

On Constructing AEs with Leakage and Faults

Thomas Peters¹, Sayandeep Saha², Yaobin Shen³ and François-Xavier Standaert¹

¹ Crypto Group, ICTEAM Institute, UCLouvain, Louvain-la-Neuve, Belgium

thomas.peters@uclouvain.be, fstandae@uclouvain.be

² Indian Institute of Technology Bombay, Mumbai, India

sayandeepsaha@cse.iitb.ac.in

³ School of Informatics, Xiamen University, Xiamen, China

yaobin.shen@xmu.edu.cn

Abstract. Physical attacks are one of the potent threats to modern cryptography. Symmetric-key primitives are well-explored with respect to passive, active, and more recently combined (i.e., simultaneously leaking and faulting) adversaries. For symmetric key operating modes, there are many proposals for encryption (ENC), Message Authentication Codes (MAC), and Authenticated Encryption (AE) covering passive attacks. However, research on active or combined adversaries is more succinct and, to the best of our knowledge, boils down to combined secure MAC due to Berti et al. (ToSC 2023, issue 1), combined secure ENC due to Dobraunig et al. (CCS’22), and active secure AE named MEM due to Saha et al. (ToSC 2022, issue 4). Our first contribution is a security model to formally reason about the combined security of AE. The model extends the fault-then-leak security of MAC introduced by Berti et al. to AE and can also be seen as a natural extension of state-of-the-art leakage resistant notions in their so-called faulty matrix framework. Armed with this model, we show as a second contribution that all the leakage-resilient, including leakage-resistant, AE constructions and the fault-resilient MEM construction are vulnerable to a single fault injection within the composition of their building blocks. These decoupling attacks remain successful even for combined-secure building blocks. As a result, we finally present **LEAF**, the first AE construction secure against fault-then-leak adversaries with a single fault injection. In encryption, **LEAF** makes only one call to a MAC and two calls to an ENC that are combined secure.

Keywords: Faults · Side-Channels · Combined Security · Authenticated Encryption

1 Introduction

Over the last two decades, the security of cryptographic implementations against physical attacks has been a topic of intense research. Such attacks mainly come in two variants: 1) when the adversary passively observes the computation, known as Side-Channel Attacks (SCA) [KJJ99, BCO04, CRR03, MDP20]; and 2) when the adversary actively perturbs the computation by injecting some external stimuli, called Fault Attacks (FA) [BS97, TMA11, FJLT13, DEK⁺18]. Recently, it has been shown that the attack surface gets significantly enhanced if the adversary can inject faults and then measure leakage from the same execution [RLK11, FGV11, SBJ⁺21, APZ21] – giving rise to the so-called *combined* adversarial model. Such combined attacks are able to recover secrets from implementations having independent SCA and FA countermeasures, and therefore imply the need for new combined countermeasures offering a joint protection.

Countermeasures for SCA are well-studied and come in two flavors: 1) implementation-level countermeasures, such as masking and shuffling [CJRR99, ISW03, HOM06, VCMKS12];

and 2) mode-level countermeasures for encryption (ENC) [PSV15], Message Authentication Code (MAC) [MOSW15,BGPS21], and Authenticated Encryption (AE) [BMOS17,GPPS19,BGP⁺20,DEM⁺]. Implementation-level countermeasures for FAs are also well-explored and utilize redundancy in various forms to detect/correct the faults [WKKG04,IPSW06,TBM14,SJR⁺20]. For a consolidated study on the state-of-the-art implementation-level FA protection, please refer to Patranabis and Mukhopadhyay [PM18]. However, only a handful of works have addressed the active (fault) security of symmetric-key modes [FG20,SKP22]. In particular, the work by Fischlin and Günther [FG20] considered fault security of AEs for the first time. The security claim was later refuted by Saha et al. [SKP22] with new integrity attacks. Improved constructions preventing the attacks were also proposed [SKP22].

Countermeasures against combined attacks are relatively less studied. At the implementation level, some formal as well as practical attempts have been made to address this problem, and provably secure, composable gadgets have been proposed very recently [RDMB⁺18,DN20,FRBSG22,FGM⁺23,BEF⁺23]. At the mode level, the work by Dobraunig et al. [DMP22] presents a sponge-based stream-cipher with combined security. Independently, Berti et al. [BGP⁺22] present a combined secure MAC based on the unpredictability of Tweakable Block Ciphers (TBCs) under a combined adversary. To the best of our knowledge, the combined security of AEs has not been explored.

The goal of this paper is to investigate combined attack-resilient AEs. In particular, we are interested in the question whether integrating a MAC that is secure against a combined adversary with a (combined-secure) ENC scheme can give an AE that is secure against a combined adversary. The combined adversary considered here is defined according to the leakage and fault paradigm by Berti et al. [BGP⁺22], i.e., a *fault-then-leak* adversary which can induce a fault in the computation and then observe its impact even with leakage. The fault models (differential and stuck-at) are also similar to the ones proposed by Berti et al. [BGP⁺22]. As we show in this paper, constructing such a combined-secure AE is non-trivial even for an adversary capable of introducing only a single fault per query. A single-faulting adversary is one which can inject at most one fault per query where faults are allowed, i.e., in each call to the encryption oracle and decryption oracle for the integrity, and in each call to the encryption oracle for the privacy (except for the challenge encryption query). The composition between the building blocks (i.e., ENC, MAC) and the articulation of the inputs to different blocks play a crucial role in ensuring security. We explain this by presenting (single fault-based) attacks on state-of-the-art leakage-resilient and fault-resilient constructions. In particular, leakage-resilient constructions are found to have multiple issues regarding confidentiality and integrity, even if a single fault per query is considered. We also find that the *decoupling attack*, an integrity attack proposed by Saha et al. [SKP22], affects all generic constructions (i.e., ENC-the-MAC, and N2, A5, A6), which can achieve leakage-resilience [BMOS17]. The decoupling attack was already shown to work on TEDT [BGP⁺20] (and similar constructions) by Saha et al. [SKP22], so that we now cover the entire spectrum of leakage resilient/resistant AE¹.

We then additionally describe new confidentiality attacks related to the nonce on these generic schemes, showing that nonces are quite sensitive to faults. Regarding fault-resilient constructions, the only existing secure one is MEM, which is found vulnerable to the stuck-at faults, with the single-faulting adversary². Notably, all the attacks that we describe (for both leakage-resilient and fault-resilient schemes) consider fault injection at the inputs and outputs of the ENC and MAC blocks. Moreover, a single fault in a single

¹“Leakage resilience” refers to a security model where (for privacy) all *except* the encryption challenge query can leak. In “Leakage resistance”(for privacy), even the encryption challenge query may leak. “Fault-resilience” refers to a security model where (for privacy) all *except* the encryption challenge query can be faulted, and decryption queries cannot be faulted. Both encryption queries and decryption queries can, however, be leaked/faulted for integrity. For further details on these terms, please refer to Berti et al. [BGP⁺20,BGP⁺22].

²We note that this vulnerability does not affect the claimed security of MEM, as it was only proven secure against (single-faulting) differential faults.

encryption query is sufficient for all of our attacks, which is a nominal requirement while moving towards achieving security against faults. The attacks arising from composition issues also emphasize that existing leakage-resilient constructions are not easily extendable for combined security. For the fault-resilient construction **MEM**, we point out multiple issues which hinder its extension towards combined security. Extending **MEM** for combined security (by strengthening the constituent blocks and disregarding the stuck-at faults) requires the constituent MAC to preserve message confidentiality in the presence of leakage. This is a relatively strong and non-standard assumption. Since we generally deem more judicious to look at design choices based on well-explored assumptions, we conclude that our observations motivate the need of new constructions for the combined security problem, and we next provide a first attempt in this direction.

Observing and analyzing the issues with existing constructions, we present **LEAF** – the first AE construction secure against a combined fault-then-leak adversary capable of doing a single fault per query, given the constituent blocks have combined security. The security of the construction stems from the redundant encoding of the nonce (ref. Sec. 4.1) along with redundant computations. Furthermore, it relies on standard cryptographic assumptions as much as possible. We only need to enhance the CPA assumption slightly with respect to leakage, which is easy to ensure/falsify in practice. In particular, we need the AE to remain secure against leakage, fault, and combined adversaries, even if the adversary obtains two leakage traces corresponding to the constituent ENC (with exactly the same inputs) blocks from the challenge encryption query. With such traces carrying the same information, the adversary can essentially do some averaging and reduce the measurement noise. However, no significant advantage corresponding to Differential Power Analysis (DPA) should be gained (i.e., leakage is repeatedly observed for the same device state). We note that such an assumption over leakage is not new and has previously been used, for example, by Bellizia et al. [BBC⁺20], named as *SPA with averaging*. Usually, achieving SPA security is considered easier than achieving DPA security, and therefore, having security with such an assumption is expected to be less demanding in terms of resources.

In order to prove the security of **LEAF** we extend the notion of *leakage-resistance* proposed by Guo et al. [GPPS19] with faults in a natural way. It separates the notion of authenticity from the one of confidentiality with some differences in the adversarial capabilities. Our definitions are described according to the faulty matrix framework due to Bertoni et al. [BGP⁺22] which allows formally defining where and how faults can occur. These definitions are expressed in full generality for adversaries that can inject many faults at a time and of different types (differential or stuck-at), which we view as our first contribution. Then, we prove that **LEAF** is the first secure AE scheme against combined attacks when the adversary is restricted to inject a single fault per query³.

More precisely, for authenticity, we define a new ciphertext integrity game for AE with nonce misuse and fault-then-leak attacks in both encryption and decryption. The resulting definition called **CIML2F2** can be found in Sec. 3.1. To prove the **CIML2F2** security of **LEAF**, it is assumed that the constituent MAC is strongly unforgeable in the presence of leakage and faults in both authentication and verify (**SUF-FL2**) – a security notion for MACs proposed by Bertoni et al. [BGP⁺22]. For confidentiality, we propose the notion of **CCAmL2f1** (ref. Sec. 3.1), which stands for CCA security in the presence of leakage and faults, where the faults are limited to only non-challenge encryption queries. Furthermore, the nonce for the challenge encryption query cannot be reused for a non-challenge query. In order to prove the **CCAmL2f1** security of **LEAF**, we assume that the underlying encryption blocks (ENC) of the AE are CPA secure with leakage and faults with all the aforementioned restrictions imposed on the adversary (called **CPAmL2f1** security). We also need the **SUF-**

³Such a single-fault adversary can however completely alter a whole intermediate input such as a nonce or a block of message or ciphertext.

FL2 assumption on the MAC blocks. For the confidentiality of LEAF, we further assume some bounded leakage model (which, concretely, can be viewed as a SPA-with-averaging leakage assumption). For its integrity, the leakage can be unbounded till the exact values of the variables except for the long-term secrets. The connections between the blocks and the inputs to the blocks may leak unboundedly. The connections can also be faulted with differential and stuck-at faults, with one injection corrupting only one variable (i.e., an output of a block or input to a block) at a time.

The rest of the paper is organized as follows. In Sec. 2, we present the technical background and recall the matrix model proposed by Berti et al. [BGP⁺22] to represent faults. In Sec. 3, we formally define our new security notions for AE in the fault-then-leak model in Sec. 3.1, and we go through the existing AEs to show their insecurity in Sec. 3.2. We also pinpoint the main design challenges in this section. In Sec. 4, we address these challenges and present the LEAF construction. The proofs of integrity and confidentiality are articulated in Sec. 5. We conclude in Sec. 6, discussing some future challenges.

2 Preliminaries

For the sake of readability, we do not provide the numerous security definitions for Message Authentication Code (MAC), symmetric encryption scheme (ENC), and Authenticated Encryption (AE) with and without fault or leakage in this paper. Rather, we recall some of the stronger practical notions from which one can easily derive standard weaker notions as we will explain. But, first, let us give the notations used in this work.

2.1 Notations

Let $\{0, 1\}^n$ be the set of all n -bit strings and $\{0, 1\}^*$ the set of all finite strings, including the empty string ε . For a finite set, we denote by $x \xleftarrow{\$} \mathcal{X}$ the uniform sampling from $\mathcal{X}$ and assign the value to x . A key variable is always denoted with an uppercase letter K .

Adversary. By a $(q_1, \dots, q_\omega, t)$ -bounded adversary $\mathcal{A}$ we mean a probabilistic algorithm that has access to ω oracles, $O_1, \dots, O_\omega$, making at most q_i queries to its i -th oracle O_i , and performing computation bounded by running time t . We denote by $y \leftarrow \mathcal{A}^{O_1, \dots, O_\omega}(x)$ the event that the adversary $\mathcal{A}$ outputs y on input x after interaction with those oracles.

Leaking Algorithm. A leaking version of an algorithm **Algo** is denoted **LAlgo**. It runs both **Algo** and a *leakage function* L_{Algo} which captures the additional information given by an implementation of **Algo** during its execution⁴. **LAlgo** simply returns the outputs of both **Algo** and L_{Algo} which all take the same input.

2.2 Leakage-Resistant Authenticated Encryption [GPPS19]

The purpose of authenticated encryption is to provide both confidentiality and integrity in a single cryptographic scheme. First, we provide the general description of such a scheme.

Definition 1 (Nonce-Based Authenticated Encryption with Associated Data). A *nonce-based authenticated encryption scheme with associated data* is a tuple $\text{AE} = (\text{Gen}, \text{Enc}, \text{Dec})$ of algorithms such that, the key-generation algorithm **Gen** samples a key K uniformly at random from the key space $\mathcal{K}$, and (1) the encryption algorithm $\text{Enc} : \mathcal{K} \times \mathcal{N} \times \mathcal{AD} \times \mathcal{M} \rightarrow \mathcal{C}$ deterministically maps a key from $\mathcal{K}$, a nonce from $\mathcal{N}$, an associated data from $\mathcal{AD}$, and a message from $\mathcal{M}$ to a ciphertext in $\mathcal{C}$; (2) the decryption algorithm $\text{Dec} : \mathcal{K} \times \mathcal{N} \times \mathcal{AD} \times \mathcal{C} \rightarrow \mathcal{M} \cup \{\perp\}$ deterministically maps a key from $\mathcal{K}$, a nonce from $\mathcal{N}$, an associated data from

⁴The leakage modelled by L_{Algo} as the algorithm modelled by **Algo** are abstract. The definitional framework does not restrict their types or their specification.

$\mathcal{AD}$, and a ciphertext from $\mathcal{C}$ to a message in $\mathcal{M}$ if integrity checking succeeds or to a special symbol $\perp$ if integrity checking fails.

Since we only focus on nonce-based authenticated encryption with associated data, we simply refer to it as *authenticated encryption* or AE. Given a ciphertext $C \in \mathcal{C}$ of an encrypted message $M \in \mathcal{M}$, the decryption algorithm should always be able to recover M . This is captured by the *correctness* property: for any key $K \in \mathcal{K}$ generated by Gen , any nonce $N \in \mathcal{N}$, any associated data $A \in \mathcal{AD}$, and any message $M \in \mathcal{M}$, $\text{Dec}_K(N, A, \text{Enc}_K(N, A, M)) = M$. Whenever the algorithm Dec_K returns a message $M \neq \perp$ on input (N, A, C) , we say that the ciphertext is *valid*.

