IACR Transactions on Symmetric Cryptology
ISSN 2519-173X, Vol. 0, No. 0, pp. 0–0. DOI:10.46586/tosc.v0.i0.0-0

Exploring Integrity of AEADs with Faults: Definitions and Constructions

Sayandeep Saha[1], Mustafa Khairallah[1,2] and Thomas Peyrin[1]

[1] School of Physical and Mathematical Sciences
Nanyang Technological University (NTU), Singapore, Singapore
{sayandeep.saha,thomas.peyrin}@ntu.edu.sg
[2] Seagate Research, Singapore, Singapore
mustafa.khairallah@seagate.com

Abstract. Implementation-based attacks are major concerns for modern cryptography. For symmetric-key cryptography, a significant amount of exploration has taken place in this regard for primitives such as block ciphers. Concerning symmetric-key operating modes, such as Authenticated Encryption with Associated Data (AEAD), the state-of-the-art mainly addresses the passive Side-Channel Attacks (SCA) in the form of leakage resilient cryptography. So far, only a handful of work address Fault Attacks (FA) in the context of AEADs concerning the fundamental properties – integrity and confidentiality. In this paper, we address this gap by exploring mode-level issues arising due to FAs. We emphasize that FAs can be fatal even in cases where the adversary does not aim to extract the long-term secret, but rather tries to violate the basic security requirements (integrity and confidentiality). Notably, we show novel integrity attack examples on state-of-the-art AEAD constructions and even on a prior fault-resilient AEAD construction called SIV$. On the constructive side, we first present new security notions of fault-resilience, for PRF (frPRF), MAC (frMAC) and AEAD (frAE), the latter can be seen as an improved version of the notion introduced by Fischlin and Gunther at CT-RSA'20. Then, we propose new constructions to turn a frPRF into a fault-resilient MAC frMAC (hash-then-frPRF) and into a fault-resilient AEAD frAE (MAC-then-Encrypt-then-MAC or MEM).

Keywords: Fault Attack · Side-Channel Attack · Authenticated Encryption with Associated Data

1 Introduction

Right from their introduction, Fault Attacks (FA) [BDL97, BS97] have received significant attention from the research community. Over the years, both analysis and fault injection techniques have improved significantly [TMA11, FJLT13, SBHS15, SH07, SBR+20, DEK+18, PCNM15, ZLZ+18, MOG+20, DEK+18, DEG+18, SBR+20, SBJ+21], making FAs practical for a large spectrum of devices ranging from embedded platforms to cloud environments. However, most of the existing research on FA in symmetric-key cryptography is dedicated to block ciphers. Given that FAs often target key recovery, and block ciphers are the most widely deployed symmetric-key primitives, this is not surprising. However, the fact that a given computation can be tampered with at any point by a fault makes the scope of FAs much wider. In other words, it is important to explore state-of-the-art symmetric-key operating modes with respect to FAs where, other than key recovery, faults can also disrupt the mode execution.

Authenticated Encryption with Associated Data (AEAD) is a class of symmetric-key operating modes that has gained prominence in the research community in the past years.

After their introduction by Bellare and Namprempre [BN00] and Katz and Yung [KY00], AEADs have become the standard for secure symmetric-key communication. They have been the main topic of at least two international design competitions[1] [2], one of which is still ongoing[2]. The NIST call for lightweight cryptography[2] enlists SCA security as a desired condition. Naturally, security against FAs also comes into context.

Fault Attacks on AEADs: Most of the existing FA proposals on AEADs aim to recover the long-term secrets with fault injection [DEK+16, SC16, DMMP18]. Both Differential Fault Analysis (DFA) [SC16, DRSA16, KS+20, JSP20, SLXY20, QAMSB+19, SOX+21] and Statistical Fault Attacks (SFA/SIFA) [DEK+16, DMMP18, RAD19, QAMSB+17] have been exploited in this regard. Some of these attacks require the nonce to be repeated, whereas the statistical ones are free of any such requirements. While recovering the long-term secret is indeed the ultimate form of attack, it is not the only goal of an attacker. Rather, several mode-level artifacts enable much simpler attack surfaces that can be exploited to violate the integrity and confidentiality requirements. As a simple example of integrity violation, one may consider an attack where the adversary tampers with the nonce with a fault at the beginning of the encryption and generates the ciphertext. It generates the forgery by replacing the correct nonce with a faulty one during the query made to the decryption oracle [IMG+15, SSB+18, SMS+18]. Another forgery attack, specific to the AE modes CLOC [IMGM14] and SILC [IMG+14], was proposed by Roy *et al.* in [RCC+16]. Satisfying the confidentiality requirements in the presence of a fault adversary also seems challenging even without key recovery. One simple attack is to make the nonce repeat by injecting faults, which readily breaks the real-vs-random security of the scheme [FG20]. However, fault assisted nonce-reuse is just one among several attack surfaces that may arise due to faults. Recently, Dobraunig *et al.* proposed a unified model for analyzing leakage due to faults and SCA [DMP20]. However, their work mainly addresses the issue of recovering the long-term secret from a permutation-based construction. The difficulty of long-term secret recovery has also been addressed in the ISAP proposal [DEM+20].

To the best of our knowledge, the only work that aims to formalize fault-resilient security notions for AEADs is [FG20]. Their main idea was to include randomness in the SIV-based AEAD schemes (named as SIV$). However, we found that their constructions and assumptions are not enough to ensure integrity in the presence of faults.

1.1 Our Contributions:

Novel Fault-based Attack on Integrity. In this paper, we shed light on the integrity issues arising due to FAs. We first characterize different attack surfaces violating the integrity, and then propose a new class of fault-assisted integrity attack called *decoupling attack*. For any given message M, the decoupling attack can generate a valid forgery with message $M \oplus \Delta$, where Δ is any difference. It also extends to message length modifying attacks. We show that the fault resilient scheme in [FG20] is vulnerable to this attack.

We carefully distinguish the proposed decoupling attack from attacks which corrupt the message/nonce/Associated Data (AD) at the input of the AEAD with a fault Δ, and uses the output of the encryption module as a forgery. One may notice that the attack on the nonce input mentioned previously also falls in this category. In the decoupling attack, the fault is injected during the computation and not at the beginning. While corrupting the inputs indeed qualifies as a practical attack, their triviality follows from the fact that the inputs can also be tampered at their very sources to create an equivalent effect. Such tampers are typically "out-of-the-scope" for the AEAD security and these "generic" attacks are excluded from potential attacks in this work. Our security model (ref. next paragraph)

[1] https://competitions.cr.yp.to/caesar-submissions.html
[2] https://csrc.nist.gov/projects/lightweight-cryptography

abstracts away input faults where faults happen at the input itself or slightly after, thus, drawing a boundary between what is considered an input fault and what is not. Upon fault injection, the adversary might get a valid ciphertext equivalent to manipulating the input (i.e. a valid plaintext ciphertext-pair) but nothing more. Regardless of the validity of the ciphertext, it cannot generate/predict a new ciphertext.

New Security Notions for Fault Resilience. Faults may corrupt different parts of an execution – the data-flow, control-flow, the memory elements, and even the constants and tables. Furthermore, the nature of the corruption can be diverse – from flipping a few bits to fixing a (full/partial) state to a predefined value. Finally, a fault can be a transient or a persistent fault staying in the system for several encryptions. We capture this diversity within a unified template. Such unified modeling helps us to formally argue the security of constructions with respect to adversaries having varying power. Next, we introduce new security notions called fault-resilient-PRF (frPRF), fault-resilient-Random Oracle (frRO), fault-resilient-MAC (frMAC) and fault-resilient-AEAD (frAE). Informally, a frPRF is simply a PRF where the PRF security is not affected by faulty queries. Practically speaking, this property can be ensured if the PRF keys cannot be recovered with faulty queries, so it can be obtained from a primitive with fault countermeasures. This frPRF notion will then be used as a building block for our constructions.

Fault Resilient MAC and AEAD from Fault Resilient PRF. Our final contributions are two new constructions. First, we describe hash-then-frPRF, a fault-resilient MAC (frMAC) based on the frPRF notion, having similar properties as frPRF along with MAC security. The hash-then-frPRF construction takes a randomness-augmented input and generates a tag. The security of this construction is ensured for an adversary capable of doing a single differential fault per query, before the challenge query is submitted.

Regarding AEAD, a straightforward way to achieve security against faults is to duplicate the entire computation multiple times and check for any mismatch in the results. More precisely, one may encrypt the same (N, A, M) tuple t times and only output if the results match for all the cases. However, this entails a computational overhead of t times. Another alternative is to apply t different AEADs (or maybe the same AEAD algorithm with different keys) on the same message and let the decryption module check if all the decrypted messages are the same. This approach typically increases the ciphertext length and the amount of computation by t times. In search of a more elegant and practical solution, we found that it is somewhat inevitable to get rid of the redundancy completely, at least for the MAC computations. However, the order of the encryption and the MAC operations plays a crucial role in the security. After a careful analysis, we propose a three-pass mode called *MAC-then-Encrypt-then-MAC* or MEM, from which the MAC and the encryption steps are realized with frMAC and frPRF primitives, respectively. The reason for realizing a three-pass mode is that single and two pass modes, in general, fail to prevent the proposed decoupling attacks. It was also interesting to observe that several other possible three-pass constructions may not achieve the desired security as the decoupling attack works on them.

MEM ensures integrity for single differential fault injected per encryption query. Additionally, it achieves confidentiality if there is no fault in the challenge queries. An AEAD security game is designed for this purpose, which carefully excludes the generic attacks mentioned before. We note that faulty encryption and non-faulty decryption emulates numerous practical scenarios where the encryption module can be accessed and faulted by an adversary, but the decryption module runs in a secure environment. Also, single differential fault per encryption is the most common scenario that can be realized for a large class of practical devices.

The rest of the paper is organized as follows. In Sec. 2, we present the notations, and concepts utilized throughout the paper. We formalize the fault modeling in Sec. 3, followed by a discussion on fault-tolerance of conventional AEADs in Sec. 4. We present the decoupling attack and discuss the vulnerability of SIV$ to this attack in Sec.5. We

present the new security notions and the new hash-then-frPRF MAC construction in Sec. 6, and the MEM construction with associated security games in Sec. 7. We conclude the paper in Sec. 8 with some open issues discussed in Appendix D (supp. material).

2 Preliminaries

2.1 Notation and Terminology

We denote deterministic assignments with '$\leftarrow$'. Assignments with uniform random sampling from a set are denoted as'$\xleftarrow{\$}$'. Definitional assignments are represented with ':='. By $|X|$ we denote the length, if X is a string and size, if X is a set. ε represents an empty string, and $\emptyset$ denotes an empty set. $x \oplus y$ denote the bitwise XOR operation between two strings x and y, provided $|x| = |y|$. Similarly, for strings x_1, x_2, $\cdots$, x_n, $x_1||x_2|| \cdots x_n$ denotes their concatenation. For two sets A and B, $A \cup B$ denotes the set union and $A \cap B$ denotes the set intersection. For two sets D, R, $f : D \rightarrow R$ denotes a function with domain D, range R and $Im(f) := \{f(x) : x \in D\} \subseteq R$. $\star$ denotes a placeholder.

2.2 Definitions and Concepts

Definition 1 (Authenticated Encryption with Associated Data (AEAD))**.** A nonce-based AEAD $\mathcal{AE}$ consists of following efficient algorithms:

- $\mathsf{Enc}(K, N, M, A; r)$: On input a secret key $K \xleftarrow{\$} \mathcal{K}$, a nonce $N \in \mathcal{N}$, a message $M \in \mathcal{M}$, an associated data $A \in \mathcal{AD}$, and an (optional) internally generated randomness $r \in \mathcal{R}$, the Enc outputs $C \in \mathcal{C}$ (i.e. $C \leftarrow \mathsf{Enc}(K, N, M, A; r)$). Here $\mathcal{K} := \{0,1\}^{\mathcal{AE}.\mathsf{klen}}$ is the keyspace, $\mathcal{N} := \{0,1\}^{\mathcal{AE}.\mathsf{nlen}}$ is the nonce space, $\mathcal{M} := \{0,1\}^*$ is the message space, $\mathcal{AD} := \{0,1\}^*$ is the AD space, $\mathcal{R} := \{0,1\}^{\mathcal{AE}.rlen}$ is the randomness space, and $\mathcal{C} := \{0,1\}^*$ is the ciphertext space. $\mathcal{AE}.\mathsf{klen}$, $\mathcal{AE}.\mathsf{nlen}$ and $\mathcal{AE}.\mathsf{rlen}$ are the lengths of the key, nonce and the randomness, respectively.

- $\mathsf{Dec}(K, N, A, C)$: On input a secret key K, a nonce $N \in \mathcal{N}$, an AD $A \in \mathcal{AD}$, and a ciphertext $C \in \mathcal{C}$, the deterministic Dec outputs $M \in \mathcal{M} \cup \perp$ ($M \leftarrow \mathsf{Dec}(K, N, A, C)$).

Definition 2 (Pseudo-Random Function (PRF))**.** A keyed function $F : \mathcal{K} \times \mathcal{M} \rightarrow \mathcal{Z}$, denoted as $F(K, M)$ (or $F_K(M)$) takes a secret key $K \in \mathcal{K}$, and a message $M \in \mathcal{M}$ and returns a fixed length string $Z \in \mathcal{Z}$. For $K \in \mathcal{K}$, let $\mathsf{PRF}_K(M)$ be an oracle which takes as input a message $M \in \mathcal{M}$ and returns $F_K(M)$. A (q_m, t)-adversary against the prf security of F is an adversary $\mathcal{A}$ with oracle access to PRF_K, making at most q_m queries, and running in time at most t. F qualifies as a PRF, if for all $\mathcal{A}$, trying to distinguish the real world PRF_K, and an ideal world oracle RF returning independent random values (instantiating a truly random function) the advantage:

$$\mathbf{Adv}_F^{\mathsf{prf}}(q_m, t) \leq \max_{\mathcal{A}} |\Pr[K \xleftarrow{\$} \mathcal{K} : 1 \leftarrow \mathcal{A}^{\mathsf{PRF}_K}] - \Pr[1 \leftarrow \mathcal{A}^{\mathsf{RF}}]|,$$

is negligible in the security parameter. An adversary $\mathcal{A}$ is not allowed to repeat queries.

Definition 3 (Message Authentication Code (MAC))**.** For $K \in \mathcal{K}$, let $\mathsf{Mac}_K(M)$ be the MAC oracle which takes as input a message $M \in \mathcal{M}$ and returns $F_K(M)$, and $\mathsf{Verify}_K(M)$ be the verification oracle that takes $(M, \tau) \in \mathcal{M} \times \mathcal{T}$ as input and returns 1 if $F_K(M) = \tau$ and 0, otherwise. We assume that an adversary makes queries to the two oracles Mac_K and Verify_K for a secret key $K \in \mathcal{K}$.

A (q_m, q_v, t)-adversary against the mac security of F is an adversary $\mathcal{A}$ with oracle access to Mac_K and Verify_K, making at most q_m queries to its first oracle and at most q_v

verification queries to its second oracle, and running in time at most t. We say that $\mathcal{A}$ forges if any of its queries to Ver_K returns 1. The advantage of $\mathcal{A}$ against the mac security of F is defined as:

$$\mathbf{Adv}^{\mathsf{mac}}_F(\mathcal{A}) := \Pr[K \xleftarrow{\$} \mathcal{K} : \mathcal{A}^{\mathsf{Mac}_K, \mathsf{Ver}_K} \text{ forges}]$$

where the probability is also taken over the random coins of $\mathcal{A}$, if any. The adversary is not allowed to ask a Verify_K query (M, τ) if a previous query M to Mac_K returned τ. However, it is still possible that a Verify_K query (M, τ) is first made, possibly rejected, and a Mac_K query M is subsequently made. We define $\mathsf{Adv}^{\mathsf{mac}}_F(q_m, q_v, t)$ as the maximum of $\mathsf{Adv}^{\mathsf{mac}}_F(\mathcal{A})$ over all (q_m, q_v, t)-adversaries. Finally, we represent the mac security using the advantage of an adversary trying to distinguish the real world $(\mathsf{Mac}_K, \mathsf{Verify}_K)$ and the ideal world. The ideal world oracles are $(\mathsf{RF}, \mathsf{Rej})$, where RF returns an independent random value (instantiating a truly random function) and Rej returns 0 for every verification query.

$$\mathbf{Adv}^{\mathsf{mac}}_F(q_m, q_v, t) \leq \max_{\mathcal{A}} |\Pr[K \xleftarrow{\$} \mathcal{K} : 1 \leftarrow \mathcal{A}^{\mathsf{Mac}_K, \mathsf{Verify}_K}] - \Pr[1 \leftarrow \mathcal{A}^{\mathsf{RF}, \mathsf{Rej}}]|,$$

whereas for mac security, an adversary makes at most q_m queries to its first oracle and q_v verification queries to its second oracle, runs in time at most t, and returns a decision bit. We note that $q_v = 0$, this is equivalent to PRF security.

