Providing Integrity for Authenticated Encryption in the Presence of Joint Faults and Leakage

Francesco Berti, Itamar Levi

Bar-Ilan University, Ramat-Gan, Israel
`francesco.berti@biu.ac.il,itamar.levi@biu.ac.il`

Abstract. Passive (leakage exploitation) and active (fault injection) physical attacks pose a significant threat to cryptographic schemes. Although leakage-resistant cryptography is well studied, there is little work on mode-level security in the presence of joint faults and leakage exploiting adversaries. In this paper, we focus on integrity for authenticated encryption (AE).

First, we point out that there is an inherent attack in the fault-resilience model presented at ToSC 2023. This shows how fragile the freshness condition of a forgery is when faults are injected into either the tag-generation or the encryption algorithm. Therefore, we provide new integrity definitions for AE in the presence of leakage and faults, and we follow the atomic model, in which the scheme is divided into atoms (or components, e.g. a call to a block cipher) and allows the adversary to inject a fault only into the inputs of an atom. We envision this model as a first step for *leveled implementations* in the faults context, the granularity of atoms can be made finer or coarser (for example, instead of considering a call to a block cipher, we can consider atoms to be rounds of the block cipher). We hold the underlying belief that it would be easier to protect smaller blocks than a full scheme. The proposed model is very flexible and allows us to understand where to apply faults countermeasures (in some very interesting cases this model can reduce faults inside atoms to faults on their outputs, as we discuss).

We then show that an AE-scheme using a single call to a highly leakage-protected (and thus very expensive) component, CONCRETE (presented at Africacrypt 2019), maintains integrity in the presence of leakage in both encryption and decryption, and faults only in decryption. On the other hand, a single fault in encryption is enough to forge. Therefore, we first introduce a weaker definition (which restricts the meaning of freshness), *weak integrity*, which CONCRETE achieves even if the adversary can introduce faults in the encryption queries (with some restrictions on the number and type of faults). Finally, we provide a variant, CONCRETE2, which is slightly more computationally expensive, but still uses a single call to a strongly protected component that provides integrity in the presence of leakage and faults.

Keywords: AE · Fault Injection · Fault-resistance · Integrity · Leakage-resistance · Model-Level Security · Side Channels · SCA

1 Introduction

Classically, in the standard-model an adversary can attack a cryptographic scheme by choosing its inputs and seeing its answer, the so-called *black-box* model [32]. However, considering an adversary which has physical access to the device, she can perform more powerful attacks, both passive and active: On the one hand, passively, she can measure the physical quantities involved in the cryptographic computation, such as time, electromagnetic radiation, power consumption, etc. [33, 34, 41]; on the other hand, she can actively inject faults into the computation [18, 17, 2], obtaining critical information from the device's answers, either to retrieve secret-material (e.g., the key) [25, 30, 49] or enough information to forge it (e.g. by replacing the nonce [48]), see [47] for more details.

Finally, a combined attack can be performed and manifest a concrete threat [46, 45].

With respect to leakage-resistant symmetric cryptography, there is a large body of work that provides achievable security definitions for privacy [40], integrity [12, 13], and for AE [29, 3], and various constructions.

A particularly interesting line of research for construction is *leveled implementation*: As any mode-of-operation (mode-level) symmetric scheme is based on primitives such as block ciphers (BC) and hash functions (H), instead of protecting all primitives uniformly, one can design a scheme using weakly protected implementation for all of primitives and a very well protected for just one or a few [40, 24, 5]. Such an approach gained large interest and acceptance in the community and found to be particularly efficient because strong countermeasures against side-channel attacks are expensive, e.g. higher-order masking can be thousands of times slower (or energy hungry) [19, 28, 31, 50]. There are many examples, both based on (tweakable [1]) block ciphers [12, 13, 8, 37] or sponges [22, 4]. In all these constructions, the bulk of the construction is carried by the weakly protected implementations, and they use ephemeral keys (i.e. keys generated in that particular query and used only a few times), while the strongly protected implementations are reserved for a few calls (usually one or two) where the master key of the scheme is used. To study the integrity of a MAC (Message Authenticated Code) or a AE (Authenticated Encryption) scheme, there is a leakage model that works well with leveled implementations: the *unbounded leakage model* [12]. It assumes that all inputs (even the key) and outputs of the weakly implemented components are leaked, while the strongly protected components leak their inputs and their outputs, but not the secret-inputs (e.g., key): these components are modeled either as *leak-free* [12], i.e. except of the inputs (not the key) or the outputs of a block cipher (BC), no other information is obtained from the leakage, or *strongly unpredictable* ($\mathsf{SUP\text{-}L2}$) [7], i.e. given a leakage of a blockcipher $\mathsf{F}_k(\cdot)$ and its inverse $\mathsf{F}^{-1}(\cdot)$, it is hard to find an input/output pair (x, y) that is *fresh* (never received as an answer before) and *valid* ($y = \mathsf{F}_k(x)$). In the unbounded leakage model, it is possible to construct a MAC that provides authenticity when the tag-generation algorithm leaks, for example the well-known Hash-then-BC, i.e. $\tau = \mathsf{F}_k(\mathsf{H}(m))$, assuming a single call to a leak-free block cipher [40, 12]. On the other hand, decryption leakage is much more tricky, since the correct tag is often recomputed (and thus can be leaked), i.e. $\mathsf{Vrfy}_k(m, \tau)$ accepts if $\tau \overset{?}{=} \mathsf{Mac}_k(m)$. To avoid this, it has been proposed to use the inverse in the verification, by doing a check on a different value [13]: instead of comparing τ with the correct tag, $\mathsf{Vrfy}_k(m, \tau)$ inverts the tag and checks if $\tilde{h} = \mathsf{F}_k^{-1}$ is equal to the digest $h = \mathsf{H}(m)$. The idea is that $\tilde{h}$ cannot be used for forgery because it is random as the output of a leak-free block cipher, and the adversary has to find a pre-image for this value (and we can assume that the hash function is range-oriented pre-image resistant). In the random oracle model, the previous MAC provides integrity in the presence of leakage in both tag-generation and verification, even if we assume that F is $\mathsf{SUP\text{-}L2}$ [7]. To avoid using the random oracle model, which is undesirable and would be a too optimistic assumption, and security would then be impossible to achieve [10] (Asiacrypt 2021), a MAC, $\mathsf{LR\text{-}MAC1}$ was proposed which uses a *strongly unpredictable with leakage* tweakable-BC: The idea is to use a fixed input, 0^n, as the input and the message hash, $h = \mathsf{H}(m)$, as the tweak, i.e. $\tau = \mathsf{F}_k^h(0^n)$. This way, if $\mathsf{F}_k^{-1,h}(\tau) \neq 0^n$ during verification, the adversary cannot use the output of F^{-1} to forge.

Not surprisingly, the idea of using the inverse in decryption has also been used to build AE schemes, which provide authenticity with leakage in both encryption and decryption in the unbounded leakage model [13, 8, 4, 37]. Most schemes use 2 calls to the strongly protected implementation of a (tweakable) block cipher, one to authenticate either the

[1]A *tweakable block cipher* (TBC) is a block cipher which has an additional input, the *tweak*, for flexibility [36]: $\mathsf{F}_k^{tw}(x)$ where tw is the tweak.

message or the ciphertext, and another to generate a first ephemeral key which is used to encrypt the message either with a rekeying-based encryption scheme (e.g. the one proposed at CCS 2015 [40]) or with a sponge. At Africacrypt 2019, a leakage-resistant AE scheme $\mathsf{CONCRETE}$ (see Fig. 1 and Alg. 6) using a single call to a strongly protected TBC. $\mathsf{CONCRETE}$ provides integrity in the unbounded leakage model when both encryption and decryption leak [15]. It picks a random ephemeral key k_0, then computes a commitment on that key, then uses that first ephemeral key to encrypt using PSV (a leakage-resistant encryption scheme based on rekeying [40]) to obtain a ciphertext c, then it encrypts the ephemeral key k_0 with a TBC, where the tweak is the digest of the message, $h = \mathsf{H}(c)$, $c_{l+1} = \mathsf{F}_k^h(k_0)$. The adversary in the decryption first retrieves the key k_0, checks if the committing is correct, and then can retrieve m correctly from k_0.

Similarly to the leakage resistant schemes, it is possible to withstand fault attacks with specific countermeasures, see [6] for example. On the other hand, for leakage-resistant schemes, it is possible to design schemes that are inherently more secure, when working at the mode-level. However, regarding fault-resistance, the literature is much more scarce. There are two relevant fault models discussed so far, as detailed next.

In the first model, the adversary can inject a fault on a value kept in memory when it is used by the scheme, either setting it to a certain value or with a differential fault (moreover, this fault can be permanent or transient) [27]. The authors have also provided a fault-resilient signature-scheme and AE scheme which withstand fault in encryption. With a similar model [1], the security of hedged Fiat-Shamir signatures has been assessed against faults. In another model, equipped with fault injection and leakage collection capabilities, the adversary accumulates information, the so-called *accumulated interference* [26]. In [47] Saha et al. introduce the concept of fault-resilient PRF: the adversary can fault a real PRF, then, after she has done all the faulted queries she has to distinguish the real PRF from an ideal (clearly she is not allowed to repeat queries between the two different phases of her game). From this notion, they define authenticity in the presence of faults for both MAC and AE. In the fault resilience model [47], the adversary performs all the faulted queries before making any query to distinguish queries to the real scheme (that is, either to Mac and Vrfy, or to Enc and Dec in the AE case) from an ideal scheme (that is, either to an oracle which outputs a random value and another which always outputs reject). Moreover, the adversary can only inject faults in the direct queries (that is, to Mac or to Enc). They also provide a probabilistic MAC and a probabilistic AE scheme achieving the fault-resilience as defined. Finally, using the basic model developed in [27, 1], the authors of [9] extended it with the so-called *atomic model* where for example it is possible to divide a MAC into atoms (e.g. the hash, a call to BC) and assume that the adversary can only inject faults in the input of the atoms. Surprisingly, protecting against faults in verification is easier than in tag-generation: $\mathsf{LR\text{-}MAC1}$ achieves this security without any modification (even if the adversary obtains leakage in both tag-generation and verification in the unbounded leakage model), even if the adversary can fault every input of every atom (except the master key used by the leak-free TBC). However, if the adversary can inject faults into the tag-generation, the adversary can easily forge because she can choose the inputs to the leak-free TBC F_k and thus has oracle access to it.

To prevent such attacks, in [9], the authors either abandon the unbounded leakage model with two calls to a leak-free TBC, assuming that the output of the first leak-free TBC is not leaked, or they use a randomized version (there is an additional random input to Mac, making internal values random), but it withstands only differential faults.