In the leakage environment, the adversary $\mathcal{A}$ not only has access to encryption oracle Enc and decryption oracle Dec , but also to their corresponding leakage functions LEnc and LDec , where LEnc takes as input (K, N, A, M) and returns the leakage information $\ell_{\text{Enc}} = \text{LEnc}(K, N, A, M)$ and LDec takes as input (K, N, A, C) and returns the leakage information $\ell_{\text{Dec}} \leftarrow \text{LDec}(K, N, A, C)$.

Integrity with leakage Authenticity of authenticated encryption is defined as the infeasibility for any efficient adversary to produce a valid *fresh* ciphertext, known as a *forgery*. A ciphertext is fresh if it is not the output of any previous honest encryption that the adversary would have been able to eavesdrop.

The following security notion is called *ciphertext integrity* (CI) as for black-box security of AE, but with enhance adversarial capabilities. To highlight that the adversary may repeat nonces in encryption and decryption oracles, as well as in the forgery, we use the term *nonce-misuse* (M). The ability of the adversary to observe the *leakage* (L) *both* of the encryption and decryption algorithms *during* their computation is represented by the number (2).

Definition 2 (CIML2 [BPPS17]). An authenticated encryption $\text{AE} = (\text{Gen}, \text{Enc}, \text{Dec})$ with leakage function pair $\text{L} = (\text{LEnc}, \text{LDec})$ satisfies (q, t, ϵ) -*ciphertext integrity with nonce misuse and leakage both in encryption and decryption*, denoted as CIML2, if for all (q, t) -bounded adversaries $\mathcal{A}$, we have:

$$\text{Adv}_{\text{AE}}^{\text{CIML2}}(\mathcal{A}) := \Pr [\text{CIML2}_{\text{AE}, \text{L}}(\mathcal{A}) \Rightarrow 1] \leq \epsilon$$

where the security game $\text{CIML2}_{\text{AE}, \text{L}}(\mathcal{A})$ is defined in Table 1, and $q = (q_e, q_d)$ is an upper bound on the total number of queries made to the leaking encryption and leaking decryption oracles, respectively.

Table 1: CIML2 game.

CIML2 _{AE, L} ($\mathcal{A}$) experiment	
Initialize: $K \leftarrow \text{Gen}$ $\mathcal{S} \leftarrow \emptyset$	Oracle $\text{LEnc}(N, A, M)$: $\ell_{\text{Enc}} \leftarrow \text{LEnc}(K, N, A, M)$ $C \leftarrow \text{Enc}_K(N, A, M)$ $\mathcal{S} \leftarrow \mathcal{S} \cup \{(N, A, C)\}$ return (C, ℓ_{Enc})
Procedure: $(N, A, C) \leftarrow \mathcal{A}^{\text{LEnc}, \text{LDec}}$ If $(N, A, C) \in \mathcal{S}$, return 0 If $\perp = \text{Dec}_K(N, A, C)$, return 0 Return 1	Oracle $\text{LDec}(N, A, C)$: $\ell_{\text{Dec}} \leftarrow \text{LDec}(K, N, A, C)$ $M \leftarrow \text{Dec}_K(N, A, C)$ Return (M, ℓ_{Dec})

To get the black-box ciphertext integrity, the reader may replace the oracles and the algorithms LEnc and LDec by Enc and Dec , respectively, so that $\ell_{\text{Enc}} = \ell_{\text{Dec}} = \varepsilon$.

Confidentiality with leakage We recall the notion of CCAmL2 security (Chosen-Ciphertext Attacks security with misuse-resilience and Leakage in both encryption and decryption) [GPPS19]. In this setting, the adversary has access to encryption and decryption oracles both with leakage, and tries to attack the confidentiality of messages encrypted with unique and fresh nonces. The security game is formalized in the left-or-right framework. That is, the probability that the adversary can distinguish the encryption of two different messages M^0 and M^1 of equal length should be negligible. The reason to use this framework is that it is hard to conceptually define the leakage of idealized objects in the real-or-random game as discussed by Guo et al. [GPPS19]. In the CCAmL2 security game illustrated in Table 5, the adversary may repeat nonces in queries to the first leaking encryption oracle LEnc_1 , but nonces in queries (a.k.a. challenging queries) to the second leaking encryption oracle LEnc_2 should be unique and fresh. In addition, the adversary has access to the leaking decryption oracle LDec . To avoid trivial win and at the same time capturing practical scenarios such as firmware updates, the adversary is allowed to forward challenging queries from the second encryption oracle to LDec , but can only obtain the leakage information.

Definition 3 (CCAmL2). An authenticated encryption $\text{AE} = (\text{Gen}, \text{Enc}, \text{Dec})$ with leakage function pair $\text{L} = (\text{LEnc}, \text{LDec})$ is (q, t, ϵ) -chosen-ciphertext secure with nonce misuse-resilience and leakage both in encryption and decryption, denoted CCAmL2, if for any (q, t) -bounded adversary $\mathcal{A}$, we have:

$$\text{Adv}_{\text{AE}}^{\text{CCAmL2}}(\mathcal{A}) := |\Pr [\text{CCAmL2}_{\text{AE}, \text{L}}^0(\mathcal{A}) \Rightarrow 1] - \Pr [\text{CCAmL2}_{\text{AE}, \text{L}}^1(\mathcal{A}) \Rightarrow 1]| \leq \epsilon,$$

where the security game $\text{CCAmL2}_{\text{AE}, \text{L}}^b(\mathcal{A})$ is defined in Figure 2, and $q = (q_e, q_d)$ is an upper bound on the total number of queries made to the leaking encryption and leaking decryption oracles, respectively.

Table 2: CCAmL2 game.

$\text{CCAmL2}_{\text{AE}, \text{L}}^b(\mathcal{A})$ experiment	
Initialize: 1: $K \leftarrow \text{Gen}$ 2: $\mathcal{S} \leftarrow \emptyset$ Procedure: 1: $b' \leftarrow \mathcal{A}^{\text{LEnc}_1, \text{LEnc}_2, \text{LDec}}$ 4: Return b'	Oracle $\text{LEnc}_1(N, A, M)$: $\ell_{\text{Enc}} \leftarrow \text{LEnc}(K, N, A, M)$ $C \leftarrow \text{Enc}_K(N, A, M)$ Return (C, ℓ_{Enc})
	Oracle $\text{LEnc}_2(N, A, M^0, M^1)$: If $ M^0 \neq M^1 $, return $\perp$ $\ell_{\text{Enc}}^b \leftarrow \text{LEnc}(K, N, A, M^b)$ $C^b \leftarrow \text{Enc}_K(N, A, M^b)$ $\mathcal{S} \leftarrow \mathcal{S} \cup \{(N, A, C^b)\}$ Return $(C^b, \ell_{\text{Enc}}^b)$
	Oracle $\text{LDec}(N, A, C)$: $\ell_{\text{Dec}} \leftarrow \text{LDec}(K, N, A, C)$ $M \leftarrow \text{Dec}_K(N, A, C)$ If $(N, A, C) \in \mathcal{S}$, return ℓ_{Dec} Else return (M, ℓ_{Dec})

By removing the leaking decryption queries from the experiment, one easily gets the CPAmL2 notion. By removing all the leakage from the definitions, we can thus find back the black-box CCA and the CPA notions in the nonce-misuse *resilience* setting.

2.3 The Faulty Matrix Framework [BGP⁺23]

We recall the framework due to Berti et al. [BGP⁺23] which allows easily representing queries with faults, i.e., faulty queries, to an oracle evaluating an algorithm Algo_K . This framework requires as input an implementation and is thus directly compatible with modeling the corresponding leakage function. An example of the framework is given on page 207.

Dependency Matrix Let $(f_1, \dots, f_m)$ be an implementation of a cryptographic algorithm Algo_K , where the key K may be encoded as a parameter in some of the f_j functions. By mean of $\text{Algo}_k(x) = y$ with input $x = (x_1, \dots, x_n)$, we associate the following sequence of computations:

$$\begin{aligned} f_1(x_1, x_2, \dots, x_n) &= y_1, \\ f_2(x_1, x_2, \dots, x_n, y_1) &= y_2, \\ &\vdots \\ f_m(x_1, x_2, \dots, x_n, y_1, y_2, \dots, y_{m-1}) &= y_m, \end{aligned}$$

that returns $\text{Select}(y_1, \dots, y_m) = y$ as output. **Select** is a public deterministic function made implicit in the implementation since no adversary can gain information by injecting faults during the computation of **Select**. The sequence $(f_1, \dots, f_m)$ does not force the computation to be strictly sequential in practice as the functions f_j may be independent of some the entries of $(x_1, \dots, x_n, y_1, \dots, y_{j-1})$. The input variables on which f_j really depends are said “active.” By representing non-active input variables by the empty string ε , the computation of Algo_K is captured by the following $m \times (n + m - 1)$ matrix:

$$\begin{pmatrix} \tilde{x}_{11} & \tilde{x}_{12} & \cdots & \tilde{x}_{1n} & \varepsilon & \varepsilon & \cdots & \varepsilon \\ \tilde{x}_{21} & \tilde{x}_{22} & \cdots & \tilde{x}_{2n} & \tilde{y}_{21} & \varepsilon & \cdots & \varepsilon \\ \vdots & & & \vdots & & & \ddots & \vdots \\ \tilde{x}_{m1} & \tilde{x}_{m2} & \cdots & \tilde{x}_{mn} & \tilde{y}_{m1} & \tilde{y}_{m2} & \cdots & \tilde{y}_{m\ m-1} \end{pmatrix}$$

where the j -th line corresponds to f_j ’s inputs so that $\tilde{x}_{ji} = x_i$ if f_j depends on x_i and $\tilde{x}_{ji} = \varepsilon$ otherwise, and $\tilde{y}_{ji} = y_i$ if f_j depends on the f_i ’s output y_i and $\tilde{y}_{ji} = \varepsilon$ otherwise. This is the *dependency matrix* of the implementation $(f_1, \dots, f_m)$ of Algo_k . Each column contains at most 2 distinct values that are x_i and ε for column $1 \leq i \leq n$, and y_i and ε for the $(n + i)$ -th column with $1 \leq i \leq m - 1$. We stress that the order in which an implementation accesses the memory of an input variables might have an impact on the security. Carefully choosing the f_j functions is thus of utmost importance to reflect when this order matters.

The Faulty Matrix Function The positions of the active variables in the dependency matrix of $(f_1, \dots, f_m)$ represent all the possible places to inject a fault that might influence the result of the computation of Algo_K . Without any further specification, the adversary may provide additional faulty variables to be inserted at some of these positions of its choice. The faulty algorithm will simply compute the sequence of the f_j functions based on the input x , but will replace the local input variables of f_j by the additional faulty variables specified by the adversary. The representation of which faulty variables and where it has to be injected is made through the input vector z of faulty variables and the *faulty matrix function*, denoted $\mathcal{F}_{\text{Algo}}$, by Berti et al. [BGP⁺23]. The input z is chosen by the adversary in each query, while $\mathcal{F}_{\text{Algo}}$ is fixed by the implementation. Given (x, y) , the oracle for Algo_K computes the f_j functions based on $\mathcal{F}_{\text{Algo}}(z)$. The function $\mathcal{F}_{\text{Algo}}$

can be seen as the dependency matrix where all the active variables have been replaced by the symbol $*$ that represents an empty position that the adversary can feed with a component of z . By convention, the number of variables in z is exactly the number of $*$ in the “emptied” dependency matrix. Moreover, this matrix is fed with z in the reading order. If the adversary does not want to inject a fault at a position, it simply specifies $*$ in the corresponding component of z . (The example provided on page 207 shows such a case with $z = (z_1, *, *, z_4)$.)

In practice, implementers could provide fault countermeasures to prevent the adversaries from modifying local input variables of some of the f_j functions during the computation of a faulty query. In that case, $\mathcal{F}_{\text{Algo}}$ contains the symbol $\perp$ in the corresponding entries where such a protection is provided. That is, $\mathcal{F}_{\text{Algo}}$ is determined by $(f_1, \dots, f_m)$ and the potential additional protection. If we let h be the number of $*$ entries in $\mathcal{F}_{\text{Algo}}$ where $h \leq (2n + m - 1)m/2$ and $z = (z_1, z_2, \dots, z_h)$, we have:

$$\mathcal{F}_{\text{Algo}}(z) = \begin{pmatrix} \tilde{z}_{11} & \tilde{z}_{12} & \cdots & \tilde{z}_{1n} & \varepsilon & \varepsilon & \cdots & \varepsilon \\ \tilde{z}_{21} & \tilde{z}_{22} & \cdots & \tilde{z}_{2n} & \tilde{z}_{2n+1} & \varepsilon & \cdots & \varepsilon \\ \vdots & & & \vdots & & & \ddots & \vdots \\ \tilde{z}_{m1} & \tilde{z}_{m2} & \cdots & \tilde{z}_{mn} & \tilde{z}_{m n+1} & \tilde{z}_{m n+2} & \cdots & \tilde{z}_{m m-1} \end{pmatrix}$$

such that: $\tilde{z}_{ji} = \varepsilon$, if ε is the entry of the dependency matrix at position (j, i) ; and $\tilde{z}_{ji} = \perp$, if the active variable at position (j, i) of the dependency matrix is protected. The remaining entries are in one-to-one correspondence with z (that may include some $*$) defined by the reading order.

Our new security notions in Section 3.1 retains this generality of the framework, but our construction in Section 4 does not assume any additional protection on the active entries so that the related faulty matrix function does not contain any $\perp$ symbol. As we target a generic construction of AE, our implementation $(f_1, \dots, f_m)$ will directly correspond to the building blocks.

Types of Fault The faulty matrix function above is defined for *stuck-at* fault, where z contains the input variables that will overwrite the corresponding active variables during the computation. The framework is compatible with *differential* fault as well by adding the appropriate operational symbol in z like $z = (z_1, \oplus z_2, \dots, z_h)$ for instance. In that example, the active variable corresponding to z_2 will be XOR-ed by z_2 , etc.

The framework allows injecting *transient* and *persistent* faults. The transient fault on a variable only corrupts a single copy of that variable during read, write, or some arithmetic/logical operations. On the other hand, a persistent fault modifies the variable’s memory. Once introduced, a persistent fault remains in the system throughout the execution and affects all the copies of a corrupted variable after the fault has been introduced. In the case of a persistent fault on the variable x_i at some point during the computation, and assuming that no other fault from z will modify it a second time, the i -th column of the matrix $\mathcal{F}_{\text{Algo}}(z)$ will end with the same faulty input on as many rows as the number of time this variable will still be used to answer the query. Even if z contains more than one entry distinct of $*$, it can thus be the case that the adversary injects a single fault that affects all the remaining evaluations of the f_j functions. This distinction is useful to be able to count the actual number of faults introduced by the adversary in the query.

