $u \in \mathbb{R}_{2^n}$ from Eq. 2 represents a word in a coset of $\mathcal{C}$ (as does the polynomial $u(x)$ in Eq. 1). Therefore, a check matrix of the code $\mathcal{C}$ should be used to recover r and hence v from u (refer to Eq. 2 and 3). One of check matrices is $H = (I_{m\times m} | B^T_{(n-m)\times m})$.

Example 1. Let $m = 4, n = 7$ and $P_0(x) = x^4 + x + 1, P(x) = x^4 + x^3 + 1$. Denote by α a root of P_0. The four roots of P in $\mathbb{F}_2[x]/P_0(x)$ and the corresponding isomorphisms are:

$$\beta_1 = \alpha^7 = \alpha^3 + \alpha + 1 = (1101) \quad \rightarrow \quad L_1^{-1} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 1 \\ 1 & 0 & 0 & 1 \\ 0 & 0 & 1 & 1 \end{pmatrix}, L_1 = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 \\ 1 & 0 & 1 & 1 \\ 1 & 0 & 1 & 0 \end{pmatrix}$$

$$\beta_2 = \alpha^{11} = \alpha^3 + \alpha^2 + \alpha = (0111) \quad \rightarrow \quad L_2^{-1} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 \\ 1 & 1 & 0 & 1 \\ 0 & 0 & 0 & 1 \end{pmatrix}, L_2 = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 1 & 0 & 1 & 1 \\ 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

$$\beta_3 = \alpha^{13} = \alpha^3 + \alpha^2 + 1 = (1011) \quad \rightarrow \quad L_3^{-1} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 1 & 0 & 1 & 1 \\ 0 & 1 & 1 & 1 \\ 0 & 1 & 0 & 1 \end{pmatrix}, L_3 = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 \end{pmatrix}$$

$$\beta_4 = \alpha^{14} = \alpha^3 + 1 = (1001) \quad \rightarrow \quad L_4^{-1} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 \\ 1 & 0 & 1 & 1 \\ 1 & 1 & 1 & 1 \end{pmatrix}, L_4 = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 \\ 0 & 1 & 1 & 0 \\ 1 & 1 & 0 & 0 \end{pmatrix}$$

The generator and the check matrices for the code $\mathcal{C}$ generated by $P(x)$ are

$$G^s_{(n-m)\times n} \;=\; (B|I) = \left(\begin{array}{cccc|c} 1 & 0 & 0 & 1 & \\ 1 & 1 & 0 & 1 & I \\ 1 & 1 & 1 & 1 & \end{array}\right), \qquad H_{m\times n} = (I|B^T).$$

Note that there are three irreducible primitive polynomials of degree 4. Thus, the order of our masking scheme is $d_{CLM} = ((7-4) + 7/8 \cdot \log_2(3 \cdot 4))/4 = 1.5342$ which is larger than the order of the corresponding conventional first order masking ($d = 1, m = 4, n = 8$) with two shares per bit.

3.5 Elementary calculations in the mask domain $\mathbb{R}_{2^n}$

In general, the SBOX operation is defined over $\mathbb{F}_{2^m}$. That is, its output is calculated by applying elementary addition and multiplication operations between elements, say v_1 and v_2 in the field $\mathbb{F}_{2^m}$. However, in masking-based implementation, each elementary calculation in $\mathbb{F}_{2^m}$ is replaced by a sequence of new elementary calculations operating on the n-bit vectors u_1 and u_2 (or, as in the conventional case, on the $(d+1)$ shares). To minimize possible information leakage via side channels, after every atomic operation in $\mathbb{R}_2^n$ (equivalently, in $\mathbb{R}_{2^n}$), the n-bit vector is replaced by another element in its equivalence class.

To obtain ring homomorphism, the procedure is to set $h(x) = P(x)Q(x)$ ($\deg(h(x)) = n$) and perform the computation over $\mathbb{R}_2[x]/h(x)$.

The choice of Q (and hence $h(x)$) does not affect the result of the $ADD(u_1, u_2)$ operation since it involves addition of a polynomial of degree $< n$.

Similarly, the choice of Q does not affect the result of the $MUL(u_1, u_2)$ operation because a mask refresh is performed after each operation. Formally, multiplication of two polynomials of degree less than n results in a polynomial of degree less than $2n-1$. Denote by $\mathcal{C}$ and $\hat{\mathcal{C}}$ (shortened) cyclic codes of lengths n and $2n-1$ generated by $P(x)$, respectively. The group $\mathbb{G}_{2^{2n-1}}$ can then be partitioned into 2^m cosets of the code $\hat{\mathcal{C}}$. The n-bit codewords of $\mathcal{C}$ can be referred to as polynomials in $\mathbb{R}_2[x]/h(x)$. As such, they are contained in the (shortened) cyclic code $\hat{\mathcal{C}}$. Consequently, the $(2n-1)$ bit vectors $v_1 L$ and u_1 are in the same coset of $\hat{\mathcal{C}}$, and so is $v_2 L$ and u_2.

Therefore, the multiplication of $u_1(x)$ and $u_2(x)$ results in a polynomial $w(x)$ whose corresponding vector in $\mathbb{G}_2^{2n-1}$ is in the same coset as the vector associated with the product $v_1 L(x) \cdot v_2 L(x) \in \mathbb{F}_2[x]/P(x)$. Since $w(x)$ may be of degree greater than $n-1$, it needs to be replaced by one of the members in its coset of the proper degree. Formally, define $u_1(x)u_2(x) = w(x) = (w_0, w_1, \ldots, w_{n-1}, w_n, \ldots w_{2n-2})$. Then, if $Q(x)$ is fixed, a mask refresh is required, as described in multiplication and modular reductions algorithms 3 and 4[2]. Otherwise, the security is compromised, as illustrated in Sub-section 4.1. That is, $w(x)$ is replaced by $u_3(x)$ as follows:

$$\begin{aligned} u_3(x) &=_{(i)} \quad w(x) \mod h(x) + r(x)P(x) \\ &= \quad w(x) - a(x)Q(x)P(x) + r(x)P(x) \\ &=_{(ii)} \quad w(x) - b(x)P(x), \end{aligned} \tag{10}$$

where $r(x)P(x)$ represents a refresh within the modulo $h(x)$ computation (i), and $b(x) = a(x)Q(x) + r(x)$ is a random polynomial (ii). Since $Q(x)$ is a polynomial of degree $n-m$ and r is a vector of length $n-m$, a random $Q(x)$ and a random $r(x)$ share the same amount of randomness. In other words, one can use a random polynomial Q of degree $n-m$ and a fixed $r(x)$, say $r(x) = 0$. ***This in turn obviates the need for mask refresh.*** The polynomial $h(x) = Q(x)P(X)$ becomes random and $w(x)$ is replaced by $u_3(x) = w(x) \mod h(x)$.

When $Q(x) = x^{n-m} + \sum_{i=0}^{n-m-1} q_i x^i$ is picked at random, $u_3(x) \in \mathbb{R}_2[x]/h(x)$ is obtained from $w(x)$ by subtracting a random codeword $\hat{c}(x) = b(x)P(x)$ that must have a specific form in order to nullify the first $n-1$ most significant bits of w (Fig. 1). In other words, $\hat{c}$ must be a codeword of the form

$$\hat{c} = (*, *, \cdots *, w_n, \ldots w_{2n-2}).$$

There are 2^{n-m} codewords of this form in $\hat{\mathcal{C}}$. The choice of Q determines the exact codeword. Thus, a systematic encoding of the $(2n-1-m)$-bit information word

$$z = (\underbrace{q_0, q_1, \ldots q_{n-1-m}}_{n-m \text{ bits}}, \underbrace{w_n, \ldots w_{2n-2}}_{n-1 \text{ bits}}),$$

eliminates the need to compute the h from the random Q and then calculate $w(x) \mod h(x)$. Instead, we pick a random codeword of the desired form. In matrix notation this is equivalent to

$$u_3 = (w_0, w_1, \ldots, w_{n-1}) + z \cdot A \tag{11}$$

where A consists of the first left-hand n columns of the generator matrix $G^s_{(2n-m-1)\times(2n-1)}$ of $\hat{\mathcal{C}}$. That is,

$$\begin{aligned} G^s_{(2n-m-1)\times(2n-1)} &= \left(A_{(2n-m-1)\times n} \;\middle|\; \begin{array}{c} 0_{(n-m)\times(n-1)} \\ \hline I_{(n-1)\times(n-1)} \end{array} \right) \\ &= \left(B_{(2n-m-1)\times m} \;\middle|\; \begin{array}{c|c} I_{(n-m)\times(n-m)} & 0_{(n-m)\times(n-1)} \\ \hline 0_{(n-1)\times(n-m)} & I_{(n-1)\times(n-1)} \end{array} \right) \end{aligned} \tag{12}$$

From an implementation cost perspective, the formulation of the possibility to randomize Q (and its equivalence) as illustrated in Eq. 10, is highly advantageous: *instead of computing an (expensive) modular reduction that requires mask-refresh at every step*[3] *one can simply*

[2]we elaborate on this point and give examples in the simulation section below
[3]as discussed in Appendix A

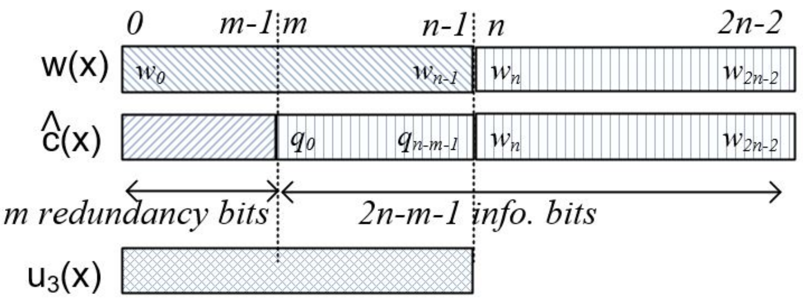

Figure 1: The structure of $w(x)\hat{c}(x)$ and the resulting $u_3(x)$

plug the randomness (i.e., the random coefficients of $q(x)$) into the variable z (as formulated in Eq. 11).

The following example illustrates the MUL and ADD operations over $\mathbb{R}_2[x]/h(x)$:

Example 2. Let $m = 4, n = 7$ and $P_0(x) = x^4 + x + 1, P(x) = x^4 + x^3 + 1$. Let $v_1 = \alpha^5 = (0110)$ and $v_2 = \alpha^{11} = (0111) \in \mathbb{F}_2[x]/P_0(x)$, then (direct computation):

$$v_3 = v_1 \cdot v_2 = \alpha = (0100).$$

In the mask domain, the MUL operation works as follows: Let $L = L_1$ from Ex. 1. Let $r_1 = (001), r_2 = (101)$ and $q = (110)$. Then, from Eq. 2 we have, $u = (v, r)M$ where

$$M = \left(\begin{array}{cc} L_{4\times4} & \mathbf{0}_{4\times3} \\ \hline B_{3\times4} & I_{3\times3} \end{array}\right) = \left(\begin{array}{cccc|c} 1 & 0 & 0 & 0 & \\ 0 & 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 1 & \\ 1 & 0 & 1 & 0 & \\ \hline 1 & 0 & 0 & 1 & \\ 1 & 1 & 0 & 1 & I \\ 1 & 1 & 1 & 1 & \end{array}\right), A = \left(\begin{array}{cccc|c} \multicolumn{4}{c|}{B_{3\times4}} & I_{3\times3} \\ \hline 1 & 1 & 1 & 0 & \\ 0 & 1 & 1 & 1 & \\ 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 & \\ 1 & 0 & 1 & 1 & \\ 1 & 1 & 0 & 0 & \end{array}\right)$$

This implies that,

$$\begin{aligned} u_1 &= (v_1, r_1) \cdot M = (0010\ 001) = x^2 + x^6 \\ u_2 &= (v_2, r_2) \cdot M = (0001\ 101) = x^3 + x^4 + x^6 \\ w &= u_1 \cdot u_2 = (0000\ 011\ \mathbf{011101}) \\ z &= (110\ \mathbf{011101}) \\ u_3 &= (0000\ 011) + z \cdot A = (0000101) = x^4 + x^6 \end{aligned} \tag{13}$$

Consequently,

$$v_3 = u_3 \cdot H^T \cdot L^{-1} = (0110)L^{-1} = (0100)$$

as expected from the direct computation in $\mathbb{F}_2[x]/P_0(x)$.