3 Modelling the Faults

The first step towards formalizing fault resilience is to model the faults with a unified structure, which encodes the fault location, width, physical model,...*etc.* Such a unified representation is useful for quantifying the security. The unified representation that we present in this section is similar to the one presented in [FG20], but is a bit more elaborate[3]. It captures different variable classes for fault injection, which was not covered in [FG20].

Before formally presenting the model let us briefly describe the type of faults practically observed on devices. In practice, localized (say within one bit/byte of a variable) and temporally precise (at a specific line/statement in the code) transient faults are observed in most of the devices [SGD08, DEK+18, MOG+20]. The faults may result in random corruption of the affected variable, flipping of specific bits, or specific data-dependent corruption such as bit set/reset [SBHS15, SBJ+21]. Persistent bit-flips at tables/constants are also feasible, where the fault remains in the system till a reset [ZLZ+18]. For software implementations, instruction-corruption/skip faults are observed which eventually results in corrupting the data-flow or control-flow of the program [KPB+17, MHER14]. Overall, the main impact of any fault in a cryptographic setting results in a data/control flow corruption. The precision, width, and repeatability of a fault are largely dependent on the quality of the injection setup. For example, it is difficult to make targeted bit-flips with clock-glitch, but such targeted bit flips are common with laser-setups. Our unified representation and fault models, however, abstracts necessary traits of practical faults.

The fault in a computation is characterized by the tuple:

$$\mathcal{F} := \langle \{v_i, mod_i, t_i, w_i\}_{i=1}^{nf} \rangle \tag{1}$$

The parameters in Eq. (1) are defined in Table 1, along with a classification of the variables to be faulted, and a classification of the fault models. In order to explain why classifying variables is important, we distinguish between data and control faults. Control faults, for example, may skip an entire function call, which cannot be represented easily with a

[3] For our security proofs, we only use a subset of the fault models and faultable variables covered by this unified structure. However, we believe that the scope of this representation is not limited to this work, and it may be of further use in future research.

data fault. In general, control faults can be realized by corrupting the loop-counter, a loop-decision variable, or the decision variable of an if-else statement, which do not always contain the data/state being encrypted. Similarly, faults in constants may have different consequences on both control and data. For example, fault in a constant domain separator may open up scopes for several attacks. Therefore, a holistic analysis should capture each of these variable classes separately, to clearly specify the scope of the claimed security with respect to faults.

The fault models capture the physical faults. The temporal and spacial precision of faults can be captured by the variables, transient/persistent nature, and the fault width parameters. At the level of a pseudocode, the temporal precision of a fault can be captured by specifying the position (e.g., line number) of the statement where the target variable resides. Also, if the statement resides inside a loop, the loop counter is used as an extra parameter to specify the exact timing of the fault during loop execution. All of these parameters depend on the target implementations, and, therefore, helps to capture different (hardware/software) implementations under our unified template. The diff faults model a large class of faults (including single/multi-bit flips) changing the value of a variable. The rand faults model the random corruption of variables and is a special case of diff faults. fix is the strongest fault model in this case in terms of physical realization. It requires setting a variable (or a part of it) to a desired value, which is practically difficult for a large fault width (w_i). However, fix models the data dependent bit set/reset faults. It can also model a a specific case of diff faults, where the value of the target variable is known. In this case, fix faults can be be interpreted as diff faults having a different Δ corresponding to each valuation of the target variable. Regarding persistent faults, it is worth mentioning that we do not assume the faults to persist during the challenge queries of the schemes we are going to propose in this work. This is a natural choice as in this work we shall only argue security for non-faulty challenges.

Table 1: Unified Representation of Faults

Fault Representation	
Params	Description
v_i	Denote the variables corrupted by faults. $v_i \in$ data $\cup$ control $\cup$ constant.
nf	The number of faults injected throughout the computation (in the same or different clock cycles).
w_i	Denote the width (how many bits within a target variable are corrupted) of a fault ($0 \leq w_i \leq \lvert v_i \rvert$).
mod_i	The logical abstraction of physical nature of faults (fault models). $mod_i \in$ fix $\cup$ diff $\cup$ rand $\cup$ nof.
t_i	Denote if the fault is transient/persistent and the temporal fault location.

Variable Classification	
Params	Description
data	Denotes the set of data-flow variables (input, output and intermediate states of the computation).
control	Denotes the set of control-flow variables (branch statements).
constant	Denotes the set of constants, tables, and domain separators of the AEAD algorithm.

Fault Models	
fix	Denote faults where the adversary is allowed to fix w_i bits of the target variable to some desired value.
diff	Denote the differential faults where the adversary is allowed to select a bitwise differential Δ_i for variable v_i (with $HW(\Delta_i) = w_i$) and set $v_i^{'} = v_i \oplus \Delta_i$. Here $v_i^{'}$ is the faulty version of v_i and $HW(\cdot)$ denote the Hamming weight.
rand	Same as diff except the fact that $\Delta_i \xleftarrow{\$} \{0,1\}^{\lvert v_i \rvert}$ and $HW(\Delta_i) = w_i$.
nof	Denotes the case when the adversary chooses not to inject a fault in the execution.

Representing Faults in an Algorithm

In order to define the security games flexibly with faults, we follow a specific syntax of defining faults with an oracle. The interaction between a faulting adversary $\mathcal{A}(\mathcal{F})$, and an oracle $\mathcal{O}_{flt}$ is defined in Table 2. $\mathcal{O}_{flt}$ maintains a list $\mathcal{V}_{flt}$ of the values taken by every variable appearing during the computation. If there is a fault in any of the variables (or

Table 2: Interaction between faulting adversary and oracle

$CodeMem \leftarrow$ Initialize() ▷ Initializes the code and memory $X \leftarrow \mathcal{A}^{\mathcal{O}_{flt}}(\mathcal{F})$ ▷ Oracle call
$\mathcal{O}_{flt}$: // Oracle computation prior and during fault injection $\mathcal{V}_{flt} \leftarrow \mathcal{V}_{flt} \| \mathsf{Fault}(\mathcal{F}.v_i)$ ▷ Fault simulates the desired fault. // Oracle computation after fault injection **return** *output*

any of its occurrence specified in $\mathcal{F}$), it maintains the faulty value in $\mathcal{V}_{flt}$. More precisely, in case a variable is utilized multiple times, $\mathcal{V}_{flt}$ maintains whatever (faulty/non-faulty) values have been assumed by this variable in its multiple occurrences[4]. Furthermore, $\mathcal{A}(\mathcal{F})$ interacts with the code and the memory elements of the underlying algorithm only through the oracle (we call it *oracle call*). In case the fault is transient, the adversary only temporarily modifies the value of the variable and keeps the original value intact in the memory. In the case of multiple transient faults, every fault occurs with respect to the original non-faulted value in the memory. However, for persistent faults, the original value in the memory is corrupted and used throughout the computation by the oracle.

At this point, it is important to distinguish the faults in two types: a) *information-leaking* faults, and b) *altering faults*. While every fault indeed alters the data or flow of computation, *information-leaking* faults require a special mention as the corruption is always meant for leaking some secret. By the very nature of FAs, such leakage happens from the outputs of queries faulted with information leaking faults. Our oracles does not distinguish between these altering and information-leaking faults, but only keeps the scope that some secret might leak through the faulty outcomes.

4 Conventional AEADs and Faults

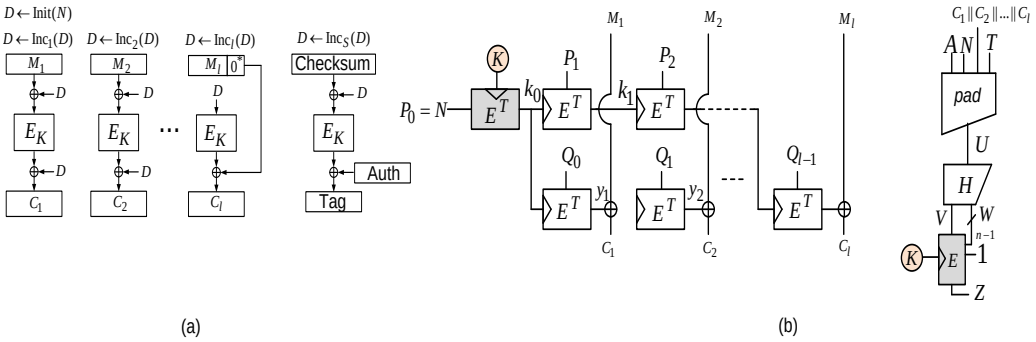

Figure 1: Impact of faults on long-term secret: (a) OCB; (b) TEDT.

There are several ways in which faults can be exploited for attacking AEAD schemes. According to the basic security goals, they can be classified into two broad classes – attacks on confidentiality; and attacks on integrity. The main focus of this paper is to shed light on integrity violations. We restrict ourselves to encryption faults.

[4]In Table 2, we use the abstract function Fault(·) (also called *fault oracle*) to represent results of faulty computation for some variable or function.

4.1 Recovery of Long-Term Secrets

The most obvious attack on integrity is to recover the long-term secret associated with the MAC by means of faults. If the underlying scheme utilizes the same long-term key for both encryption and MAC generation, then recovering the key from either the encryption or the MAC block is feasible. Most of the existing AEADs do not include explicit fault countermeasures. Therefore, attacking the primitives processing the long-term key is always a feasible attack surface.

In order to give a concrete example, we refer to the OCB scheme [KR16] depicted in Fig. 1(a). The encryption (E_K) processes each message block (M_i) XOR-ed with a nonce-dependent mask (D), and then further XORs the same mask with the result to produce the ciphertext. In the last incomplete block, only the mask D is encrypted and the padded message is XORed with the encryption output. The MAC computes a checksum over the message blocks, which is then XOR-ed with D. Next, an encryption operation is performed on this followed by an XOR with the $Auth$ (AD). To attack the MAC, without loss of generality, we consider the AD part to be empty. As a result, the adversary can obtain the encryption of the message checksum XOR-ed with the mask D. As the ciphertext (i.e. the tag) is available, the adversary can launch several key recovery attacks such as PFA [ZLZ+18], SFA [DMMP18] or SIFA [DEK+18] to recover this long-term secret (encryption key K) without nonce repeat. For example, if the encryption (E_K) is realized with AES, single-bit faults can be injected at the 10th round S-Box input and several faulty tags can be collected. Then the adversary can guess one key byte of the last round at a time to partially decrypt a byte of the faulty tags reaching the fault injection point. If the key guess is correct, then the single-bit corruption should be observed at the injection point. Computing the Hamming Distance (HD) between all partially decrypted faulty tags would expose this single-bit corruption – the key guess with lowest aggregate HD denotes a correct key byte [GYTS14]. This attack would eventually expose several key bytes. Alternatively, an adversary can perform this attack on the encryption of the incomplete message block recovering the entire key. Recovery of the long-term key is sufficient to create universal forgery attacks. Such key-recovery attacks are feasible for many other schemes beyond OCB. Therefore, each block cipher call for such schemes should be protected against FAs.

The recovery of long-term secrets can be considered as a leakage. Therefore, it is important to explore the leakage-resilient schemes in this regard. The processing of long-term secrets is often restricted to a few primitive calls in leakage resilient schemes. Let us consider the TEDT [BGP+20] scheme as an example (ref. Fig. 1(b)). TEDT performs PSV [PSV15] encryption followed by MAC generation (Encrypt-then-MAC). The IV of the PSV encryption is generated by encrypting the nonce (N) with a leak-free TBC with a "public key" (PK) as a tweak (i.e $T = PK$) and a long-term secret key K. The MAC part hashes the ciphertext, the AD, the nonce, and PK, and then encrypts the hash with a leak-free TBC with the same secret key. In case this leak-free block of TEDT is not protected against FAs, the attacks on it become straightforward. Since TEDT is nonce misuse-resilient, a large class of attacks (such as DFA) becomes feasible on the leak-free block during the MAC operation if the nonce repeats. One should note that the unavailability of the ciphertexts from the leak-free block (during the encryption of the nonce) is not an issue here. Attacks, such as FTA [SBR+20], can work on the leak-free block (during the encryption of the nonce) without ciphertexts only based on the information whether the computation is faulty or not. Even for nonce respecting use-cases, SIFA/SFA remains feasible [DMMP18]. Finally, the leakage from the output of the leak-free components (which is allowed) also enables combined attacks [SBJ+21]. Therefore, it is evident that the leak-free components must include FA countermeasures.

4.2 Generic Attacks on Integrity Without Secret Recovery

A crucial question is, therefore, can integrity be violated even without extracting the long-term secrets? As we shall describe in this subsection, the answer to this question is affirmative.

Our first set of attacks consider the faults to be injected at the inputs of the AEAD. The attacks are described as follows:

1. Query $\mathcal{AE}.\mathsf{Enc}$ with the tuple $(K, N, A, M; r)$.
2. Inject a fault $\mathcal{F}$ in $\mathcal{AE}.\mathsf{Enc}$ with $\mathcal{F}.v_i := M$ (resp. $\mathcal{F}.v_i := N$, $\mathcal{F}.v_i := A$, $\mathcal{F}.v_i := r$), $\mathcal{F}.mod_i \in \mathsf{fix} \cup \mathsf{diff} \cup \mathsf{rand}$, $\mathcal{F}.t_i$ transient/persistent, $0 \leq \mathcal{F}.w_i \leq |\mathcal{F}.v_i|$ and $\mathcal{F}.nf = 1$. Without loss of generality, we define a fault mask $\Delta_i := \mathcal{F}.v_i \oplus v_i^{'}$. For $\mathcal{F}.v_i := M$, we have $v_i^{'} := M^{'} = M \oplus \Delta_i$ (resp. $N \oplus \Delta_i$, $A \oplus \Delta_i$, or $r \oplus \Delta_i$).
3. $\mathcal{AE}.\mathsf{Enc}$ returns $C^{'}$, which corresponds to $M^{'} := M \oplus \Delta_i$ (or $N^{'} := N \oplus \Delta_i$, $A^{'} := A \oplus \Delta_i$, $r^{'} := r \oplus \Delta_i$).
4. The adversary uses $(N, A, C^{'})$ (resp. $(N^{'}, A, C^{'})$ or $(N, A^{'}, C^{'})$) as forgery.

One should note that the injection on the message M is considered trivial by any integrity game, as the adversary has to repeat $(N, A, C^{'})$ as a forgery, which has been seen by the encryption oracle, and listed as trivial query by default. The same is true for a fault injection on r. However, if the fault is injected on the nonce or AD, the adversary can fool the traditional game by using $(N^{'}, A, C^{'})$ (or $(N, A^{'}, C^{'})$) as a forgery, assuming the encryption oracle has only seen N and A (and not $N^{'}$, and $A^{'}$) paired with $C^{'}$. Such issue with traditional security games is natural as these games do not consider faults. We resolve this issue by augmenting the security game considering faults in our game definition in Sec. 7. Nevertheless, as already pointed out in the introduction, all these attacks are considered generic (corrupting the inputs) and their scope is beyond the AEAD security. A relevant question is, can there be a non-trivial attack on AEADs without performing an input fault or long-term key recovery? Next section shows that it is indeed feasible.

5 The Decoupling Attack

In this section, we present the decoupling attack. For the sake of illustration, we consider the leakage resilient scheme TEDT (ref. Fig. 1(b)). However, the attack applies to most of the existing schemes. The following attack utilizes the Encrypt-then-MAC structure of TEDT. The attack can be described as follows:

1. Query $\mathcal{AE}.\mathsf{Enc}$ with the tuple $(K, N, A, M; r)$.
2. The fault is injected at the input of the MAC operation corrupting the ciphertext. More precisely, the input to the MAC is corrupted to $(N, A, C \oplus \Delta_i, PK)$ from (N, A, C, PK). The original ciphertext, however, remains unchanged.
3. $\mathcal{AE}.\mathsf{Enc}$ returns $(C||\tau)$, where $\tau = MAC(C \oplus \Delta_i)$.
4. The adversary modifies $(C||\tau)$ to $(C \oplus \Delta_i||\tau)$, which is a valid forgery.

The proposed attack also works if the fault model is rand. However, it becomes probabilistic and remains practical for small fault widths. The traditional integrity game ($\mathsf{INT\text{-}CTXT}$) or its leakage-enhanced versions cannot deal with the decoupling attack. This is because the ciphertext coming out of the encryption module, in this case, is an invalid one, and the adversary can make it valid by some offline computation. Moreover, this attack strongly depends on the structure of the AEAD scheme and should, therefore, be

handled within the scope of AEAD. In the next subsection, we show that the fault resilient scheme proposed in [FG20] is also vulnerable to this decoupling attack. We shall also argue that the attack is not limited to linear encryption schemes.