Number of Faults and Validity of a Faulty Query If the adversary is not restricted in the number of faults injected per query, it is called *unbounded*. Otherwise, if z cannot represent more than d injections (without counting the multiplicity in the case of persistent faults), the adversary is said *d-bounded*.

Eventually, a faulty query (x, z) is called *valid*, if for all x_i of $x = (x_1, \dots, x_n)$, x_i is not replaced by a persistent faulty x'_i , i.e., the i -th column of $\mathcal{F}(z)$ does not only

contain x'_i and possibly ε . Otherwise, the query is equivalent to (x', z') , where $x' = (x_1, \dots, x_{i-1}, x'_i, x_{i+1}, \dots, x_n)$ and $\mathcal{F}(z')$ is only distinct from $\mathcal{F}(z)$ in its i -th column that only contains “*” and possibly ε . Only valid queries should be processed further by the oracle. As any invalid faulty query is equivalent to a valid faulty query, this restriction does not limit the adversarial capability. The validity of faulty queries also comes in handy when the security notion under consideration rejects some *trivial attacks*. It makes the detection of a trivial attack (that the security experiment might discard) easier, which would have been hidden in the structure of an invalid query as an artifact of the framework. We stress that it is not the faulty framework that defines what is a trivial attack or not. Moreover, the invalidity of faulty queries only concerns those where an input variable in x is directly and permanently modified by a faulty value. Of course, it does not prevent making transient faulty queries even in the computation of f_1 .

Example Let (f_1, f_2, f_3) be implementation of Algo_k such that, on input $x = (x_1, x_2)$, the algorithm output $y = (y_1, y_3)$, where $f_1(x_1) = y_1$, $f_2(x_2) = y_2$, and $f_3(x_1, y_1, y_2) = y_3$. We thus have $\text{Select}(y_1, y_2, y_3) = (y_1, y_3)$ and the following dependency matrix:

$$\begin{pmatrix} x_1 & \varepsilon & \varepsilon & \varepsilon \\ \varepsilon & x_2 & \varepsilon & \varepsilon \\ x_1 & \varepsilon & y_1 & y_2 \end{pmatrix}$$

This matrix shows that f_1 and f_2 can be computed in parallel as f_2 does not depend on y_1 . Assuming that the y_1 -entry is protected by the implementation in the computation of f_3 , we get the following faulty matrix function, and for $z = (z_1, *, *, z_4)$:

$$\mathcal{F}_{\text{Algo}} = \begin{pmatrix} * & \varepsilon & \varepsilon & \varepsilon \\ \varepsilon & * & \varepsilon & \varepsilon \\ * & \varepsilon & \perp & * \end{pmatrix}, \quad \mathcal{F}_{\text{Algo}}(z) = \begin{pmatrix} z_1 & \varepsilon & \varepsilon & \varepsilon \\ \varepsilon & * & \varepsilon & \varepsilon \\ * & \varepsilon & \perp & z_4 \end{pmatrix},$$

which evaluates to the faulty output $y' = (f_1(z_1), f_3(y'_1, y_2, z_4))$, where $y'_1 = f_1(z_1)$ (instead of $y_1 = f_1(x_1)$) which is protected in f_3 and $y_2 = f_2(x_2)$ as a non-faulty part. Furthermore, the faulty query $((x_1, x_2), (z_1, z_2, z_1, *))$ is invalid since it is equivalent to $((z_1, x_2), (*, z_2, *, *))$ that is still invalid since it is equivalent to $((z_1, z_2), (*, *, *, *))$.

Faulty Predicate Given the faulty matrix function $\mathcal{F}_{\text{Algo}}$ and a faulty query (x, z) , Berti et al. [BGP⁺23] defines the *faulty predicate*⁵ $\mathcal{F}_{\text{Algo}}(x, z)$ that returns 1 only if the query is valid. Here, we extend this predicate into $\mathcal{F}_{\text{Algo}}^d(x, z)$ to *also* satisfy the condition that the fault z results in injecting less than d faulty values (without their multiplicities in the case of persistent faults) to simplify the description of the security notion in the case of d -bounded adversary. In the unbounded case, we may have $d = \infty$. For our construction, we set $d = 1$.

2.4 Fault-then-Leak Model and FL-Secure MAC

We recall the Berti et al. security model against fault and leakage [BGP⁺23] as well as the corresponding security notion for Message Authentication Code (MAC). Both rely on the faulty matrix framework for which we slightly extend the leakage model.

The Fault-then-Leak Attack Model In the Fault-then-Leak (FL) attack model, the adversary can inject faults on any ephemeral values during a computation and observe the impact of the faults through leakage on the same query. However, the adversary has to first choose the faults to inject before making the query (or seeing the leakage

⁵The name “faulty predicate” is from us.

from the query). It cannot adaptively choose the faults for an ongoing query based on the leakages from the same query. Such a restriction is motivated by practice, as fault injections require careful setup manipulations which cannot be instantaneous launched during an execution depending on an on-the-fly measurement. However, the adversary can adaptively choose its faults across different queries based on the previously observed faulty and leaky computation.

If the implementation of Algo_K is given by $(f_1, \dots, f_m)$ in the faulty matrix framework, the leakage function $\mathbf{L}_{\text{Algo}}$ is defined as the tuple of leakage functions $(\mathbf{L}_1, \dots, \mathbf{L}_m)$, where each $\mathbf{L}_j$ is the leakage function corresponding to f_j . If K is a parameter of f_j , so is its leakage function for which we can make this explicit in the notation as $\mathbf{L}_j(K, \cdot)$.

If (x, z) is a faulty query for $\text{Algo}_K(x)$ with fault z , we denote the oracle answer by $\text{FAlgo}_K(x, z)$, where this output is defined by running $(f_1, \dots, f_m)$ on input $\mathcal{F}_{\text{Algo}}(z)$, where $\mathcal{F}_{\text{Algo}}$ is the faulty matrix function according to the faulty matrix framework. Somehow, we extend $\text{Algo}_K(x)$ into $\text{FAlgo}_K(x, z)$ with the faulty input z . A fault-then-leak query to Algo_K on input is written $\text{FLAlgo}_K(x, z)$ which returns both $\text{FAlgo}_K(x, z)$ and $\mathbf{L}_{\text{Algo}}(K, x, z)$, where $\mathbf{L}_{\text{Algo}}(K, x, z)$ is the leakage returned by the $(\mathbf{L}_1, \dots, \mathbf{L}_m)$ on the same input as for $(f_1, \dots, f_m)$ from x and $\mathcal{F}_{\text{Algo}}(z)$. By definition of the model, the implementation, the leakage functions, and the faulty matrix are public information. As a result, faulty and leaky effects can be profiled.

Types of Leakage We recall that the type of leakage depends directly on the implementation $(f_1, \dots, f_m)$ and on the device where Algo will be running. It can be of any type, but it is in general the goal of cryptographic designers to base security analyzes on weak and realistic (fault and) leakage assumptions. Concretely, and even though the leakage resulting from the computation of $f_i(x_1, x_2, \dots, x_n, y_1, y_2, \dots, y_{i-1}) = y_i$ is defined as $\mathbf{L}_i(x_1, x_2, \dots, x_n, y_1, y_2, \dots, y_{i-1}) = \ell_i$ if f_i does not depend on the key K of Algo_K and $\mathbf{L}_i(K, x_1, x_2, \dots, x_n, y_1, y_2, \dots, y_{i-1}) = \ell_i$ otherwise according to [BGP⁺23] (and so that if f_i only uses a part of the key, only that part is an input of $\mathbf{L}_i$ accordingly) when the only leakage to capture comes from the atomic functions f_i 's, we can also model the leakage respectively as $\mathbf{L}_i^*(x_1, x_2, \dots, x_n, y_1, y_2, \dots, y_{i-1}) = \ell_i^*$ or $\mathbf{L}_i^*(K, x_1, x_2, \dots, x_n, y_1, y_2, \dots, y_{i-1}) = \ell_i^*$, where $\ell_i^* = (\ell_i, y_i)$, to explicitly capture the leakage from the *wires*. That is, the output y_i can be part of the leakage if an implementation does not protect the wires from the f_i 's output (or its storage/load) to the next functions f_j 's for which y_i is an active input. For instance, we can have $\mathbf{L}_{\text{Algo}} = (\mathbf{L}_1, \mathbf{L}_2^*, \mathbf{L}_3)$ in the example above, where y_2 is explicitly part of the leakage. We next write $\mathbf{L}_{\text{Algo}}^* = (\mathbf{L}_1^*, \dots, \mathbf{L}_m^*)$ when all the wires leak, and say that *the wires leak in full*.

Fault-then-Leak Security for MAC A message authentication code is a triplet $\text{MAC} = (\text{Gen}, \text{Auth}, \text{Ver})$ of algorithms. The key generation algorithm Gen samples a key K uniformly at random from the key space $\mathcal{K}$. The authentication algorithm Auth takes as input a key $K \in \mathcal{K}$ and a message $M \in \mathcal{M}$, and outputs a tag $T \in \mathcal{T}$. We occasionally denote K as a parameter to Auth rather than input and denote it as Auth_K . The verification algorithm takes as input a key $K \in \mathcal{K}$, a message $M \in \mathcal{M}$ and a tag T and outputs 1 if T is a valid tag for M , and 0, otherwise. Similarly, we also denote Ver_K with K as a parameter. A MAC must satisfy correctness: $\forall (K, M) \in \mathcal{K} \times \mathcal{M}, \text{Ver}(K, M, \text{Auth}(K, M)) = 1$.

Let $\text{MAC} = (\text{Gen}, \text{Auth}, \text{Ver})$ be a MAC scheme with authentication tag implementation $(f_1, \dots, f_m)$ with n inputs, and verification implementation $(g_1, \dots, g_{n'})$ with n' inputs. Consequently, the leakage function pair $\mathbf{L} = (\mathbf{L}_{\text{Auth}}, \mathbf{L}_{\text{Ver}})$ is well-defined up to the specification of the wires that leak. Let $\mathcal{F} = (\mathcal{F}_{\text{Auth}}, \mathcal{F}_{\text{Ver}})$ be the (protected) faulty matrix function pair. This also precisely defines the faulty versions FAuth and FVer of the authentication algorithm and the verification algorithm, respectively.

Definition 4 (SUF-FL2). A message authentication code $\text{MAC} = (\text{Gen}, \text{Auth}, \text{Ver})$ with

leakage function pair $L = (L_{\text{Auth}}, L_{\text{Ver}})$ and faulty matrix function pair $\mathcal{F} = (\mathcal{F}_{\text{Auth}}, \mathcal{F}_{\text{Ver}})$ is (q_m, q_v, t, ϵ) -strongly existentially unforgeable against stuck-at (resp. differential), unbounded (resp. d -bounded) fault-then-leak attacks in authentication tag generation and tag verification, denoted SUF-FL2 , if for any (q_m, q_v, t) -adversaries $\mathcal{A}$,

$$\text{Adv}_{\text{MAC}}^{\text{SUF-FL2}}(\mathcal{A}) := \Pr[\text{SUF-FL2}_{\text{MAC}, \mathcal{F}, L}(\mathcal{A}) \Rightarrow 1] \leq \epsilon,$$

where the adversary makes at most q_m fault-then-leak authentication queries, and q_v fault-then-leak verification queries in the experiment in Table 3.

Table 3: SUF-FL2 game.

SUF-FL2 _{MAC, F, L} ($\mathcal{A}$) experiment	
Initialize: 1: $K \leftarrow \text{Gen}$ 2: $\mathcal{S} \leftarrow \emptyset$ Procedure: 1: $(M, T) \leftarrow \mathcal{A}^{\text{FLAuth}, \text{FLVer}}$ 2: If $(M, T) \in \mathcal{S}$, return 0 3: Return $\text{Ver}_K(M, T)$	Oracle $\text{FLAuth}(M, z)$: 1: If $\mathcal{F}_{\text{Auth}}^d(M, z) = 0$, return $\perp$ 2: $T \leftarrow \text{FAuth}_K(M, z)$ 3: $\ell_{\text{Auth}} \leftarrow L_{\text{Auth}}(K, M, z)$ 4: $\mathcal{S} \leftarrow \mathcal{S} \cup \{(M, T)\}$ 5: Return (T, ℓ_{Auth}) Oracle $\text{FLVer}(M, T, z)$: 1: If $\mathcal{F}_{\text{Ver}}^d(M, T, z) = 0$, return $\perp$ 2: $V \leftarrow \text{FVer}_K(M, T, z)$ 3: $\ell_{\text{Ver}} \leftarrow L_{\text{Ver}}(K, M, T, z)$ 4: Return (V, ℓ_{Ver})

3 FL-Security for AE and Insecure Constructions

In this section, we first model the AE security in the fault-then-leak model. Then, we show that all previous generic constructions of AE that either satisfy a leakage security or fault security fall short to remain secure in our combined attack model. While our new security notions are fully general based on the faulty matrix framework with leakage that allows adversary to inject many faults per query, all the attacks show that the existing schemes are insecure even with a single fault injection.

3.1 Fault-then-Leak Security of AE

Our new security notions extend the leakage-resistant security definitions of AE recalled in Section 2.2 by including the faulty matrix model to also capture fault attacks.

Ciphertext Integrity in the Fault-then-Leak Model We capture combined attacks against the integrity of an AE by extending the CIML2 notion into the faulty matrix model.

Definition 5 (CIML2F2). An authenticated encryption $\text{AE} := (\text{Gen}, \text{Enc}, \text{Dec})$ with leakage function pair $L = (L_{\text{Enc}}, L_{\text{Dec}})$ and faulty matrix function pair $\mathcal{F} = (\mathcal{F}_{\text{Enc}}, \mathcal{F}_{\text{Dec}})$ satisfies (q_e, q_d, t, ϵ) -ciphertext integrity with nonce misuse and leakage both in encryption and decryption against stuck-at (resp. differential), unbounded (resp. d -bounded) faults both in encryption and decryption in the fault-then-leak model, or is (q_e, q_d, t, ϵ) -CIML2F2 secure, if for any (q_e, q_d, t) -adversary $\mathcal{A}$,

$$\text{Adv}_{\text{AE}}^{\text{CIML2F2}}(\mathcal{A}) := \Pr[\text{CIML2F2}_{\text{AE}, \mathcal{F}, L}(\mathcal{A}) \Rightarrow 1] \leq \epsilon,$$

where $\mathcal{A}$ makes at most q_e fault-then-leak encryption queries and q_d fault-then-leak decryption queries in the experiment detailed in Table 4.