1.1 Our contribution

In this paper, we first study the theoretical aspect of integrity in the presence of faults. We point out a subtle problem of the fault-resilient framework [47]: since one cannot prevent the adversary from faulting the return function of any algorithm, for example if the fault is in $\tau \leftarrow \mathsf{Mac}_k(m)$ by switching the first bit of τ, the resulted (m, τ) is considered fresh (since we did not get τ, but $\tau \oplus 10\ldots0$). This attack is clearly not significant since we can view (m, τ) as not fresh, which is indeed the case. However, it illustrates that the model should be studied carefully and precisely. In our perspective, this is an inherent security consideration that highlights how difficult and delicate it is to model security against faults and leakage attacks. Fortunately, in the atomic model, the previous attack cannot occur because the adversary can only fault the input of the atoms, and output of atoms only when they are also intermediate computations. For example, in LR−MAC1 the output τ is the output of the F_k atom, while in CONCRETE, the adversary can inject a fault in c when it is used as input to H, but as ciphertext, c is the output of the encryption part. Combining these observations, we first extend the atomic model to the AE case: we provide the integrity definition in the presence of faults and leakage. Moreover, we define a weaker version by which we do not allow the adversary to forge with any of the values that the ciphertext may take in encryption request owing to faults, denoted *weak integrity.*

Second, we study the integrity of CONCRETE in the unbounded leakage model in the presence of faults. We choose CONCRETE because it is a probabilistic scheme (which helps against leakage for direct queries) and uses only one leak-free call. Similar to LR−MAC1, an adversary cannot forge with any fault in the decryption (except on the master key). On the other hand, a single fault (even a differential) is enough for an adversary to forge. However, if the adversary can only inject one set fault (and any differential fault she wants) per encryption query, CONCRETE provides *weak integrity* as proven in this paper. This is consistent with [9], which suggests that for authenticity, the direct queries (either Mac or Enc) must be randomized, and the adversary should not be able to derandomize them with faults (i.e. choosing all inputs for some critical computations, thus having oracle access to those atoms). On the other hand, we do not have this requirement for the inverse queries (either Vrfy or Dec), where by using the inverse of a (T)BC, F_k^{-1}, we can prevent the adversary from forging even if she has oracle access to F_k^{-1}.

Third, we modify CONCRETE to withstand faults in encryption as well. We identify that CONCRETE authenticates any ciphertext produced by the encryption part (independent of the encrypted message), second, since $c_i = \mathsf{E}_{k_i}(p_B) \oplus m_i$, where k_i is the i^{th} ephemeral key, where p_B is a constant, thus an adversary can predict how modifying the message would modify the corresponding ciphertext (i.e, $c_i' = c_i \oplus \Delta$ encrypts $m_i' = m_i \oplus \Delta$). Therefore, in the proposed scheme we modify the encryption $c_i = \mathsf{E}_{k_i}(m_i)$ and in addition a value obtained by a MAC of the message m is encrypted, deriving CONCRETE2 (Fig. 3), full proofs and intuition are discussed in the paper. With these changes, CONCRETE2 can provide integrity even in the unbounded leakage model, where the adversary can inject any fault in decryption, and one set fault and any differential fault per encryption query (note that with 2 set faults, the adversary obtains oracle access to $\mathsf{F}_k^{\cdot}(\cdot)$ because she can sets both inputs of F_k, therefore she can choose k_0, compute the commitment, the MAC and the encryption of m from k_0, and the digest h of these values, then she forces, in an encryption query with other inputs, to compute $\mathsf{F}_k^h(k_0)$, so she can forge). Note that all faults discussed here are variable faults, e.g., up to n-bit faults where n is the variable size and previous constructions usually withstand a single fault per query (except [9]).

CONCRETE2 is a 2-pass scheme using a single call to a leak-free TBC, where these 2 passes cannot be done in parallel. However, notably CONCRETE2 may be very efficient in terms of performance in encryption, as compared to a 1-pass scheme, because the second pass may commence with a very small delay from the start of the first pass (quasi 1-pass). For example, the second pass, the one that computes h, digesting the ciphertext blocks, may begin as soon as at least one of their blocks is produced (consider a round-based or mode-of-operation based construction). Comparing with [47], their fault-resilient AE-scheme, MEM is a 3-pass scheme (in encryption quasi 2-pass), and can withstand a single fault per encryption query, and no fault in decryption. On the other, MEM provides privacy even if a single fault is injected in an encryption query, while, here we do not focus on this problem (however, we do suspect that CONCRETE2 also provides privacy in the presence of faults). Finally, another contribution we put forward is a variant of CONCRETE2 using the popular sponge construction.

We show for the first time that it is possible to construct a scheme in the unbounded leakage model that provides integrity even in the presence of faults, with modified and meaningful definitions adapted to the faulted setting.

2 Background

Notations We denote with $\{0,1\}^n$ the set of all the n-bit strings and $\{0,1\}^*$ the set of all finite strings, with $x\|y$ the concatenation of the two strings x and y, and with $|x|$ the length of the string x. We denote with $\pi_{n,l}(x)$ the n leftmost bits of the string x, and $\pi_{n,r}(x)$ the n rightmost bits. With $x \xleftarrow{\$} \mathcal{S}$, we denote that x is picked uniformly at random from the set $\mathcal{S}$, with $\pi_l(x)$ we denote the l-left bits of x, and with 0^n we denote a sequence of n zeros. When we *parse* a string x in n-bits blocks, we divide $x = (x_1, \dots, x_l)$ with $|x_1| = \dots = |x_{l-1}| = n$, and $|x_l| \leq n$, where $x = x_1\| \dots \|x_l$. When we *special parse*(j) a string x in n-bits blocks, we divide $x = (x_1, \dots, x_l)$ with $|x_1| = \dots = |x_{j-1}| = |x_{j+1}| = \dots = |x_l| = n$, and $|x_j| \leq n$, where $x = x_1\| \dots \|x_l$. A $(q_1, \dots, q_d, t)$-adversary A is a probabilistic algorithm, which is allowed q_i queries to oracle $\mathcal{O}_i$ and runs in time bounded by t. With $y \leftarrow \mathsf{A}^{\mathcal{O}_1,\dots,\mathcal{O}_d}(x)$, we denote that adversary A on input x, with access to oracles $\mathcal{O}_1, \dots, \mathcal{O}_d$ outputs y. Let Alg be a probabilistic algorithm. With $y \leftarrow \mathsf{Alg}(x)$, we denote that we ran Alg on input x obtaining y. If we want to explicitly denote the randomness r used in the previous execution, we write $y \leftarrow \mathsf{Alg}(x; r)$.

2.1 Cryptographic primitives- Hash and (Tweakable) Block-ciphers

In our schemes, we will use hash functions and tweakable blockciphers (TBC).

We use hash functions to compress data. For an adversary, it should be difficult to find a *collision*, (2 different inputs with the same output):

Definition 1. A hash function $\mathsf{H} : \mathcal{HK} \times \{0,1\}^* \rightarrow \{0,1\}^n$ is (t,ϵ)-*collision resistant* (CR) if $\forall$ t-adversaries A: $\Pr[\ (m_0, m_1) \leftarrow \mathsf{A}(s)$ s.t. $\mathsf{H}_s(m_0) = \mathsf{H}_s(m_1),\ m_0 \neq m_1 \mid s \xleftarrow{\$} \mathcal{HK}] \leq \epsilon$.

We use (tweakable) block-ciphers to produce pseudorandom random values.

Definition 2 ([36])**.** A *tweakable block-cipher* (TBC) $\mathsf{F} : \mathcal{K} \times \mathcal{TW} \times \{0,1\}^n \rightarrow \{0,1\}^n$ is a family of permutations, where $\mathsf{F}(k, tw, \cdot) : \{0,1\}^n \rightarrow \{0,1\}^n$ is a permutation $\forall (k, tw) \in \mathcal{K} \times \mathcal{TW}$. For simplicity, we often denote $\mathsf{F}_k^{tw}(x)$, for $\mathsf{F}(k, tw, x)$. With $\mathsf{F}_k^{-1,tw}$, we denote the inverse of F_k^{tw}.
A *block-cipher* (BC) is a TBC without any tweak, that is $|\mathcal{TW}| = 1$.

Definition 3. A TBC $\mathsf{F} : \mathcal{K} \times \mathcal{TW} \times \{0,1\}^n \rightarrow \{0,1\}^n$ is a $(q, t, \epsilon_{\mathsf{sTPRP}})$-sTPRP (*strong Tweakable Pseudo Random Permutation*) if for any (q_E, q_I, t)-adversary A

$$|\Pr[1 \leftarrow \mathsf{A}^{\mathsf{F}_k^{\cdot}(\cdot), \mathsf{F}_k^{-1,\cdot}}] - \Pr[1 \leftarrow \mathsf{A}^{\mathsf{f}^{\cdot}(\cdot), \mathsf{f}^{-1,\cdot}}] \leq \epsilon$$

where $q_E + q_I \leq q$, $k \xleftarrow{\$} \mathcal{K}$, and $\mathsf{f} \xleftarrow{\$} \mathcal{TPERM}$, where $\mathcal{TPERM}$ is the set of the functions with the same signature as F_k, that is the functions $\mathsf{f} : \mathcal{TW} \times \{0,1\}^n \rightarrow \{0,1\}^n$ s.t. $\forall tw \in \mathcal{TW}$ $\mathsf{f}^{tw}(\cdot) : \{0,1\}^n \rightarrow \{0,1\}^n$ is a permutation. If F is a BC, we have strong PRP. If $q_I = 0$, F is a TPRP if F is a TBC, and PRP if F is a BC (and PRF if we remove the constraint that f is a permutation and f is picked from the set of functions $\mathsf{f} : \{0,1\}^n \rightarrow \{0,1\}^n$).

In some cases, we need to model the block-cipher as an ideal -cipher

Definition 4 ([20])**.** A block-cipher $\mathsf{E} : \mathcal{K} \times \{0,1\}^n$ is an *ideal cipher* if it has been chosen uniformly at random among all block-ciphers with the same signature.

That is, an ideal cipher is a family of $|\mathcal{K}|$ independent permutations. When an ideal cipher E is used, even only by the oracles, the adversary is always allowed to query directly it choosing both the key and the input for each call she does.

2.2 MACs, and (Authenticated) Encryption Schemes

In the previous section, we have formally defined hash functions, and (tweakable) blockciphers. These are the building blocks of the MAC, encryption and AE primitives, which we define here. We use a Message Authentication Code, MAC, to authenticate (see Def. 12, App. A.1), while ivE schemes for encryption (see Def. 14, App. A.3).