5.1 Issues with Existing Fault Resilient AEAD Schemes

In this section, we show that the decoupling attack also works on the fault resilient scheme SIV$ [FG20]. The security game corresponding to [FG20] is presented in Table 11 in Appendix E (we do not need this to describe the attack). The SIV$ construction is described in Table 3, which uses a PRF for generation of an IV. The IV itself works as a tag and is also utilized to seed the encryption operation. The main difference of SIV$ from classical SIV is the incorporation of a random value r for the computation of the PRF. However, this simple trick does not ensure integrity. It is worth noting that the attacks designated as generic by us are also excluded by the game in [FG20].

Table 3: The SIV$ Scheme

SIV$	
KGen: 1: $(K_1, K_2) \xleftarrow{\$} \mathcal{K}$ 2: **return** (K_1, K_2) $\mathsf{Enc}_K^{\mathsf{SIV\$}}(N, A, M; r)$: 1: $(K_1, K_2) \leftarrow K$ 2: $IV \leftarrow \mathsf{PRF}(K_1, N, A, M; r)$ 3: $C \leftarrow \mathsf{Enc}(K_2, r\|\|M, IV)$ 4: **return** (IV, C)	$\mathsf{Dec}_K^{\mathsf{SIV\$}}(N, A, C^{'})$: 1: $(K_1, K_2) \leftarrow K$ 2: $(IV, C) \leftarrow C^{'}$ 3: $r\|\|M \leftarrow \mathsf{Dec}(K_2, C, IV)$ 4: $IV^{''} \leftarrow \mathsf{PRF}(K_1, N, A, M, r)$ 5: **if** $IV = IV^{''}$ 6: **return** M 7: **else**: 8: **return** $\perp$

Let us consider a fault in the message M at the input of the PRF step. However, M remains uncorrupted during the encryption. This can be ensured by inserting a transient fault while reading the message from the memory for the MAC step. Without loss of generality, we consider a differential fault in this case, although the results are not limited to this fault model. After injecting this fault, the IV (let us denote the faulty IV as $IV^{'}$) corresponds to the message $M \oplus \Delta_i$, while the ciphertext $C^{'}$ corresponds to M. Indeed this is an invalid output in the sense that it cannot be decrypted and verified successfully. However, if the encryption operation of SIV$ is linear (that is, if it generates a pseudorandom string that is XOR-ed with the message to encrypt it), the adversary can create a valid forgery from this by computing $C^{''} = C^{'} \oplus \Delta_i$. $(IV^{'}, C^{''})$ corresponds to the message $M \oplus \Delta_i$.

The linearity assumption on the encryption can be relaxed in certain cases. As a pathological example, let us consider an instantiation of SIV$ with CBC encryption. Instead of changing the value of the message M, we can drop a few message blocks during the PRF computation. This can be achieved by faulting the message-length parameter (which is a common input in many practical AEAD syntax) with a data fault, or by injecting a control fault to abort some loops before their completion. In this case, the attack works beyond linear encryptions, provided it encrypts the message block-wise. In essence, the adversary can generate the encryption of a truncated message as a valid forgery. However, applicability of such truncation attacks depends upon the scheme under consideration. For example, it can be prevented if the message-length parameter is also encrypted with each message block. The decoupling attack, in general, work against most common AEAD modes including the single-pass ones. A detailed discussion on this is provided in Appendix A (supp. material).

Table 4: The frPRF Oracles

frPRF	
Real World	Ideal World
INIT 1: $K \xleftarrow{\$} \mathcal{K}$ 2: $d \leftarrow 0$ 3: $\mathcal{S}_{flt} \leftarrow \emptyset$ 4: $\mathcal{S} \leftarrow \emptyset$ $\mathsf{PRF}^f_K(M, \mathcal{F})$ 1: **if** $d = 1$ 2: **return** $\perp$ 3: $n_{\mathsf{pre}} \leftarrow 0$ 4: $(\mathcal{M}_{flt}, X) \leftarrow \mathsf{Fault}(F_K(M), \mathcal{F})$ 5: **for** $Y \in \mathcal{M}_{flt}$ 6: **if** $X = F_K(Y) \ \wedge \ (Y, X) \notin \mathcal{S}_{flt}$ 7: $\mathcal{S}_{flt} \leftarrow \mathcal{S}_{flt} \cup \{(Y, X)\}$ 8: $n_{\mathsf{pre}} \leftarrow n_{\mathsf{pre}} + 1$ 9: **return** X $\mathsf{PRF}_K(M)$ 1: **if** $\exists (M, X), \ s.t. \ (M, X) \in \mathcal{S}_{flt} \cup \mathcal{S}$ 2: **return** $\perp$ 3: $X \leftarrow F_K(M)$ 4: $\mathcal{S} \leftarrow \mathcal{S} \cup \{(M, X)\}$ 5: $d \leftarrow 1$ 6: **return** X	INIT 1: $K \xleftarrow{\$} \mathcal{K}$ 2: $d \leftarrow 0$ 3: $\mathcal{S}_{flt} \leftarrow \emptyset$ 4: $\mathcal{S} \leftarrow \emptyset$ $\mathsf{PRF}^f_K(M, \mathcal{F})$ 1: **if** $d = 1$ 2: **return** $\perp$ 3: $n_{\mathsf{pre}} \leftarrow 0$ 4: $(\mathcal{M}_{flt}, X) \leftarrow \mathsf{Fault}(F_K(M), \mathcal{F})$ 5: **for** $Y \in \mathcal{M}_{flt}$ 6: **if** $X = F_K(Y) \ \wedge \ (Y, X) \notin \mathcal{S}_{flt}$ 7: $\mathcal{S}_{flt} \leftarrow \mathcal{S}_{flt} \cup \{(Y, X)\}$ 8: $n_{\mathsf{pre}} \leftarrow n_{\mathsf{pre}} + 1$ 9: **if** $n_{\mathsf{pre}} > 1$ 10: **return** $\perp$ 11: **return** X $\mathsf{RF}(M)$ 1: **if** $\exists (M, X), \ s.t. \ (M, X) \in \mathcal{S}_{flt} \cup \mathcal{S}$ 2: **return** $\perp$ 3: $X \xleftarrow{\$} \{0,1\}^{\lvert X \rvert}$ 4: $\mathcal{S} \leftarrow \mathcal{S} \cup \{(M, X)\}$ 5: $d \leftarrow 1$ 6: **return** X

6 Fault Resilient PRF and MAC

We begin with defining a new primitive in this section called fault-resilient PRF or frPRF. We also construct hash-then-frPRF, a fault-resilient MAC (frMAC) using the frPRF as a base primitive. These constructions will be utilized for constructing a fault-resilient AEAD.

6.1 Permissible Fault Models

Our fault modelling in Sec. 3 covers possible fault scenarios. However, we only claim security against a subset of these models. We begin by restricting the adversary to inject only a single fault (i.e. $nf = 1$ in $\mathcal{F}$; the subscript i is also dropped) during the entire encryption operation which is practical for most devices. Further, we limit the adversary to differential faults only[5]. Regarding the faultable variables, we assume that both data and a subset of control and constant faults are feasible. More precisely, at the mode-level in an AEAD pseudocode, control faults may corrupt a loop counter, or a loop-decision variable for the loops running over the message or ciphertext lengths. control faults may also happen for branch-decision (e.g., if-else) variables, if any. Likewise, constant faults are allowed for the used public constants. We note that other than mode-level, both control and constant faults can happen at a primitive-level, where the block cipher/TBC/permutation computations are corrupted. control faults may also cause skip of an entire block cipher/TBC/permutation function call at mode-level[6]. However, considering such faults would require concrete instantiations of these primitives. We leave this as a future work. We further assume that there are certain components, for which fault injection either results in a correct outcome, a $\perp$, or uniformly random outcome. Such components are denoted as "fault-free components". They can be realized by incorporating primitive-level FA countermeasures. Finally, leakage of some secrets is, allowed by means of information leaking differential faults except from the fault-free components.

[5]While we note that fix faults are feasible, they require extremely powerful injection setups.
[6]We consider a very limited subset of control faults in the form of length-modifying attacks.

6.2 Modelling Faults in Idealized Primitives

One of the challenges of addressing fault resilience is modelling faults in idealized primitives, *e.g.*, random oracles and random functions. These primitives are monolithic in nature; they do not have an implementation that can be faulted. On the other hand, these constructions are always used in a certain context, mainly to be compared with a real-world primitive. Such real-world primitive has an implementation description. Hence, we use this real-world primitive to simulate a fault behaviour that we can impose on the idealized primitive. For primitives keyed with a long term key, we assume that faults are always performed on the real-world primitive but the challenge queries in the ideal world are performed on the ideal-world primitive. For public functions, we use the difference between the faulty and non-faulty executions of the real-world function to deduce the output fault difference in the ideal-world primitive.

6.3 Fault-Resilient PRF

A fault-resilient PRF is a PRF that comes with a second oracle $\mathsf{PRF}_K^f(M, \mathcal{F})$, which we call a faulty PRF oracle. According a given fault model, the adversary can induce faults during the execution of PRF_K^f. A (q_f, q_m, t)-adversary against the frprf security of a PRF F is an adversary $\mathcal{A}$ that makes q_f faulty queries PRF_K^f and q_m queries to PRF_K and runs in time t. Table 4 depicts the PRF_K and PRF_K^f oracles. In Table 4, we separate $\mathcal{S}$ and $\mathcal{S}_{flt}$ to capture the resilience notion. $\mathcal{S}$ is the list of queries to the non-faulty oracle and $\mathcal{S}_{flt}$ is the list of queries to the faulty oracle. However, $\mathcal{S}_{flt}$ can also include non-faulty executions. If the $\mathsf{Fault}(F_K(M), \mathcal{F})$ oracle is called with $\mathcal{F}.mod = \mathsf{nof}$, it outputs $F_K(M)$. $\mathsf{Fault}(F_K(M), \mathcal{F})$ also returns the set $\mathcal{M}_{flt}$ which contains all the values of M that appear during the execution, including faulty values. Note that the faulty oracle keeps track of the number of valid input/output pairs (or pre-images) that are leaked from the faulty implementation as n_{pre}. In the ideal world, if $n_{\mathsf{pre}} > 1$, the oracle returns $\perp$. Hence, if an implementation leaks more than one input/output pairs for any call, the two worlds can be easily distinguished. An adversary against frprf security of the PRF_K must perform all PRF_K^f queries before querying PRF_K. Hence, at a given query to PRF_K, the adversary is considered to have performed q_f queries to PRF_K^f followed by $q_m - 1$ queries to PRF_K. For a PRF outputting n bits, the adversary aims to distinguish the last q_m outputs from a random string of length nq_m. Let n^i_{pre} be the number of pre-images for faulty query i and $\mathcal{Q}_f(\mathcal{A}) = \{1, \cdots q_f\}$ if $\mathcal{A}$ makes q_f faulty queries. The frprf advantage is given by:

$$\mathbf{Adv}^{\mathsf{frprf}}_{F,\mathsf{Fault}}(q_f, q_m, t) \leq \max_{\mathcal{A}} \Pr[\exists i \in \mathcal{Q}_f(\mathcal{A}) \; s.t. \; n^i_{\mathsf{pre}} > 1] +$$

$$\max_{\mathcal{A}} |\Pr[K \xleftarrow{\$} \mathcal{K} : 1 \leftarrow \mathcal{A}^{\mathsf{PRF}_K, \mathsf{PRF}_K^f} | \forall 1 < i < q_f, n^i_{\mathsf{pre}} \leq 1] -$$

$$\Pr[K \xleftarrow{\$} \mathcal{K} : 1 \leftarrow \mathcal{A}^{\mathsf{RF}, \mathsf{PRF}_K^f} | \forall 1 < i < q_f, n^i_{\mathsf{pre}} \leq 1]|.$$

This bound follows from a simple hybrid argument. The first term bounds the advantage of distinguishing the real-world game from a similar transitional game where the faulty oracle is replaced by the ideal world faulty oracle (it returns $\perp$ when more than 1 pre-image is leaked in one query). The second term bounds the advantage of distinguishing the transitional game from the ideal world game. We call F_K a fault-resilient PRF, if $\mathbf{Adv}^{\mathsf{frprf}}_F(q_f, q_m, t)$ is negligible for all adversaries that do not query PRF_K with a trivial query, such that q_f and q_m are polynomial in some security parameter. Trivial queries are defined according to the set $\mathcal{S} \cup \mathcal{S}_{flt}$ in Table 4.

It is crucial to note that the fault-resilient PRF must perform all faulty (and some fault-free) queries before making the challenge queries. We do not claim security for faults in post-challenge queries. Without loss of generality, let us consider a challenge

query as M_c for which the frPRF oracle returns X (it is interacting with PRF_K). Next, in a post-challenge query, the adversary queries $M_c^{'}$, and faults it to M_c (and therefore, queries PRF_K^f). In this case, the adversary will be able to distinguish the ideal world from the real world, using the PRF_K^f outcome. However, gaining information about future queries are not possible due to the structure of the game. We leave the security for faulty post-challenge queries an open issue at this point, and only claim security for faults at pre-challenge queries. One intuition is to use a randomized PRF to overcome this issue. However, since our construction is based on a deterministic PRF, we restrict ourselves to this model. The reasoning for this approach is that, while it might be tempting to consider a randomized PRF, such definition is not clear and leads to issues in our assumption and also in practice. The security of our construction relies on being able to build and test the deterministic frPRF in practice. For example, it can be built from a block cipher with fault countermeasure (*e.g.* duplication or other forms of redundancy). This can be easily tested, as we simply require two simple properties; the block cipher is secure, and fault-based key recovery attacks on the implementation are not possible. If the PRF is randomized then additional and unclear testing would be required for the PRF and this would lead to stronger assumptions. However, even restricting the faults "pre-challenge" covers several practical situations where the adversary only has access to the device for a limited time and it wants to disrupt future queries exploiting that access. Our model also differentiates between a classical adversary ($q_f = 0$) and an adversary who performs non-faulty queries during the training phase (the first q_f queries where $q_f \neq 0$). The difference is in the ideal world, where in the latter case, the first q_f queries always use the real-world oracle, while in the classical case all the queries will be random an independent. While this can be considered a limitation in some analyses, we see this as an inherent feature from the two-phase security notion we propose, and in our view it captures the concept of an adversary that has access to the implementation for a training phase before issuing challenge queries. We leave it as future work whether stronger security notions are better at capturing real life scenarios or not and whether a deterministic fault-resilient PRF can be built in those stronger models.

6.4 Modelling Faults in a Hash Function

A hash function family H_s is $(t, \epsilon_{\mathsf{cr}})$-collision resistant with faults if for every adversary $\mathcal{A}$ that runs in at most time t, the probability that $\mathcal{A}(s)$ outputs a pair of inputs $(M_0, M_1) \in (\{0,1\}^*)^2$, such that $H_s(M_0) = H_s(M_1)$ and $M_0 \neq M_1$, is bounded by ϵ_{cr}, with $s \xleftarrow{\$} \mathcal{HK}$ ($\mathcal{HK} := \{0,1\}^{\mathsf{hklen}}$, where hklen is the key length for the hash).

$$Pr[s \xleftarrow{\$} \mathcal{HK}, \mathcal{A}(s) \rightarrow (M_0, M_1) \in (\{0,1\}^*)^2 \; s.t. \; M_0 \neq M_1, H_s(M_0) = H_s(M_1)] \leq \epsilon_{\mathsf{cr}}.$$

Here $\mathcal{A}$ can also query faulty oracle $\mathsf{Fault}(H(\cdot), \mathcal{F})$, which we sometimes refer to as $H^f(\cdot, \mathcal{F})$ for simplicity, but a collision on H_s^f is not considered valid.

Observation. *A $(t, \epsilon_{\mathsf{cr}})$-collision resistant hash function is also a $(t, \epsilon_{\mathsf{cr}})$-collision resistant in the presence of faults for the non-faulty queries.*

Since H_s is a public function, the adversary can find a collision by simulating the hash function themselves, with or without faults. However, finding a collision on a faulted version (H_s^f) does not imply a collision on the non-faulty H_s, as these two are different public functions, and collisions on H_s^f are unlikely to lead to a collision on H_s. For simplicity, from now we omit the suffix "s" and use H for hash function.

When the adversary induces a fault in an execution of H, one of following three cases may happen: 1) **Ineffective fault:** the fault is ineffective. In this case $H(M) = H^f(M)$; 2) **Predictable-preimage fault:** the fault is effective, and we get $H^f(M) = H(M^{'})$ for

which $M^{'}$ is easy to predict; 3) **Unpredictable-preimage fault:** The fault is effective, and we get $H^f(M) = X$ where no pre-image is known for X. In order to capture this formally, we consider a faulty hash oracle $H^f(M, \mathcal{F})$ that works as follows: it takes as input a message M and fault specification $\mathcal{F}$, then returns (M^f, h) such that $H(M^f) = h$ or $M^f = \perp$. In case of ineffective fault, $M^f = M$. Finally, since H is a public function, attacks such as key recovery or state recovery are trivial and not useful.