3.6 The affine transformation in the mask domain

Denote by $\mathcal{A}_{\mathbb{F}}(v) = vT + t$ the original affine transformation of the m-bit binary vector v. Here, $v, t \in \mathbb{F}_2^m$, and T is a binary $m \times m$ matrix over $\mathbb{F}_2$. An affine transformation of the $(m+d)$-bit binary vector $u \in \mathbb{R}_{2^n}$ in the mask domain, $\mathcal{A}_{\mathbb{R}}(u) = uW + w$ where $u, w \in \mathbb{F}_2^{m+d}$

and W is a binary $(m+d)\times(m+d)$ matrix over $\mathbb{F}_2$ that preserves homomorphism is defined as follows:

Let G and H be the systematic generator and check matrices of dimension $n-m$ and m of a shortened cyclic code $\mathcal{C}$ of length n defined by the generator polynomial $P(x)$, respectively. That is,

$$\begin{aligned} G &= \begin{pmatrix} B & I \end{pmatrix}, \\ H &= \begin{pmatrix} I & B^T \end{pmatrix}, \\ M &= \begin{pmatrix} L & 0 \\ B & I \end{pmatrix}, \end{aligned}$$

and $u=(v,r)M$ and $v=uH^TL^{-1}$. Consider a transformation matrix W of the form

$$W=\begin{pmatrix} W_{1,1} & 0 \\ W_{2,1} & W_{2,2} \end{pmatrix}.$$

We require that,

$$\begin{aligned} vT+t &= (uW+w)H^TL^{-1} \\ &= (v,r)MWH^TL^{-1}+wH^TL^{-1} \\ &= (v,r)\begin{pmatrix} LW_{1,1} \\ BW_{1,1}+W_{2,1}+W_{2,2}B \end{pmatrix} L^{-1}+wH^TL^{-1} \end{aligned}$$

Equivalently, w should be a member of the coset whose syndrome is $s=wH=tL$. $W_{1,1}=L^{-1}TL$ and the matrices $W_{2,1}$ and $W_{2,2}$ should satisfy

$$BW_{1,1}+W_{2,1}+W_{2,2}B=0.$$

3.7 Other security metrics

The effectiveness of masking against a probing attack or a fault injection is evaluated by the following security orders [BCC+14, KP20] :

- Bit-level security order d_b against a probing attack on single or/and several bits that can be probed simultaneously. In general, the security order of a masking scheme is defined as the largest number d_b above which it is possible to exploit secret information from the protected implementation, e.g., by probing wires or register-stored values of the protected circuit. In Boolean masking, $d_b=d_{\mathcal{C}}^{\perp}-1$ where $d_{\mathcal{C}}^{\perp}$ is the minimum distance of $\mathcal{C}^{\perp}$ (the dual code of $\mathcal{C}$). In other words, it equals the maximal number of independent columns in G.

  For example, the bit-level security order in Ex. 1 is $d_b=1$. Namely, two probes located, for example, on u_2 and u_6 can reveal information on the third bit of vL regardless of how $r(x)$ has been encoded.

  In fact, $n-m$ probes placed on the $n-m$ most significant bits of u are sufficient to recover the exact codeword that masks the m bits of vL. However, the randomness in the selection of L ensures that this does not lead to a significant security breach.

- Bit-level error detection order d_f against random (benign) and malicious fault injections. d_f is the maximal number of bit flips that can always be detected by every codeword in $\mathcal{C}$. In this paper, we refer to the polynomial $P(x)$ as a generator polynomial of a (shortened) cyclic code $\mathcal{C}$ of length n and dimension $n-m$. Moreover, if n is less or equal to the order of the roots of $P(x)$ the code is of distance 3 and hence $d_f=2$, otherwise, it is of distance 2 and $d_f=1$.

- The arbitrary-error detection rate, d_a, is the probability that the worst injected error pattern will pass unnoticed. The value of d_a indicates how robust the code is against a sophisticated attacker that can apply any number of bit-lips it chooses. Since this paper deals with linear masking, there are error patterns that can never be detected. That is, $d_a = 1$.

4 Simulations

In this section we provide several simulation-based examples to (1) improve the pseudo-algorithm provided in the multiplication (MULT) and the modular reduction (MOD) proposed in [BBA+22] and the need for atomic-operations refresh, (2) illustrate the threats encapsulated in the brute-force parameter selection or in so-called non-significant leakage as discussed in [BBA+22]. Finally, (3) we illustrate the strengths embedded within CLM.

4.1 Leakage from Modular Multiplication in the masked domain

We simulated leakage traces with a Hamming weight (HW) leakage model and additive normal Gaussian noise with a zero mean and standard deviation matching a SNR of 10^{-1}. We started by simulating a multiplication with a specific (fixed) P and the chosen Q polynomials, as was devised in [BBA+22]. Algorithms 1, 2 detail the multiplication. Here we used $m = 8$ and $d = n - m = 8$. We further utilized a *generator matrix* of the systematic code, i.e.,

$$G^{sys}_{(n-m)\times n} = (B_{(n-m)\times m}|I_{(n-m)\times(n-m)})$$

and implemented a single (16-bit) multiplier in software. That is, for purposes of illustration and simplicity's sake no algorithmic noise was incorporated in this simulation. In each and every *step* in the sequence of the computation (see algorithms) we took the computed internal value and added an independent and additive noise to produce a *pseudo* leakage trace.