The cryptographic primitive that provides privacy and authenticity is *authenticated encryption* (AE). Following [15, 47] we use a probabilistic encryption scheme. The syntax can be found in App. A.2. Note that, differently from the standard nAE definition, see [42, 44], we allow AEnc to be probabilistic as in [47]. We leave the standard security notion [39], in the black-box model (that is when the adversary can choose the inputs and only sees the outputs of their oracles), Def. 15 to App. A.4.

2.3 Leakage

The previous definitions are *black-box*, because the adversary chooses the inputs of their oracles and sees only their outputs. On the other hand, cryptographic algorithms are usually implemented on electronic devices. Thus, an adversary can measure the physical quantities involved during the computations, as instantaneous power consumption, time, electromagnetic/ radio-frequency (RF) radiation etc. [33, 34, 41, 21]. This is the so-called *leakage* which can give useful information to the adversaries.

Notations. We denote that an oracle O leaks, adding the suffix L, that is with OL. The leakage of the implementation of the oracle $\mathsf{O}_k(x)$ on input (x) is given by the *leakage function* $\mathsf{L}_{\mathsf{O}}(x;k)$. The leakage function represents the information that the adversary gets via leakage from the computation of oracle on input x. When $\mathsf{O}_k(x)$ is a probabilistic oracle, we assume that it has access to a random tape carrying the value r, the leakage function takes also r as input. Similar definitions and notations may be found in [11].

2.4 Integrity in the presence of leakage.

When the adversary not only chooses the inputs of the oracles and sees their outputs, but also has access to the leakage when they perform their computations, it is better not to have a comprehensive security definition for both privacy and authenticity. In fact, if we naively modify Eq. 2 for AE-*security* (Def. 15 in App. A), adding the leakage of all the oracles, we should be able to compute a leakage for the ideal primitives ($ and $\perp$), which is indistinguishable from the leakage of the real primitives (AEnc and ADec). As discussed in [38] this is quite cumbersome and complex. Thus, it is possible to give different definitions for authenticity and privacy in presence of leakage [12]. Note that these two definitions and security goals are inherently different, for authenticity we need that it would be difficult to produce a complete (fresh and) valid ciphertext, while for privacy we require that the ciphertext will give no information about its plaintext (except for the message length). Berti et al. [13] introduced a security definition for integrity in the presence of leakage. We give the definition modified for the case of probabilistic encryption [15]:

Definition 5. An AE-scheme $\Pi = (\mathsf{Gen}, \mathsf{AEnc}, \mathsf{ADec})$ provides (q_E, q_D, t, ϵ)-*ciphertext integrity with leakage in encryption and decryption*, CIL2, if for any (q_E, q_D, t)-adversary A

$$\Pr[\ (a,c) \leftarrow \mathsf{A}^{\mathsf{AEncL}_k(\cdot,\cdot),\mathsf{ADecL}_k(\cdot,\cdot)} \mid \text{s.t. } (a,c) \text{ is fresh and valid}] \leq \epsilon. \quad (1)$$

With *fresh* we denote that c has never been obtained as an answer from a $\mathsf{AEncL}_k(a,m)$ query for any m, and with *valid* that $\mathsf{ADec}_k(a,c) \neq \perp$. (CIL means that only AEnc leaks).

Modelling leakage - the unbounded leakage model. The previous definition (Def. 5) assumes that when the adversary asks a query to either the AEnc_k or ADec_k oracle, she receives the output and the leakage of its computation. As in all definitions involving leakage, the leakage function must be realistic, to have a security definition that is relevant in practice, but not too generous, (for example assuming that the key is leaked), otherwise it is impossible to provide security.

For integrity, we use the *unbounded leakage model* [12] in which we assume that the scheme uses a *leveled implementation*: the primitives (hash functions, TBC, BC) are divided into two types: *strongly protected primitives* which are modeled as *leak-free*, that is, they leak nothing about their internals, and *weakly protected primitives.* When a strongly protected primitive is used, $\mathsf{L_O}$ gives its inputs and output, while when a weakly protected primitive is used, its inputs, output *and* its secret key. It can be schematically illustrated: we use green color for the inputs and outputs of the scheme, orange for the internal value computed and the key of the weakly protected primitives, and red for the keys of the strongly protected primitives (see for example Fig. 1). In the unbounded leakage model, $\mathsf{L_O}$ gives all the orange values.

Note that strongly protected components are usually much slower than unprotected (or slightly protected) components [28, 31, 50, 19], thus, when we desire to reduce their usage.

2.5 CONCRETE

Pereira et al. [40] proposed a scheme which is CPAL-secure, PSV (see Alg. 4 in App. J). It used a leveled implementation. They use the master key k to create the first ephemeral key k_1 from a random value called iv via a strongly protected BC F (modeled as leak-free): $k_1 = \mathsf{F}_k(\mathsf{iv})$. This ephemeral key is used twice with a weakly protected BC, E, once to create a new ephemeral key, $k_2 = \mathsf{E}_{k_1}(p_A)$ (*refreshing* it), and the other time to create a pseudo-random value which is XORed to the first message block m_1 (we parse the message in n-bit blocks, where n is the blocks-size of E), resulting in the first ciphertext block $c_1 = \mathsf{E}_{k_1}(p_B) \oplus m_1$. Then, we iterate, creating new ephemeral keys, and encrypting new blocks of the message until we have encrypted the last message block m_l. The decryption algorithm takes as input $(\mathsf{iv}, c_1\|..., c_l)$, recomputes the first ephemeral key k_1 from iv, and, then, it can correctly decrypt.

Berti et al. [13], based on PSV, proposed a scheme, EDT (Encrypt-then-Tag), which is CIL2-secure in the unbounded leakage model [2]. They started from PSV and, to authenticate the ciphertext, they used the basic hash-then-MAC: they hash the ciphertext and reusing the master key with a strongly protected TBC to compute the tag $\tau = \mathsf{F}^1_k(\mathsf{H}(c_1\|...\|c_l))$ where $m = (m_1, ..., m_l)$. In decryption, instead of recomputing the tag $\tilde{\tau}$ and checking whether τ is correct (that is, $\tau \overset{?}{=} \tilde{\tau}$), to prevent leakage of $\tilde{\tau}$, they do not compute it, instead they invert the TBC, computing $\tilde{h} = \mathsf{F}^{-1,1}_k(\tau)$, and check if $\tilde{h} \overset{?}{=} h$, with $h = \mathsf{H}(c_1\|...\|c_l)$. The idea is that to use a $\tilde{h}$, which is leaked, for a forgery, an adversary has to find a pre-image for it, which should be difficult for a good hash function since $\tilde{h}$ is random (we need to assume that H is range-oriented pre-image resistant [13]).

To reduce the number of times the strongly protected TBC is used Berti et al. [15, 14] proposed a new AE-construction: CONCRETE (Commit-Encrypt-Send-the-Key) (see Fig. 1, and Alg. 6, App. J). Instead of computing the first ephemeral key k_0, they pick it randomly, then, they do a commitment on the key, with $c_0 = \mathsf{E}_{k_0}(p_B)$, and refresh it with $k_1 = \mathsf{E}_{k_0}(p_A)$. Then, to encrypt the message, they use PSV with k_1 as the first ephemeral key. Finally, to send the first ephemeral key k_0, they used the strongly protected TBC and the master key k, with $c_{l+1} = \mathsf{F}^h_k(k_0)$, with $h = \mathsf{H}'(c_0\|c_1\|...\|c_l, a)$ a multi-input collision resistant [23] (see Def. 21, App. D.1. There, we also give a possible hash function). In decryption, from the ciphertext $c = (c_0, c_1, ..., c_l, c_{l+1})$, the decryption algorithm, first, retrieves

[2]We give a simplified version, for the full details see the original paper.

$k_0 = \mathsf{F}_k^{-1,h}(c_{l+1})$, with $h = \mathsf{H}'(c_0\|c_1\|...\|c_l, a)$, checks correctness of the commitment, and proceeds to decrypt. CONCRETE has been proved to be AE secure in the black-box model, CIL2 in the unbounded leakage model assuming that F is leak-free and CPAL2 secure [15]. For completeness, we give the results in App. B.

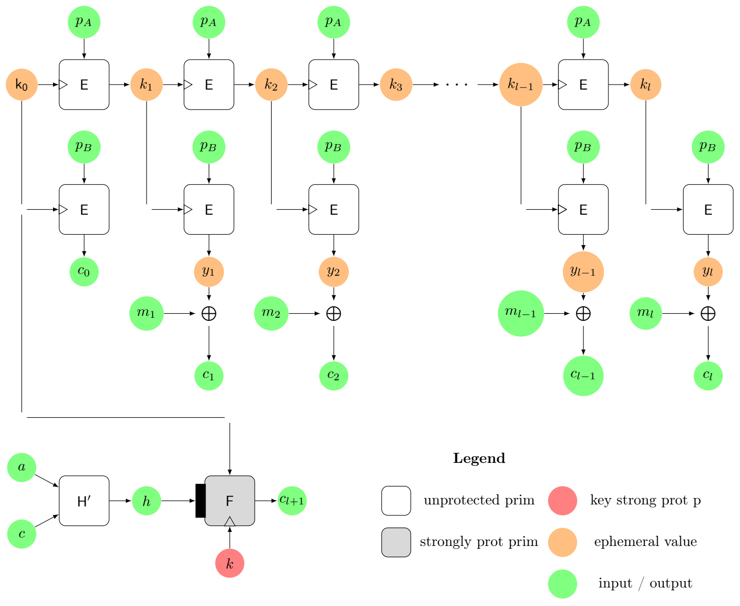

Figure 1: CONCRETE [15]. $c = c_0\|...\|c_l$.

2.6 Integrity in presence of faults and leakage

Here we present the two approaches for integrity (and leakage) in the presence of faults: [9] and [45]. In both definitions, the key-generation algorithm is assumed to be leakage-resistant (since it is done only once and/or can be protected [3]).

Notations. We denote that an oracle can be faulted adding the suffix Fa to it. The fault is seen as an additional input of the oracle. Finally, with $\mathsf{A}^{\mathsf{L},\mathsf{Fa}}$ we denote an adversary which holds a leaky and faulty model.

2.6.1 Atomic model.

Berti et al. [9] aim to start from the standard black-box authenticity definition of a MAC and add leakage and faults.

They assume that the adversary must choose all the faults *before* the start of the actual computation. Thus, when the adversary wants to see the output (and the leakage) of a faulted computation of the implementation of algorithm Algo, she chooses the inputs (the leakage) and the fault $\mathsf{Fa}_{\mathsf{Algo}}$ at the same time [4].

[3]Since our adversary is very strong, some assumptions and security mechanisms will be required, otherwise security will either be impossible with the proposed tools used or simply too expensive; we believe these are standard scenarios and questions asked by security architects in the different implementation levels. For example, taking the master key. We may assume that we implement the (T)BC with an implementation where the key is either: (1) hard-coded, (2) partially hard coded (3) uses a secured memory or derived from a secure primitive (perhaps with a TPM) (4) may be generated aided a PUF mechanism.