Since the adversary will be able to inject differential faults in any location inside the hash function (including its inputs and outputs), we need its state to be uniformly distributed. Hence, we need to perform the analysis in the random oracle model. We also need to define what exactly do we mean when we refer to a *"fault-resilient"* random oracle. Note that the frRO is a theoretical construction in nature. We assume its existence and conjecture that if a hash function is indifferentiable from a random oracle, then its faulty version is also indifferentiable from a frRO. Showing that such property is satisfied for a specific hash function (*e.g.* sponge-based hash functions) is one of our future research questions and is out of scope of this paper.

Fault-resilient Random Oracle (frRO):

In our constructions, we rely on the fact that the hash function used behaves like a random oracle even when *differential* faults can be induced. A random oracle is fault resilient if:

1. When the input is randomized, the internal state and output are also randomized and unpredictable. Injecting a differential fault leads to an unpredictable state.

2. Regardless of previous faulty queries, when an input is fresh (was not queried before) and random, the internal states and output are unpredictable.

In order to formalize these properties, we define a frRO.

Definition 4. A fault-resilient random oracle frRO is an oracle that implements a random function $\mathsf{RO} : \mathcal{M} \times \{0,1\}^{|r|} \rightarrow \{0,1\}^{|h|}$, where RO is sampled uniformly from the set of all random functions of the same domain and range. The frRO is a randomized oracle that selects a new random salt $r \in \{0,1\}^{|r|}$ in each invocation. Table 5 depicts this oracle.

Since a random oracle is a monolithic idealized primitive, we cannot inject faults except at its inputs or outputs. As long as $\mathcal{F}.mod \in \{\mathsf{nof}, \mathsf{diff}, \mathsf{rand}\}$ and $\mathcal{R}_{flt}$ is collision free (*i.e.* bad is never asserted), the frRO behaves as a random oracle, since a uniformly sampled output will be generated for each invocation, and the fault can change the output with a specific difference. In practice, a hash function has an implementation that can also be faulted at any intermediate operation. In this case, the output is not simply, $H(r\|x) \oplus \Delta$, but it can be a value related to $H(r\|x)$ and the fault location and value. However, since r is returned with the hash value, the adversary can compute $H(r\|x)$ and deduce Δ. The frRO uses the implementation of the hash function to also deduce the output difference using both the faulty and non-faulty output. This is depicted in lines 23-25 in Table 5. The definition of the frRO is parametrized by which hash function H is being evaluated. An adversary tries to distinguish between the case of a true random oracle (ideal world) and the real world depicted in Table 5. The distinguishing advantage is given by

$$\mathbf{Adv}^{\mathsf{frRO}}_{\mathsf{RO},H,\mathsf{Fault}}(\mathcal{A}) \stackrel{\text{def}}{=} |[1 \leftarrow \mathcal{A}^{\mathsf{frRO}^{\mathsf{f}},\mathsf{frRO}}] - [1 \leftarrow \mathcal{A}^{\mathsf{RO},\mathsf{RO}}]|$$

where RO is the random oracle that takes as inputs r and M and returns a random output.

Theorem 1. *As long as the* bad *event is never set, then* frRO *is indistinguishable from a fault-free random oracle.*

Proof. Let q be the number of calls to frRO and q_f be the number of calls to frRO^f. Since each query adds at most 2 values of r to $\mathcal{R}_{flt}$, the probability of setting $\mathtt{bad}$ is bound by

$$\frac{\binom{2q_f+q}{2}}{2^{|r|}}.$$

If $\mathtt{bad}$ is never set, then the output of each query is uniformly distributed, which is indistiguishable from the output of a fault-free random oracle. □

Conjecture 1. *If a hash function H is indifferentiable from a random oracle, then its faulty implementation with differential faults is indifferentiable from an* frRO.

It is clear that in case the fault is injected inside the hash function implementation, then the random oracle and the hash function do not behave in the same way. However, the description of frRO is consistent and if $\mathtt{bad}$ is never set, the output of the hash function (under the random oracle model) is uniformly distributed. A real-world faulty hash function is similar to Table 5, except that RO is replaced with H. If the hash function is constructed as an iterative function using a compression function and the faults can only be injected outside the compression function, then the conjecture above is straightforward, as an adversary against a fault-free random oracle can simulate the faulty queries using the compression function oracle. However, if faults can also be injected inside the compression function then a more careful analysis is needed.

Table 5: frRO using lazy sampling.

The frRO Oracle

INIT
for $y \in \{0,1\}^*$
  $\mathsf{RO}(y) \xleftarrow{\$} \perp$
$\mathcal{R}_{flt} \leftarrow \emptyset$

$\mathsf{frRO}^f(x; r, \mathcal{F})$
if $r \in \mathcal{R}_{flt}$ **then** $\mathtt{bad}$
$\mathcal{R}_{flt} \leftarrow \mathcal{R}_{flt} \cup \{r\}$
if $\mathcal{F}.mod = \mathsf{nof}$
  if $\mathsf{RO}(r\|x) = \perp$
    $\mathsf{RO}(r\|x) \xleftarrow{\$} \{0,1\}^{|h|}$
  $Z \leftarrow \mathsf{RO}(r\|x)$
else if $\mathcal{F}.v = r$
  $r \leftarrow r \oplus \Delta$
  if $r \in \mathcal{R}_{flt}$ **then** $\mathtt{bad}$
  $\mathcal{R}_{flt} \leftarrow \mathcal{R}_{flt} \cup \{r\}$
  if $\mathsf{RO}(r\|x) = \perp$
    $\mathsf{RO}(r\|x) \xleftarrow{\$} \{0,1\}^{|h|}$
  $Z \leftarrow \mathsf{RO}(r\|x)$
else
  if $\mathsf{RO}(r\|x) = \perp$
    $\mathsf{RO}(r\|x) \xleftarrow{\$} \{0,1\}^{|h|}$
  $(M^f, h) \leftarrow \mathsf{Fault}(H(r\|x), \mathcal{F})$
  $\Delta \leftarrow H(r\|x) \oplus h$
  $Z \leftarrow \mathsf{RO}(r\|x) \oplus \Delta$
return (r, M^f, Z)

$\mathsf{frRO}(x; r)$
  $\mathcal{F}.mod \leftarrow \mathsf{nof}$
  return $\mathsf{frRO}^f(x; r, \mathcal{F})$

6.5 Fault-Resilient Message Authentication Code (frMAC)

For $K \in \mathcal{K}$, let $\mathsf{Mac}_K(M)$ be the MAC oracle which takes as input a message $M \in M$ and returns $F_K(M)$, and $\mathsf{Verify}_K(M)$ be the verification oracle that takes $(M, \tau) \in \times\mathcal{M} \times \mathcal{T}$ as input and returns 1 if $F_K(M) = \tau$ and 0, otherwise. We assume that an adversary makes queries to the two oracles Mac_K and Verify_K for a secret key $K \in \mathcal{K}$. Besides, we also define the faulty MAC oracle $\mathsf{Mac}_K^f(M, \mathcal{F})$, which takes as input a message M and fault specifications $\mathcal{F}$ and returns (M^f, τ), such that $\mathsf{Mac}_K(M^f) = \tau$ or $M^f = \perp$. We define the frMAC game as in Table 6, similar to the frPRF game in Table 4.

A (q_f, q_m, q_v, t)-adversary against the frmac security of F is an adversary $\mathcal{A}$ with oracle access to Mac_K^f, Mac_K and Verify_K, making at most q_f queries to its Mac_K^f oracle followed by at most q_m and q_v queries to its Mac_K and Verify_K oracles, respectively, running in time at most t, and returning a decision bit. Similar to mac, we represent the frmac security using the advantage of an adversary trying to distinguish the real world

Table 6: The frMAC Oracles

frMAC

Real World	Ideal World
INIT $K \xleftarrow{\$} \mathcal{K}$ $d \leftarrow 0$ $\mathcal{S}_{flt} \leftarrow \emptyset$ $\mathcal{S} \leftarrow \emptyset$	**INIT** $K \xleftarrow{\$} \mathcal{K}$ $d \leftarrow 0$ $\mathcal{S}_{flt} \leftarrow \emptyset$ $\mathcal{S} \leftarrow \emptyset$
$\mathsf{Mac}^f_K(M, \mathcal{F})$ **if** $d = 1$ **return** $\perp$ $n_{\mathsf{pre}} \leftarrow 0$ $(\mathcal{M}_{flt}, X) \leftarrow \mathsf{Fault}(F_K(M), \mathcal{F})$ **for** $Y \in \mathcal{M}_{flt}$ **if** $\mathsf{Verify}_K(Y, X) \neq 0 \ \wedge \ (Y, X) \notin \mathcal{S}_{flt}$ $\mathcal{S}_{flt} \leftarrow \mathcal{S}_{flt} \cup \{(Y, X)\}$ $n_{\mathsf{pre}} \leftarrow n_{\mathsf{pre}} + 1$ **return** X	$\mathsf{Mac}^f_K(M, \mathcal{F})$ **if** $d = 1$ **return** $\perp$ $n_{\mathsf{pre}} \leftarrow 0$ $(\mathcal{M}_{flt}, X) \leftarrow \mathsf{Fault}(F_K(M), \mathcal{F})$ **for** $Y \in \mathcal{M}_{flt}$ **if** $\mathsf{Verify}_K(Y, X) \neq 0 \ \wedge \ (Y, X) \notin \mathcal{S}_{flt}$ $\mathcal{S}_{flt} \leftarrow \mathcal{S}_{flt} \cup \{(Y, X)\}$ $n_{\mathsf{pre}} \leftarrow n_{\mathsf{pre}} + 1$ **if** $n_{\mathsf{pre}} > 1$ **return** $\perp$ **return** X
$\mathsf{Mac}_K(M)$ **if** $\exists (M, X), \ s.t. \ (M, X) \in \mathcal{S}_{flt} \cup \mathcal{S}$ **return** $\perp$ $X \leftarrow F_K(M)$ $\mathcal{S} \leftarrow \mathcal{S} \cup \{(M, X)\}$ $d \leftarrow 1$ **return** X	$\mathsf{RF}(M)$ **if** $\exists (M, X), \ s.t. \ (M, X) \in \mathcal{S}_{flt} \cup \mathcal{S}$ **return** $\perp$ $X \xleftarrow{\$} \{0,1\}^{\lvert X \rvert}$ $\mathcal{S} \leftarrow \mathcal{S} \cup \{(M, X)\}$ $d \leftarrow 1$ **return** X
$\mathsf{Verify}_K(M, T)$ **if** $\exists (M, X), \ s.t. \ (M, X) \in \mathcal{S}_{flt} \cup \mathcal{S}$ **return** 1 $X \leftarrow F_K(M)$ **if** $X = T$ **return** 1 **else** **return** 0	$\mathsf{Rej}(M)$ **if** $\exists (M, X), \ s.t. \ (M, X) \in \mathcal{S}_{flt} \cup \mathcal{S}$ **return** 1 **return** 0

$(\mathsf{Mac}^f_K, \mathsf{Mac}_K, \mathsf{Verify}_K)$ and the ideal world. The ideal world oracles are $(\mathsf{Mac}^f_K, \mathsf{RF}, \mathsf{Rej})$, where RF returns an independent random value (instantiating a truly random function) and Rej returns 0 for every verification query. Then,

$$\mathbf{Adv}^{\mathsf{frmac}}_{F,\mathsf{Fault}}(q_f, q_m, q_v, t) \leq \max_{\mathcal{A}} | \Pr[K \xleftarrow{\$} \mathcal{K} : 1 \leftarrow \mathcal{A}^{\mathsf{Mac}^f_K, \mathsf{Mac}_K, \mathsf{Verify}_K}] -$$

$$\Pr[K \xleftarrow{\$} \mathcal{K} : 1 \leftarrow \mathcal{A}^{\mathsf{Mac}^f_K, \mathsf{RF}, \mathsf{Rej}}] |.$$

We note that when $q_v = 0$, this is equivalent to frPRF security. The adversary is not allowed to repeat queries to its Mac_K oracle, or ask for queries to its Mac_K.

6.6 Extending Fixed-length frPRF to Arbitrary-length frMAC

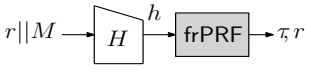

Figure 2: Fault-Resilient hash-then-PRF MAC.

Consider the hash-then-frPRF MAC, where the internal PRF is a frPRF, as depicted in Fig. 2. The frPRF can be realized with block ciphers or TBCs with fault countermeasures. We will claim that such construction is also a frMAC if M is randomized. Before that, we motivate the analysis by considering some of the attacks an adversary may try. Note that by the rules of the game in Table 6, the last q_m queries must not have appeared in the set $\mathcal{S}_{flt}$ after the first q_f queries. At a given query j, the adversary can try to:

1. Find M such that $H(r\|M) = H(r'\|M')$ for a previously queried message $r'\|M'$, potentially using either a non-faulty query, ineffective fault or predictable-preimage

fault. If $r\|M = r'\|M'$, then the attack is trivial. If $r\|M \neq r'\|M'$, then the adversary has found a collision for H.

2. Find M such that $H(r\|M) = h$ where h has appeared as the input to frPRF during at least one of the first $q_f + j - 1$ queries, using an unpredictable-preimage fault. In this case, the adversary needs to predict the value of r.

3. Find bias in τ in queries for which $H(r\|M)$ has never appeared in any of the first $q_f + j - 1$ queries. This needs the adversary to break the frprf security of base frPRF.

4. The adversary can try to find a near-collision such that $H(r\|M) = H(r'\|M') \oplus \Delta$ where Δ is a fault they can inject. However, to use such fault, the adversary needs to predict when r and/or r' will appear. Since the random value is unpredictable, this attack is not feasible.

In all cases, the adversary needs to break a strong security notion on one of the primitives or predict a random value, and the random values are shared with the verification oracle in a secure way, beyond the scope of the construction.

Table 7: The hash-then-frPRF Oracles

Hash-then-frPRF Oracles

INIT
$K \xleftarrow{\$} \mathcal{K}$
$d \leftarrow 0$
$\mathcal{S}_{flt} \leftarrow \emptyset$
$\mathcal{S} \leftarrow \emptyset$
$\mathcal{R}_{flt} \leftarrow \emptyset$
$\mathcal{R}_p \leftarrow \emptyset$
return s

$\mathsf{Mac}^f_K(M; r, \mathcal{F})$
if $d = 1$
  return $\perp$
$\mathcal{M}_{flt} \leftarrow \{r\|M\}$
$\mathcal{R}_{flt} \leftarrow \mathcal{R}_{flt} \cup \{r\}$
$r' \leftarrow r$
$n_{\mathsf{pre}} \leftarrow 0$
if $\mathcal{F}.mod = \mathsf{nof}$
  $X \leftarrow F_K(H(r\|M))$
else if $\mathcal{F}.v = r$
  $X \leftarrow F_K(H(r \oplus \Delta\|M))$ ▷ $\mathcal{F}.mod = \mathsf{diff} \vee \mathsf{rand}$
  $\mathcal{M}_{flt} \leftarrow \mathcal{M}_{flt} \cup \{r \oplus \Delta\|M\}$
else if $\mathcal{F}.v = \mathsf{hash}$
  $(r\|M^f, h) \leftarrow \mathsf{Fault}(H(r\|M), \mathcal{F})$
  $X \leftarrow F_K(h)$
  $\mathcal{M}_{flt} \leftarrow \mathcal{M}_{flt} \cup \{r\|M^f\}$
else if $\mathcal{F}.v = \mathsf{prf}$
  $X \leftarrow \mathsf{Fault}(F_K(H(r\|M)), \mathcal{F})$
for $Y \in \mathcal{M}_{flt}$
  if $\mathsf{Verify}_K(Y, X) \neq 0 \ \wedge \ (Y, X) \notin \mathcal{S}_{flt}$
    $\mathcal{S}_{flt} \leftarrow \mathcal{S}_{flt} \cup \{(Y, X)\}$
    $n_{\mathsf{pre}} \leftarrow n_{\mathsf{pre}} + 1$
if $r \neq r'$
  $\mathcal{R}_{flt} \leftarrow \mathcal{R}_{flt} \cup \{r\}$
return (r, X)

$\mathsf{Mac}_K(M; r)$
$\mathcal{R}_{flt} \leftarrow \mathcal{R}_{flt} \cup \{r\}$
$X \leftarrow F_K(H(r\|M))$
$\mathcal{S} \leftarrow \mathcal{S} \cup \{(r||M, X)\}$
$d \leftarrow 1$
return (r, X)

$\mathsf{Verify}_K(M, \tau; r)$
if $(r\|M, \tau) \in \mathcal{S}_{flt} \cup \mathcal{S}$
  return $\perp$
$X \leftarrow F_K(H(r\|M))$
if $X = \tau$
  return 1
else
  return 0

$\mathsf{Hash}(r, X)$
$\mathcal{R}_p \leftarrow \mathcal{R}_p \cup \{r\}$
return $H(X)$

Theorem 2. *If $H : \mathcal{M} \times \mathcal{HK} \to \mathcal{Z}$ is a* frRO *and F_K is a secure fault-resilient PRF against all $(q_f, q_m + q_v, t)$-adversaries, then* hash-then-frPRF *is a secure randomized fault-resilient MAC (frMAC) against all (q_f, q_m, q_v, q_p, t)-adversaries that perform differential faults, s.t.*

$$\mathbf{Adv}^{\mathtt{frmac}}_{\mathsf{htfrprf},\mathsf{Fault}}(q_f, q_m, q_v, q_p, t) \leq \mathbf{Adv}^{\mathtt{frprf}}_{F,\mathsf{Fault}}(q_f, q_m + q_v, t) + \frac{\binom{q_e}{2}}{2^{|r|}} + \frac{q_e q_p}{2^{|r|}} +$$

$$\frac{2(q_p + q_e + q_v)^2}{2^{|h|}} + \frac{2(q_e + q_v + 1)}{2^{|h|}} + \frac{2q_f q_v}{2^{|h|}} + \frac{q_v}{2^{|\tau|}}.$$

Here $q_e = 2q_f + q_m$ and q_p is the number of queries made to the last oracle Hash.