Table 4: CIML2F2 game

CIML2F2 _{AE, F, L} ($\mathcal{A}$) experiment	
Initialize: 1: $K \leftarrow \text{Gen}$ 2: $\mathcal{S} \leftarrow \emptyset$ Procedure: 1: $(N, A, C) \leftarrow \mathcal{A}^{\text{FLEnc, FLDec}}$ 2: If $(N, A, C) \in \mathcal{S}$, return 0 3: If $\text{Dec}_K(N, A, C) = \perp$, return 0 4: Return 1	Oracle $\text{FLEnc}(N, A, M, z)$: 1: If $\mathcal{F}_{\text{Enc}}^d(N, A, M, z) = 0$, return $\perp$ 2: $C \leftarrow \text{FEnc}_K(N, A, M, z)$ 3: $\ell_{\text{Enc}} \leftarrow \text{LEnc}(K, N, A, M, z)$ 4: $\mathcal{S} \leftarrow \mathcal{S} \cup \{(N, A, C)\}$ 5: Return (C, ℓ_{Enc}) Oracle $\text{FLDec}(N, A, C, z)$: 1: If $\mathcal{F}_{\text{Dec}}^d(N, A, C, z) = 0$, return $\perp$ 2: $M \leftarrow \text{FDec}_K(N, A, C, z)$ in $\mathcal{M} \cup \{\perp\}$ 3: $\ell_{\text{Dec}} \leftarrow \text{LDec}(K, N, A, C, z)$ 4: Return (M, ℓ_{Dec})

Confidentiality in the Fault-then-Leak Model We capture combined attacks against the confidentiality of an AE by extending the CCAmL2 notion into the faulty matrix model. We recall that in Definition 3, the (L) refers to leakage *resistance* since the challenge query is also leaky. However, for the faults, we target as a first step in this paper what we call fault *resilience* with (f) as the challenge ciphertext cannot be faulty. In the CCAmL2f1 security game as defined in Table 5, the adversary may repeat nonces in queries to the faulty and leaking encryption oracle FLEnc_1 , while nonces in queries (a.k.a. challenging queries) to the leaking encryption oracle LEnc_2 should always be unique and fresh. The adversary also has access to the leaking decryption oracle LDec . In the case that the adversary forwards challenging queries from the oracle LEnc_2 , she cannot obtain the plaintext but only the leakage (which would otherwise enable a trivial win).

Definition 6 (CCAmL2f1). An authenticated encryption $\text{AE} := (\text{Gen}, \text{Enc}, \text{Dec})$ with leakage function pair $\text{L} = (\text{LEnc}, \text{LDec})$ and faulty matrix function $\mathcal{F}_{\text{Enc}}$ is (q_e, q_d, t, ϵ) -chosen-ciphertext secure with nonce misuse-resilience and leakage both in encryption and decryption against stuck-at (resp. differential), unbounded (resp. d -bounded) faults in encryption in the fault-then-leak model, or (q_e, q_d, t, ϵ) -CCAmL2f1 secure, if for any (q_e, q_d, t) -adversary $\mathcal{A}$,

$$\text{Adv}_{\text{AE}}^{\text{CCAmL2f1}}(\mathcal{A}) := \left| \Pr \left[\text{CCAmL2f1}_{\text{AE}, \mathcal{F}_{\text{Enc}}, \text{L}}^0(\mathcal{A}) \Rightarrow 1 \right] - \Pr \left[\text{CCAmL2f1}_{\text{AE}, \mathcal{F}_{\text{Enc}}, \text{L}}^1(\mathcal{A}) \Rightarrow 1 \right] \right| \leq \epsilon,$$

where $\mathcal{A}$ makes at most q_e fault-then-leak encryption queries and q_d leaking decryption queries in the experiment of Table 5.

By removing all the LDec oracles from the experiment, we get $\text{CPAmL2f1}_{\text{AE}, \mathcal{F}_{\text{Enc}}, \text{L}}^b(\mathcal{A})$ leading to the CPAmL2f1 security notion as

$$\text{Adv}_{\text{AE}}^{\text{CPAmL2f1}}(\mathcal{A}) := \left| \Pr \left[\text{CPAmL2f1}_{\text{AE}, \mathcal{F}_{\text{Enc}}, \text{L}}^0(\mathcal{A}) \Rightarrow 1 \right] - \Pr \left[\text{CPAmL2f1}_{\text{AE}, \mathcal{F}_{\text{Enc}}, \text{L}}^1(\mathcal{A}) \Rightarrow 1 \right] \right| \leq \epsilon.$$

For completeness, we provide the stronger notion CCAmL2f2 in Appendix A as a target for future research as the following subsection already shows the difficulty to achieve CCAmL2f1 from the literature even for a single fault per query.

3.2 Insecurity of Known Constructions

There is a large body of work on leakage resilient/resistant AE schemes, and a comprehensive discussion can be found in the work by Guo et al. [GPPS19]. Regarding fault

Table 5: CCAmL2f1 game.

CCAmL2f1 _{AE, F_{Enc}, L} ^b (A) experiment	
Initialize: 1: $K \leftarrow \text{Gen}$ 2: $\mathcal{S} \leftarrow \emptyset$ Procedure: 1: $b' \leftarrow \mathcal{A}^{\text{FLEnc}_1, \text{LEnc}_2, \text{LDec}}$ 4: Return b'	Oracle $\text{FLEnc}_1(N, A, M, z)$: If $\mathcal{F}_{\text{Enc}}^d(N, A, M, z) = 0$, return $\perp$ $\ell_{\text{Enc}} \leftarrow \text{LEnc}(K, N, A, M, z)$ $C \leftarrow \text{FEnc}_K(N, A, M, z)$ Return (C, ℓ_{Enc})
	Oracle $\text{LEnc}_2(N, A, M^0, M^1)$: If $M^0 \neq M^1$, return $\perp$ $\ell_{\text{Enc}}^b \leftarrow \text{LEnc}(K, N, A, M^b)$ $C^b \leftarrow \text{Enc}_K(N, A, M^b)$ $\mathcal{S} \leftarrow \mathcal{S} \cup \{(N, A, C^b)\}$ Return $(C^b, \ell_{\text{Enc}}^b)$
	Oracle $\text{LDec}(N, A, C)$: $\ell_{\text{Dec}} \leftarrow \text{LDec}(K, N, A, C)$ $M \leftarrow \text{Dec}_K(N, A, C)$ If $(N, A, C) \in \mathcal{S}$, return ℓ_{Dec} Else return (M, ℓ_{Dec})

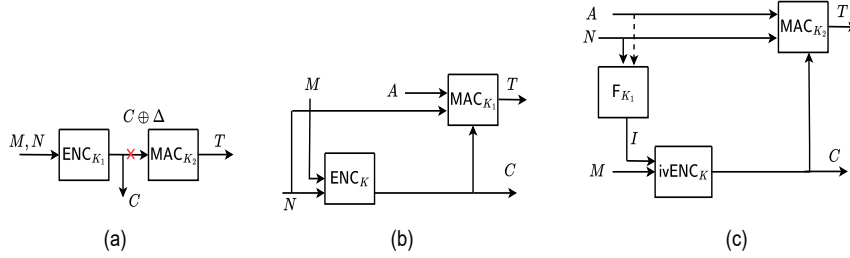


Figure 1: Leakage resilient constructions: (a) Generic ENC-then-MAC and decoupling attack on it; (b) N2; (c) A5 (without the dotted line) and A6 (with the dotted line). The block F_{K_1} represents a PRF secure against leakage according to the definition of [BMOS17].

resilience, the only secure construction is the MEM (a.k.a. MAC-then-ENC-then-MAC) due to Saha et al. [SKP22]. In this section, we discuss the insecurity of these existing leakage or fault-targeted constructions with respect to combined attacks with a single fault injection per query in our model and without explicit leakage from the wires. The single-faulting adversary refers to any adversary that can inject at most one fault per query (of differential or stuck-at type) where faults are allowed, i.e., a single faulty component in z in each call to encryption and decryption for the integrity and in encryption for the privacy (except for the challenge query). Our discussion will expose the main challenges in designing combined attack-resilient AE. We focus our discussion on generic constructions from the encryption scheme and message authentication code. For brevity, in this section, we denote an encryption block as ENC, a decryption block as DEC, and the authentication tag generation block as MAC.

Issues with Existing Constructions Addressing Leakage We consider the leakage resilient schemes as per the model by Barwell et al. [BMOS17] (misuse-resistance with no challenge leakage), and also the leakage resistant schemes (challenge encryption and decryption leakage with only misuse resilience for confidentiality) as per the model by Guo

et al. [GPPS19]. The fault inductions happen only at the inputs to the blocks (i.e., ENC or MAC) as per the matrix model, where the implementation is defined by the algorithm of the building blocks (as in our construction in Section 4). Furthermore, faults are only allowed on non-challenge encryption queries (the fl model considered throughout this work). Finally, we limit the discussion to a 1-bounded fault adversary. As we shall see in this subsection, a 1-bounded fault adversary is sufficient to break the existing designs.

The simplest generic constructions that can be converted to leakage-resilient (misuse-resistance with no challenge leakage) designs are ENC-then-MAC type, as argued by Barwell et al. [BMOS17]. There also exists specially crafted ENC-and-MAC or MAC-then-ENC type constructions, which are leakage-resilient (no challenge leakage and decryption leakage) [BKP⁺18]. However, all of these constructions become vulnerable to the so-called decoupling attack proposed by Saha et al. [SKP22]. Without loss of generality, we first describe a CIML2F2 attack for a generic ENC-then-MAC construction (ref. Fig. 1(a)), where $C = \text{ENC}_{K_1}(N, M)$ and $T = \text{MAC}_{K_2}(C)$ (without associated data) and output ciphertext (C, T) .

- **Query 1:** The adversary queries any (N, M) to the encryption oracle with a differential fault Δ on C (making it $(C \oplus \Delta)$) at the input of the MAC. That is, the adversary makes an FLEnc query on input $(N, M, (*, *, \oplus \Delta))$. This is a valid query according to $\mathcal{F}_{\text{ENC}}^1$, and the output ciphertext is $(C, T = \text{MAC}_{K_2}(C \oplus \Delta))$.
- **Forgery:** The adversary outputs $(C \oplus \Delta, T)$ which will be accepted by the decryption of the AE as $T = \text{MAC}_{K_2}(C \oplus \Delta)$ that outputs the message $\text{DEC}_{K_2}(C \oplus \Delta) \neq \perp$.

The attack is applicable to any linear encryption scheme where a keystream is XOR-ed with the message to generate the ciphertext. It may also result in message-length modification forgeries in certain cases, as discussed by Saha et al. [SKP22]. It is straightforward to observe that the decoupling attack also scales to the more recent generic constructions, denoted as N2, A5 and A6 (ref. Fig. 1), which can be made leakage-resilient as shown by Barwell et al. [BMOS17]. Broadly, all of these three schemes are ENC-then-MAC constructions. Therefore, the adversary only needs to corrupt the ciphertext input to the MAC block with fault and later needs to modify the output ciphertext according to the fault value. It also scales to the other classical schemes like MAC-then-ENC.

The three generic constructions presented in Fig. 1 cover most existing leakage-resilient AE schemes, except some specially crafted single-pass schemes. We note that single-pass MAC-then-ENC or MAC-and-ENC schemes provide only restricted security against leakage, and many of them are also vulnerable to decoupling attacks [SKP22]. Finally, Saha et al. [SKP22] show that TEDT [BGP⁺20], which is leakage-resistant in the sense of definition by Guo et al. [GPPS19], is also vulnerable to decoupling attacks. As we observe here, the main issue in any case is how the ENCs and MACs are connected to each other and how their inputs are articulated. Also, the leakage model is mostly not an issue for the decoupling attack as it is purely a fault attack.

Now, we show how block composition issues lead to other problems beyond integrity. Let us reconsider N2, A5 and A6. These new schemes deal with the nonce (N) and associated data (A) in different ways. Let us first consider the N2 scheme. It includes a nonce N going to two different modules. We have $\text{ENC}_K(N, M) = C$ and then $\text{MAC}_{K_1}(N, A, C) = T$ resulting in the output ciphertext (C, T) . We mount a CCAmL2f1 attack as follows:

- **Challenge Query:** The adversary queries $(N^{ch}, A^{ch}, M^0, M^1)$ as challenge and obtains the output (C^b, T^b) .
- **Post-Challenge Query:** The adversary makes an FLEnc query with (N, A^{ch}, M^0) , where $N \neq N^{ch}$, with fault N^{ch} only on the input of the ENC block. That is, it makes the query $\text{FLEnc}(N, A^{ch}, M^0, (N^{ch}, *, *, *, *))$, where the fault only affects

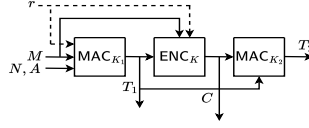


Figure 2: The MEM construction.

the first call to the nonce. This is then a transient fault that is valid according to $\mathcal{F}_{\text{ENC}}^1$. If the faulty input was $z = (N^{ch}, *, N^{ch}, *, *)$, it would not have been valid. As the encryption block is deterministic, the oracle returns (C, T) , where $C = \text{ENC}_K(N^{ch}, M^0)$ is exactly C^b if and only if $b = 0$, which allows distinguishing b . This is also CPAmL2f1 attack in the weaker security notion.

The exact same attack applies to the A5 and A6 schemes and their variants [DJS19]. Without any modification, this attack also applies to certain leakage-resistant schemes such as TEDT, where the nonce is input to both the ENC and MAC blocks.

Issues with Fault Resilient Constructions The MEM construction is depicted in Fig. 2, and is the only existing construction which takes care of the block composition issue to prevent fault attacks. This construction protects against decoupling attacks against an adversary capable of doing 1-bounded (differential) faults as we target. The security model of MEM assumes no fault in the decryption oracle and in the challenge query, which we represent as (f1) in our model. We note that the constituent MACs and encryption must also be protected against 1-bounded differential faults to guarantee their claim. In the work by Saha et al. [SKP22], the security claim relied on a fault-resilient PRF assumption, a fault-resilient random oracle (RO), and also on external randomness used along with the nonce. Interestingly, the building blocks (the MAC and the ENC) of MEM closely resemble the ones used for leakage resilience, raising the hope that the same construction would also show good resilience against a combined adversary. However, as we show in the following paragraphs, there are several issues in materializing this intuition.

We have $\text{MEM.Enc}_{K,K_1,K_2}(N, A, M) = (C, T_1, T_2)$ computed as $\text{MAC}_{K_1}(r; N, A, M) = T_1$, $\text{ENC}_K(r||M, T_1) = C$ and $\text{MAC}_{K_2}(T_1, C) = T_2$. Here r is a (internally generated) random string which is also encrypted along with M by ENC_K . Also, ENC_K is a linear encryption scheme where the keystream is XOR-ed with $r||M$. Therefore, MAC_{K_1} is randomized, whereas MAC_{K_2} and ENC_K are deterministic. Unlike the previously described leakage resilient schemes (and leakage resistant schemes such as TEDT), MEM does not use the nonce (and A) in multiple blocks. Rather, the nonce is only used once with the MAC_{K_1} . Therefore, whatever the fault we induce on the nonce, the query will be deemed invalid. Nevertheless, one can still perform the following CCAmL2f1 attack:

- **Challenge Query:** The adversary queries $(N^{ch}, A^{ch}, M^0, M^1)$ as challenge and obtains the output (C^b, T_1^b, T_2^b) .
- **Post-Challenge Query:** The adversary queries (N, A^{ch}, M^0) to the FLEnc oracle with $N \neq N^{ch}$ and fault at the T_1 input of the ENC block making it equal to T_1^b . That is, it makes the query $\text{FLEnc}(N, A^{ch}, M^0, (*, *, *, *, T_1^b, *, *))$. This is clearly a valid fault according to $\mathcal{F}_{\text{MEM.Enc}}^1$, which results in a ciphertext (C', T_1, T_2) . The adversary outputs $b = 0$ if $C' \approx C^b$ and $b = 1$, otherwise⁶. which is a successful

⁶Note that $C^b = \text{ENC}_K(r||M^b) = (r||M^b) \oplus \text{Kstream}(T_1^b, K)$, and $C' = \text{ENC}_K(r'||M^0) = (r'||M^0) \oplus \text{Kstream}(T_1, K)$, where $r \neq r'$, and Kstream is the keystream generated by the deterministic ENC_K . Therefore, the first few bits of C' and C^b will be unequal as they encrypt the randomness. The rest of the bits corresponding to M^b will be equal in case M^0 was encrypted for the challenge query. This is why we use $\approx$ symbol.

attack based on a deterministic linear ENC. This time also, this is a CPAmL2f1 attack.