[4]They assume that the adversary cannot see the leakage of part of the computation and then chooses the faults accordingly. This is a strong requirement. At a lower level, this requirement has not been asked by Berndt et al. [6].

To represent the fault Fa that the adversary wants to inject they introduce the *dependency matrix*. If an algorithm Algo from inputs $(x_1, x_2, ..., x_n)$ outputs y, we can reduce its implementation to a sequence of steps, called *atoms*: $f_1(x_1, x_2, ..., x_n) = z_1,\ f_2(x_1, x_2, ..., x_n, z_1) = z_2,\ \dots,\ f_N(x_1, x_2, ..., x_n, z_1, ..., z_{N_1}) = z_N$. where f_i is any deterministic function or the random picking. Note that this description is not unique. As an example, we describe $\mathsf{CONCRETE}$ with these functions in Sec. 4.1, and of $\mathsf{LR\text{-}MACr}$ in the following of this section. The output y is obtained from the outputs of the f_is choosing some of them (and eventually rearranging) via the function Select. This function depends only on the length of the inputs. That is, $y = \mathsf{Select}(y_1, ..., y_N)$ with $y = y_{i_1} \| \dots \| y_{i_L}$ with $L = g(|x_1|, \dots, |x_n|)$, where $g : \mathbb{N}^n \to \mathbb{N}$, and $i_1, \dots, i_h$ are in $[N]$ (we can see the indexes $i_1, \dots i_L$ as the output of a function $g_{out} : \mathbb{N}^n \to \mathcal{P}([N])$, which takes as input only the lengths of the inputs).

But, usually, we do not effectively use all the inputs for the f_is. For example, in $\mathsf{CONCRETE}$ when computing $k_{i+1} = \mathsf{E}_{k_i}(p_A)$, thus all inputs and all ephemeral values are ineffective except for k_i. The *dependency matrix* is built from the inputs of the f_is, by writing *only* the effective inputs[5] and putting the string ϵ for all others. As an example, we give the dependency matrix for $\mathsf{CONCRETE}$ in Eq. 3, 4 and of $\mathsf{LR\text{-}MACr}$ in App. E.

They assume that the adversary can only set faults between the atoms. Considering *leveled implementations* in the faults context, the granularity of atoms can be made finer or coarser as discussed below. We believe that it would be easier to protect smaller blocks than a full scheme. This model is quite flexible and allows us to understand where faults countermeasures are needed. Moreover, the adversary is not allowed to input any fault in the selection function Select. That is, she can only fault the inputs of the atoms. Thus, we can represent the fault $\mathsf{Fa}_{\mathsf{Algo}}$ with the *fault matrix* which shows how each input is modified. That is, if in f_i, the input x_j is replaced with a stuck-at fault with x'_j, the fault matrix represents this setting the i,j-th element to x'_j, if x_j is faulted with the differential fault Δ, we set the i,j-th element to $\cdot \oplus \Delta$, or if there is no fault, we set it to $\cdot$.

Clearly, we cannot achieve security if we allow the adversary to fault all inputs and outputs (for example, the adversary can set one bit of the key to 0 and see if it affects the output). Thus, there are inputs of certain atoms, which cannot be faulted, and we assume that these inputs are protected[3]. When we assume that the x_i input of the jth atom is protected and cannot be faulted, we denote this setting the (j,i)th element of the fault matrix to $\perp$; if it can only be faulted with differential faults, we use $\perp/\oplus$. The set of all these protected inputs is denoted with $\mathcal{I}$. Additionally, we can bound the number and/or the type of faults the adversary can do. The set of all admissible faults is denoted with $\mathcal{F}$.

Finally, we do not allow the adversary to *always* replace an input x_i, when it is effectively used, with another input, x'_i (that is, if with the fault Fa we have turned the computation of $\mathsf{Mac}_k(m)$ in the computation of $\mathsf{Mac}_k(m')$ by replacing m with m' every time m is used, then, the output $\tau \leftarrow \mathsf{Mac}_k(m', \mathsf{Fa})$ is clearly s.t. $\mathsf{Vrfy}_k(m', \tau) = \top$, but we consider (m', τ) an invalid forgery).

Note that two different implementations of the same algorithm can be divided into different atoms, thus there are two different fault matrices (capturing the dependency of faults from the implementations) [9].

Having presented the model, we can give Berti et al.' definition of security in the presence of leakage and faults:

Definition 6. A $(\mathcal{I}_{\mathsf{Mac}}, \mathcal{I}_{\mathsf{Vrfy}})$-protected implementation of MAC, with leaking function pair $\mathsf{L} = (\mathsf{L}_{\mathsf{Mac}}, \mathsf{L}_{\mathsf{Vrfy}})$ and fault admissible sets $\mathcal{F} = (\mathcal{F}_{\mathsf{Mac}}, \mathcal{F}_{\mathsf{Vrfy}})$, is $(q_{FL}, q_M, q_V, t, \epsilon)$ *strongly-existentially unforgeable against $\mathcal{F}$-fault-then-leakage attacks in tag-generation and verification (*$\mathsf{SUF\text{-}FL2}$*)* if for all (q_{FL}, q_M, q_V, t)-adversaries $\mathsf{A}^{\mathsf{L},\mathsf{Fa}}$ we have

$$\Pr[\mathsf{SUF\text{-}FL2}_{\mathsf{MAC},\mathcal{F},\mathsf{L},\mathsf{A}} \Rightarrow 1] \le \epsilon$$

[5]We say that the input y is *ineffective* for the function $f(x,y)$, if $\forall x, y, y'\ f(x,y) = f(x,y')$. All the other inputs are effective.

Table 1: The SUF−FL2 experiment. To simplify the description of the experiment, we have omitted the checks that $\mathsf{Fa} \in \mathcal{F}_M$ (resp. $\mathcal{F}_V$) during tag-generation (resp. verification) queries and $\mathsf{Fa} \in \mathcal{F}_F$ for the output.

```
The SUF−FL2_{MAC,F,L,F,A^{LFa}} experiment

Initialization:
    $k \leftarrow \mathsf{Gen}$
    $\mathcal{S} \leftarrow \emptyset$

Finalization:
    $(m, \tau) \leftarrow \mathsf{A}^{\mathsf{L},\mathsf{Fa},\mathsf{MacLFa}_k,\mathsf{VrfyLFa}_k}$
    If $(m, \tau) \in \mathcal{S}$ or $\mathsf{Vrfy}_k(m, \tau) = \perp$
        Return 0
    Return 1

Oracle $\mathsf{MacLFa}_k(m, \mathsf{Fa})$:
    $\tau = \mathsf{Mac}_k(m, \mathsf{Fa})$
    $\mathcal{S} \leftarrow \mathcal{S} \cup (m, \tau)$
    Return $(\tau, \mathsf{L}_M(\mathsf{Mac}_k(m, \mathsf{Fa})))$

Oracle $\mathsf{VrfyLFa}_k(m, \tau, \mathsf{Fa})$:
    $\mathsf{ans} = \mathsf{Vrfy}_k(m, \tau, \mathsf{Fa})$
    $\ell = \mathsf{L}_V(\mathsf{Vrfy}_k(m, \tau, \mathsf{Fa}))$
Return $(\mathsf{ans}, \ell)$
```

where the $\mathsf{SUF{-}FL2}_{\mathsf{MAC},\mathcal{F},\mathsf{L},\mathsf{A}}$-experiment is defined in Tab. 1.

In App. E, we give an example of application of the atomic model to a MAC [9].

Observations. First, we observe that the atoms could be finer or coarser (we can consider atoms, for example, a call of a blockcipher, or its rounds or even the gates). Moreover, this model covers both *transient* (faults which modify a value for a single computation) and *persistent* (faults which modify a value for all subsequent computations) faults. In addition, when all the inputs of a component are known before the start of the computation any (deterministic) fault inside them can be seen as a fault on its output (e.g., considering the well-known MAC Hash-then BC if we compute $\mathsf{Mac}_k(m) = \mathsf{F}_k(\mathsf{H}(m))$, with F a BC and H a hash function, if we consider H and F the atoms of this computation, since the adversary chooses m, any faults in the computation of $\mathsf{H}(m)$ can be replaced by a correspondent fault on $h = \mathsf{H}(m)$ in the computation of $\tau = \mathsf{F}_k(m)$). *Finally, when the forgery of the adversary is checked, no faults are allowed, because if the adversary can force the* Vrfy *algorithm to output always* $\top$*, no cryptographic countermeasure can be useful.*

2.6.2 Fault resilient PRF (frPRF), MAC, encryption scheme and frAE

Saha et al. [47] introduced the notion of fault-resilient PRF (frPRF). They assume first, that every faulted query does not leak more than one correct couple (input/output), that is, $n^i_{\mathsf{pre}} \leq 1$ and after having done *all* faulted queries, for any fresh query, the outputs from F_k are indistinguishable from random ones. We leave the formal definition (Def. 18) to App. A.6. Similarly, they define a fault-resilient random oracle (which gives random outputs for fresh inputs even if the adversary has faulted previous queries).

When we move from a PRF to a MAC, we have to consider that we have to forge (that is, distinguishing if the last oracle is implemented with Vrfy_k or $\perp$) and we are not interested in the randomness of the output of Mac_k. Thus, we have the frMAC definition:

Definition 7. A $(q_{Fa}, q_M, q_D, t, \epsilon)$*-fault resilient* MAC *(*frMAC*)* if $\forall$ (q_{Fa}, q_M, q_V, t)-adversary A

$$|\Pr[\mathsf{A}^{\mathsf{MacFa}_k,\mathsf{Mac}_k,\mathsf{Vrfy}_k} \Rightarrow 1] - \Pr[\mathsf{A}^{\mathsf{MacFa}_k,\$,\perp} \Rightarrow 1]| \leq \epsilon,$$

where the adversary is not allowed to forward queries between different oracles. That is, there is a list $\mathcal{S}$ which contains all the couples (m, τ), with m the queries asked to Mac_k or \$, and τ is the oracle's answer, and a list $\mathcal{S}_{flt}$ of all queries to MacFa_k which for every query on input m, contains the couple message (m', τ), with m' any value that m takes via fault, such that $\mathsf{Vrfy}_k(m', \tau) \neq \perp$; the adversary cannot ask to Vrfy_k any query in $\mathcal{S} \cup \mathcal{S}_{flt}$ and to the $\mathsf{Mac}_k/\$$ oracle a query on input m, if there is a couple (m, τ) in $\mathcal{S}_{flt}$ for any τ.

Note that to MacFa_k oracle the adversary can ask queries with no faults. All the queries to $\mathsf{Mac}_k\mathsf{Fa}$ must be done *before* the first $\mathsf{Mac}_k/\$$ oracle query.