The proof of this theorem is presented in Appendix B (supp. material). Table 7 presents the oracles for hash-then-frPRF.

7 Fault-Resilient AEAD

In this section we propose our construction for a fault-resilient AEAD (frAE). One may observe that a straightforward combination of frMAC and some encryption scheme (either as "Encrypt-then-MAC", "MAC-then-Encrypt" or "MAC-and-Encrypt") is not going to work in this case. This is because the adversary can fault the input of the frMAC and enable the decoupling attack. We, therefore propose a three-pass construction.

7.1 MAC-then-Encrypt-then-MAC (MEM)

Table 8: The MAC-then-Enc-then-MAC (MEM) Scheme

MEM	
KGen: 1: $(K_1, K_2, K_3) \xleftarrow{\$} \mathcal{K}$ 2: **return** (K_1, K_2, K_3)	5: $\tau_2 \leftarrow \mathsf{Mac}_{K_3}(C; \tau_1)$ 5a: $h_2 \leftarrow H(\tau_1 \| C)$ 5b: $\tau_2 \leftarrow F^*_{K_3}(h_2)$ 6: **return** $(\tau_1 \| C \| \tau_2)$
$\mathsf{MEMEnc}_K(N, A, M; r)$: // parse $M = (M_1, M_2, \cdots, M_l)$ 1: $(K_1, K_2, K_3) \leftarrow K$ 2: $X \leftarrow \mathsf{pad}(N, A, M)$ 3: $\tau_1 \leftarrow \mathsf{Mac}_{K_1}(X; r)$ 3a: $h_1 \leftarrow H(r \| X)$ 3b: $\tau_1 \leftarrow F^*_{K_1}(h_1)$ 4: $C \leftarrow \mathsf{frEnc}(K_2, r \| M, \tau_1)$ 4a: $k_0 \leftarrow F^*_{K_2}(\tau_1)$ 4b: $C_0 \leftarrow F_{k_0}(p_b) \oplus r$ 4c: $k_i \leftarrow F_{k_{i-1}}(p_a)$ 4d: $C_i \leftarrow F_{k_i}(p_b) \oplus M_i$ 4e: $C \leftarrow (C_0, C_1, \cdots C_l)$	$\mathsf{MEMDec}_K(N, A, C)$: // parse $C = (\tau_1, C_0 \| C_1 \| \cdots \| C_l, \tau_2)$ 1: $h_2^{ch} \leftarrow H(\tau_1 \| C)$ 2: $\tau_2' \leftarrow \mathsf{F}^*_{K_3}(h_2^{ch})$ 3: **if** $\tau_2 = \tau_2'$ 4: $(r, M) \leftarrow \mathsf{frEnc}(K_2, C, \tau_1)$ 5: $X \leftarrow \mathsf{pad}(N, A, M)$ 6: $h_1^{ch} \leftarrow H(r \| X)$ 7: $\tau_1' \leftarrow \mathsf{F}^*_{K_1}(h_1^{ch})$ 8: **if** $\tau_1' = \tau_1$ 9: **return** M 10: **else return** $\perp$

Our proposal for frAE is presented in Table 8 and Fig. 3. The crux of this scheme lies in further authenticating the $(\tau_1 || C)$. Both the MACs (denoted as Mac_{K_1} and Mac_{K_3}) are constructed by applying the frMAC. The fault-resilient encryption (frEnc) is performed with PSV [PSV15] initiated with a frPRF. However, it can be abstracted as any pseudorandom bit generator seeded with a frPRF[7]. We assume distinct frPRF and hash functions for Mac_{K_1} and Mac_{K_3} and PSV enabled by different key for each frPRF[8]. The encryption operation also requires two distinct constants constants (p_a, p_b). Finally, an injective function pad is required at the input of Mac_{K_1}.

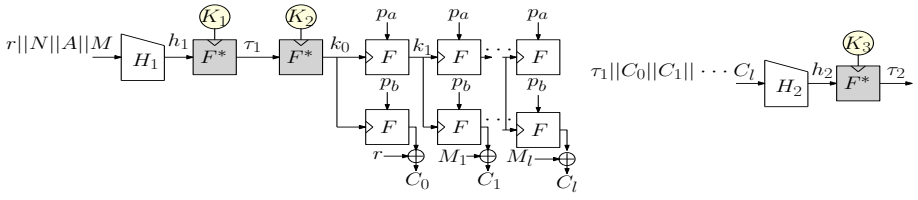

Figure 3: The MEM construction. F^* denotes the frPRFs realized with fault-free blocks.

[7]In fact, PSV can be replaced with other lightweight options such as CTR-DBRG [BK+07] keyed with frPRF. We chose PSV for its leakage-resilience so that it can be analyzed in future for combined security.

[8]For concrete constructions, one might also utilize domain separation with tweakable block ciphers to realize distinct PRFs. We assumed distinct keys for the sake of simplicity.

One should note that our construction is computationally efficient compared to the simple duplication-based schemes. We do not need to duplicate the encryption operation. Furthermore, the MACs are computed on different variables, namely the message and the ciphertext tag pairs. This further limits the scope for decoupling attacks as the MACs has to be performed in a sequential manner and none of these variables appear at the same clock cycle. A single fault adversary cannot corrupt the MACs together to create a decoupling. Interestingly, we found that not every three-pass construction has such nice properties, and many of them are vulnerable. This point is discussed in Appendix D. Our construction also bears some structural similarities with the leakage-resilient scheme SIVAT in [BMOS17]. However, SIVAT does not include frPRFs and randomness, which are essential for security against faults (to prevent fault-assisted nonce repeat, and manipulation of the hash values). Also, SIVAT does not check one of the tags at the decryption. This is crucial in FA, as otherwise, the decoupling attack will happen.

7.2 The Fault Resilient Security Game

We first define the security game for AEAD considering encryption faults ($\mathsf{Expt}^{\mathsf{frAE},b}_{\mathcal{A},\mathcal{F}}$). The game is similar to the one for frPRF – we keep a faulty encryption oracle. However, after a certain point, the adversary is only allowed to make non-faulty queries. The set of faulty queries is denoted as $\mathcal{S}_{flt}$, and the set of correct queries is denoted as $\mathcal{S}$. $\mathcal{S}_{flt} \cup \mathcal{S}$ comprises all trivial queries. During a query to Enc^f_K, the encryption oracle keeps track of the correct and faulty valuations of each corrupted variable in the set $\mathcal{V}_{flt}$. Referring to the game in Table 9, $\mathcal{V}_{flt} := \mathcal{N}_{flt} \cup \mathcal{AD}_{flt}$. Here $\mathcal{N}_{flt}$ and $\mathcal{AD}_{flt}$ denote the set of all possible (correct/faulty) valuations of nonces and ADs, respectively. For easy explanation, we directly refer to $\mathcal{N}_{flt}$ and $\mathcal{AD}_{flt}$ instead of $\mathcal{V}_{flt}$ in the game and the proof. We choose to keep track of those variables only, which take part in the decryption. Therefore, we do not maintain the faulty messages or intermediates explicitly. The reason behind such a choice is that the faulty encryption oracle here tries to decrypt (without faults) the resulting ciphertext with whatever values (correct/faulty) it has observed for the inputs (nonce and AD). The goal of this step is to only check whether or not any possible output tuple $(N^{'}, A^{'}, C)$ gets decrypted successfully. The set of such decryptable tuples is denoted by $\mathcal{I}_{vld}$, which is later added to $\mathcal{S}_{flt}$. One may note that this extra step in the encryption oracle helps us to avoid trivial queries. Additionally, we exclude the final outcome of the Enc_K as the fault injection variable, as such faults also lead to generic attacks.

7.3 Security Proof

Proof Sketch: The proof of the following theorem follows directly from the security of the frPRFs and frMACs used in the scheme, and bounding the probability of a collision on the random value r, or the first tag τ_1. Intuitively, the fault injected in the first MAC or the encryption step will be committed by the second MAC, such that τ_1 and C cannot be decoupled. On the other hand, a fault injected in the second MAC will lead to τ_2 that is decoupled from the input message, and the message corresponding to the forged tag will be unpredictable due to the randomness. The fault-resilient confidentiality of the scheme follows naturally from the resilience of the frPRFs. In order to show the security of the MEM construction, we need an additional definition.

Definition 5 (IV-based Encryption)**.** An IV-based encryption scheme $E : \mathcal{K} \times \mathcal{M} \times \mathcal{IV} \rightarrow \mathcal{C}$ is an encryption scheme that takes a message $M \in \mathcal{M}$, an initial vector $IV \in \mathcal{IV}$, and a secret key $K \in \mathcal{K}$, and returns a ciphertext $C \in \mathcal{C}$, such that $|M| = |C|$. An IV-based encryption scheme is fault-resilient against all (q_f, q_m, t)-adversaries, if for any adversary that runs in time t, and performs q_f faulty queries and q_m non-faulty queries using non-trivial IVs, the advantage of distinguishing the last q_m ciphertexts from a set of random

Table 9: The frAE Game.

$\mathsf{Expt}^{\mathsf{frAE},b}_{\mathcal{A},\mathcal{F}}$

INIT

$K \xleftarrow{\$} \mathcal{K}$
$b \xleftarrow{\$} \{0,1\}$
$d \leftarrow 0$
$\mathcal{S}_{flt} \leftarrow \emptyset$
$\mathcal{S} \leftarrow \emptyset$

FIN

$b' \leftarrow \mathcal{A}^{\mathsf{Enc}^f(\cdot),\mathsf{Enc}(\cdot),\mathsf{Dec}(\cdot)}(\mathcal{F})$
return b'

$\mathsf{Enc}^f_K(N, A, M, \mathcal{F})$

if $d = 1$
  return $\perp$
$r \xleftarrow{\$} \mathcal{R}$
$(\mathcal{N}_{flt}, \mathcal{AD}_{flt}, C) \leftarrow \mathsf{Fault}(\mathsf{MEMEnc}_K(N, A, M; r), \mathcal{F})$
$\mathcal{I}_{vld} \leftarrow \{(N', A') \in \mathcal{N}_{flt} \times \mathcal{AD}_{flt} \mid \mathsf{MEMDec}_K(N', A', C) \neq \perp \wedge (N', A', C) \notin \mathcal{S}_{flt}\}$
if $b = 1 \wedge |\mathcal{I}_{vld}| > 1$
  return $\perp$
$\mathcal{S}_{flt} \leftarrow \mathcal{S}_{flt} \cup \{(N', A', C) \mid (N', A') \in \mathcal{I}_{vld}\}$
return C

$\mathsf{Enc}_K(N, A, M)$

$r \xleftarrow{\$} \mathcal{R}$
$C \leftarrow \mathsf{MEMEnc}_K(N, A, M; r)$
$\mathcal{S} \leftarrow \mathcal{S} \cup \{(N, A, C)\}$
if $b = 1$
  $C \xleftarrow{\$} \{0,1\}^{|M|+|\tau_1|+|\tau_2|+|r|}$
$d \leftarrow 1$
return C

$\mathsf{Dec}_K(N, A, C)$

if $b = 1 \vee (N, A, C) \in \mathcal{S}_{flt} \cup \mathcal{S}$
  return $\perp$
else
  return $\mathsf{MEMDec}_K(N, A, C)$

strings of equal lengths is negligible. We define this advantage as

$$\mathbf{Adv}^{\mathtt{ivfrind}}_{E_K,\mathsf{Fault}}(q_f, q_m, t) := \max_{\mathcal{A}} |\Pr[K \xleftarrow{\$} \mathcal{K} : 1 \leftarrow \mathcal{A}^{E^f_K, E_K}] - \Pr[K \xleftarrow{\$} \mathcal{K} : 1 \leftarrow \mathcal{A}^{E^f_K, \$}]|$$

where \$ takes the (IV, M) pair and returns a random string of length $|M|$. We build a fault-resilient IV-based encryption scheme using a secure frPRF F_K, and a pseudorandom key generator KG, as: $E_K = KG(F_K(IV)) \oplus M$.

Theorem 3. *If Mac_{K_1} and Mac_{K_3} are a two independent fault-resilient MACs against all (q_f, q_m, q_v, t)-adversaries and E_{K_2} is a secure IV-and-fault-resilient IND-CPA encryption scheme against all (q_f, q_m, t)-adversaries, built using a secure frPRF and a pseudorandom key generator KG, such that $E_{K_2} = KG(F_{K2}(IV)) \oplus M$, then randomized MAC-then-Encrypt-then-MAC is a secure randomized fault-resilient authenticated encryption with associated data (frAE) against all (q_f, q_m, q_v, t)-adversaries that perform differential faults:*

$$\mathbf{Adv}^{\mathtt{frae}}_{\mathsf{MEM},\mathsf{Fault}}(q_f, q_m, q_v, q_p, t) \leq \mathbf{Adv}^{\mathtt{frmac}}_{\mathsf{Mac}_{K_1}}(q_f, q_m, q_v, q_p, t) \;+$$

$$\mathbf{Adv}^{\mathtt{frmac}}_{\mathsf{Mac}_{K_3}}(q_f, q_m, q_v, q_p, t) + \mathbf{Adv}^{\mathtt{ivfrind}}_{E_{K_2}}(q_f, q_m, t) + \frac{q_e q_v}{2^{|r|}} + \frac{\binom{q_e}{2}}{2^{|\tau_1|}} + \frac{\binom{q_e}{2}}{2^{|r|}}$$

where $q_e = 2q_f + q_m$, and q_p is the maximum number of queries made to either H_1 or H_2.

The proof of this theorem is presented in Appendix C (supp. material).

7.4 The Security of the Encryption Layer

In the security analysis and proof of the MEM construction, we assumed the middle encryption layer is resilient against IV repetition and faults. In other words, as long as the non-faulty queries use IVs that have not been used before (including faulty IVs), the outcome of these queries is indistinguishable from random strings. To construct the exact scheme, we use the PSV [PSV15] construction. Its security in this model is captured in our security claim by $\mathbf{Adv}^{\mathtt{ivfrind}}$, and it can be seen that as long as fresh values for τ_1 appear, the initial key k_0 is uniformly distributed due to the fault-resilience property of the second frPRF. Therefore, PSV fits well for encryption here.

8 Conclusion

In this paper, we take the first step to address the problem of fault-assisted integrity violations in AEADs. We examine a new attack called *decoupling attack* which can affect many AEADs including the leakage-resilient schemes and existing fault-resilient AEADs (SIV$). We then formalize the notion of fault-resilience (frPRF, frMAC, frRO, and frAE) and propose a resilient construction (MEM) to prevent fault-assisted integrity attacks.

However, there still exist several crucial open issues. For example, our proposals still cannot withstand faults in the challenge queries. While extensions for multiple faults in the encryption is feasible, it is not explored in this paper and requires significant amount of research in the future. Extending the security claims for the fix faults (mostly if it is a full-state fault) is another point, where further exploration is needed. Finally, the ideas presented can be combined with the state-of-the-art leakage-resilient schemes providing combined security. However, a formal exploration of this requires further research. A discussion on these open issues is presented in Appendix D as supplementary material.

Acknowledgment

We would like to thank Ashwin Jha and the anonymous reviewers of ToSC for their valuable comments. We would also like to thank Kazuhiko Minematsu and Damian Vizar for their comments on an earlier draft of this article. The second author would like to thank the organizers of FrisiaCrypt 2022, where a preliminary version of this work was presented. This work was supported by a joint Wallenberg AI, Autonomous Systems and Software Program-Nanyang Technological Universy (WASP-NTU) grant.

References

[BCI+20] Subhadeep Banik, Avik Chakraborti, Tetsu Iwata, Kazuhiko Minematsu, Mridul Nandi, Thomas Peyrin, Yu Sasaki, Siang Meng Sim, and Yosuke Todo. GIFT-COFB. *Cryptology ePrint Archive*, 2020.

[BDL97] Dan Boneh, Richard A DeMillo, and Richard J Lipton. On the importance of checking cryptographic protocols for faults. In *Proceedings of 15th International Conference on the Theory and Applications of Cryptographic Techniques (EUROCRYPT)*, pages 37–51. Springer, 1997.

[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.