In some sense, the attack mimics the nonce repeat scenario without faulting the nonce directly. If the adversary knows N_{ch} before querying it, then the same attack can be performed by faulting a pre-challenge query as well. The attack requires a stronger fault model than the differential faults and does not falsify the claimed security of MEM. However, the resilience against differential faults comes at the cost of an assumption that MAC_{K_1} is randomized (i.e., takes a fresh random input along with the message and the nonce). Without the randomization, the adversary can obtain T_1^b from the challenge query and $T_1 = T_1'$ from the post-challenge query. Then it can repeat the post-challenge query once again and inject a differential fault $\Delta = T_1^b \oplus T_1'$ at the T_1 input of the ENC block, obtaining the same attack as above. While the randomness prevents this differential attack, the effect of randomness is nullified with the stuck-at fault. Therefore, the cascading of MAC, ENC and MAC alone is not robust enough against even single faults, and extra design complexities (i.e., randomized MAC) on the blocks also do not enhance the security significantly.

On the Leakage Security of MEM: The (differential) fault security of MEM strongly relies on two integrity checks at decryption and one of them happens on the message. Therefore, the first MAC in the encryption (MAC_{K_1}) is indispensable, which directly works on the message. In order to ensure combined confidentiality, it is essential that this MAC also preserves confidentiality in the presence of leakage. If confidentiality is not ensured for the MAC, and challenge leakage is allowed (as in CCAmL2 model), then we can trivially distinguish between M^0 and M^1 observing the challenge encryption leakage. However, the confidentiality assumption is significantly strong for a MAC as no practical MAC is constructed keeping message confidentiality as a design goal. Quite naturally, it would be difficult (and impractical) to impose this extra assumption in a leakage or combined setting, where the adversary can observe (even if in a bounded manner) the internal computation. It would, therefore, be a better option to explore whether or not well-understood and easier-to-achieve assumptions can provide combined security.

The second MAC block (MAC_{K_2}) of MEM does not take the nonce as input. While this design choice may look beneficial with respect to the attack on the nonce (due to invalid faults), it actually makes the security hard to establish in the presence of leakage. To see why, let us consider a leaking decryption oracle as per the CCAmL2 model. Let the adversary make a query (C^b, T_1^b, T_2^b, N) with a fresh nonce $N \neq N_{ch}$ to this leaking decryption oracle. The verification of MAC_{K_2} is based on (C^b, T_2^b) and will be passed. Next, DEC_K will be triggered with inputs C^b and T_1^b . Consequently, the leakage from DEC_K will depend on the message M^b . While leakage on M^b still does not immediately violate the constraints of the CCAmL2 game (as the $\mathcal{L}_{\text{decch}}$ oracle returns the leakage from the challenge decryption query), it is not natural as the leakage of M^b happens from the leaking decryption oracle here instead of the special $\mathcal{L}_{\text{decch}}$ oracle. Consequently, it becomes hard to argue the security of MEM in the presence of leakage, even without faults.

Remarks In order to keep track of the design challenges, we now summarize our main observations in this section:

- (1) Security against faults strongly depends upon a) the composition between the ENC and the MAC blocks and b) how the nonce is used among the blocks.
- (2) None of the existing generic leakage-resilient constructions can resist fault attacks. This observation holds true even for some well-established leakage-resistant constructions such as TEDT. The only existing fault-resilient construction is MEM, which only resolves the issue with respect to differential faults. However, MEM requires

fresh random inputs for each query to ensure this security. So far, *no construction resists fault attacks for both stuck-at and differential faults, even while the faults are limited to non-challenge encryption queries and are 1-bounded.*

- (3) The MEM construction is not well-compatible with the existing models of leakage resistance. In the presence of leakage, the MACs (at least the MAC_{K_1}) of MEM need to ensure confidentiality. Expecting confidentiality from a MAC is a strong and nontrivial assumption, especially in the presence of leakage. This leads us to the third observation – c) the block composition also imposes nontrivial properties on the blocks to be ensured in the presence of leakage and faults. In general, a block-level construction should ensure that such “extra” assumptions on the blocks are as close to the standard assumptions as possible.

4 Our Construction: LEAF

The previous section show three issues to address to develop an FL-secure AE. Next, we show how to handle them and propose our new scheme called **LEAF** whose C1ML2F2 and CCAmL2f1 security analysis are given in Section 5. The proposed construction is secure against a single-faulting adversary as defined in Sec. 3.2.

We stress again that **LEAF** is a generic composition that relies on weaker components. Namely, an encryption scheme, and a MAC. Therefore, the implementation and the leakage functions of our AE are inherited from those building blocks. That is, as long as **LEAF** runs on a device on which the encryption and the MAC are secure in their respective sense, whatsoever is the type of leakage in that case, its combined security holds for the same type of leakage. Our results hold even if all the wires leak as defined in Section 2.4.

4.1 Handling the Nonce Issue

In order to address the issue with nonce repeat faults, we introduce redundancy in nonce. More precisely, with this strategy, we ensure that even if the nonce is used in multiple blocks, the fault cannot enable the nonce to repeat in one or multiple of them by simply faulting the nonce inputs of these blocks. Using redundancy for fault detection/correction is not new in the fault attack literature [PM18], where it is mainly used to protect the primitives. However, in our case, we do not perform any detection/correction over the nonce. Rather, we exploit the fact that a redundant encoding makes it difficult to control the nonce by means of faults.

Redundant Encoding of Nonce. Let us define the input syntax of a (nonce-based) ENC block as $\text{ENC}(K, N^w, M)$ (alternatively, $\text{ENC}_K(N^w, M)$), where $N^w := N_1 || N_2 || \dots || N_w$ with $N_i = N, \forall i$. The input syntax of the (nonce-based) MAC block MAC can also be defined in the same way. N^w represents a *redundant encoding* of the nonce. We do not specify how the building blocks process N^w , as it is to be handled internally by the blocks. Also, we do not change the syntax of the AE, which takes a single nonce of fixed length. Rather, from a given nonce N , we generate N^w at the beginning of the AE computation. In this work, we target a single fault adversary, for which $w = 2$. Therefore, we must have two redundant copies N_1 and N_2 of N .

Assumptions on the Redundant Encoding. For the standard AE syntax where N is provided as an input, it is crucial that the N_1 and N_2 are generated at the beginning of the computation and are stored in two distinct memory locations. The inputs to the building blocks are made from these two different memory locations. The implementation must ensure that a single fault injection does not end up corrupting both N_1 and N_2 . We argue that it is indeed possible to ensure this implementation assumption in practice. For example, one may consider making one copy of the nonce N to a separate memory

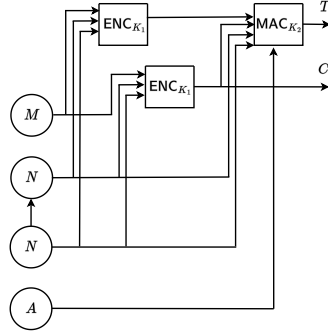


Figure 3: The illustration of LEAF encryption.

location after the inputs have been received and name these two copies as N_1 and N_2 . If we consider this copy to be an atomic function in the matrix model, then the only way to corrupt both N_1 and N_2 would be to induce the fault in N itself at the input of the copy. However, this is equivalent to a trivial attack. Any other attempt of faulting both N_1 and N_2 would require two faults for this implementation. It can be observed that if only one of N_1 or N_2 becomes faulty, it results in an invalid encoding ($N_1 \neq N_2$), which does not correspond to any valid nonce N . Therefore, such faults lead to a completely unrelated computation from the challenge nonce (which the adversary wants to repeat at least for one building block in the fault attacks involving nonce).

Several applications may require the nonce to be generated on-chip rather than supplying it from outside. In such cases, there might be situations when the adversary can fault the nonce during generation. In other words, the nonce generation may also be a part of the AE scheme in such cases. A straightforward solution to this issue is to maintain two independent albeit identical circuit or logic block to generate N_1 and N_2 . In order to enable nonce repeat, the adversary has to identically fault both the blocks with two different faults, which is anyway beyond the capability of the single faulting adversary. Also, keeping two circuits is equivalent to keeping two independent memories in this case. Therefore, the security remains unaltered with this minor variation of usability. To summarize, we rely on implementation strategies to ensure that a valid redundant encoding cannot be created with a single fault unless the adversary faults the nonce at the AE input.

Remarks. It is worth noting that there can be other alternative ways to protect the nonce. For example, the work by Dobraunig et al. [DMP20] proposed to keep a state for the nonce inside the ENC block and protect it at the implementation level (with some redundancy). In their case, the ENC block also outputs the nonce along with the ciphertext. This amounts to some modifications of the classical syntax of ENC. Broadly the idea is similar to ours – somehow making the nonce resilient to faults. However, in the AE context, the nonce can be faulted at several other places, as pointed out in the attacks described previously, and protecting it only inside the ENC is not sufficient. Therefore, encoding the nonce with some redundancy, and maintaining this encoding throughout the AE computation seems a more judicious approach. Another alternative is to use randomness along with the nonce or use a completely random nonce to make nonce repeat faults more difficult to inject. This strategy already works in the presence of differential faults, as also shown for the MEM construction. However, we note that guaranteeing good randomness is always challenging. Overall, it seems that the idea of using redundant encoding is simple and effective – at least for the block-level constructions we target.

Algorithm 1 The encryption procedure $\text{Enc}(K, N, A, M)$ of the AE construction **LEAF**.

```

1:  $K_1 \parallel K_2 \leftarrow K$ 
2:  $(N_1, N_2) \leftarrow N$   $\triangleright N_1 = N_2$ 
3:  $C \leftarrow \text{ENC}_{K_1}(N_1, N_2, M)$ 
4:  $C' \leftarrow \text{ENC}_{K_1}(N_1, N_2, M)$ 
5:  $T \leftarrow \text{MAC}_{K_2}(N_1, N_2, A, C, C')$ 
6: return  $(C, T)$ 
```

Algorithm 2 The decryption procedure $\text{Dec}(K, N, A, C, T)$ of the AE construction **LEAF**.

```

1:  $K_1 \parallel K_2 \leftarrow K$ 
2:  $(N_1, N_2) \leftarrow N$   $\triangleright N_1 = N_2$ 
3:  $(C, C') \leftarrow C$ 
4: if  $\text{Ver}_{K_2}(N_1, N_2, A, C, C', T) = \perp$  then
5:   return  $\perp$ 
6: end if
7:  $M \leftarrow \text{DEC}_{K_1}(N_1, N_2, C)$ 
8: return  $M$ 
```

4.2 Constructing the Mode

The next major concern while constructing a combined attack-resilient mode is that it should not lead to decoupling attacks or some similar integrity attack originating from the combination of leakage and faults. So far, the only known strategy for preventing decoupling is to perform multiple checks at the decryption. However, in this section, we present an alternative strategy, which requires a single integrity check, thereby simplifying the decryption operation significantly. The idea is to use redundancy once again, but this time in the computation rather than the data encoding.

The LEAF Construction: The proposed combined attack-resilient construction called **LEAF**, is depicted in Fig. 3. On input (N, A, M) , it uses two ENC's and one MAC to generate the ciphertext and the tag (C, T) . The encryption in **LEAF** begins by encrypting the message with ENC_{K_1} two times, independent of each other, both of which take the message M and the redundantly encoded nonce (N_1, N_2) as inputs. The ENC's produce two ciphertexts C and C' , and under the correct operation $C = C'$. The MAC (i.e., MAC_{K_2}) takes (N_1, N_2) , A , C and C' as input and generates the tag T . The encryption and decryption algorithms of **LEAF** are presented in Algorithm 1 and Algorithm 2. The decryption begins with checking the tag T by creating two copies of the input ciphertext C , followed by the decryption of the ciphertext (if the tag check is passed) using DEC_K . The decryption either returns the message M or $\perp$.

The resilience of the scheme against decoupling attacks is easy to follow. At any point in time, the adversary can only control one of the C or C' inputs to the MAC. Since decoupling requires controlling all the ciphertext inputs to the MAC operation, it is impossible to do this attack with a single fault in this case. The nonce here must be provided to all the blocks, which may enable the nonce-repeat attack mentioned before.

However, using the encoded nonce, we prevent this attack even against stuck-at fault models. Overall, the scheme provides security against a single faulting adversary in the CIML2F2 and CCAmL2f1 model, which we shall formally establish in the following section. As a side note, our construction does not rely on any extra randomness for fault security. Also, we avoid the linear increase in tag size, as in the case of MEM. Finally, LEAF enjoys some parallelization between the encryption blocks, while MEM was completely sequential. While this is definitely an advantage in terms of performance, it may allow an adversary to fault two encryptions simultaneously with a single injection. Hence, implementers additionally need to ensure that such injections do not lead to equal faults for both encryptions (so that they can be caught by the MAC check).

Remark. The LEAF construction provides stronger guarantees albeit being much simpler than the MEM. In the work by Saha et al. [SKP22], a similar construction was briefly mentioned (and never formally analyzed) in the appendix for fault security, and it was argued that the security depends on how it is implemented. More precisely, if the adversary can somehow fault both the ciphertexts with the same fault at the input of the MAC, the decoupling attack may still occur. However, according to our adversarial model, it would require two injections, as we assume a single injection can corrupt at most one variable. This assumption over faults, however, has to be ensured and verified at the implementation level.

The Cryptographic Assumptions: The next question is, what cryptographic properties should we expect from the ENC and the MAC blocks of LEAF? The minimum property to expect is CPA security in the presence of leakage and faults for the ENCs. More formally, we want the ENC operations to be CPAmL2f1 – that is, CPA secure while being misuse-resilient, leakage-resistant for leakages from both encryption and decryption and fault-resilient, where the fault is only limited to the encryption and does not affect the challenge query. However, the construction requires the ENCs to be *deterministic* (since C' is not included in the ciphertext and C must play the role of C' in decryption). *The CPA security of the encryption has to be ensured with leakage given the adversary obtains two identical leakage traces for each query (since C and C' are and must be computed independently).* By identical, we mean that both the traces contain the same information. This assumption is also an artifact of the construction as it uses two identical deterministic encryptions with the same key. However, it is not new and is the same with the *SPA with averaging security* in the work of Bellizia et al. [BBC⁺20]. In particular, this assumption does not increase the likelihood of DPA attacks as the information content of the two traces are the same. The only advantage for the adversary is that it can do some averaging to remove some physical noise. While such averaging may ease the SPA attacks to some extent, the security implication of this assumption is easy to verify. Also with reasonable noise, this assumption is easy to achieve practically on state-of-the-art devices [BBC⁺20].