For the TVLA experiment, we asserted *random* multiplications, i.e., the multiplier and multiplicand were quarried with $v_1 \leftarrow \$$; $v_2 \leftarrow \$$, and *fixed*-value multiplications,

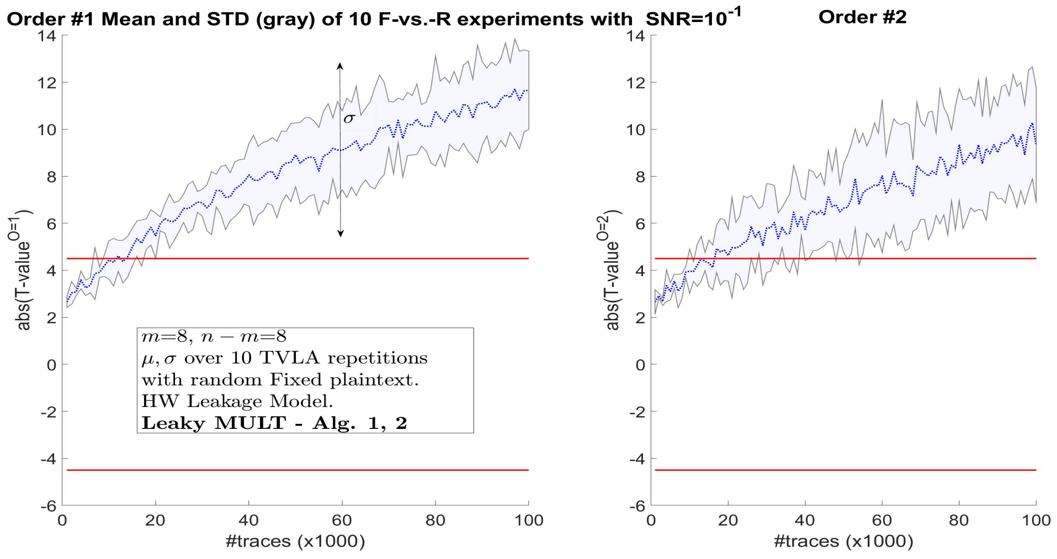

Figure 2: Original Multiplier: Fixed vs. random t-test based TVLA of the first two statistical orders over a (masked) leaky MULT calculation

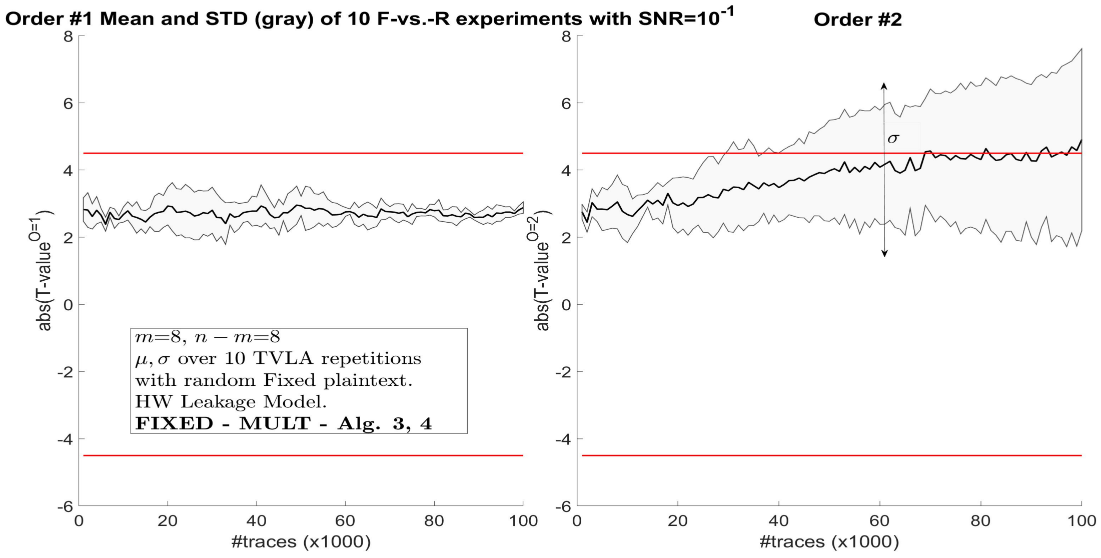

Figure 3: Leakage free Multiplier: fixed vs. random t-test based TVLA of the first two statistical orders over a (masked) MULT calculation

i.e., the multiplier and multiplicand were quarried with $v_1 \leftarrow Val_{v_1}$; $v_2 \leftarrow Val_{v_2}$, for 10^5 measurements (computations), which was enough for the given low SNR value. The *random* and *fixed* quarries were randomly interleaved (mixed). This process of implementing a fixed versus a random TVLA t-test was repeated 10 times (for demonstration), i.e., with 10 different, randomly selected *fixed* values (Val_{v_1}, Val_{v_2}). Note that \$ represents a uniform at random draw.

The software implementation of these algorithms when utilizing the Shift-and-Add paradigm as it appears in the original [BBA+22] is somewhat problematic. The two places where we found leaks in the algorithms are marked in red: (1) the lack of refresh between all atomic operations that turned out to be crucial for minimizing the leakage in Subsection 3.5 and (2) timing side-channels caused by the lack of an *else* condition (non-constant time implementation). In Fig. 2 we illustrate the mean (μ, dashed blue curve) and the standard deviation (σ, light blue halo around the mean) of the max t-test values plotted against the number of traces in the simulation. Without faults there should be no leakage above the test threshold, which was falsified as indicated in the 1^{st} statistical moment as d=1.

These issues must be handled before evaluating the consolidated linear masking scheme. The naive corrections are listed in Algorithms 3 and 4, and highlighted in green . We implemented a constant time code and refresh in each loop iteration. As illustrated in Fig. 3 the mean (μ) and standard deviation (σ) of the max t-test values did not leak in the 1^{st}-order statistics, as expected. From this point on, we conducted all the simulations with these two algorithms.