[BK+07] Elaine B Barker, John Michael Kelsey, et al. *Recommendation for random number generation using deterministic random bit generators (revised)*. US Department of Commerce, Technology Administration, National Institute of Standards and Technology, Computer Security Division, Information Technology Laboratory, 2007.

[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 *International Conference on the Theory and Application of Cryptology and Information Security (ASIACRYPT)*, pages 693–723. Springer, 2017.

[BN00] Mihir Bellare and Chanathip Namprempre. Authenticated encryption: Relations among notions and analysis of the generic composition paradigm. In *International Conference on the Theory and Application of Cryptology and Information Security (ASIACRYPT)*, pages 531–545. Springer, 2000.

[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.

[DEG+18] Christoph Dobraunig, Maria Eichlseder, Hannes Gross, Stefan Mangard, Florian Mendel, and Robert Primas. Statistical ineffective fault attacks on masked AES with fault countermeasures. In *Proceedings of 24th International Conference on the Theory and Application of Cryptology and Information Security (ASIACRYPT)*, pages 315–342. Springer, 2018.

[DEK+16] Christoph Dobraunig, Maria Eichlseder, Thomas Korak, Victor Lomné, and Florian Mendel. Statistical fault attacks on nonce-based authenticated encryption schemes. In *International Conference on the Theory and Application of Cryptology and Information Security (ASIACRYPT)*, pages 369–395. Springer, 2016.

[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+20] C.E Dobraunig, Maria Eichlseder, Stefan Mangard, Florian Mendel, B.J.M Mennink, Robert Primas, and Thomas Unterluggauer. ISAP v2. 0. 2020.

[DEMS21] Christoph Dobraunig, Maria Eichlseder, Florian Mendel, and Martin Schläffer. Ascon v1. 2: Lightweight authenticated encryption and hashing. *Journal of Cryptology*, 34(3):1–42, 2021.

[DMMP18] Christoph Dobraunig, Stefan Mangard, Florian Mendel, and Robert Primas. Fault attacks on nonce-based authenticated encryption: Application to Keyak and Ketje. In *International Conference on Selected Areas in Cryptography (SAC)*, pages 257–277. Springer, 2018.

[DMP20] Christoph Dobraunig, Bart Mennink, and Robert Primas. Leakage and tamper resilient permutation-based cryptography. *Cryptology ePrint Archive*, 2020.

[DRSA16] Prakash Dey, Raghvendra Singh Rohit, Santanu Sarkar, and Avishek Adhikari. Differential fault analysis on Tiaoxin and AEGIS family of ciphers. In *International Symposium on Security in Computing and Communication*, pages 74–86. Springer, 2016.

[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*, page 56–84, San Francisco, CA, USA, 2020. Springer.

[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.

[GYTS14] Nahid Farhady Ghalaty, Bilgiday Yuce, Mostafa Taha, and Patrick Schaumont. Differential fault intensity analysis. In *Proceedings of 11th IEEE Workshop on Fault Diagnosis and Tolerance in Cryptography (FDTC)*, pages 49–58. IEEE, 2014.

[IMG+14] Tetsu Iwata, Kazuhiko Minematsu, Jian Guo, Sumio Morioka, and Eita Kobayashi. SILC: simple lightweight CFB. *CAESAR submission*, 2014.

[IMG+15] Tetsu Iwata, Kazuhiko Minematsu, Jian Guo, Sumio Morioka, and Eita Kobayashi. Re: Fault based forgery on CLOC and SILC. https://groups.google.com/g/crypto-competitions/c/_qxORmqcSrY, 2015.

[IMGM14] Tetsu Iwata, Kazuhiko Minematsu, Jian Guo, and Sumio Morioka. Cloc: authenticated encryption for short input. In *International Workshop on Fast Software Encryption*, pages 149–167. Springer, 2014.

[JSP20] Amit Jana, Dhiman Saha, and Goutam Paul. Differential fault analysis of NORX. In *Proceedings of the 4th ACM Workshop on Attacks and Solutions in Hardware Security*, pages 67–79, 2020.

[KPB+17] SV Dilip Kumar, Sikhar Patranabis, Jakub Breier, Debdeep Mukhopadhyay, Shivam Bhasin, Anupam Chattopadhyay, and Anubhab Baksi. A practical fault attack on ARX-like ciphers with a case study on ChaCha20. In *2017 Workshop on Fault Diagnosis and Tolerance in Cryptography (FDTC)*, pages 33–40. IEEE, 2017.

[KR16] Ted Krovetz and Phillip Rogaway. OCB (v1. 1). *Submission to the CAESAR Competition*, 2016.

[KS+20] Banashri Karmakar, Dhiman Saha, et al. DESIV: Differential fault analysis of SIV-Rijndael256 with a single fault. In *2020 IEEE International Symposium on Hardware Oriented Security and Trust (HOST)*, pages 241–251. IEEE, 2020.

[KY00] Jonathan Katz and Moti Yung. Unforgeable encryption and chosen ciphertext secure modes of operation. In *International Workshop on Fast Software Encryption*, pages 284–299. Springer, 2000.

[MHER14] Nicolas Moro, Karine Heydemann, Emmanuelle Encrenaz, and Bruno Robisson. Formal verification of a software countermeasure against instruction skip attacks. *Journal of Cryptographic Engineering*, 4(3):145–156, 2014.

[MOG+20] Kit Murdock, David Oswald, Flavio D Garcia, Jo Van Bulck, Daniel Gruss, and Frank Piessens. Plundervolt: Software-based fault injection attacks against Intel SGX. In *Proceedings of 41st IEEE Symposium on Security and Privacy (S&P)*, pages 1466–1482. IEEE, 2020.

[PCNM15] Sikhar Patranabis, Abhishek Chakraborty, Phuong Ha Nguyen, and Debdeep Mukhopadhyay. A biased fault attack on the time redundancy countermeasure for AES. In *Proceedings of 6th International Workshop on Constructive Side-Channel Analysis and Secure Design (COSADE)*, pages 189–203. Springer, 2015.

[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 (ACMCCS)*, pages 96–108, 2015.

[QAMSB+17] Hassan Qahur Al Mahri, Leonie Simpson, Harry Bartlett, Ed Dawson, and Kenneth Koon-Ho Wong. Fault attacks on XEX mode with application to certain authenticated encryption modes. In *Australasian Conference on Information Security and Privacy*, pages 285–305. Springer, 2017.

[QAMSB+19] Hassan Qahur Al Mahri, Leonie Simpson, Harry Bartlett, Ed Dawson, and Kenneth Koon-Ho Wong. Fault analysis of AEZ. *Concurrency and Computation: Practice and Experience*, 31(23):e4785, 2019.

[RAD19] Keyvan Ramezanpour, Paul Ampadu, and William Diehl. A statistical fault analysis methodology for the ascon authenticated cipher. In *2019 IEEE International Symposium on Hardware Oriented Security and Trust (HOST)*, pages 41–50. IEEE, 2019.

[RCC+16] Debapriya Basu Roy, Avik Chakraborti, Donghoon Chang, SV Kumar, Debdeep Mukhopadhyay, and Mridul Nandi. Fault based almost universal forgeries on CLOC and SILC. In *International Conference on Security, Privacy, and Applied Cryptography Engineering (SPACE)*, pages 66–86. Springer, 2016.

[SBHS15] Bodo Selmke, Stefan Brummer, Johann Heyszl, and Georg Sigl. Precise laser fault injections into 90nm and 45nm SRAM-cells. In *Proceedings of 14th International Conference on Smart Card Research and Advanced Applications (CARDIS)*, pages 193–205, Bochum, Germany, Nov 2015. Springer.

[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 *ASIACRYPT 2021*, 2021.

[SBR+20] Sayandeep Saha, Arnab Bag, Debapriya Basu Roy, Sikhar Patranabis, and Debdeep Mukhopadhyay. Fault template attacks on block ciphers exploiting fault propagation. In *Proceedings of 39th Annual International Conference on the Theory and Applications of Cryptographic Techniques (EUROCRYPT)*, pages 612–643. Springer, 2020.

[SC16] Dhiman Saha and Dipanwita Roy Chowdhury. EnCounter: on breaking the nonce barrier in differential fault analysis with a case-study on PAEQ. In *International Conference on Cryptographic Hardware and Embedded Systems*, pages 581–601. Springer, 2016.

[SGD08] Nidhal Selmane, Sylvain Guilley, and Jean-Luc Danger. Practical setup time violation attacks on AES. In *Proceedings of 7th European Dependable Computing Conference (EDCC)*, pages 91–96. IEEE, 2008.

[SH07] J. M. Schmidt and Michael Hutter. Optical and EM fault-attacks on CRT-based RSA: Concrete results. In *Proceedings of 15th Austrian Workhop on Microelectronics (Austrochip)*, pages 61–67, Graz, Austria,, Oct 2007. Verlag der Technischen Universität Graz.

[SLXY20] Iftekhar Salam, Kim Young Law, Luxin Xue, and Wei-Chuen Yau. Differential fault based key recovery attacks on TRIAD. In *Proceedings of the 23rd International Conference on Information Security and Cryptology*, pages 273–287. Springer, 2020.

[SMS+18] Iftekhar Salam, Hassan Qahur Al Mahri, Leonie Simpson, Harry Bartlett, Ed Dawson, and Kenneth Koon-Ho Wong. Fault attacks on Tiaoxin-346. In *Proceedings of the 16th Australasian Computer Science Week Multiconference*, pages 1–9, 2018.

[SOX+21] Iftekhar Salam, Thian Hooi Ooi, Luxin Xue, Wei-Chuen Yau, Josef Pieprzyk, and Raphaël C-W Phan. Random differential fault attacks on the lightweight authenticated encryption stream cipher Grain-128AEAD. *IEEE Access*, 9:72568–72586, 2021.

[SSB+18] Iftekhar Salam, Leonie Simpson, Harry Bartlett, Ed Dawson, and Kenneth Koon-Ho Wong. Fault attacks on the authenticated encryption stream cipher MORUS. *Cryptography*, 2(1):4, 2018.

[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.

[ZLZ+18] Fan Zhang, Xiaoxuan Lou, Xinjie Zhao, Shivam Bhasin, Wei He, Ruyi Ding, Samiya Qureshi, and Kui Ren. Persistent fault analysis on block ciphers. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, 2018(3):150–172, 2018.

Supplementary Material

A Decoupling Attack on Different Modes

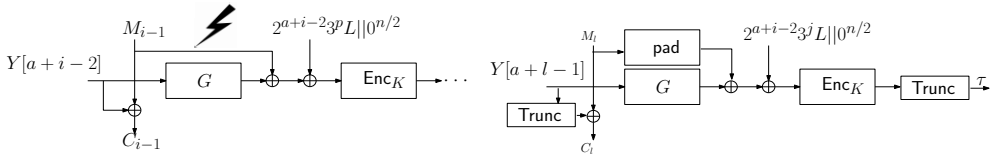

Figure 4: Single-pass mode example: GIFT-COFB.

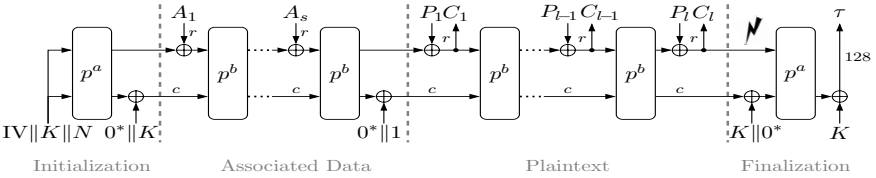

Figure 5: Single-pass mode example: ASCON.

The decoupling attack is not limited to the "Encrypt-then-MAC" structures like TEDT. It is also applicable to "MAC-then-Encrypt", "MAC-and-Encrypt" and even for single-pass schemes. The attack on SIV\$ (ref. Sec. 5.1) works as a representative to show how decoupling can be performed for "MAC-then-Encrypt" schemes. To see how it applies to "MAC-and-Encrypt" (or "Encrypt-and-MAC"), we refer to the OCB mode once again (see Sec. 4.1 for a short description of OCB). During the encryption of a message $M = M_1||M_2|| \cdots ||(M^*\mathsf{pad})$, we target the final incomplete message block (M^*) for attack. During the checksum computation, we corrupt this message block making it $M^* \oplus \Delta_i$. However, the encryption of M^* remains uncorrupted. The MAC stage, therefore, computes $\tau' = MAC(M_1||M_2|| \cdots ||(M^*\mathsf{pad}) \oplus \Delta_i)$. The $\mathcal{AE}.\mathsf{Enc}$ finally outputs $(C||\tau')$. The adversary creates a forgery by computing $((C \oplus \Delta_i)||\tau')$.

We next argue that single-pass schemes like GIFT-COFB [BCI+20] are also vulnerable. A careful investigation of GIFT-COFB (ref. Fig 4) shows that even for those schemes, one message block is processed two times – once for generating the ciphertext, and once for consuming it within the state for MAC. A fault during the second processing enables the decoupling attack. In general, decoupling can take place whenever any variable, which is either an input or output of the AEAD, is used multiple times. The usage includes reading from or writing back to the memory. For example, one may also target the ASCON [DEMS21] single-pass scheme. In this case, a ciphertext block is required to be corrupted, just when it is consumed by the next permutation block for MAC (Fig. 5). Such a fault can be created in many ways, such as by corrupting initial rounds of the permutation.

B Proof of Theorem 2

Proof. We start by replacing the F_K function with an *ideal-world* frPRF: (F_K^f, RF) (Table 4; column 2). We assume the hash function with a frRO. The adversary $\mathcal{A}$ can make direct queries to the random oracle of the form $h_p = H(r_p\|M_p)$. It maintains two tables $\mathcal{R}_p$ and $\mathcal{H}_p$, such that $\mathcal{R}_p$ includes all the values r_p used in these queries and $\mathcal{H}_p$ includes all the pairs $(r_p\|M_p, h_p)$. The number of such queries is q_p.

During the game the set $\mathcal{S}_{flt} \cup \mathcal{S}$ contains the trivial queries. If $(r\|M, \tau) \notin \mathcal{S}_{flt} \cup \mathcal{S}$, we call $(r\|M)$ a fresh message. Besides, if $\tau = \mathsf{Mac}_K(r\|M)$, we call $(r\|M, \tau)$ a valid pair. We describe the following games. The event E_i is the event that the adversary against **Game i** terminates, outputting 1. We note that during any faulty query, at most two values can be added to any of the transcripting sets; $\mathcal{M}_{flt}$ or $\mathcal{R}_{flt}$, since we only consider single faults. At the same time, if during a given call, the adversary observes two messages with the same tag, this will either imply a collision is found for the frRO or distinguishing the frPRF based on the value n_{pre} in Table 4.

Game 0: The game described in Table 7 is used with (F_K^f, RF) being a ideal-world frPRF. E_0 is the event that $\exists j \in \{1, \cdots, q_v\}$ *s.t.* $r^j\|M^j$ is fresh and $(r^j\|M^j, \tau^j)$ is a valid pair.

Game 1: The game is similar to **Game 0** but it terminates if a collision is found in the set $\mathcal{R}_{flt}$, or between a value in $\mathcal{R}_p$ and a value in $\mathcal{R}_{flt}$. Note that an adversary can make queries to the random oracle with $r_p = r_i$ received from a previous query to Mac_K or Mac_K^f leading to a trivial condition. However, we are interested in the adversary's ability to predict a random value. In order to construct **Game 1**, we need to construct an adversary $\mathcal{A}'$ that simulates the oracles in Table 7 using the H, H^f, RF and F_K^f oracles (F_K^f is the PRF_K^f oracle from Table 4). After the i^{th} query, it checks the sets $\mathcal{R}_{flt}$ and $\mathcal{R}_p$ and returns 1 if there one of the following two conditions is satisfied:

1. $\mathcal{R}_{flt}$ has a collision.
2. The i^{th} query is a query to Mac_K or Mac_K^f and $r^i \in \mathcal{R}_p$

Since r^i is uniformly distributed and unpredictable to the adversary before it is used, then all the values in $\mathcal{R}_{flt}$ are *almost* uniformly distributed. Note that not all pairs in R_{flt} are equally likely to collide, *e.g.,* r and $r \oplus \Delta$, where Δ is a fault value will never collide. This reduces the collision probability from the case where all the values are uniformly distributed (it reduces the number of collision-pair candidates). We can upper bound the probability by taking the uniform case, which is the collision probability over $|\mathcal{R}_{flt}|$ uniformly random values, and $|\mathcal{R}_{flt}| \leq 2q_f + q_m = q_e$. The probability that the second condition is satisfied after the i^{th} query is bounded by $q_p/2^{|r|}$ if it a query to Mac_K and $2q_p/2^{|r|}$ if it a query to Mac_K^f. Hence, the transition is bounded by

$$|\Pr[E_0] - \Pr[E_1]| \leq \frac{\binom{q_e}{2}}{2^{|r|}} + \frac{q_e q_p}{2^{|r|}}$$

Game 2: The game is similar to **Game 1** but it aborts if a collision is found in the H.