We note that CPA security ensures the fact that no information regarding the message leaks from the ciphertext – a property that we expect in this case. It is also practical to achieve with leakage and faults, as already shown by Dobraunig et al. [DMP20]. Although the ENC construction by Dobraunig et al. [DMP20] is slightly different from what we expect (as it maintains a nonce internally, rather than taking it as input), fundamentally, it also relies on protecting the nonce, which we also achieve in a different way. Therefore, it is expected that practically achieving CPAmL2f1 should be feasible. However, explicitly constructing such an ENC is out of the scope of this paper as we focus on the block-level constructions and their expected properties here. The assumption on the MAC is simple – we only need unforgeability in the presence of leakage and faults (SUF-FL2). Successful constructions in this regard have already been proposed by Berti et al. [BGP⁺22]. Overall, we successfully avoid any confidentiality assumption for the MAC in the presence of leakage and faults, and still propose a successful construction, without restricting the fault models

and with reasonable and well-established leakage assumptions.

On Extension to Multiple Faults: Extending our security claims to an d -bounded adversary with $d > 1$ is a nontrivial task. To see why it is so for **LEAF**, we note that this construction relies on the redundant encoding of the nonce. This redundant encoding is generated “on-the-fly” from a single nonce. While it might seem that independently replicating $d + 1$ times would result in security against d faults, we observe that it is not. Creating several copies of the nonce N involves multiple *copy* operations either from a single memory (N) or from different memory locations corresponding to different redundant copies (N_{ts}). For the former case, a persistent fault on the memory of N , sometimes in between these series of copy operations, may end up corrupting several subsequent N_{ts} . We note that this fault does not result in a trivial attack as it corrupts the redundant encoding of the nonce where some of the redundant copies correspond to the actual nonce value N and some of them correspond to a faulty value N' . Therefore, it cannot be replaced by an equivalent non-faulty nonce. For the later case, where copies are created from already generated N_{ts} (maybe as a tree of copy operations), the same effect can be realized by faulting one of the N_{ts} . In a nutshell, we can corrupt multiple redundant copies even with a single injection, and, therefore, can achieve any adversarial goal with a lower number of faults d' than the allowed count d . Replicating the calls to the **ENC** module several times also becomes futile after such corruption of the nonce. Therefore, it seems that at least without some extra assumptions (e.g., some of the copy operations are fault-free), it is challenging to extend the current version of **LEAF** for $d(> 1)$ faults. Note that it is not an issue when there are only two copies and $d = 1$ (i.e., for **LEAF**).

5 Integrity and Confidentiality Analyses of LEAF

In this section, we provide both integrity and confidentiality analyses of **LEAF** against combined adversaries. Our analyses show that **LEAF** can achieve security against combined side-channel and fault attacks under reasonable assumptions. We start by describing the faulty matrix associated with the generic composition implementation.

5.1 Fault Matrix and Leakage Function of LEAF

To prove security, we first need to describe the faulty matrix function pair $\mathcal{F} = (\mathcal{F}_{\text{Enc}}, \mathcal{F}_{\text{Dec}})$ and the associated leakage function pair $\mathbf{L} = (\mathbf{L}_{\text{Enc}}, \mathbf{L}_{\text{Dec}})$. Our generic composition implementation for encryption is f_1 is the identity function to model copying N in a second memory, $f_2 = \text{ENC}_{K_1}(\cdot)$, $f_3 = \text{ENC}_{K_1}(\cdot)$, and $f_4 = \text{MAC}_{K_2}(\cdot)$. For input (N, A, M) , we have $x_1 = N$, $x_2 = A$, $x_3 = M$, with $y_1 = x_1$, $y_2 = \text{ENC}_{K_1}(x_1, y_1, x_3) = \text{ENC}_{K_1}(N, N, M)$, $y_3 = \text{ENC}_{K_1}(x_1, y_1, x_3) = \text{ENC}_{K_1}(N, N, M)$, as well as $y_4 = \text{MAC}_{K_2}(x_1, y_1, x_2, y_2, y_3) = \text{MAC}_{K_2}(N, N, A, C, C')$, and the output is $(y_2, y_4) = (C, T)$. Then, $\mathbf{L}_{\text{Enc}}(K, \cdot)$ is given by $(\mathbf{L}_{\text{ENC}}(K_1, \cdot), \mathbf{L}_{\text{ENC}}(K_1, \cdot), \mathbf{L}_{\text{MAC}}(K_2, \cdot))$ and we further assume that all the wires leak. As detailed above, we note $\mathbf{L}_{\text{Enc}}^*(K, \cdot) = (\mathbf{L}_{\text{ENC}}^*(K_1, \cdot), \mathbf{L}_{\text{ENC}}^*(K_1, \cdot), \mathbf{L}_{\text{MAC}}^*(K_2, \cdot))$ to also emphasize that (y_1, y_2, y_3, y_4) leaks. The dependency matrix is

$$\begin{pmatrix} x_1 & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon \\ x_1 & \varepsilon & x_3 & y_1 & \varepsilon & \varepsilon \\ x_1 & \varepsilon & x_3 & y_1 & \varepsilon & \varepsilon \\ x_1 & x_2 & \varepsilon & y_1 & y_2 & y_3 \end{pmatrix}$$

and the faulty leaky encryption query has the form $\text{FLEnc}_K(N, A, M, (z_1, \dots, z_{12}))$ according to the faulty matrix function

$$\mathcal{F}_{\text{Enc}}(z) = \begin{pmatrix} z_1 & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon \\ z_2 & \varepsilon & z_3 & z_4 & \varepsilon & \varepsilon \\ z_5 & \varepsilon & z_6 & z_7 & \varepsilon & \varepsilon \\ z_8 & z_9 & \varepsilon & z_{10} & z_{11} & z_{12} \end{pmatrix}.$$

We move on to our generic composition implementation for decryption. We have f_1 is the identity function to model copying N in a second memory, f_2 is the identity function to model copying C in a second memory, $f_3 = \text{Ver}_{K_2}(\cdot)$, and $f_4 = \text{Dec}_{K_1}(\cdot)$. We stress that we can always implement LEAF's decryption so that f_4 is not computed at all if f_3 detects invalid tag T , and we assume so in the following and accordingly to the specification Algorithm 2. For input (N, A, C, T) , we have $x_1 = N$, $x_2 = A$, $x_3 = C$, $x_4 = T$, $y_1 = x_1$, $y_2 = x_3$, $y_3 = \text{Ver}_{K_2}(x_1, y_1, x_2, x_3, y_2, x_4) = \text{Ver}_{K_2}(N, N, A, C, C', T)$, and $y_4 = \text{Dec}_{K_1}(x_1, y_1, x_3) = \text{Dec}_{K_1}(N, N, C)$ only if $y_3 \neq 0$ (and nothing otherwise)⁷ and the output is $\perp$ if $y_3 = 0$, and y_4 , otherwise. Then, $\text{L}_{\text{Dec}}(K, \cdot)$ is given by $(\text{L}_{\text{Ver}}(K_2, \cdot), \text{L}_{\text{Dec}}(K_1, \cdot))$ if the ciphertext is deemed valid, i.e., Ver outputs 1 so that Dec does not output $\perp$, and simply by $\text{L}_{\text{Ver}}(K_2, \cdot)$ otherwise. Here again, we consider that wires leak so that $(y_1, y_2, y_3, y_4) = (N, C, 1, M)$ is added to the leakage of valid ciphertexts, and $(y_1, y_2, y_3) = (N, C, 0)$ is added to the leakage of invalid ciphertext without adding any sensitive information (that was not already given). Hence, we will use the notation L_{Dec}^* in the theorems. The dependency matrix is

$$\begin{pmatrix} x_1 & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon \\ \varepsilon & \varepsilon & x_3 & \varepsilon & \varepsilon & \varepsilon & \varepsilon \\ x_1 & x_2 & x_3 & x_4 & y_1 & y_2 & \varepsilon \\ x_1 & \varepsilon & x_3 & \varepsilon & y_1 & \varepsilon & \varepsilon \end{pmatrix}$$

and the faulty leaky decryption query has the form $\text{FLDec}_K(N, A, C, T, (z_1, \dots, z_{11}))$ according to the faulty matrix function

$$\mathcal{F}_{\text{Dec}}(z) = \begin{pmatrix} z_1 & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon \\ \varepsilon & \varepsilon & z_2 & \varepsilon & \varepsilon & \varepsilon & \varepsilon \\ z_3 & z_4 & z_5 & z_6 & z_7 & z_8 & \varepsilon \\ z_9 & \varepsilon & z_{10} & \varepsilon & z_{11} & \varepsilon & \varepsilon \end{pmatrix}.$$

5.2 Integrity Analysis of LEAF

The following theorem states that LEAF is CIML2F2 security given a SUF-FL2 secure MAC. Note that building a CIML2F2-secure AE from a SUF-FL2 secure MAC is non-trivial. As shown in the previous sections, having a SUF-FL2 MAC is not a sufficient condition to have a CIML2F2-secure AE.

Theorem 1. *Let MAC be $(q_m, q_v, t, \epsilon_{\text{SUF-FL2}})$ -SUF-FL2 against 1-bounded faults with leakage function pairs $(\text{L}_{\text{MAC}}, \text{L}_{\text{Ver}})$ and ENC be a correct deterministic encryption scheme with leakage function pairs $(\text{L}_{\text{Enc}}, \text{L}_{\text{Dec}})$. Then for any (q_e, q_d, t_1) -adversary $\mathcal{A}$ with the derived leakage function pair $\text{L}^* = (\text{L}_{\text{Enc}}^*, \text{L}_{\text{Dec}}^*)$ with wires leaking in full and faulty matrix function pair $\mathcal{F} = (\mathcal{F}_{\text{Enc}}, \mathcal{F}_{\text{Dec}})$ as formally specified in Section 5.1, LEAF is $(q_e, q_d, t, \epsilon_{\text{CIML2F2}})$ -CIML2F2 secure against 1-bounded fault attacks, where*

$$\epsilon_{\text{CIML2F2}} \leq (q_d + 1)\epsilon_{\text{SUF-FL2}},$$

and $q_m \leq q_e$, $q_v \leq q_d$, $t_1 \leq t + (q_e + q_d)(2t_{\text{Enc}} + t_{\text{MAC}})$.

⁷Since the function “If $y_3 = 0$ stop and output $\perp$, and compute f_4 otherwise” is key-less and that y_3 is computable from the output decryption, we do not introduce this function in the dependency/faulty matrix for the sake of exposure simplicity.

Proof. We use a series of games to proceed the proof. Game G_0 is the same as CIML2F2 game where the (q_e, q_d, t) -adversary $\mathcal{A}$ tries to output a valid decryption tuple against LEAF.

Game G_1 is the same as game G_0 except that we abort if for some decryption query (N, A, C, T, z) made by $\mathcal{A}$, it can be transformed into a valid forgery against MAC_{K_2} of LEAF. To bound the difference between G_0 and G_1 , we build a sequence of $q_d + 2$ games $G_1^0, \dots, G_1^{q_d+1}$ as follows. Game G_1^i is the same as game G_0 except that we abort if one of the first i decryption queries can be transformed into a valid forgery against MAC_{K_2} in LEAF. The adversary $\mathcal{A}$'s final output (N, A, C, T) is deemed as the $(q_d + 1)$ -th decryption query, where there is no faults since otherwise resulting in a trivial forgery.

We now construct adversaries $\mathcal{C}^i$ to bound the distance between two consequent games G_1^i and G_1^{i+1} . Adversary $\mathcal{C}^i$ plays the **SUF-FL2** game against MAC_{K_2} . At the start of game, adversary $\mathcal{C}^i$ picks up key K_1 uniformly at random from $\mathcal{K}$ for ENC. Adversary $\mathcal{C}^i$ can simulate the oracles correctly for adversary $\mathcal{A}$ and return the corresponding leakages to $\mathcal{A}$ as (i) she has key K_1 of the encryption ENC; (ii) she has the full control of queries to her oracles; (iii) she can obtain leakages of ENC and MAC. When adversary $\mathcal{A}$ asks her $(j + 1)$ -th decryption query (N_j, A_j, C_j, T_j) where there is no fault in this decryption query, adversary $\mathcal{C}^i$ outputs $(N_j, N_j, A_j, C_j, C_j, T_j)$ as her forgery against MAC_{K_2} . Note that if there is no fault in this decryption query from adversary $\mathcal{A}$, namely the decryption query is (N_j, A_j, C_j, T_j) which is also the case for the final decryption (challenging) query, then the transformed tuple $(N_j, N_j, A_j, C_j, C_j, T_j)$ against MAC_{K_2} must be a fresh tuple that is explained as follows. If any of (N_j, N_j) , A_j , T_j never appeared in previous encryption queries, then obviously this is a fresh tuple. If there is a previous tuple $(N_{a,1}, N_{a,2}, A_a, C_{a,1}, C_{a,2}, T_a)$ corresponding to the encryption query $(N_a, A_a, M_a, C_a, C_a, T_a)$ such that $(N_{a,1}, N_{a,2}) = (N_j, N_j)$, $A_a = A_j$, $T_a = T_j$, then if $C_{a,b} \neq C_j$ for some $b \in 1, 2$, this is still a fresh tuple. If $C_{a,b} = C_j$ for some $b \in 1, 2$, then it implies that the adversary injects a fault at the position of $C_{a,b}$ that makes it becomes C_j while the correct value at this position should be another value that is denoted by $C_{a,1-b}$. Then $C_{a,1-b}$ is still not equal to C_j as the adversary can inject at most one fault in this query. Hence, the above tuple is still fresh. We can also make similar arguments for the case when there is some fault in this decryption, except that there is some possibility that the transformed tuple has appeared in previous queries against MAC_{K_2} . But in this case, it cannot be regarded as a valid forgery against MAC_{K_2} . Thus,

$$|\Pr[G_1^i(\mathcal{C}^i)] - \Pr[G_1^{i+1}(\mathcal{C}^i)]| \leq \epsilon_{\text{SUF-FL2}},$$

where adversary $\mathcal{C}^i$ makes at most $q_m \leq q_e$ authentication queries, $q_v \leq q_d$ verification queries, and runs in time at most $t_1 \leq t + (q_e + q_d)(2t_{\text{ENC}} + t_{\text{MAC}})$, where t_{ENC} is the running time of a single ENC algorithm and t_{MAC} is the running time of a single MAC algorithm.