4.2 Sensitivity to heuristic based selection of (P, L, Q)-tuples

In general, for a given (m, n) pair [BBA+22] provides (P, L, Q)-tuples. That is, RAMBAM associates with each irreducible polynomial $P(x)$ of degree $m = 8$ an L matrix and (heuristically chosen) $Q(x)$ polynomial of degree $n - m$. In this section we discuss the implications of working with pre-determined (P, L, Q)-tuples.

We simulated leakage traces with a Hamming weight (HW) leakage model and additive normal Gaussian noise with zero mean and a standard deviation matching a SNR of 10^{-1},

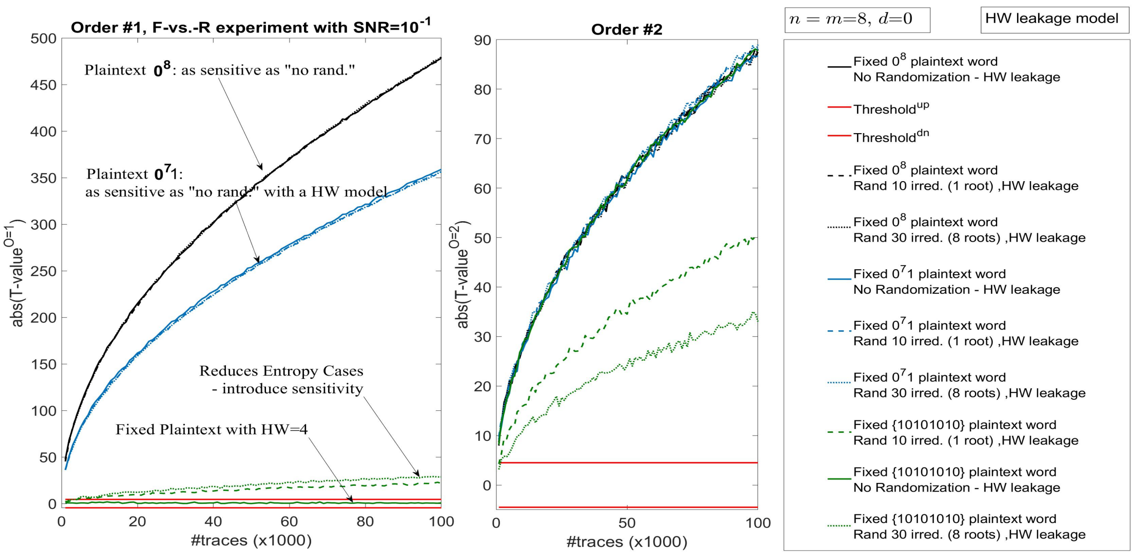

Figure 4: Fixed vs. random t-test based TVLA of the first two statistical orders, with chosen fixed plaintext of 0, 1 and HW=$m/2$, in an $n - m = 0$ reduced-CLM instances which does not fully utilize the entire randomization space provided by isomorphism and L randomization (such as [BBA+22]), illustrating possible information leakage.

of some internal value (after mapping). After the simulation we computed a *fixed* versus *random* TVLA t-test showing:

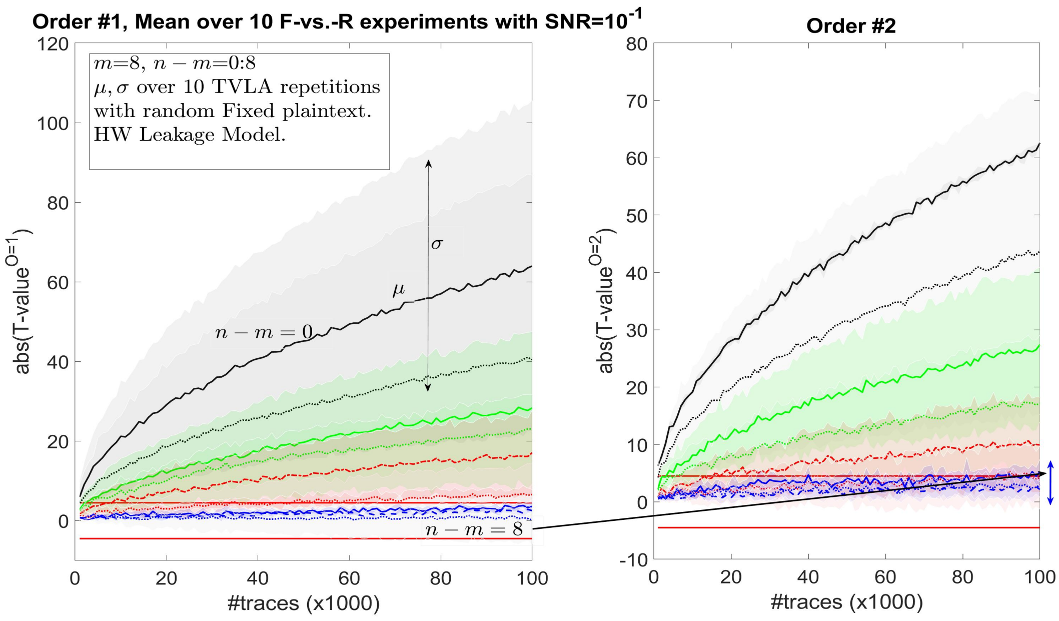

Figure 5: 10 Fixed vs. random t-test based TVLA of the first two statistical orders, with *random* fixed plaintext for all $n - m = [0 : 8]$ cases with a fixed L, i.e., RAMBAM or highly reduced-CLM instance.