Similar definitions of fault-resilient encryption schemes (Def. 19), AE (Def. 20) can be found in App. A.6; as well as a frMAC (frMAC) and a fault-resilient AE-scheme (MEM).

3 Integrity vs. faults

This section introduces our integrity framework and definition in the presence of faults and leakage. To motivate it, we start by exposing a trivial fault attack that is not covered by the frMAC and frAE security definitions (Sec. 2.6.2, and App. A.6).

3.1 A trivial attack on frMAC and frAE

In our opinion, the frMAC (and frAE) framework is not completely suited to provide security because it does not cover a trivial attack (which depends solely on the framework, and not, in our opinion, on the inherent security of a scheme). We exploit the fact that in the frMAC security definition, for a faulted query on input (m, Fa), we do keep in memory the couples (m', τ), s.t. $\mathsf{Vrfy}_k(m', \tau) = \top$, with $\tau \leftarrow \mathsf{Mac}_k(m, \mathsf{Fa})$ and m' is any value that m takes via fault (Sec. 2.6.2), but we *do not* keep in memory the couples (m', τ'), s.t. $\mathsf{Vrfy}_k(m', \tau') = \top$, where m' and τ' are *any* value that m *and* τ takes via fault.

This can easily be exploited: Let us query Mac_k on input (m, Fa), where Fa is the flipping of the first bit of the tag τ when it is output (that is, we inject a fault in the Return part of the algorithm). Thus, the faulted computation outputs $\tau' = \tau \oplus 1\|0^{|\tau|-1}$ instead of the correct tag τ. Now, we forge Mac with (m, τ). Obviously, $\mathsf{Vrfy}_k(m, \tau) = \top$, thus our forgery is valid. Moreover, it is fresh, since we only deem not-fresh queries of the type (m', τ') s.t. $\mathsf{Vrfy}(m', \tau') \neq \perp$, with m' any value that the variable assumes via faults. In our case, there are no such couples. Note that the previous attack does not use any information learned via leakage and can be done against any MAC scheme.

Crucial observations. The same attack can be done to a frAE-secure encryption scheme when the tag (or any block of the ciphertext) is computed. The previous attack may be deemed trivial, but nonetheless exposes a hole in the security model of [47]. In our opinion, the previous attack does not pose any security problem, and it is completely acceptable to consider the previous forgery not fresh, making the adversary unable to win with such a forgery. But the model should cover such a situation.

This attack shows the complexity and subtility of modeling security against fault attacks when the adversary can inject faults into the Mac *or* Enc *algorithm.* [6]

The previous attack may not happen in the atomic model (Sec. 2.6.1), because we assume that the output of the scheme is formed by rearranging (and dropping) the outputs of the atoms via the Select function. In particular, for LR-MAC1, this cannot happen since the tag is the output of the block-cipher F.

3.2 Security definitions for integrity in the presence of faults and leakage

Here, we give our integrity definition for AE in the presence of faults and leakage. Our model assumes that the key generation does not leak and cannot be faulted, as in all leakage-resilient and fault-resilient models [40, 3, 12, 29, 27, 1, 9, 47].

In view of the attack explained in the previous section, we follow the Berti et al. [9] framework (see Sec. 2.6). Adapting Def. 6 to the AE syntax we obtain:

[6]It is out of the paper's scope, but modeling such attacks is challenging in defining security notions as CCA with faults, when we want to prevent the adversary from forwarding queries between her oracles.

Table 2: The $\mathsf{CIL2F2}$ experiment. To simplify the description of the experiment we have omitted the checks that $\mathsf{Fa} \in \mathcal{F}_E$ (resp. $\mathcal{F}_D$) during encryption (resp. decryption) queries and $\mathsf{Fa} \in \mathcal{F}_F$ for the output.

The $\mathsf{CIL2F2}_{\Pi,\mathcal{F},\mathsf{L},\mathcal{F},\mathsf{A}^{\mathsf{LFa}}}$ experiment	
Initialization:	Oracle $\mathsf{AEncLFa}_k(m, \mathsf{Fa})$:
$k \leftarrow \mathsf{Gen}$	$c = \mathsf{AEnc}_k(a, m)$
$\mathcal{S} \leftarrow \emptyset$	$(a', c) \leftarrow \mathsf{Faulted}(a, m, \mathsf{Fa})$
	$\mathcal{S} \leftarrow \{(a', c)\} \cup \mathcal{S}$
Finalization:	Return $(c, \mathsf{L}_E(\mathsf{AEnc}_k(a, m, \mathsf{Fa})))$
$(a^*, c^*) \leftarrow \mathsf{A}^{\mathsf{L},\mathsf{Fa},\mathsf{AEncLFa}_k,\mathsf{ADecLFa}_k}$	
If $(a^*, c^*) \in \mathcal{S}$ or $\mathsf{ADec}_k(a^*, c^*) = \perp$	Oracle $\mathsf{ADecLFa}_k(a, c, \mathsf{Fa})$:
Return 0	$\mathsf{ans} = \mathsf{ADec}_k(a, c, \mathsf{Fa})$
If invalid $= 1$ Return 1	$\ell = \mathsf{L}_V(\mathsf{Vrfy}_k(m, \tau, \mathsf{Fa}))$
Return 1	Return (ans, ℓ)

Definition 8. A $(\mathcal{I}_{\mathsf{AEnc}}, \mathcal{I}_{\mathsf{ADec}})$-protected implementation of the AE-scheme $\Pi = (\mathsf{Gen}, \mathsf{AEnc}, \mathsf{ADec})$ with leaking function pair $\mathsf{L} = (\mathsf{L}_{\mathsf{AEnc}}, \mathsf{L}_{\mathsf{ADec}})$ and fault admissible sets $\mathcal{F} = (\mathcal{F}_E, \mathcal{F}_D)$ is $(q_{FL}, q_E, q_D, t, \epsilon)$ *ciphertext integrity in the presence of leakage and faults against $\mathcal{F}$-fault-then-leakage attacks in encryption and decryption (*$\mathsf{CIL2F2}$*)* if for all (q_{FL}, q_E, q_D, t)-adversaries $\mathsf{A}^{\mathsf{L},\mathsf{Fa}}$ we have: $\Pr[\mathsf{CIL2F2}_{\mathcal{F},\mathsf{L},\mathsf{A}} \Rightarrow 1] \leq \epsilon$.
Where the $\mathsf{CIL2F2}_{\Pi,\mathcal{F},\mathsf{L},\mathsf{A}}$-experiment is defined in Tab. 2, and $\mathsf{Faulted}$ is an algorithm that takes as input all the values that a assumes during the encryption and outputs the couple (a', c) s.t. $\mathsf{ADec}_k(a', c) \neq \perp$. Moreover, if there is more than one valid couple Fa blocks the execution and returns the flag invalid which makes the adversary wins the experiment. We may add a suffix $\mathsf{-de}$ or $\mathsf{-dd}$ or d to denote that we allow only differential faults in encryption or in decryption or in both, respectively.

The fact that the adversary can win, if she forces $\mathsf{Faulted}$ to output two valid couples (a', c), and (a'', c'') (where a' and a'' are two different values that a assumes due to the faults), has been inspired by [27].

The q_{FL} queries are for the adversary to model the leakage in the presence of faults. Thus, as in [40], the adversary chooses all the inputs and the key. In the unbounded leakage model, where the strongly protected components are modeled as leak-free, there is nothing to model (because the adversary cannot obtain from leakage more than what we assume she can obtain, that is all inputs and outputs of every component except for the key of the leak-free), thus, we omit the q_{FL} queries.

Looking ahead, since $\mathsf{CONCRETE}$ uses only once a. Thus, according to the atomic model, the adversary cannot fault a, $\mathsf{Faulted}$ will always answer (a, c), and the adversary cannot win making $\mathsf{Faulted}$ obtaining two different valid couples (a', c), and (a'', c). The decoupling attack [45] is covered by the previous definition.

On the other hand, we may consider the previous security definition too strict. In fact, an adversary could forge simply taking some values that the ciphertext assumes. That is, let $c = \mathsf{Select}(y_1, ..., y_m)$, the forgery will be (a', c') with a' be one of the values that a assume during one encryption query, say the ith, and $c' = \mathsf{Select}(y'_1, ..., y'_N)$ with y'_j one of the value that y_j during the ith encryption query, as the decoupling attack does. We can be satisfied with a scheme which is forgeable only with such forgeries, as long an adversary is only able to find a single valid forgery per encryption query. Thus, we introduce the weakly integrity in presence of faults and leakage.

Definition 9. A $(\mathcal{I}_{\mathsf{AEnc}}, \mathcal{I}_{\mathsf{ADec}})$-protected implementation of the AE-scheme $\Pi = (\mathsf{Gen}, \mathsf{AEnc}, \mathsf{ADec})$ with leaking function pair $\mathsf{L} = (\mathsf{L}_{\mathsf{AEnc}}, \mathsf{L}_{\mathsf{ADec}})$ and fault admissible sets $\mathcal{F} = (\mathcal{F}_E, \mathcal{F}_D)$ is $(q_{FL}, q_E, q_D, t, \epsilon)$ *weak ciphertext integrity in the presence of leakage and* *faults against $\mathcal{F}$-fault-then-leakage attacks in encryption and decryption (*$\mathsf{wCILF2}$*)* if for all (q_{FL}, q_E, q_D, t)-adversaries $\mathsf{A}^{\mathsf{L},\mathsf{Fa}}$ we have

$$\Pr[\mathsf{wCILF2}_{\mathcal{F},\mathsf{L},\mathsf{A}} \Rightarrow 1] \leq \epsilon$$

where the $\mathsf{wCILF2}_{\Pi,\mathcal{F},\mathsf{L},\mathsf{A}}$-experiment is defined in Tab. 3 (App. J), and $\mathsf{Faulted}$ is an algorithm that takes as input all the values that a, and c assume during the encryption and outputs the couple (a', c') s.t. $\mathsf{ADec}_k(a'.c') \neq \perp$.

With respect to similar definitions (for example [27, 45]) we do not ask that for each encryption query, there is only a valid decryption query that can be produced, but simply that the adversary cannot find two valid forgeries. *That is, we move from an existential condition to a computational one. We believe that for security this is a sufficient condition, since the problem is not if two such forgeries exist, but if an adversary can find them.*

4 The integrity of CONCRETE

This section is devoted to study the integrity of CONCRETE in the presence of faults and leakage. First, we apply the atomic model to CONCRETE. Second, we prove that in the atomic model, if the adversary is not allowed to insert a fault in the key of the leak-free TBC F, CONCRETE provides security in the presence of a leaking encryption and a leaking and faulty decryption. Third, we prove that if the adversary can insert a single bit fault in encryption she can forge, breaking integrity. Fourth, we prove that if the adversary is only allowed differential (or random) faults CONCRETE provides weak-integrity with faults.