By the hybrid argument,

$$|\Pr[G_1(\mathcal{A})] - \Pr[G_0(\mathcal{A})]| \leq \sum_{i=0}^{q_d} |\Pr[G_1^i(\mathcal{C}^i)] - \Pr[G_1^{i+1}(\mathcal{C}^i)]| \leq (q_d + 1)\epsilon_{\text{SUF-FL2}}.$$

For game G_1 , the final output (N, A, C, T) of the adversary $\mathcal{A}$ cannot be a valid forgery against LEAF, as otherwise it will be transformed into a valid forgery against MAC_{K_2} in LEAF. Therefore, we have $\Pr[G_1(\mathcal{A})] = 0$. Wrapping up,

$$\Pr[G_0(\mathcal{A})] \leq (q_d + 1)\epsilon_{\text{SUF-FL2}},$$

namely $\epsilon_{\text{CIML2F2}} \leq (q_d + 1)\epsilon_{\text{SUF-FL2}}$, and conclude the proof. $\square$

5.3 Confidentiality Analysis of LEAF

Before proceeding with the security analysis of the confidentiality of LEAF, we must capture the repeated challenge leakage in the encryption of the building block ENC.

5.3.1 CPAmL2f1 Security with Challenge-Encryption Leakage with Averaging

It is not enough for ENC to be CPAmL2f1 secure as we cannot simulate ourself another challenge-encryption trace given the one received in the answer to the CPAmL2f1 challenge query. Indeed, we cannot request another computation with the same nonce to the CPAmL2f1 oracle. However, assuming the security does not vanish if we can request repeated challenge-encryption *leakage* query only comes to request an SPA with averaging in the same spirit of the L_{decch} queries in Table 5 but on the encryption.

We say that ENC is CPAmL2f1-secure with averaging if it remains secure in the CPAmL2f1 experiment even with additional access to a challenge-encryption leakage oracle L_{encch} which returns a fresh leakage trace $\text{leak}_{\text{Enc}}^b \leftarrow L_{\text{Enc}}(K, N, A, M^b)$ as if it recomputed $C^b \leftarrow \text{Enc}_K(N^{ch}, A^{ch}, M^b)$.

5.3.2 The Confidentiality Proof

We now prove that LEAF achieves CCAmL2f1 security, given a CPAmL2f1 secure encryption scheme ENC, a SUF-FL2 secure MAC. Note that building a CCAmL2f1-secure AE from a CPAmL2f1 secure encryption scheme ENC and a SUF-FL2 secure MAC is non-trivial. As shown in the previous sections, how to design the connectivity among these building blocks is an essential problem towards achieving the CCAmL2f1 security.

Discussion. Let us first discuss how an adversary can win the confidentiality game if it is not allowed to fault the challenge query and the decryption queries. The only way for it is to emulate the challenge query (at least partially) on some non-challenge encryption query by injecting faults. The easiest way to achieve this is to repeat the challenge nonce by means of faults. In the presence of leakage, the nonce-repeat need not result in a valid decryptable query. For example, if the nonce is input to both ENC and MAC, the adversary can only repeat the challenge nonce (with fault) for the ENC and collect the ciphertext leakage to check with the challenge ciphertext C^b . The outcome of the overall encryption would be invalid (i.e., not decryptable), but the adversary can still win the game (with or without accessing the decryption oracle). However, the LEAF is secure from this attack as we use redundantly encoded nonce. The other way of winning would be to fault the internal state of the constituent ENC or the MACs, which emulates a situation like nonce-repeat. According to our game restrictions, this would require obtaining (through leakage) some internal state of ENC during the challenge encryption, which only depends on the nonce. Then the ENC in a non-challenge encryption can be faulted to emulate this internal state and create a nonce-repeat. The same attack might work for a MAC if it directly processes unencrypted messages. Achieving CPAmL2f1 security for the ENC ensures that it never happens, as any success of such an attack breaks CPA security. In the following, we formally prove that our construction prevents any such strategy by the attacker.

Theorem 2. *Let ENC be $(q'_e, t_1, \epsilon_{\text{CPAmL2f1}})$ -CPAmL2f1 secure against SCA with averaging (as specified above) against 1-bounded faults with leakage function pair $(L_{\text{ENC}}, L_{\text{DEC}})$, and MAC be $(q_m, q_v, t_2, \epsilon_{\text{SUF-FL2}})$ -SUF-FL2 against 1-bounded faults and leakage function pairs $(L_{\text{MAC}}, L_{\text{VER}})$. Then, for any (q_e, q_d, t) -adversary $\mathcal{A}$ with the derived leakage function pair $L^* = (L_{\text{ENC}}^*, L_{\text{DEC}}^*)$ with wires leaking in full at the exception of the input message of the CPAmL2f1 encryption (for which the leakage assumption depends on the chosen instantiation) and faulty matrix function $\mathcal{F}_{\text{Enc}}$ as formally defined in Section 5.1, LEAF is $(q_e, q_d, t, \epsilon_{\text{CCAmL2f1}})$ secure against 1-bounded fault-then-leak attacks, where*

$$\epsilon_{\text{CCAmL2f1}} \leq \epsilon_{\text{CPAmL2f1}} + (q_d + 1)\epsilon_{\text{SUF-FL2}},$$

$$\text{and } q'_e = 2q_e, t_1 = t + q_e(2t_{\text{ENC}} + t_{\text{MAC}}), q_m \leq q_e, q_v \leq q_d, t_2 \leq t + (q_e + q_d)(2t_{\text{ENC}} + t_{\text{MAC}}).$$

Proof. Recall that the faulty matrices (and, therefore, the fault functions) are the same as the ones defined in the integrity analysis of LEAF. The leakage functions also gave the same semantics except for the leakage model. Regarding the assumption on the blocks of LEAF, ENC is assumed being CPAmL2f1 and the MAC is assumed being SUF-FL2. Regarding the assumption on wires of LEAF, they can again leak in full to the adversary, at the exception of the wire manipulating the message. This wire does not leak at all in our following confidentiality analysis. However, and as always in the treatment of confidentiality with leakage, there can be a bounded leakage from within the CPAmL2f1 encryption block itself, as per assumption of the actual block chosen when instantiating LEAF.

If any of non-trivial decryption queries can pass the decryption oracle, then it implies that the adversary can break the integrity of LEAF. Hence, we have the following equation

$$\text{Adv}_{\text{LEAF}}^{\text{CCAmL2f1}}(\mathcal{A}) \leq (q_d + 1)\epsilon_{\text{SUF-FL2}} + \text{Adv}_{\text{LEAF}}^{\text{CPAmL2f1}}(\mathcal{A}),$$

where the first term on the right comes from Theorem 1. The above reduction from the CCAmL2f1 security of LEAF to the SUF-FL2 security of MAC and the CPAmL2f1 security of LEAF works as follows. Suppose that for some decryption query (N, A, C) made by $\mathcal{A}$, it can pass the decryption oracle, then it is easy to see that (N, N, A, C, C, T) is a valid forgery against MAC_{K_2} .

We then proceed to analyze the second term on the right of equation. Recall that in LEAF, the message M is first encrypted by ENC_{K_1} twice to produce two ciphertexts C' and C . Together with nonces N and N , associated data A , these two ciphertexts C and C' are authenticated by MAC_{K_2} to generate T . If the adversary cannot win the left-or-right game by seeing C' and C , then she cannot win the same game with the additional information T as these this value will only change depending on the values of C' and C that comes from M_b while the other information like nonces and associated data are the same in the challenging query. Hence, we can reduce the CPAmL2f1 security of LEAF to the CPAmL2f1 security of ENC_{K_1} as follows.

If the adversary $\mathcal{A}$ can break the CPAmL2f1 security of LEAF, then we can construct another adversary $\mathcal{B}$ that breaks the CPAmL2f1 security of ENC_{K_1} . The reduction works as follows. The adversary $\mathcal{B}$ has oracle access to ENC_{K_1} , and she selects key K_2 uniformly at random from $\mathcal{K}$ for MAC. The adversary $\mathcal{B}$ can faithfully simulate the oracles for adversary $\mathcal{A}$ by using her own oracles and the key K_2 of MAC. When adversary $\mathcal{A}$ submits the challenging query $(N^{ch}, A^{ch}, M^0, M^1)$, adversary $\mathcal{B}$ sets $((N^{ch}, N^{ch}), M^0, M^1)$ as her challenging query and submits it to ENC_{K_1} . She then repeats this challenging query to ENC_{K_1} ⁸. She receives $C^{b'}$ and C^b and the corresponding leakage $l_{\text{ENC}}^{b'}$ and l_{ENC}^b from repeated challenging queries. After that, she invokes the MAC with key K_2 to obtain $T_b = \text{MAC}_{K_2}(N^{ch}, N^{ch}, A, C^b, C^{b'})$ and the corresponding leakage l_{MAC}^b . She returns (C^b, T_b) together with the corresponding leakages $l_{\text{ENC}}^{b'}$, l_{ENC}^b , and l_{MAC}^b to adversary $\mathcal{A}$. Finally, when adversary $\mathcal{A}$ outputs a guess bit b' , adversary $\mathcal{B}$ uses b' as her guess bit against the CPAmL2f1 security of ENC_{K_1} . Hence, if adversary $\mathcal{A}$ wins the CPAmL2f1 game against LEAF, then adversary $\mathcal{B}$ can win the CPAmL2f1 game against ENC_{K_1} , which can be formalized by the following equation

$$\text{Adv}_{\text{LEAF}}^{\text{CPAmL2f1}}(\mathcal{A}) \leq \text{Adv}_{\text{ENC}_{K_1}}^{\text{CPAmL2f1}}(\mathcal{B}),$$

where adversary $\mathcal{B}$ makes at most $2q_e$ encryption queries, and runs in time at most $t_1 = t + q_e(2t_{\text{ENC}} + t_{\text{MAC}})$. We thus can conclude the proof. $\square$

⁸Note that here the adversary $\mathcal{B}$ needs to repeat the challenge query to ENC_{K_1} instead of simply copying the previous response because, although the ciphertext $C^{b'}$ is the same as C^b , the second leakage $l_{\text{ENC}}^{b'}$ may be different from l_{ENC}^b as the trace can depend on the noise in the implementation and may be probabilistic.

5.4 Instantiation issues

Implementing LEAF still requires to instantiate an encryption scheme that is CPAmL2f1 and a MAC that is SUF-FL2. As usual, the situation with integrity guarantees in the presence of physical adversaries is easier and it is for example possible to rely on the mode-level solution by Berti et al. [BGP⁺22] for this purpose, which will lead to an efficient implementation. By contrast, and to the best of our knowledge, there are no mode-level solutions to reach CPAmL2f1 in the existing literature. As a result, this block can for now only be instantiated with a standard cryptographic primitive like the AES protected with (more expensive) implementation-level countermeasures like the ones by Feldtkeller et al. [FGM⁺23] and by Berndt et al. [BEF⁺23], included in efficient modes like OCB. Improving this instantiation is an interesting scope for further research.

6 Discussion and Conclusion

Constructing AEs in the presence of a simultaneously leaking and faulting adversary is important in physical security. In this paper, we take the first step towards combined-attack secure AE and discuss the potential challenges by attacking the existing leakage and fault-resilient AEs. We also propose a new block-level construction called LEAF secure against a leaking and faulting adversary capable of doing a single fault injection per query. While the adversarial capabilities we consider are a natural first step, we note that it would be interesting to scale the construction for an d -faulting (i.e. d -bounded) adversary. Achieving this goal with reasonable assumptions is an open problem. Another open problem is to further enhance the adversarial model (for confidentiality), by including decryption and/or challenge query faults. While even in a single fault setting this currently looks challenging to achieve, this remains a practical requirement for certain applications, as in the case of leakage. Finally, a last natural problem is to construct concrete constructions with block ciphers, TBCs, or permutations based on the generic constructions proposed in this work. We believe that going concrete in that direction may open up many scopes of optimization without degrading the security, which is worth exploring. Such direction comes to study the construction with a more granular matrix representation.

Acknowledgments

Thomas Peters and François-Xavier Standaert are respectively associate researcher and research director of the Belgian Fund for Scientific Research (F.R.S.-FNRS). This work and its presentation have been funded in parts by the ERC Advanced Grant 101096871 (BRIDGE). Yaobin Shen is supported by the National Key Research and Development Program of China (2024YFB4504800), the National Natural Science Foundation of China (62402404). Views and opinions expressed are those of the authors only and do not necessarily reflect those of the EU or the European Research Council. Neither the European Union nor the granting authority can be held responsible for them.

References

- [APZ21] Melissa Azouaoui, Kostas Papagiannopoulos, and Dominik Zürner. Blind side-channel SIFA. In *2021 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pages 555–560. IEEE, 2021.
- [BBC⁺20] Davide Bellizia, Olivier Bronchain, Gaëtan Cassiers, Vincent Grosso, Chun Guo, Charles Momin, Olivier Pereira, Thomas Peters, and François-Xavier

- Standaert. Mode-level vs. implementation-level physical security in symmetric cryptography - A practical guide through the leakage-resistance jungle. In *Advances in Cryptology - CRYPTO 2020 - 40th Annual International Cryptology Conference, CRYPTO 2020, Santa Barbara, CA, USA, August 17-21, 2020, Proceedings, Part I*, pages 369–400, 2020.
- [BCO04] Eric Brier, Christophe Clavier, and Francis Olivier. Correlation power analysis with a leakage model. In *Cryptographic Hardware and Embedded Systems-CHES 2004: 6th International Workshop Cambridge, MA, USA, August 11-13, 2004. Proceedings 6*, pages 16–29. Springer, 2004.
- [BEF⁺23] Sebastian Berndt, Thomas Eisenbarth, Sebastian Faust, Marc Gourjon, Maximilian Ortl, and Okan Seker. Combined fault and leakage resilience: Composability, constructions and compiler. In *Annual International Cryptology Conference*, pages 377–409. Springer, 2023.
- [BGP⁺20] Francesco Berti, Chun Guo, Olivier Pereira, Thomas Peters, and François-Xavier Standaert. TEDT, a leakage-resilient aead mode for high (physical) security applications. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, pages 256–320, 2020.
- [BGP⁺22] Francesco Berti, Chun Guo, Thomas Peters, Yaobin Shen, and François-Xavier Standaert. Secure message authentication in the presence of leakage and faults. *Cryptology ePrint Archive*, 2022.
- [BGP⁺23] Francesco Berti, Chun Guo, Thomas Peters, Yaobin Shen, and François-Xavier Standaert. Secure message authentication in the presence of leakage and faults. *IACR Trans. Symmetric Cryptol.*, 2023(1):288–315, 2023.
- [BGPS21] Francesco Berti, Chun Guo, Thomas Peters, and François-Xavier Standaert. Efficient leakage-resilient macs without idealized assumptions. In *International Conference on the Theory and Application of Cryptology and Information Security*, pages 95–123. Springer, 2021.
- [BKP⁺18] Francesco Berti, François Koeune, Olivier Pereira, Thomas Peters, and François-Xavier Standaert. Ciphertext integrity with misuse and leakage: definition and efficient constructions with symmetric primitives. In *Proceedings of the 2018 on Asia Conference on Computer and Communications Security*, pages 37–50, 2018.
- [BMOS17] Guy Barwell, Daniel P Martin, Elisabeth Oswald, and Martijn Stam. Authenticated encryption in the face of protocol and side channel leakage. 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 693–723. Springer, 2017.
- [BPPS17] Francesco Berti, Olivier Pereira, Thomas Peters, and François-Xavier Standaert. On leakage-resilient authenticated encryption with decryption leakages. *IACR Transactions on Symmetric Cryptology*, pages 271–293, 2017.
- [BS97] Eli Biham and Adi Shamir. Differential fault analysis of secret key cryptosystems. In *Proceedings of 17th Annual International Cryptology Conference (CRYPTO)*, pages 513–525, Santa Barbara, USA, Aug 1997. Springer.