(1) in the case of n=m (no redundancy) we found failures, i.e., indications that there were information leakages. For purposes of illustration, the *fixed* input to the *fixed* SBOX input was set to 0 (which would manifest the leakiest scenario) given the reduced entropy due to the zero associated isomorphic-mapping, i.e., such plaintexts are known to leak equally to a non-protected scenario, or a fixed plaintext of 1 as shown in Fig. 4 by a blue curve, slightly improving entropy, or with a plaintext of a HW of $n/2$ (or $m/2$ in this case). The black curves for both the first and second order t-tests were associated with the fixed 0: solid black is a no- randomization simulation and dashed black represents a simulation with $|\mathcal{L}| = 10$ different L matrices (i.e., a reduced version of CLM), and the dense dashed black curve represents a simulation with $|\mathcal{L}| = 30$ (i.e., also a reduced version of CLM). The set of blue curves is identical (solid, dashed and dense dashed) but evaluated with a fixed vector of the value 1. Clearly, all the reduced versions including RAMBAM are bound to leak significantly in this reduced-entropy scenario, not only n=m and there is no randomization, but we did not exhaust the full power of the entropy from randomization over L possible representations as proposed by CLM (i.e., here $|\mathcal{L}|$<240).

(2) In this same scenario of n=m (no redundancy) we next show that with HW of four (green curves on Fig. 4), the solid curve does not leak in the first order as expected since the randomized set mean is the same. However, because of the reduced entropy due to L, some bias appears (dashed and dense-dashed green curves). This illustrates issues that are more apparent with RAMBAM fixed tuples. Clearly if CLM utilizes the entire entropy provided by isomorphic L these cases can be avoided, *i.e., an almost first order SCA security d =2 can be achieved without randomness in the representation of variables, with (almost) no leakage in the first statistical moment*[4].

(3) Next, we simulated a scenario with $n - m = [0 : 8]$ (denoting the range between 0 and 8) and no isomorphic L randomization (i.e., RAMBAM). Fig. 5 shows the mean (μ) TVLA curve and its variance (σ) over 10 TVLA tests, where in a departure from

[4]This property is provided as an example below in Fig. 7

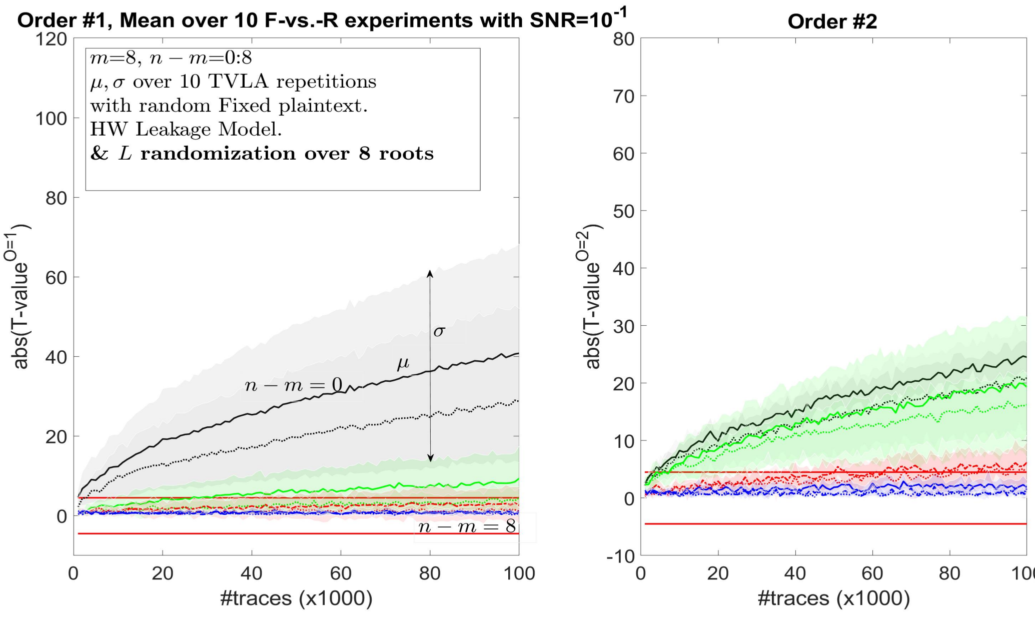

Figure 6: 10 Fixed vs. random t-test based TVLA of the first two statistical orders, with *random* fixed plaintext for all $n - m = [0 : 8]$ cases with slight isomorphic L randomization (reduced-CLM instance), with 8 possible roots.

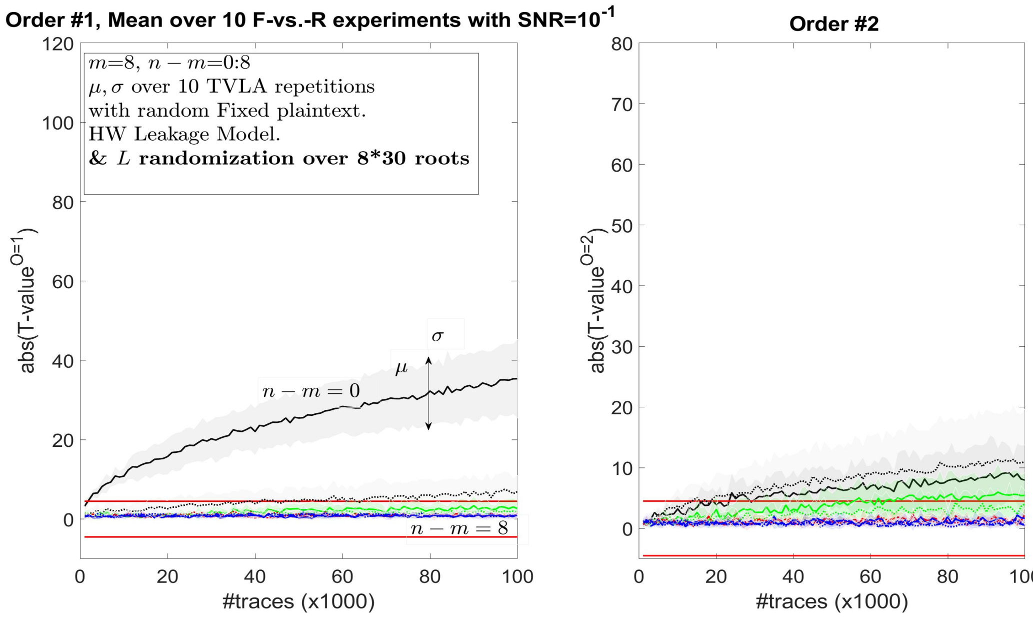

Figure 7: 10 Fixed vs. random t-test based TVLA of the first two statistical orders, with *random* fixed plaintext for all $n - m = [0 : 8]$ cases with no isomorphic L randomization full-entropy CLM instance, with $8 \cdot 30$ possible roots.