4.1 CONCRETE and its division in atoms

Here, we divide both the encryption and the decryption algorithm of CONCRETE in atoms (and we give the dependency matrix in App. E.2).

To make the notation clearer, instead of indexing with a subset in $\mathbb{N}$, we prefer to use an indexation similar to the one used by subsections in Latex (e.g., $0.1, 0.2, \ldots, 0.n, 1.1, \ldots$).

Division of CONCRETE into atoms. First, we observe that the associated data, a is used only once by CONCRETE, thus, according to the atomic model (Sec. 2.6.1), it cannot be faulted.

For the query on input (a, m), we define $(x_1, \ldots, x_l) = (m_1, \ldots, m_l)$, the parsing of m in n-bit long blocks, $x_{l+1} = a$. Similar to what done in [9], we assume that the master key is hard coded into the implementation, thus, we consider $\mathsf{F}_k^{\cdot}(\cdot)$ as an atom; moreover, we consider the production of randomness as an additional atom. So, as atoms, we consider $\mathsf{F}_k^{\cdot}(\cdot)$, $\mathsf{E}_{\cdot}(p_A)$, $\mathsf{E}_{\cdot}(p_B)$ $\mathsf{H}'_s(\cdot,\cdot)$, the XOR $\cdot \oplus \cdot$, and the random picking $\cdot \xleftarrow{\$} \{0,1\}^n$.

We define $f_{0.1} = k_0 \xleftarrow{\$} \{0,1\}^n$, $f_{0.2} = \mathsf{E}_{k_0}(p_B)$, then we iterate these three blocks for $i = 1, ..., l-1$: $f_{i.1} = k_i = \mathsf{E}_{k_{i-1}}(p_A)$, $f_{i.2} = y_i = \mathsf{E}_{k_i}(p_B)$, $f_{i.3} = c_i = y_i \oplus m_i$; after that, $f_{l.1} = k_l = \mathsf{E}_{k_{l-1}}(p_A)$, $f_{l.2} = y_l = \mathsf{E}_{k_l}(p_B)$, $f_{l.3} = c_l = \pi_{|m_l|}(y_l) \oplus m_l$, $f_{l+1.1} = h = \mathsf{H}'(c_0\| \ldots \|c_l, a)$, $f_{l+1.2} = c_{l+1} = \mathsf{F}_k^h(k_0)$.

We observe that $f_{0.1}$ uses no input, while $f_{0.2}$ only $z_{0.1} = k_0$. For all $i = 1, ..., l$, $f_{i.1}$ uses only $z_{i-1.1} = k_{i-1}$, $f_{i.2}$ uses only $z_{i.1} = k_i$, and $f_{i.3}$ only $z_{i.2} = y_i$ and $x_i = m_i$. Finally, $f_{l+1.1}$ uses $x_{l+1} = a$, and $z_{0.2} = c_0$, $z_{1.3} = c_1, \ldots, z_{l.3} = c_l$, while $f_{l+1.2}$ uses $z_{l+1.1}$. The output is $(z_{0.2}, z_{1.3}, ..., z_{l.3}, z_{l+1.2}) = (c_0, c_1, ..., c_l, c_{l+1})$, that is, $\mathsf{Select}(z_{0.0}, \ldots, z_{l+1.2}) = (z_{0.2}, z_{1.3}, ..., z_{l.3}, z_{l+1.2})$.

The dependency matrix, which is straightforwardly obtained from these equations, can be found in App. E.2, Eq. 3 for space reasons.

For decryption, the division is completely similar and can be found in App. E.2.

4.2 Integrity of CONCRETE vs faults in decryption

In this section, we prove that CONCRETE does not lose integrity if the adversary can inject any number or type of faults between the atoms in decryption. Before going into the proof, we need to introduce a new security definition for hash function.

Pre-image resistance with chosen target and random n-prefix for hash functions. In the proof, we need the following security property for a hash function $\mathsf{H} : \{0,1\}^* \to \mathcal{TW}$: given $y \in \mathcal{TW}$ chosen by the adversary and $x' \xleftarrow{\$} \{0,1\}^n$, it is difficult to find $x' \in \{0,1\}^n$ s.t. $x = x' \| x''$ and $\mathsf{H}(x) = y$. Formally,

Definition 10. A hash function $\mathsf{H} : \mathcal{HK} \times \{0,1\}^* \to \mathcal{TW}$ is (t, ϵ)*-pre-image resistant with chosen target and random n-bit prefix input (n*-PR-corpi*)* if for any t-adversary $\mathsf{A} = (\mathsf{A}_1, \mathsf{A}_2)$

$$\Pr[\mathsf{H}_s(x) = y \wedge \pi_n(x) = x' \mid x \leftarrow \mathsf{A}_2(\mathsf{st}, x'),\ x' \xleftarrow{\$} \{0,1\}^n,\ (\mathsf{st}, y) \leftarrow \mathsf{A}_1(s),\ s \xleftarrow{\$} \mathcal{HK}] \leq \epsilon.$$

The links between this security notion and collision resistance and pre-image resistance are discussed in App. D

Leakage function and faulty matrix. For the leakage function, assuming that F is implemented in a strongly protected way that we model as leak-free, we use the usual functions for the unbounded leakage model (see Sec. 2.5): for encryption, $\mathsf{L}_{\mathsf{AEnc}}(a, m, k_0; k) := k_0$ (differently from CIML2, here, the adversary does not choose k_0, so we will give it as leakage); for decryption $\mathsf{L}_{\mathsf{ADec}}(a, c; k) := k_0$. Regarding faults, the adversary is allowed to set any fault she wants between the atoms in decryption, without any restriction and no fault at all in encryption, that is all entries of $\mathcal{I}_{\mathsf{AEnc}}$ are $\perp$, while for $\mathcal{I}_{\mathsf{ADec}}$ we allow the adversary to choose any entry different from ϵ of the matrix defined in Eq. 4, App. E.2, as long as they are compliant with Sec. 2.6.1.

Theorem 1. *Let* F *be a strongly protected* $(q, t_1, \epsilon_{\mathsf{sTPRP}})$-sTPRP*, let* E *be an ideal block-cipher, let* A *be allowed* q_I *queries to* E*, let* H *be a* $(t_2, \epsilon_{\mathsf{CR}})$*-collision resistant and* $(t_3, \epsilon_{\mathsf{PR\text{-}corpi}})$*-*$n$-PR-corpi *hash function. Then,* $(\mathcal{I}_{\mathsf{AEnc}}, \mathcal{I}_{\mathsf{ADec}})$*-protected implementation of the* AE*-scheme* $\mathsf{CONCRETE}\Pi = (\mathsf{Gen}, \mathsf{AEnc}, \mathsf{ADec})$*, encoding messages at most Ln bits long, with leaking function pair* $\mathsf{L} = (\mathsf{L}_{\mathsf{AEnc}}, \mathsf{L}_{\mathsf{ADec}})$ *and fault admissible sets* $\mathcal{F} = (\mathcal{F}_E, \mathcal{F}_D)$ *has* $(q_I, q_E, q_D, t, \epsilon)$-ciphertext integrity in the presence of leakage and faults against $\mathcal{F}$-fault-then-leakage attacks in encryption and decryption (CIL2Fd)*, with*

$$\epsilon \leq \epsilon_{\mathsf{sTPRP}} + \epsilon_{\mathsf{CR}} + (q_D + 1)\epsilon_{\mathsf{PR\text{-}corpi}} +$$
$$\frac{(q_D + 2)(q_D + 1)}{2^{n+1}} + \frac{q_D[(q_E(L + 1) + 1)(q_D + 1)/2 + q_I]}{2^n}.$$

Idea of the proof. First, we observe that any fault sent after the second atom will not help any adversary. In fact, after k_0 is obtained, the adversary can already compute on his own if c_0 is correct. Then, let (a^*, c^*) be a forgery and let $h^* = \mathsf{H}'(c_0^* \| \ldots \| c_{l^*}^*, a^*)$ and $k_0^* = \mathsf{F}_k^{-1,h^*}(c_{l^*+1}^*)$. Suppose that the adversary has already forced a decryption query to compute $k_0^* = \mathsf{F}_k^{-1,h^*}(c_{l^*+1}^*)$. Let us suppose that this had been done for the first time in the i^{th} decryption query (the CIML2 proof is enough to prove that if $c_{l^*+2} = \mathsf{F}_k^{h^*}(k_0^*)$ in an encryption query, then the adversary cannot use it [15]). Now, there are two possibilities: 1) the adversary has already forced a correct computation of H outputting h^*. Thus, to forge the adversary either finds a collision for h^* (but H is collision resistant) or the input of this computation c^i, a^i is s.t. $c_0^i = \mathsf{E}_{k_0^i}(p_B)$ (but, k_0^i is random, thus, $\mathsf{E}_{k_0^i}(p_B)$ is random, since E is a PRF). 2) the adversary has to find a pre-image of h^* whose first n-bits are $\mathsf{E}_{k_0}(p_B)$ (which are random), but H is PR-corpi. Instead, if the adversary has not forced this computation before, then k_0 is random, thus $\mathsf{E}_{k_0}(p_B)$ is random, thus, the probability that $c_0 = \tilde{c_0}$ is 2^{-n}.

The full proof requires that we compute the probability of collision between different ephemeral keys; we leave it to App. I.1 with the time bounds.

On the need of the ideal block-cipher for E. In our security proof we need to prove the following: given a random k_0, and its direct leakage, $c_0 = \mathsf{E}_{k_0}(p_B)$ remains random. This seems straightforward to prove, but there is a problem: we need to give k_0 to the CIL2F adversary (as it is leaked). Thus, we cannot use the standard PRF-game, where a PRF adversary A' against E_{k_0}, with k_0 picked uniformly at random, uses a CIL2F adversary A, because A' should need oracle access to E_{k_0} or a random function e, but she also needs to give to A the leakage which is a function of k_0. **But if A' knows k_0, she wins easily the PRF game**. Moreover, if the adversary A has inserted a differential fault in k_0, or she has set to a fixed value some of its bits, A' should be able to compute the required queries correctly, and this is impossible without having access to k_0 [7]

The use of the ideal cipher model in the presence of leakage seems arguable at first glance. I.e., one may wonder how a block cipher could be ideal when it is instantiated and the adversary receives leakage of its computations. On the other hand, note that we are only using the ideal hypothesis to prove that given a random k_0, leakage and fault associated with $\mathsf{E}_{k_0}(p_B)$, $\mathsf{E}_{k_0}(p_B)$ remains random, and cannot be forecasted before. **We emphasize that the adversary receives only one such leakage instance under k_0**. We feel confident that this assumption does not harm the concrete security of our scheme and the use of the ideal cipher model even if it is paired with that leakage.