In order to construct **Game 2**, we need an adversary $\mathcal{B}$ that simulates the oracles in Table 7, similar to $\mathcal{A}'$, as follows: When it receives a Mac_K query M, it generates a random r and calculates $h = H(r\|M)$, stores $(r\|M, h)$ is a hash table $\mathcal{H}$. $\mathcal{B}$ then queries the frPRF oracle $\mathsf{PRF}_K(h)$, and returns the outcome τ to $\mathcal{A}$. It does the same when it receives a query to Mac_K^f with $\mathcal{F}.mod = \mathsf{nof}$. When $\mathcal{B}$ receives a query to Mac_K^f with $\mathcal{F}.v = \mathsf{prf}$, it also does the same except it queries $\mathsf{PRF}_K^f(h, \mathcal{F})$. When $\mathcal{B}$ receives a query to Mac_K^f with $\mathcal{F}.v = \mathsf{hash}$, it simulates the faulty hash function H^f. During this simulation, $\mathcal{B}$ maintains the set $\mathcal{M}_{flt}$. It sends the outcome of the faulty hash h^f to RF. If $\exists r||M \in \mathcal{M}_{flt}$ *s.t.* $h^f = H(r||M)$, then $\mathcal{B}$ adds $(r||M, h^f)$ to $\mathcal{H}$, Otherwise, it adds $(\perp, h^f)$. Finally, when $\mathcal{B}$ receives a Verify_K query $(r||M, \tau)$, it calculates $h = H(r||M)$, sends h to RF and checks if $\tau = \mathsf{RF}(h)$. It also stores $(r||M, h)$ in $\mathcal{H}$. $\mathcal{B}$ also simulates the queries $\mathcal{A}$ makes to the random oracle and appends them to $\mathcal{H}$. At the end of the game, $\mathcal{B}$ checks if there is a collision in $\mathcal{H}$. We call this event HC. It returns 1 if the collision exists and 0 otherwise. $\mathcal{A}$ returns 1 if E_0, E_1 or HC occurs. Note that for single fault, there can be

at most two values in $\mathcal{M}_{flt}$. $\mathcal{B}$ simulates **Game 1** correctly, and as long as HC does not occur, the two games are identical. Since H is a frRO, then

$$|\Pr[E_1] - \Pr[E_2]| \leq \frac{2(q_p + 2q_f + q_m + q_v)^2}{2^{|h|}} = \frac{2(q_p + q_e + q_v)^2}{2^{|h|}}$$

Game 3: This game is the same except that the F_K^f oracle is replaced with an oracle that is similar except that instead of returning τ, it returns (a, τ) *s.t.* either $F_K(a) = \tau$ or $a = \perp$, which occurs if the condition in line 5 of F_K^f of Table 4 (in both worlds) fails for all possible values. From $\mathcal{A}$'s point of view, **Game 2** and **Game 3** are identical.

$$|\Pr[E_2] - \Pr[E_3]| = 0$$

Game 4: The game is similar to **Game 3** except that we change the adversary $\mathcal{B}$ to $\mathcal{B}'$ to address adversaries trying to break the one-wayness of the random oracle. For every query h that $\mathcal{B}'$ sends to either RF or F_K^f, when $\mathcal{F}.v \neq \{\mathsf{prf}\}$, it sets $h^t = h$. If it queries F_K^f when $\mathcal{F}.v = \{\mathsf{prf}\}$, and F_K^f returns (a, τ), it sets $h^t = a$. If $a = \perp$, it does not add a corresponding entry to $\mathcal{H}$. It terminates if during any of the verification queries (r^j, M^j, τ^j), $(\star, H(r^j \| M^j)) \in \mathcal{H}$. We call this event HP. Note that if $\star \neq \perp$, then the adversary found the collision and both games will be identical. If $\star = \perp$, then one of two cases are possible:

1. The adversary found a pre-image for a target value h.
2. The adversary injected a fault during a query to Mac_K^f such that the effective input to RF or F_K^f is $H(r^j \| M^j)$.

The first case is addressed using the one-wayness of the random oracle. The second case, $\exists (i, j) \in \{1, \cdots, q_f\} \cup \{1, \cdots, q_v\}$, *s.t.* $H(r^j \| M^j) = H(r^i \| M^i) \oplus \delta$, where δ is the fault in the output of the hash function. This requires a near collision at the output of the hash function. The value r^i during the Mac_K^f query is uniform and cannot be predicted before the injection of the fault. Hence, from the properties of the faulty random oracle, the probability of this event for a given pair (i, j) is bounded by $1/2^{|h|}$. Note that since there is no collision on the random value r, the output of the hash function is always fresh, and the knowledge of δ does not help the adversary. The number of possible near collision pairs in this case is bounded by $2q_f q_v$. Thus,

$$|\Pr[E_3] - \Pr[E_4]| \leq \frac{2(2q_f + q_m + q_v + 1)}{2^{|h|}} + \frac{2q_f q_v}{2^{|h|}} = \frac{2(q_e + q_v + 1)}{2^{|h|}} + \frac{2q_f q_v}{2^{|h|}}$$

Game 5: This game is similar to **Game 4**, except the game aborts if one of the q_v queries to $\mathsf{Verify_K}$ is valid and fresh. E_4 is the event that the adversary wins this game. Note that since the previous games terminate for different possible collisions, the input to the frPRF in this game is always fresh, and the output is uniformly distributed.

In order to transition from **Game 4** to **Game 5**, we construct $q_v + 2$ hybrid games, as follows:

Game $4^{\mathbf{j}}$: is the similar to **Game 4**, except it aborts if one of the first j queries to Verify_K is valid and fresh. It can be seen that **Game $4^{\mathbf{0}}$** $\equiv$ **Game 4** and **Game $4^{\mathbf{q_v+1}}$** $\equiv$ **Game 5**.

We note that **Game $4^{\mathbf{j}}$** and **Game $4^{\mathbf{j+1}}$** are identical if query $j + 1 \in \{1, \cdots, q_v\}$ is invalid or not fresh. If the query is valid and fresh, we call this event GT^{j+1}. We note that

$$|\Pr[E_4^j] - \Pr[E_4^{j+1}]| \leq \Pr[GT^{j+1}].$$

We also note that if at query $j \in \{1, \cdots, q_v\}$ is fresh, and $HC \vee HP$ do not occur, then H^j is a fresh input to RF. Hence, the adversary has to guess a fresh output of a random function RF. Thus,

$$|\Pr[E_4^j] - \Pr[E_4^{j+1}]| \leq \Pr[GT^{j+1}] \leq \frac{1}{2^{|\tau|}}$$

and we can bound $|\Pr[E_3] - \Pr[E_4]|$ by

$$|\Pr[E_4] - \Pr[E_5]| \leq \sum_{j=0}^{q_v-1} |\Pr[E_4^j] - \Pr[E_4^{j+1}]| \leq \frac{q_v}{2^{|\tau|}} \text{ and } \Pr[E_5] = 0$$

Finally, the overall bound can be given by

$$\mathbf{Adv}^{\mathtt{frmac}}_{\mathsf{htfrprf},\mathsf{Fault}}(q_f, q_m, q_v, q_p, t) \leq \mathbf{Adv}^{\mathtt{frprf}}_{F_K}(q_f, q_m+q_v, t) + \sum_{i=0}^{4} |\Pr[E_i] - \Pr[E_{i+1}]| + \Pr[E_4] \leq$$

$$\mathbf{Adv}^{\mathtt{frprf}}_{F,\mathsf{Fault}}(q_f, q_m+q_v, t) + \frac{\binom{q_e}{2}}{2^{|r|}} + \frac{q_e q_p}{2^{|r|}} + \frac{2(q_p+q_e+q_v)^2}{2^{|h|}} + \frac{2(q_e+q_v+1)}{2^{|h|}} + \frac{2q_f q_v}{2^{|h|}} + \frac{q_v}{2^{|\tau|}}$$

where $q_e = 2q_f + q_m$. □

C Proof of Theorem 3

Proof. The adversary $\mathcal{A}$ performs q_f faulty queries to the Enc_K^f oracle, followed by q_m Enc_K queries and q_v Dec_K queries (ref. Table 9). Note that $K := (K_1, K_2, K_3)$. Mac_{K_1}, E_{K_2} and Mac_{K_3} are replaced with their ideal-world variants. The adversary additionally maintains tables $\mathcal{R}_{flt}$, and $\mathcal{T}_{1,flt}$ such that $\mathcal{R}_{flt}$ maintains all the r values queried, and $\mathcal{T}_{1,flt}$ maintains all the τ_1 values queried.

Game 0: This game is described in Table 9. An elaborated version of this game appears in the Table 10. However, the proof can be followed with Table 9 only.

Game 1: This game is similar to **Game 0** except that it terminates if there is a collision in $\mathcal{R}_{flt}$. Since in every faulty query we an have at most two random values,

$$|\Pr[E_0] - \Pr[E_1]| \leq \frac{\binom{2q_f+q_m}{2}}{2^{|r|}}.$$

Game 2: This game is similar to **Game 1** except that it terminates if there is a collision in $\mathcal{T}_{1,flt}$. Since in every fault query we an have at most two values for τ_1,

$$|\Pr[E_1] - \Pr[E_2]| \leq \frac{\binom{2q_f+q_m}{2}}{2^{|\tau_1|}}$$

Game 3: This game is similar to **Game 2** except that it terminates if the forgery $(N^*, A^*, \tau_1^*, C_1^*, \tau_2^*)$ succeeds, but triplet $(\tau_1^*, C_1^*, \tau_2^*)$ is fresh and valid for Mac_{K_3} (during the decryption). This case is covered by the security of the second MAC.

$$|\Pr[E_2] - \Pr[E_3]| = 0.$$

Game 4: This game is similar to **Game 3** except that it terminates if $(N^*, A^*, \tau_1^*, C^*, \tau_2^*)$ succeeds, but triplet $(\tau_1^*, C^*, \tau_2^*)$ is trivial for Mac_{K_3}. If

$$\nexists(N^{'}, A^{'}, r^{'}, \tau_1^*, C^*, \tau_2^*) \in \mathcal{S} \cup \mathcal{S}_{flt},$$

it implies that (τ_1^*, C_1^*) was a faulty input to Mac_{K_3} in one of the faulty encryption queries, since otherwise, the full query would be trivial. If the fault was injected in C, *i.e.,* τ_1 was generated in an encryption query, then $KG(F_{K_2}(\tau_1^*)) \oplus C^*$ leads to a fresh message for Mac_{K_1} at the decryption. In this case the security is covered by the security of Mac_{K_1}.

Since τ_1 is collision-free, if the fault was injected in τ_1, a key-stream $KG(F_{K_2}(\tau_1^*))$ is fresh and uniform, *i.e.,* τ_1^* was never used to seed KG previously. Since the key-stream used to decrypt C^* will be uniformly distributed, the input to Mac_{K_1} at the decryption is uniformly distributed. If the input to Mac_{K_1} is fresh, this is covered by the security of Mac_{K_1}. The probability that such input is not fresh is bounded by $(2q_f + q_m)q_v/2^{|r|}$. If

$$\exists(N^{'}, A^{'}, r^{'}, \tau_1^*, C^*, \tau_2^*) \in \mathcal{S} \cup \mathcal{S}_{flt},\ s.t.\ (N^{'}, A^{'}, r^{'}) \neq (N^*, A^*, r^*),$$

it implies that a fault was injected in either Mac_{K_1} or E_{K_2}. If the fault was injected in E_{K_2} in τ_1, such τ_1 was not generated by a valid Mac_{K_1} query. Hence, the forgery cannot be successful. If the fault was injected in C^*, then the input to Mac_{K_1} is fresh. Finally, if the fault was injected in Mac_{K_1}, then C^* does not encrypt the same message that was processed by Mac_{K_1} and the input to Mac_{K_1} at the decryption is fresh. Those cases are also covered by the security of Mac_{K_1}. Therefore,

$$|\Pr[E_3] - \Pr[E_4]| \leq \frac{(2q_f + q_m)q_v}{2^{|r|}}.$$

Finally, $\Pr[E_4] = 0$, since if none of the games terminate, all the inputs to the MACs are either trivial or invalid and frmac and frprf security of the components is not broken and maintains the indistiguishability of the non-faulty queries from random strings. □

Table 10: The MEM oracles (elaborated)

$\mathsf{Expt}^{\mathsf{MEM},b}_{\mathcal{A},\mathcal{F}}$

INIT

$(K_1, K_2, K_3) \xleftarrow{\$} \mathcal{K}$
$K := (K_1, K_2, K_3)$
$d \leftarrow 0$
$b \xleftarrow{\$} \{0,1\}$
$\mathcal{S}_{flt} \leftarrow \emptyset$
$\mathcal{S} \leftarrow \emptyset$
$\mathcal{R}_{flt} \leftarrow \emptyset$
$\mathcal{T}_{1,flt} \leftarrow \emptyset$
return s

$\mathsf{Enc}^f_K(N, A, M, \mathcal{F})$

if $d = 1$
  return $\perp$
$r \xleftarrow{\$} \mathcal{R}$
$\mathcal{R}_{flt} \leftarrow \mathcal{R}_{flt} \cup \{r\}$
$\mathcal{N}_{flt} \leftarrow \{N\}$
$\mathcal{AD}_{flt} \leftarrow \{A\}$
if $\mathcal{F}.mod = \mathsf{nof}$
  $\tau_1 \leftarrow \mathsf{Mac}_{K_1}(N\|A\|r\|M\|\|M|)$
  $\mathcal{T}_{1,flt} \leftarrow \mathcal{T}_{1,flt} \cup \{\tau_1\}$
  $C \leftarrow KG(F_{K_2}(\tau_1)) \oplus r\|M$
  $\tau_2 \leftarrow \mathsf{Mac}_{K_3}(\tau_1\|C)$
else if $\mathcal{F}.v = r$ ▷ at $\mathcal{AE}$ input
  $\tau_1 \leftarrow \mathsf{Mac}_{K_1}(N\|A\|r \oplus \Delta\|M\|\|M|)$
  $\mathcal{T}_{1,flt} \leftarrow \mathcal{T}_{1,flt} \cup \{\tau_1\}$
  $C \leftarrow KG(F_{K_2}(\tau_1)) \oplus (r \oplus \Delta)\|M$
  $\mathcal{R}_{flt} \leftarrow \mathcal{R}_{flt} \cup \{r \oplus \Delta\}$
  $\tau_2 \leftarrow \mathsf{Mac}_{K_3}(\tau_1\|C)$
else if $\mathcal{F}.v = r$ ▷ at MAC_1 input
  $\tau_1 \leftarrow \mathsf{Mac}_{K_1}(N\|A\|r \oplus \Delta\|M\|\|M|)$
  $\mathcal{T}_{1,flt} \leftarrow \mathcal{T}_{1,flt} \cup \{\tau_1\}$
  $C \leftarrow KG(F_{K_2}(\tau_1)) \oplus (r)\|M$
  $\mathcal{R}_{flt} \leftarrow \mathcal{R}_{flt} \cup \{r \oplus \Delta\}$
  $\tau_2 \leftarrow \mathsf{Mac}_{K_3}(\tau_1\|C)$
else if $\mathcal{F}.v = r$ ▷ at Enc input
  $\tau_1 \leftarrow \mathsf{Mac}_{K_1}(N\|A\|r\|M\|\|M|)$
  $\mathcal{T}_{1,flt} \leftarrow \mathcal{T}_{1,flt} \cup \{\tau_1\}$
  $C \leftarrow KG(F_{K_2}(\tau_1)) \oplus (r \oplus \Delta)\|M$
  $\mathcal{R}_{flt} \leftarrow \mathcal{R}_{flt} \cup \{r \oplus \Delta\}$
  $\tau_2 \leftarrow \mathsf{Mac}_{K_3}(\tau_1\|C)$
else if $\mathcal{F}.v = \mathsf{tag}$ ▷ at Mac_1 output
  $\tau_1 \leftarrow \mathsf{Mac}_{K_1}(N\|A\|r\|M\|\|M|)$
  $\mathcal{T}_{1,flt} \leftarrow \mathcal{T}_{1,flt} \cup \{\tau_1\}$
  $C \leftarrow KG(F_{K_2}(\tau_1 \oplus \Delta)) \oplus (r)\|M$
  $\mathcal{T}_{1,flt} \leftarrow \mathcal{T}_{1,flt} \cup \{\tau_1 \oplus \Delta\}$
  $\tau_2 \leftarrow \mathsf{Mac}_{K_3}(\tau_1 \oplus \Delta\|C)$
else if $\mathcal{F}.v = \mathsf{tag}$ ▷ at Enc input
  $\tau_1 \leftarrow \mathsf{Mac}_{K_1}(N\|A\|r\|M\|\|M|)$
  $\mathcal{T}_{1,flt} \leftarrow \mathcal{T}_{1,flt} \cup \{\tau_1\}$
  $C \leftarrow KG(F_{K_2}(\tau_1 \oplus \Delta)) \oplus r\|M$
  $\mathcal{T}_{1,flt} \leftarrow \mathcal{T}_{1,flt} \cup \{\tau_1 \oplus \Delta\}$
  $\tau_2 \leftarrow \mathsf{Mac}_{K_3}(\tau_1\|C)$
else if $\mathcal{F}.v = \mathsf{tag}$ ▷ at Mac_2 input
  $\tau_1 \leftarrow \mathsf{Mac}_{K_1}(N\|A\|r\|M\|\|M|)$
  $\mathcal{T}_{1,flt} \leftarrow \mathcal{T}_{1,flt} \cup \{\tau_1\}$
  $C \leftarrow KG(F_{K_2}(\tau_1)) \oplus r\|M$
  $\mathcal{T}_{1,flt} \leftarrow \mathcal{T}_{1,flt} \cup \{\tau_1 \oplus \Delta\}$
  $\tau_2 \leftarrow \mathsf{Mac}_{K_3}((\tau_1 \oplus \Delta)\|C)$
else if $\mathcal{F}.v = \mathsf{Mac}_1$
  $(N^f, A^f, r^f, M^f, \tau_1) \leftarrow$
    $\mathsf{Fault}(\mathsf{Mac}_{K_1}(N\|A\|r\|M\|\|M|), \mathcal{F})$
  $\mathcal{T}_{1,flt} \leftarrow \mathcal{T}_{1,flt} \cup \{\tau_1\}$
  $C \leftarrow KG(F_{K_2}(\tau_1)) \oplus r\|M$
  $\tau_2 \leftarrow \mathsf{Mac}_{K_3}(\tau_1\|C)$
  $\mathcal{N}_{flt} \leftarrow \mathcal{N}_{flt} \cup \{N^f\}$
  $\mathcal{AD}_{flt} \leftarrow \mathcal{AD}_{flt} \cup \{A^f\}$
  $\mathcal{R}_{flt} \leftarrow \mathcal{R}_{flt} \cup \{r^f\}$
else if $\mathcal{F}.v = \mathsf{Mac}_2$
  $\tau_1 \leftarrow \mathsf{Mac}_{K_1}(N\|A\|r\|M\|\|M|)$
  $\mathcal{T}_{1,flt} \leftarrow \mathcal{T}_{1,flt} \cup \{\tau_1\}$
  $C \leftarrow KG(F_{K_2}(\tau_1)) \oplus r\|M$
  $(\tau_1^f, C^f, \tau_2) \leftarrow \mathsf{Fault}(\mathsf{MAC}_{K_3}(\tau_1\|C), \mathcal{F})$
  $\mathcal{T}_{1,flt} \leftarrow \mathcal{T}_{1,flt} \cup \{\tau_1^f\}$
else if $\mathcal{F}.v = \mathsf{Enc}$
  $\tau_1 \leftarrow \mathsf{Mac}_{K_1}(N\|A\|r\|M\|\|M|)$
  $\mathcal{T}_{1,flt} \leftarrow \mathcal{T}_{1,flt} \cup \{\tau_1\}$
  $C \leftarrow KG(F_{K_2}(\tau_1)) \oplus (r\|M) \oplus \Delta$
  $\tau_2 \leftarrow \mathsf{Mac}_{K_3}(\tau_1\|C)$
$\mathcal{I}_{vld} \leftarrow \{(N^{'}, A^{'}) \in \mathcal{N}_{flt} \times \mathcal{AD}_{flt} \mid$
  $\mathsf{Dec}_K(N^{'}, A^{'}, C) \neq \perp \wedge (N^{'}, A^{'}, C) \notin \mathcal{S}_{flt}\}$
if $b = 1 \wedge |\mathcal{I}_{vld}| > 1$
  return $\perp$
$\mathcal{S}_{flt} \leftarrow \mathcal{S}_{flt} \cup \{(N^{'}, A^{'}, C) \mid$
  $(N^{'}, A^{'}) \in \mathcal{I}_{vld}\}$
return $(\tau_1, C, \tau_2, \mathcal{S}_{flt})$