the previous two scenarios, the *fixed* plaintext was chosen at random: {black, green, red, blue} curves represent $n - m$={1, 2, 4, 8}, respectively. Clearly the highlighted variance regions (halo) show that information does not only leak in the extreme scenarios of the tailored-*fixed* values above. The figure also shows that as $n - m = [0 : 8]$ increases, along with the expected security increase, there are lower mean (μ) curves and at the same time smaller variances (σ). Note this latter point is merely an artifact due to the small support set of 10 TVLA experiments.

4.3 Enhancing performance by releasing the reins on (P, L, Q)-tuples

In this section we demonstrate the strength of the CLM scheme that make it possible to release the reins on (P, L, Q)-tuples and hence increase the masking order.

We simulated a scenario with $n - m = [0 : 8]$ and random L. Fig. 6 depicts the TVLA results where $|\mathcal{L}| = \forall$. Specifically, $P(X)$ was fixed and the isomorphism was defined by picking one of its eight roots at random. The figure shows the mean (μ) curve and its variance (σ) over 10 TVLA tests, with random *fixed* plaintext. {black, green, red, blue} curves represent $n - m$={1, 2, 4, 8}, respectively. The figure also depicts the same trend where when $n - m = [0 : 8]$ increases, as expected, security increases, and there are lower mean (μ) curves. However, note the significant reduction in information leakage as a result of this randomization.

For illustration we extended this experiment.In Fig. 7 the representation was randomized with all $8 \cdot 30$-roots associated with cosets (i.e., 240 isomorphic Ls and associated Ps). The figure again shows the mean (μ) curve and its variance (σ) over 10 TVLA tests, with random *fixed* plaintext. Evidently the trends repeat with the increase in $n - m = [0 : 8]$. However, even before $n - m$=8 information is hard to distinguish (as is apparent in both the mean and variance of the curves) in the first statistical moment of the leakage. *This is clearly due to the entropy provided by the maximum randomization of isomorphic* L*s as compared to standard masking with* conventional *security-order* d*=0 in this case. This* *security level can only be achieved with (conventional) masking, with say* $n = 16, m = 8$.

5 Conclusions

Current masking schemes often incur significant overhead.This paper presented a comprehensive generalization of various masking schemes and described an elegant mathematical representation for masking dubbed Consolidated Linear Masking (CLM), where masking schemes are formalized by their encoding. Within this unified framework, we established a theoretical foundation linking randomized isomorphic (code) representations and entropy provided by the redundancy of the representation, which supports a revised notion of the masking order. The proposed scheme and formalization thus paves the way for significant enhancements. Specifically, we showed that it is possible to fully randomize the (P, L, Q)-tuples which define the implementation, thus considerably increasing the entropy provided by the scheme. For example, a 1^{st}-order secure design can be (almost) achieved without increasing the size of the variable representation, or an almost 3^{rd}-order secure design only needs to double the representation size. We provided formalizations showing that CLM enables randomized isomorphic field selection for improved security, a flexible choice of the randomization polynomial, and highly efficient embedded mask-refreshing via the randomized isomorphic representation. This latter property significantly reduces the randomness requirements and saves abundant modular reductions within the scheme, while providing much improved performance. Finally, we also showed that the wider range of isomorphic randomized mappings depicted here (by m) significantly increases the available randomization.

References

[BBA+22] Yaacov Belenky, Vadim Bugaenko, Leonid Azriel, Hennadii Chernyshchyk, Ira Dushar, Oleg Karavaev, Oleh Maksimenko, Yulia Ruda, Valery Teper, and Yury Kreimer. Redundancy aes masking basis for attack mitigation (rambam). *IACR Transactions on Cryptographic Hardware and Embedded Systems*, pages 69–91, 2022.

[BCC+14] Julien Bringer, Claude Carlet, Hervé Chabanne, Sylvain Guilley, and Houssem Maghrebi. Orthogonal direct sum masking. In David Naccache and Damien Sauveron, editors, *Information Security Theory and Practice. Securing the Internet of Things*, pages 40–56, Berlin, Heidelberg, 2014. Springer Berlin Heidelberg.

[BFG15] Josep Balasch, Sebastian Faust, and Benedikt Gierlichs. Inner product masking revisited. In *Advances in Cryptology–EUROCRYPT 2015: 34th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Sofia, Bulgaria, April 26-30, 2015, Proceedings, Part I 34*, pages 486–510. Springer, 2015.

[BFG+17] Josep Balasch, Sebastian Faust, Benedikt Gierlichs, Clara Paglialonga, and François-Xavier Standaert. Consolidating inner product masking. In *Advances in Cryptology–ASIACRYPT 2017: 23rd International Conference on the Theory and Applications of Cryptology and Information Security, Hong Kong, China, December 3-7, 2017, Proceedings, Part I 23*, pages 724–754. Springer, 2017.

[CGLS20] Gaëtan Cassiers, Benjamin Grégoire, Itamar Levi, and François-Xavier Standaert. Hardware private circuits: From trivial composition to full verification. *IEEE Transactions on Computers*, 2020.

[DCEM18] Thomas De Cnudde, Maik Ender, and Amir Moradi. Hardware masking, revisited. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, pages 123–148, 2018.