Faults inside the atoms. In the previous proof, we have never used the fact that E is an ideal block-cipher except, during decryption queries, for the computation of $\mathsf{E}_{k_0}(p_B)$ as discussed above, that the right commitment to the ephemeral key retrieved from F_k^{-1} (which we compute apart from the simulation of the decryption algorithm). Thus, if any fault is inserted in the computation of E during a decryption query the adversary should gain no advantage in forging. Moreover, injecting any fault in H (in decryption) is not a problem: similarly to [9], an adversary can set the output of H when it is used, which is easier and more powerful than faulting internal computation of H. Thus, our construction will only require that there are no faults in F. *That is, in the implementation of* CONCRETE *only a single call to a single primitive must be protected against faults and leakage to provide integrity in the presence of leakage and faults (in decryption).* We formalize this intuition in App. I.2, giving the model, the theorem and the proof.

4.3 CONCRETE is not secure vs fault(s) in encryption

Until now, we have not considered faults in encryption. In this section, we prove that a single fault in encryption can break the CIL2F security of CONCRETE: it is simply enough to perform the decoupling attack [45], that is, we flip a bit in $c_1, \ldots, c_l$ when $h = \mathsf{H}'_s((c_0 \| \ldots \| c_l), a)$ is computed. Let us say that the fault replaces c_1 with c'_1. Let $(c_0, c_1, \ldots, c_l, c_{l+1})$ be the output of AEnc_k with input $(a, (m_1, \ldots, m_l))$ where the computation has been faulted with the fault previously described. Then, $(a, c_0, c'_1, c_2, \ldots, c_l, c_{l+1})$ is a valid forgery (encrypting $m_1 \oplus c_1 \oplus c'_1, m_2, \ldots m_l$). Thus, CONCRETE does not provide integrity if a fault is injected during encryption (either setting or flipping a bit).

4.4 Weakly integrity of CONCRETE vs fault(s) in encryption

In the previous attack, we observe that c_1 takes the value c'_1 during the computation, thus, the previous attack does not break the weak integrity of CONCRETE. Now, we

[7]For the latter problem, we may have thought of a PRF-game where the adversary can ask key-related queries. Since this solution does not solve the first problem (the leakage of k_0), we have not pursued it, but we have used the ideal-cipher model.

study the wCILF2-security of CONCRETE. First, we observe that if the adversary can set k_0 and h, the adversary can produce a forgery. In fact, she can simply precompute the right $c_0, \ldots, c_l$ for the k_0 she has chosen, which encrypts $m_1, \ldots, m_l$. Then, she can compute $h = \mathsf{H}'_s((c_0\| \ldots \|c_l), a)$ for the c_is she has already computed and for an a of her choice. Setting the inputs of F to the h and k_0 just obtained, she gets c_{l+1}. So, $(a, (c_0, c_1, \ldots, c_l, c_{l+1}))$ is a valid forgery. Thus, there is no integrity if the adversary can set two values in a single encryption query.

Note that for the previous attack, it is necessary for the adversary to set these two values. She cannot succeed with a differential attack (because both k_0 and h would be random [8], thus, she cannot know in advance the offset to change from the k'_0 (and h') actually used in the computation to the one she wants. We prove that CONCRETE is wCILF2 if the adversary injects only differential faults in encryption (and any faults she wants in decryption) in App. I.3. The proof is the same with the difference that, for every encryption query, we can keep in memory instead of c, $c' = (a, (c'_0, \ldots, c'_l, c_{l+1}))$ where c'_i is how c_i is modified in the input of the H atom, that is, the computation $f_{l+1.1}$.

5 CONCRETE2: a scheme cilf-secure

In this section, we present CONCRETE2 which achieves CIL2F2. We start by stating why CONCRETE does not achieve CIL2F2, then we give the description and we prove its CIL2F security. Finally, we propose a possible improvement using sponges, CONCRETESponge.

5.1 Design of CONCRETE2.

Why CONCRETE is not CIL2F2. The main reasons why CONCRETE does not achieve CIL2F2 are two: first, every possible ciphertext block $c_1, \ldots, c_l$ produced is "authenticated" [9], irrespective if this is the encryption of the required message; second, given a ciphertext block c_i, encrypting m_i, an adversary can predict the encryption of m'_i (which is $c'_i = c_i \oplus m_i \oplus m'_i$).

Designing CONCRETE2. To solve the first problem, it is enough to encrypt also the commitment of the message. For example, we can hash the message, $h' = \mathsf{H}(c_0\|m)$, (c_0 is used to have something random,) and we compute a ciphertext block, $c_{l+1} = \mathsf{E}_{k_{i+1}}(p_B) \oplus h'$, with $k_{i+1} = \mathsf{E}_{k_i}(p_A)$. In decryption, we *both* check, if we have the correct commitment for the retrieved key k_0, and if we have encrypted the correct commitment of the message.

For the second problem, instead of XORing the message to a random value $y_i = \mathsf{F}_{k_i}(p_B)$, we use y_i as an ephemeral key, that is, $k_{i,E} = y_i = \mathsf{E}_{k_i}(p_B)$ and we compute $c_i = \mathsf{E}_{k_{i,E}}(m_i)$. This way, the adversary, having c_i, cannot predict c'_i which encrypts another message (moreover, although the ephemeral key $k_{i,E}$ potentially may be leaked, this cannot give any additional information than how to encrypt this particular block). We use a similar idea, using the MAC proposed at CCS15 [40] (see Alg. 5, App. J), to compute the commitment.

Thus, we have obtained CONCRETE2 which we describe in Fig. 3, App. J, and in Alg. 7, App. J.

We also provide a sponge-based construction, denoted CONCRETESponge, which uses the duplex construction to produce the ephemeral keys, as we detail in App. C. Owing to a lack of space.

With respect to MEM [47], our scheme uses the strongly protected component significantly less (once as compared to three times), and we provide authenticity even if the adversary can inject any fault in decryption and has leakage (for more details, see App. F).

[8]If k_0 is random, then, $c_0, \ldots, c_l$ are random. For the sake of the argument, we assume that $\mathsf{H}'_s((c_0, \ldots, c_l), a)$ is random, although this is not true if H is not assumed to be a random oracle. However, for a reasonable hash function, if the inputs are random, the output should be similar to random.

[9]Technically speaking, CONCRETE does not authenticate these ciphertext blocks, but uses their hash to send k_0. Since there is a commitment on k_0, this provides authenticity.

5.2 Impossibility results for the CIL2F2-security of CONCRETE2

We start giving some impossibility results regarding the number of values set via faults (or the number set per atom), then, we give the CIL2F2 security for CONCRETE2. Combining everything, we obtain a tight security result in terms of faults that an adversary can set.

5.2.1 Setting 2 values.

An adversary can forge easily if she is allowed to set 2 values per encryption queries. She can proceed as follows: she chooses k'_0 ,m', and a', then she computes $c'_0, ..., c'_{l+2}$, and $h' = \mathsf{H}(c'_0\| \dots \|c'_{l+1}, a')$ as for an encryption query where the key picked is k'_0, the message encrypted is m' and the associated data is a' (this can be done since the only value not known to the adversary is k_0 in a real encryption, but, here, the adversary chooses it). Then, she does an encryption query on input (a, m). When c_{l+2} is computed, $c_{l+2} = \mathsf{F}^h_k(k_0)$, she simply replaces using faults h with h' and k_0 with k'_0. Since no message or associated data is touched via faults, the fault chosen by the adversary is admissible. The encryption oracle outputs $(c_0, \dots, c_{l+2})$. The adversary outputs as her forgery $c^* = (c'_0, \dots, c'_{l+1}, c_{l+2})$.

Clearly, c^* is fresh. Moreover, c^* is valid since hashing $c'_0, \dots, c'_{l+1}$ and a, we obtain h' and $\mathsf{F}^{-1,h'}_k(c_{l+2}) = k'_0$ and $c'_0, \dots, c'_{l+1}$ is the correct encryption for m' (that is, c'_0 is the correct committing for k'_0 and c'_{l+1} is the correct committing for k'_0 and m'.

Thus, CONCRETE2 cannot be CIL2F2 secure when the adversary can set two values via a fault. We discuss the theoretical reason behind this result and an additional attack where the adversary can set one value per atom in App. G.

5.3 Pre-image resistance for a chosen image offset of a random value

Before stating the security result, we need to introduce a new hash security definition, which involves finding a pre-image for $h \oplus \Delta$ with $h = \mathsf{H}(x)$ where x is "random" enough.

Definition 11. Let $\mathsf{H} : \mathcal{HK} \times \{0,1\}^* \to \{0,1\}^n$ be a hash function. We say that H is (t, ϵ)*-pre-image resistant for a chosen offset of the image of a random value* (PR-coirv) if, for any t-adversary, $\mathsf{A} = (\mathsf{A}_1, \mathsf{A}_2)$

$$\Pr[x \leftarrow \mathsf{A}_2(\mathsf{st}, x, h') \text{ s.t. } \mathsf{H}_s(x) = h' \mid h' = h \oplus \Delta,$$

$$h = \mathsf{H}_s(x),\ x \overset{\mathcal{D}}{\leftarrow} \{0,1\}^*\ (\mathsf{st}, \mathcal{D}, \Delta) \leftarrow \mathsf{A}_1(s),\ s \overset{\$}{\leftarrow} \mathcal{HK}] \le \epsilon$$

where Δ is any value in $\{0,1\}^n \neq 0^n$, and $\mathcal{D}$ is any distribution over $\{0,1\}^*$ s.t.

$$\max_{x \in \{0,1\}^*} \Pr[x \overset{\mathcal{D}}{\leftarrow} \{0,1\}^*] \le 2^{-n}.$$

Substantially, the adversary, first, chooses a distribution $\mathcal{D}$ over $\{0,1\}^*$ where no value can be picked with probability larger than 2^{-n}, which is the uniform probability on the target space, and offset Δ, then, after receiving $h = \mathsf{H}_s(x)$ with $x \overset{\mathcal{D}}{\leftarrow} \{0,1\}^*$, she has to compute a pre-image for $h \oplus \Delta$. For a good hash function, we expect that h is random since $\mathcal{D}$ is a reasonably good random distribution, thus, it is perfectly equivalent to the range-oriented pre-image resistance (Def. 22, App. D).

We require that $\Delta \neq 0^n$ because, otherwise x is a pre-image, thus, an adversary trivially wins (and, looking ahead, we have no interest in this case in our proof). Implicitly, we require that the distribution $\mathcal{D}$ can be efficiently computed and described. This definition cannot hold for any distribution with the required randomness condition (think of a distribution that picks only the pre-images of a certain value x, which are infinite).