- [CJRR99] Suresh Chari, Charanjit S Jutla, Josyula R Rao, and Pankaj Rohatgi. Towards sound approaches to counteract power-analysis attacks. In *Proceedings of 19th Annual International Cryptology Conference (CRYPTO)*, pages 398–412, Santa Barbara, USA, Aug 1999. Springer.
- [CRR03] Suresh Chari, Josyula R Rao, and Pankaj Rohatgi. Template attacks. In *Cryptographic Hardware and Embedded Systems-CHES 2002: 4th International Workshop Redwood Shores, CA, USA, August 13–15, 2002 Revised Papers 4*, pages 13–28. Springer, 2003.
- [DEK⁺18] Christoph Dobraunig, Maria Eichlseder, Thomas Korak, Stefan Mangard, Florian Mendel, and Robert Primas. SIFA: exploiting ineffective fault inductions on symmetric cryptography. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, 2018(3):547–572, 2018.
- [DEM⁺] Christoph Dobraunig, Maria Eichlseder, Stefan Mangard, Florian Mendel, Bart Mennink, Robert Primas, and Thomas Unterluggauer. Nist update: Isap v2. 0.
- [DJS19] Jean Paul Degabriele, Christian Janson, and Patrick Struck. Sponges resist leakage: the case of authenticated encryption. In *International Conference on the Theory and Application of Cryptology and Information Security*, pages 209–240. Springer, 2019.
- [DMP20] Christoph Dobraunig, Bart Mennink, and Robert Primas. Leakage and tamper resilient permutation-based cryptography. *Cryptology ePrint Archive*, 2020.
- [DMP22] Christoph Dobraunig, Bart Mennink, and Robert Primas. Leakage and tamper resilient permutation-based cryptography. In *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security, CCS 2022, Los Angeles, CA, USA, November 7–11, 2022*, pages 859–873, 2022.
- [DN20] Siemen Dhooghe and Svetla Nikova. My gadget just cares for me-how nina can prove security against combined attacks. In *Topics in Cryptology-CT-RSA 2020: The Cryptographers’ Track at the RSA Conference 2020, San Francisco, CA, USA, February 24–28, 2020, Proceedings*, pages 35–55. Springer, 2020.
- [FG20] Marc Fischlin and Felix Günther. Modeling memory faults in signature and authenticated encryption schemes. In *Topics in Cryptology-CT-RSA 2020: The Cryptographers’ Track at the RSA Conference 2020, San Francisco, CA, USA, February 24–28, 2020, Proceedings*, pages 56–84. Springer, 2020.
- [FGM⁺23] Jakob Feldtkeller, Tim Güneysu, Thorben Moos, Jan Richter-Brockmann, Sayandeep Saha, Pascal Sasdrich, and François-Xavier Standaert. Combined private circuits-combined security refurbished. *Cryptology ePrint Archive*, 2023.
- [FGV11] Junfeng Fan, Benedikt Gierlichs, and Frederik Vercauteren. To infinity and beyond: Combined attack on ecc using points of low order. In *International Workshop on Cryptographic Hardware and Embedded Systems*, pages 143–159. Springer, 2011.
- [FJLT13] Thomas Fuhr, Eliane Jaulmes, Victor Lomné, and Adrian Thillard. Fault attacks on AES with faulty ciphertexts only. In *Proceedings of 10th IEEE Workshop on Fault Diagnosis and Tolerance in Cryptography (FDTC)*, pages 108–118. IEEE, 2013.

- [FRBSG22] Jakob Feldtkeller, Jan Richter-Brockmann, Pascal Sasdrich, and Tim Güneysu. Cini minis: Domain isolation for fault and combined security. In *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security*, pages 1023–1036, 2022.
- [GPPS19] Chun Guo, Olivier Pereira, Thomas Peters, and François-Xavier Standaert. Authenticated encryption with nonce misuse and physical leakage: Definitions, separation results and first construction - (extended abstract). In *Progress in Cryptology - LATINCRYPT 2019 - 6th International Conference on Cryptology and Information Security in Latin America, Santiago de Chile, Chile, October 2-4, 2019, Proceedings*, pages 150–172, 2019.
- [HOM06] Christoph Herbst, Elisabeth Oswald, and Stefan Mangard. An aes smart card implementation resistant to power analysis attacks. In *International conference on applied cryptography and network security*, pages 239–252. Springer, 2006.
- [IPSW06] Yuval Ishai, Manoj Prabhakaran, Amit Sahai, and David Wagner. Private circuits ii: Keeping secrets in tamperable circuits. In *Advances in Cryptology - EUROCRYPT 2006, 25th Annual International Conference on the Theory and Applications of Cryptographic Techniques*, volume 4004, pages 308–327. Springer, 2006.
- [ISW03] Yuval Ishai, Amit Sahai, and David Wagner. Private circuits: Securing hardware against probing attacks. In *Proceedings of 23rd Annual International Cryptology Conference (CRYPTO)*, pages 463–481, Santa Barbara, USA, Aug 2003. Springer.
- [KJJ99] Paul Kocher, Joshua Jaffe, and Benjamin Jun. Differential power analysis. In *Advances in Cryptology—CRYPTO’99: 19th Annual International Cryptology Conference Santa Barbara, California, USA, August 15–19, 1999 Proceedings 19*, pages 388–397. Springer, 1999.
- [MDP20] Loïc Masure, Cécile Dumas, and Emmanuel Prouff. A comprehensive study of deep learning for side-channel analysis. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, pages 348–375, 2020.
- [MOSW15] Daniel P Martin, Elisabeth Oswald, Martijn Stam, and Marcin Wójcik. A leakage resilient mac. In *Cryptography and Coding: 15th IMA International Conference, IMACC 2015, Oxford, UK, December 15-17, 2015. Proceedings 15*, pages 295–310. Springer, 2015.
- [PM18] Sikhar Patranabis and Debdeep Mukhopadhyay. *Fault tolerant architectures for cryptography and hardware security*. Springer, 2018.
- [PSV15] Olivier Pereira, François-Xavier Standaert, and Srinivas Vivek. Leakage-resilient authentication and encryption from symmetric cryptographic primitives. In *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security*, pages 96–108, 2015.
- [RDMB⁺18] Oscar Reparaz, Lauren De Meyer, Begül Bilgin, Victor Arribas, Svetla Nikova, Ventzislav Nikov, and Nigel Smart. CAPA: the spirit of beaver against physical attacks. In *Proceedings of 38th Annual International Cryptology Conference (CRYPTO)*, pages 121–151, Santa Barbara, USA, Aug 2018. Springer.

- [RLK11] Thomas Roche, Victor Lomné, and Karim Khalfallah. Combined fault and side-channel attack on protected implementations of aes. In *Smart Card Research and Advanced Applications: 10th IFIP WG 8.8/11.2 International Conference, CARDIS 2011, Leuven, Belgium, September 14-16, 2011, Revised Selected Papers 10*, pages 65–83. Springer, 2011.
- [SBJ⁺21] Sayandeep Saha, Arnab Bag, Dirmanto Jap, Debdeep Mukhopadhyay, and Shivam Bhasin. Divided we stand, united we fall: Security analysis of some sca+ sifa countermeasures against sca-enhanced fault template attacks. In *International Conference on the Theory and Application of Cryptology and Information Security*, pages 62–94. Springer, 2021.
- [SJR⁺20] S. Saha, D. Jap, D. Basu Roy, A. Chakraborty, S. Bhasin, and D. Mukhopadhyay. A framework to counter statistical ineffective fault analysis of block ciphers using domain transformation and error correction. *IEEE Transactions on Information Forensics and Security*, 15:1905–1919, 2020.
- [SKP22] Sayandeep Saha, Mustafa Khairallah, and Thomas Peyrin. Exploring integrity of aeads with faults: Definitions and constructions. *Cryptology ePrint Archive*, 2022.
- [TBM14] Harshal Tupsamudre, Shikha Bisht, and Debdeep Mukhopadhyay. Destroying fault invariant with randomization. In *Proceedings of 16th International Workshop on Cryptographic Hardware and Embedded Systems (CHES)*, pages 93–111, Busan, Korea, Sep 2014. Springer.
- [TMA11] Michael Tunstall, Debdeep Mukhopadhyay, and Subidh Ali. Differential fault analysis of the advanced encryption standard using a single fault. In *Proceedings of 5th Information Security Theory and Practice. Security and Privacy of Mobile Devices in Wireless Communication (WISTP)*, pages 224–233, Crete, Greece, June 2011. Springer.
- [VCMKS12] Nicolas Veyrat-Charvillon, Marcel Medwed, Stéphanie Kerckhof, and François-Xavier Standaert. Shuffling against side-channel attacks: A comprehensive study with cautionary note. In *Advances in Cryptology—ASIACRYPT 2012: 18th International Conference on the Theory and Application of Cryptology and Information Security, Beijing, China, December 2-6, 2012. Proceedings 18*, pages 740–757. Springer, 2012.
- [WKKG04] Kaijie Wu, Ramesh Karri, Grigori Kuznetsov, and Michael Goessel. Low cost concurrent error detection for the advanced encryption standard. In *Proceedings of 24th IEEE International Conference on Test (ITC)*, pages 1242–1248, Charlotte, NC, USA, Oct 2004. IEEE.

A Definition of CCAmL2f2

In this section, we present the definition of CCAmL2f2. The different with respect to CCAmL2f1 of Definition 6 is that the adversary can inject faults in decryption queries.

Definition 7 (CCAmL2f2). An authenticated encryption AE $:= (\text{Gen}, \text{Enc}, \text{Dec})$ with leakage function pair $L = (L_{\text{Enc}}, L_{\text{Dec}})$ and faulty matrix function $\mathcal{F}_{\text{Enc}}$ is (q_e, q_d, t, ϵ) -chosen-ciphertext secure with nonce misuse-resilience and leakage both in encryption and decryption against stuck-at (resp. differential), unbounded (resp. d -bounded) faults in encryption

and decryption in the fault-then-leak model, or (q_e, q_d, t, ϵ) -CCAmL2f2 secure, if for any (q_e, q_d, t) -adversary $\mathcal{A}$,

$$\text{Adv}_{\text{AE}}^{\text{CCAmL2f2}}(\mathcal{A}) := |\Pr [\text{CCAmL2f2}_{\text{AE}, \mathcal{F}_{\text{Enc}}, \text{L}}^0(\mathcal{A}) \Rightarrow 1] - \Pr [\text{CCAmL2f2}_{\text{AE}, \mathcal{F}_{\text{Enc}}, \text{L}}^1(\mathcal{A}) \Rightarrow 1]| \leq \epsilon,$$

where $\mathcal{A}$ makes at most q_e fault-then-leak encryption queries and q_d leaking decryption queries in the experiment of Table 6.

By removing all the LDec oracles from the experiment, we get $\text{CPAmL2f2}_{\text{AE}, \mathcal{F}_{\text{Enc}}, \text{L}}^b(\mathcal{A})$ and

$$\text{Adv}_{\text{AE}}^{\text{CPAmL2f2}}(\mathcal{A}) := |\Pr [\text{CPAmL2f2}_{\text{AE}, \mathcal{F}_{\text{Enc}}, \text{L}}^0(\mathcal{A}) \Rightarrow 1] - \Pr [\text{CPAmL2f2}_{\text{AE}, \mathcal{F}_{\text{Enc}}, \text{L}}^1(\mathcal{A}) \Rightarrow 1]| \leq \epsilon,$$

defined as the CPAmL2f2 security notion.

Table 6: CCAmL2f2 game.

CCAmL2f2 _{AE, F_{Enc}, L} ^b ($\mathcal{A}$) experiment	
Initialize: 1: $K \leftarrow \text{Gen}$ 2: $\mathcal{S} \leftarrow \emptyset$ Procedure: 1: $b' \leftarrow \mathcal{A}^{\text{FLEnc}_1, \text{LEnc}_2, \text{LDec}}$ 4: Return b'	Oracle $\text{FLEnc}_1(N, A, M, z)$: If $\mathcal{F}_{\text{Enc}}^d(N, A, M, z) = 0$, return $\perp$ $\ell_{\text{Enc}} \leftarrow \text{LEnc}(K, N, A, M, z)$ $C \leftarrow \text{FEnc}_K(N, A, M, z)$ Return (C, ℓ_{Enc}) Oracle $\text{LEnc}_2(N, A, M^0, M^1)$: If $M^0 \neq M^1$, return $\perp$ $\ell_{\text{Enc}}^b \leftarrow \text{LEnc}(K, N, A, M^b)$ $C^b \leftarrow \text{Enc}_K(N, A, M^b)$ $\mathcal{S} \leftarrow \mathcal{S} \cup \{(N, A, C^b)\}$ Return $(C^b, \ell_{\text{Enc}}^b)$ Oracle $\text{FLDec}(N, A, C, z)$: If $\mathcal{F}_{\text{Dec}}^d(N, A, C, z) = 0$, return $\perp$ $\ell_{\text{Dec}} \leftarrow \text{LDec}(K, N, A, C, z)$ $M \leftarrow \text{FDec}_K(N, A, C, z)$ If $(N, A, C) \in \mathcal{S}$, return ℓ_{Dec} Else return (M, ℓ_{Dec})

On the difficulty to achieve CCAmL2f2 All the schemes described in this paper are insecure against CCAmL2f2 attacks even with a single fault per query. This is obviously the case of existing strategies prior to LEAF as shown in Section 3.2 and thus to any AE prior to our construction. Next, we briefly explain why LEAF falls short to achieve the CCAmL2f2 security without additional contrived protection. The idea is simply to enforce the verification of the tag to be valid by a fault injection on input a challenge-related ciphertext. In more details, let (C, T) be the outcome of the challenge phase on input (N, A, M^0, M^1) according to the encryption of LEAF in Algorithm 1. The adversary queries decryption on (N, A, C, T') for $T \neq T'$ with a fault on the Boolean check at step 4 of the decryption in Algorithm 2. In other words, the adversary injects a stuck-at fault on y_3 to be 1 according to the notation introduced in Section 5.1 for decryption. The answer to this faulty decryption query will thus be M^b . Furthermore, this attack does not involve any leakage. Finding means to circumvent attacks against the Boolean variable of the validity

of the ciphertext as well as the additional leakage given at the same time by post-challenge fault-then-leak decryption queries does not look straightforward and is left as a scope for further research.