[GIB18] Hannes Groß, Rinat Iusupov, and Roderick Bloem. Generic low-latency masking in hardware. *IACR transactions on cryptographic hardware and embedded systems*, pages 1–21, 2018.

[GMK18] Hannes Groß, Stefan Mangard, and Thomas Korak. *Domain-Oriented Masking–*. PhD thesis, Graz University of Technology, Austria, 2018.

[ISW03] Yuval Ishai, Amit Sahai, and David Wagner. Private circuits: Securing hardware against probing attacks. In *Annual International Cryptology Conference*, pages 463–481. Springer, 2003.

[KP20] Osnat Keren and Ilia Polian. IPM-RED : combining higher-order masking with robust error detection. *Journal of cryptographic engineering*, 11(2):147–160, 2020.

[MMSS19] Thorben Moos, Amir Moradi, Tobias Schneider, and François-Xavier Standaert. Glitch-resistant masking revisited. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, pages 256–292, 2019.

[RBN$^+$15] Oscar Reparaz, Begül Bilgin, Svetla Nikova, Benedikt Gierlichs, and Ingrid Verbauwhede. Consolidating masking schemes. In *Advances in Cryptology–CRYPTO 2015: 35th Annual Cryptology Conference, Santa Barbara, CA, USA, August 16-20, 2015, Proceedings, Part I 35*, pages 764–783. Springer, 2015.

[SL23] Dor Salomon and Itamar Levi. Masksimd-lib: on the performance gap of a generic c optimized assembly and wide vector extensions for masked software with an ascon-p test case. *Journal of Cryptographic Engineering*, 13(3):325–342, 2023.

[SM21] Aein Rezaei Shahmirzadi and Amir Moradi. Re-consolidating first-order masking schemes: Nullifying fresh randomness. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, pages 305–342, 2021.

A Software Multiplication and Modular reduction

In these algorithm listing we show the software algorithms listed in [BBA$^+$22] constructions, where the multiplication and modulus algorithms are generally well known based on Shift-and-Add paradigm, and were adapted to RAMBAM (Algorithms 1 and 2).

We denote within the multiplication and modular reduction algorithms stages where they have some conceptual weaknesses appearing in red color in the algorithm. Either owing to a lack of refresh (as discussed above, as needed between all atomic operations - Subsection 3.5) or timing side-channels owing to trivial lack of *else* conditions (non constant time implementation). We correct these leakage sources in order to evaluate CLM properties with leakage-free multiplication, as listed in Algorithms 3 and 4. Note that $[\cdot]$ represent a modular reduction (following a refresh), and $rand \leftarrow \$$ represent a random drawing to any (random) register or variable. Note, that changes appear in green color text.

Algorithm 1 Multiplication

```
Function $Mul_{Z,d}(a_{in}, b_{in})$:
    Input  : $a_{in}, b_{in}$ – two (8 + d)-bit values, each one representing a byte
    Output : $c_{out}$ – a (8 + d)-bit value representing $a_{\text{in}} \cdot b_{\text{in}} \mod P$
    $c_{\text{out}} \leftarrow 0$
    $deg \leftarrow b_{\text{in}}$
    for $i \leftarrow 0$ to $7 + d$ do
        if $((a_{in} \gg i)\&1) = 1$ then
            $c_{\text{out}} \leftarrow c_{\text{out}} + \text{deg}$ ⚡ ⚡ ;        // Here values from $a_{in}, b_{in}$ recombine
        end
        ⚡ ⚡ ;                                                  // No 'else' - trivial leakage
        $deg \leftarrow \text{ModularShiftLeft}_{Z,d}(deg)$
    end
    return $c_{out}$
```

Algorithm 2 ModularShiftLeft

```
Function $ModularShiftLeft_{Z,d}(x_{in})$:
    Input  : $x_{in}$ – a (8 + d)-bit value
    Output : $x_{out}$ – a (8 + d)-bit value
    $x_{\text{out}} \leftarrow x_{\text{in}} \ll 1$
    if $((x_{out} \gg (8 + d))\&1) = 1$ then
        $x_{\text{out}} \leftarrow x_{\text{out}} + Z$ ⚡ ⚡ ;          // Here bits from $x_{in}$ may recombine
    end
    ⚡ ⚡ ;                                                  // No 'else' - trivial leakage
    return $x_{out}$
```

Corrected pseudo-algo. Note that this software implementation is clearly much more expensive than what CLM proposes. With CLM and full- randomization over Q all internal refreshed (and modulus operations) can be spared, as discussed in sub-section 3.5.

Algorithm 3 Modified Multiplication

```
Function $Mul_{Z,d}(a_{in}, b_{in})$:
    Input  : a_in, b_in – two (8 + d)-bit values, each one representing a byte
    Output : c_out – a (8 + d)-bit value representing $a_{in} \cdot b_{in} \mod P$
    $c_{out} \leftarrow 0 \;\; deg \leftarrow b_{in}$
    for $i \leftarrow 0$ to $7 + d$ do
        if $((a_{in} \gg i)\&1) = 1$ then
            $c_{out} \leftarrow [c_{out} + r \cdot P] + deg$
        end
        else
            **$rand \leftarrow \$$ ;               // Here $rand$ may represent any random register
        end
        $deg \leftarrow ModularShiftLeft_{Z,d}(deg)$
    end
    return $c_{out}$
```

** - Otherwise trivial timing attack (time dependent latency).

Algorithm 4 Modified ModularShiftLeft

```
Function $ModularShiftLeft_{Z,d}(x_{in})$:
    Input  : $x_{in}$ – a (8 + d)-bit value
    Output : $x_{out}$ – a (8 + d)-bit value
    $x_{\mathrm{out}} \leftarrow x_{\mathrm{in}} \ll 1$
    if $((x_{out} \gg (8+d))\&1) = 1$ then
        $x_{\mathrm{out}} \leftarrow [x_{\mathrm{out}} + r \cdot P] + Z$
    end
    else
        $**rand \leftarrow \$$
    end
    return $x_{out}$
```