$\mathsf{Enc}_K(N, A, M)$

$d \leftarrow 1$
$r \xleftarrow{\$} \mathcal{R}$
$\mathcal{R}_{flt} \leftarrow \mathcal{R}_{flt} \cup \{r\}$
if $M \in \mathcal{S}_{flt} \cup \mathcal{S}$
  return $\perp$
$\tau_1 \leftarrow \mathsf{Mac}_{K_1}(N\|A\|r\|M\|\|M|)$
$\mathcal{T}_{1,flt} \leftarrow \mathcal{T}_{1,flt} \cup \{\tau_1\}$
$C \leftarrow KG(F_{K_2}(\tau_1)) \oplus r\|M$
$\tau_2 \leftarrow \mathsf{Mac}_{K_3}(\tau_1\|C)$
$\mathcal{S} \leftarrow \mathcal{S} \cup \{(N, A, \tau_1, C, \tau_2)\}$
if $b = 1$
  $(\tau_1, C, \tau_2) \xleftarrow{\$} \{0,1\}^{|\tau_1|+|\tau_2|+|r|+|M|}$
return τ_1, C, τ_2

$\mathsf{Dec}(N, A, \tau_1, C, \tau_2)$

if $b = 1 \vee (N, A, \tau_1, C, \tau_2) \in \mathcal{S}_{flt} \cup \mathcal{S}$
  return $\perp$
$\tau_2^{'} \leftarrow \mathsf{Mac}_{K_3}(\tau_1\|C)$
$r\|M \leftarrow KG(F_{K_2}(\tau_1)) \oplus C$
$\tau_1^{'} \leftarrow \mathsf{Mac}_{K_1}(N\|A\|r\|M\|\|M|))$
if $(\tau_2 \neq \tau_2^{'}) \vee (\tau_1 \neq \tau_1^{'})$
  return $\perp$
else
  return M

D Discussion and Open Issues

The proposed MEM construction is meant for preventing single-fault attacks which it achieves successfully. It has already been argued that most of the two-pass schemes are not suitable in this regard. However, an important question is, how other three-pass options work against faults. In this section, we discuss a few other three-pass schemes in the context of fault attacks. Next, we present some open issues regarding the fault-resilient constructions and their potential extensions.

D.1 Other Three-Pass Constructions

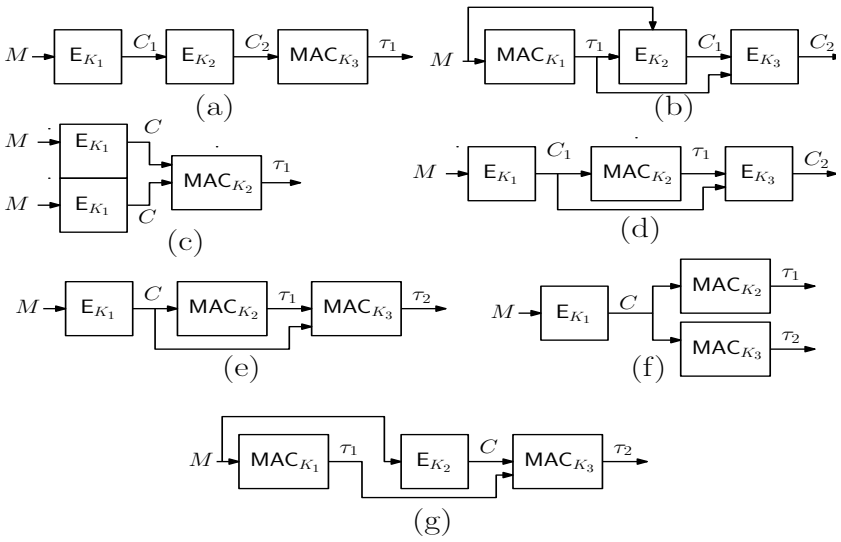

Figure 6: Alternative Three-pass Constructions.

There can be several basic constructions having more than one pass of encryption and MAC steps. A tempting question is whether or not they all present security against FAs with respect to integrity. Several such potential combinations are described in Fig. 6. It is interesting to observe that most of them are vulnerable to single-fault injections. We restrict our discussions to the schemes which only allow a constant increase in the ciphertext length.

First, let us consider schemes having two encryption operations and one MAC. Two consecutive encryptions (ref. Fig. 6(a),(b)) do not help as together they can be considered as one encryption operation. Fig. 6(c) performs two parallel encryptions with the same key followed by a MAC. While this scheme may look secure until only one of the ciphertexts is corrupted, we note that it depends upon how the MAC is implemented. In the case of parallel implementations, both the ciphertexts will be processed simultaneously. A single fault can corrupt both ciphertexts at this time instant and result in a decoupling attack. If two encryptions are separated by a MAC operation (ref. Fig. 6(d)), it can be considered as "MAC-then-Enc" over an encrypted message. Decoupling attacks take place if the MAC input is corrupted with faults.

Next, we consider cases having two MAC operations. The first two constructions in this class are presented in Fig. 6(e) and (f), where the two MACs operate on the ciphertext, and the concatenation of the ciphertext and the first MAC, respectively. A decoupling fault on the ciphertext at the beginning of the MAC phase enables an attack in this case (truncation attacks are also feasible). A very similar situation arises if the encryption

follows the two consecutive MACs[9]. One may abstract these consecutive MACs as a single MAC which converts the schemes to "Encrypt-then-MAC" and "MAC-then-Encrypt" modes, where attacks are clearly feasible[10].

One may observe that the main issue with the schemes described in the last paragraph is that both MACs operate on the same variables – ciphertext and the first tag (for the second MAC operation). Faulting the ciphertext once, therefore, serves the purpose of the adversary. The third class of schemes performs the MACs on different variables – the plaintext and the ciphertext (or ciphertext + tag). Clearly, MEM is a member of this class. Another potential member is presented in Fig. 6(g) which also prevents the attacks for single fault injection. However, we note that the overall computational complexity of this scheme is the same as MEM – two MAC computations are always required. In this paper, we choose MEM due to its structural simplicity which makes the security proofs simple and enables combination with the PSV encryption. However, the other candidate in this class is also expected to show similar properties. Finally, we note that certain single-pass schemes (computing the encryption and the MAC in one pass) can be converted to a member of this class if they are augmented with another MAC operation at the end. We leave the analysis of the security and computational complexity of such potential schemes as future work.

D.2 Security Against Multiple Faults

Let us consider a scenario where the adversary is allowed to inject at least two faults per encryption query. In this case the adversary can corrupt the computation of MAC_{K_1} for a given message M such that $\tau_1 \leftarrow \mathsf{MAC}_{K_1}(M \oplus \Delta)$. It then injects the same Δ fault during the computation of MAC_{K_2} resulting in $\tau_2 \leftarrow \mathsf{MAC}_{K_2}(C \oplus \Delta || \tau_1)$. In case the encryption is linear (such as in PSV), $\tau_2 \leftarrow \mathsf{MAC}_{K_2}(M \oplus Y \oplus \Delta || \tau_1)$ where Y is the keystream generated by the encryption function. The adversary, therefore can create a successful forgery by asking for a decryption of the message $(\tau_1 || (C \oplus \Delta) || \tau_2)$.

While it is evident that the attack works with two faults, it still requires the faults to be equal-valued, which is hard to achieve for many practical implementations. It becomes even more difficult practically as in two different instants, two different variables are to be faulted. However, from a theoretical perspective it is still important to extend the security for multiple fault injections. While MEM allows such extension by interleaving the encryption and MAC blocks with a final MAC at the end, a detailed analysis in this regard is left as a future work.

D.3 Security Against fix Faults

Our proofs for the frMAC security relies upon the assumption that the fault is differential in nature. In the case of fix faults, the following attack can be performed on the frMAC construction.

Consider an adversary who maintains a table U of offline computations for the public hash H used in the frMAC construction. U contains entries of the form $((r_j||M_j||N_j||A_j), h_j)$. Now if the adversary is capable of doing a fix fault over the full state, then during the frMAC query for any given message $(r_i||M_i||N_i||A_i)$, it can replace the desired hash value h_i with h_j eventually generating a tag τ_j corresponding to $(r_j||M_j||N_j||A_j)$. Now since this message has never been queried to the frMAC oracles, the adversary can use it as a valid forgery. This attack is, however, not possible for differential faults, as the output of the hash is always randomized and remains randomized even after a differential fault injection. Another reason behind not ensuring security against fix model is that fixing

[9] We do not describe it here as it is obvious.

[10] The attacks can be countered if the implementation performs two separate memory reads for two MAC operations. However, this is more an implementation-level protection than a mode-level solution.

the randomization to a certain value may break the security in several ways. The most obvious impact would be on confidentiality, as repeating the randomness will result in a nonce repetition.

It is worth mentioning that if we only consider the attack on the hash output with fix faults, it is solely an issue for the frMAC construction. In other words, the attack does not apply for the MEM scheme, as the verification will always fail for the second MAC. This is because the second MAC also takes the ciphertext as an input. However, analyzing this scenario formally with frMAC is not straightforward, and we leave it for the future. Currently, we only claim security with respect to differential faults.

D.4 Security against Decryption Faults

In this work, we do not claim security against decryption faults. In fact it is easy to observe that having both encryption and decryption fault can be fatal for security if the decryption fault is injected during the challenge. Considering the integrity scenario, the adversary can first fault any one of the MACs (say MAC_2) during the encryption operation as mentioned before. He can then modify the ciphertext offline and use it as a challenge query. While the query is still invalid in this case (as the other MAC still does not match), during the decryption the adversary can fault the other MAC (MAC_1) by skipping the final check. This indeed leads to a valid forgery scenario even for our protected constructions. With our current constructions, addressing the decryption faults, therefore, seems non-trivial. We leave it as one of the major open challenges to be addressed in the context of fault-resilient AEADs.

D.5 Security Against Leakage

Our "fr" constructions do not claim security against a scenario where both leakage and faults are present. However, we observe certain similarities between our frMAC construction and the leakage-resilient MACs [BKP+18], which makes us believe that claiming combined security against both attacks should be feasible. There can be several ways for combining leakage with faults. One possible way is to limit both leakage and faults for non-challenge queries only (the simplest scenario). We believe that proving security for such scenarios should be relatively easier. The situation becomes a bit more complex if we consider both encryption and decryption leakage. Finally, allowing challenge leakage and fault (for both encryption and decryption) is the most general albeit challenging scenario. Allowing challenge leakage may lead to situations where Simple Power Analysis (SPA) attacks might be feasible, and one must ensure that such attacks do not take place. DPA attacks are relatively easier to prevent as the PSV scheme already enjoys some security against them. However, challenge faults can make the scenario even more complex. In the last paragraph, we described the situation with respect to integrity. However, challenge faults in the encryption may also lead to the recovery of the ephemeral key which breaks confidentiality. Formalizing these issues is out-of-the-scope for the current paper and, therefore, left as a future work.

E The Security Game due to [FG20]

The security game for SIV$ in [FG20] is given as follows:

Table 11: The Security Game in [FG20]

$\mathsf{Expt}^{\mathsf{frAE\$},b}_{\mathcal{AE},\mathcal{A}}$

INIT :

1: $K \xleftarrow{\$} \mathcal{K}$
2: $\mathcal{S} \leftarrow \emptyset$
3: $\mathsf{clash} \leftarrow \mathsf{false}$

FIN :

1: $b^{'} \leftarrow \mathcal{A}^{\mathcal{O}^{\mathsf{Enc}^{f}_{K}(\cdot)},\mathcal{O}^{\mathsf{Dec}_{K}(\cdot)}(\mathcal{F})}$
2: **if** clash: $b^{'} \leftarrow b$
3: **return** $b^{'}$

Oracle $\mathcal{O}^{\mathsf{Dec}_K}(N, A, C)$:
1: **if** $b = 1$ or $(N, A, C) \in \mathcal{S}$
2: **return** $\perp$
3: **else**
4: **return** $(\mathsf{Dec}_K(N, A, C)$

Oracle $\mathcal{O}^{\mathsf{Enc}^{f}_{K}}(N, A, M, \mathcal{F})$:
1: $C_1 \xleftarrow{\$} \{0,1\}^{|C_0|}$
2: $C_0 \leftarrow \mathsf{Fault}(\mathsf{Enc}_K(N, A, M), \mathcal{F})$
3: $\mathcal{NA}^{\mathsf{Enc}}_{vld} \leftarrow \{(N^{'}, A^{'})$
$\in \mathcal{N}_{flt} \times \mathcal{AD}_{flt}$ |
$\mathsf{Dec}(K, N^{'}, A^{'}, C_b) \neq \perp$
$\wedge (N^{'}, A^{'}, C_b) \notin \mathcal{S}\}$
4: **if** $|\mathcal{NA}^{\mathsf{Enc}}_{vld}| \geq 2$
5: $\mathsf{clash} \leftarrow \mathsf{true}$
6: $\mathcal{S} \leftarrow \mathcal{S} \cup \{(N^{'}, A^{'}, C_b)$ |
7: $(N^{'}, A^{'}) \in \mathcal{NA}^{\mathsf{Enc}}_{vld}\}$
8: **return** C_b