This definition is in the standard model, it is trivial to see that this property holds if we model H as a random oracle.

5.4 The CIL2F2-security of CONCRETE2

This section is devoted to stating and proving the integrity in the presence of leakage and faults of CONCRETE2. We start by giving the leakage functions and the faulty matrices for CONCRETE2.

Leakage function and faulty matrix. For the leakage function, we use the same leakage function as for CONCRETE: $\mathsf{L}_{\mathsf{AEnc}}(a, m, k_0; k) := k_0$, and $\mathsf{L}_{\mathsf{ADec}}(a, c; k) := k_0$, because from k_0 the adversary can compute correctly all ephemeral values. Regarding faults, the adversary is allowed to set any fault she wants between the atoms in decryption, without any restriction; in encryption, she can use any differential fault she wants, moreover, she is allowed to set a single value, that is all entries of $\mathcal{I}_{\mathsf{AEnc}}$ are either ϵ (when the corresponding value in the dependency matrix is ϵ) or $\perp/\oplus$ with the exception of a single value; while $\mathcal{I}_{\mathsf{ADec}}$ is empty, that is, we allow the adversary to choose any entry different from ϵ of the matrix defined in Eq. **??** as long as they are compliant with Sec. 2.6.1. $\mathcal{F}_E$ and $\mathcal{F}_D$ are defined accordingly. We leave the formal division in atoms to App. E.3 (it is very similar to what we've done for CONCRETE).

Theorem 2. *Let* F *be a strongly protected* $(q, t_1, \epsilon_{\mathsf{sTPRP}})$-sTPRP*, let* E *be an ideal blockcipher, let* A *be allowed* q_I *queries to* E*, let* H *be a* $(t_1, \epsilon_{\mathsf{CR}})$*-collision resistant,* $(t_2, \epsilon_{\mathsf{PR\text{-}coirv}})$ *pre-image resistant for a chosen offset of the image of a random value, and* $(t_3, \epsilon_{\mathsf{PR\text{-}corpi}})$ *pre-image resistant with chosen target and random n-bit prefix input. Then, the* $(\mathcal{I}_{\mathsf{AEnc}}, \mathcal{I}_{\mathsf{ADec}})$*-protected implementation of the* AE*-scheme* CONCRETE2 $\Pi = (\mathsf{Gen}, \mathsf{AEnc}, \mathsf{ADec})$*, encrypting messages at most Ln bits long, with leaking function pair* $\mathsf{L} = (\mathsf{L}_{\mathsf{AEnc}}, \mathsf{L}_{\mathsf{ADec}})$ *and fault admissible sets* $\mathcal{F} = (\mathcal{F}_E, \mathcal{F}_D)$ *has* $(q_I, q_E, q_D, t, \epsilon)$ *provides* ciphertext integrity in the presence of leakage and faults against $\mathcal{F}$-fault-then-leakage attacks in encryption and decryption ($\mathsf{CIL2F2} - (\mathsf{de})$) *with*

$$\epsilon \leq \epsilon_{\mathsf{sTPRP}} + \epsilon_{\mathsf{CR}} + q\epsilon_{\mathsf{PR\text{-}corpi}} + 2q_E\epsilon_{\mathsf{PR\text{-}coirv}} +$$
$$\textcircled{a}\{4q_E + q(q+1) + 1 + q_D + q_E(\frac{q_E - 1}{2})q_I(2 + 2q_D + 5q_E) +$$
$$\textcircled{b}(3L+4)[6q_Eq_D + 2q_D + 2q_D\frac{q_D+1}{2} + 2q_E + 5q_E\frac{q_E-1}{2})]\}2^{-n}.$$

Note that the terms marked with $\textcircled{a}$ and $\textcircled{b}$ are due to the probability of collisions between ephemeral keys (when faults do not set them).

Proof idea: First, we observe that CONCRETE2 is CIL2F2-de. An adversary cannot forge using faults in decryption for the same reason as for CONCRETE. Regarding differential faults in encryption, let (h, k_0, c_{l+2}) be the triple associated to F, for this encryption query (i.e., the inputs and the output of the F atom). If $h = \mathsf{H}(c_0\|\dots\|c_{l+1}, a)$, thus, to forge using this triple, the adversary has to find another pre-image for h (but H is collision-resistant) or to use it in the forgery; instead, if this is not the case or $(c_0, \dots, c_{l+1})$ is not a valid encryption, the adversary has to find a pre-image for h with a random prefix (c_0). But, we claim that $(c_0, \dots, c_{l+1})$ is a valid encryption[10] using the key k_0 for any message m only if the adversary has injected no fault. Thus, the adversary cannot use $(c_0, \dots, c_{l+1}, c_{l+2})$ in the forgery.

Now, we prove the claim: Let m be the message that the adversary wants to encrypt, let $c^\dagger = c^\dagger_0, \dots, c^\dagger_{l+1}$ the correct (that is, without any faults injected) encryption of m using the ephemeral key k_0, and let $k^\dagger_1, \dots, k^\dagger_{l+1}, k^\dagger_{0,E}, \dots, k^\dagger_{l+1,E}, k^\dagger_{0,A}, \dots, k^\dagger_{l+1,A}$ be the correct ephemeral keys. All these values are random, due to k_0 be random, and E be an ideal cipher, thus, the probability that an adversary correctly guesses any of them is negligible. Let us suppose $c_i \neq c^\dagger_i$, thus, since $k_{i,E}$ is random the message block c_i encrypts, m'_i is random (since $m'_i = \mathsf{E}^{-1}_{k_{i,E}}(c_i)$), thus, the correct $k'_{i,A} = \mathsf{E}_{k_{i-1,A}}(m'_i)$ is random and all subsequent $k'_{j,A}$ for $j > i$ are random and consequently c'_{l+1} is random too. Now, the probability that the adversary can correct this chain with a fault is negligible because she has to guess the right value (or the right offset) which is random.

So, we only need to consider what the adversary can do when she sets a value in encryption. We classify encryption queries where the set value is injected in the computation

[10]It means that if in Enc encryption we use k_0 and m, we obtain exactly those values.

$\mathsf{AEnc}(a, m)$ where the key picked is k_0, as listed next.
In the computation of F*:*

- *changing k_0 to k_0':* It implies that the adversary knows a valid triple for F_k (h, k_0', c_{l+2}). She can use this information to forge if she can find $c_0', \dots, c_l'$ which correctly encrypts m' starting from k_0', and a' s.t. $\mathsf{H}(c_0' \| \dots \| c_{l+1}', a') = h$. We have two cases regarding the H atom in the encryption query:
  - *h is the actual output of the* H *atom*: Again we have two possibilities:
    * *the inputs of the* H *atom are* $(c_0' \| \dots \| c_{l+1}', a')$: The probability that this happens is negligible because c_0 is random (since k_0 is random and the adversary can only use differential faults and E is an ideal cipher) and the probability that the adversary with differential fault is able to arrive to c_0' is negligible (which is equal to 2^{-n}).
    * *the inputs of the* H *atom are not* $(c_0' \| \dots \| c_{l+1}', a')$: Thus, we have found a collision for the hash function because both the actual input of the hash during the encryption query and $(c_0' \| \dots \| c_{l+1}', a')$ have the same output.
  - *the actual output of the hash function is $h' \neq h$:* So we cannot use the previous result because it is not given that a pre-image for h' has been ever computed. But, the ciphertext blocks $c_0, \dots, c_{l+1}$ are random, due to the fact that k_0 is random, E is an ideal cipher, and we are only using differential faults. She only has the pre-image for $h \oplus \Delta = h'$ where Δ is the offset added to h in the computation of F due to the fault Fa. Thus, the adversary has to break the PR-coirv-security of H [11]
- *changing h to h':* So the adversary knows a valid triple for F_k (h', k_0, c_{l+2}). She can use this information to forge if she can find $c_0', \dots, c_l'$ which correctly encrypts m' starting from k_0', and a' s.t. $\mathsf{H}(c_0' \| \dots \| c_{l+1}', a') = h'$. Now, k_0 is random (because it comes from a random value that is only affected by differential faults), thus $c_0 = \mathsf{E}_{k_{0,E}}(p_B)$, with $k_{0,E} = \mathsf{E}_{k_0}(p_A)$, is random. Therefore, to forge an adversary has to break the PR-corpi-resistance of H.

Not in the computation of F*:* It implies that the adversary knows a valid triple for F_k (h, k_0, c_{l+2}), with k_0 random (because it is randomly picked and then the adversary can only add differential faults). In particular, let $c_0' = \mathsf{E}_{\mathsf{E}_{k_0}(p_A)}(p_B)$, it is random due to the fact that k_0 is random. We have two possibilities for h:

- *h is the actual output of the* H *atom:* Now we consider the H atom, two possibilities:
  - *In the computation of* H *the c_0 input is not c_0' (obtained with faults).* Thus, she needs to find a pre-image for h with prefix $c_0' = \mathsf{E}_{\mathsf{E}_{k_0}(p_A)}(p_B)$. So, we can find two pre-images for h, thus breaking the collision resistance of H.
  - *In the computation of* H *the c_0 input is c_0' (obtained with faults).* Let $((c_0, \dots, c_{l+1}), a)$ be the inputs of H atom[12]. We have two cases:
    * $(c_0, \dots, c_{l+1})$ *is a valid* [13] *encryption using the key k_0 for any message m*: As we have already discussed before this cannot happen (even if she can set one value). Thus, in this case, since the adversary has injected no fault in this encryption query, since $c_0, \dots, c_{l+1}$ is the actual encryption of m using k_0 as a key, she has to find another valid encryption whose hash is equal h (thus, she has found a collision).
    * $(c_0, \dots, c_{l+1})$ *is not a valid encryption using the key k_0 for any message m*: Thus, the adversary has to find a valid encryption using the key k_0 for any message m, $c_0', \dots, c_{l+1}'$ which is a valid encryption for any message m and with ephemeral key k_0 and a' s.t. $\mathsf{H}((c_0' \| \dots \| c_{l+1}'), a') = h$, thus, we have

[11]Note that if the adversary can set k_0, and can set additional values, she can invalidate the assumption that h' is random either setting it, or setting other values forcing the inputs of H not to be random anymore, as we did with the attack presented in Sec. 5.2 and App. G.

[12]Note that the adversary cannot fault a since here it is the only time it is used

[13]It means that if in Enc encryption we use k_0 and m, we obtain exactly those values.

found a collision for the hash function

- *h is not the actual output of the* H *atom*: She can use this information to forge if she can find $c'_0, \dots, c'_l$ which correctly encrypts m' starting from k_0, and a' s.t. $\mathsf{H}(c'_0\| \dots \|c'_{l+1}, a') = h'$. Now, k_0 is random (because it comes from a random value that is only affected by differential faults), thus $c_0 = \mathsf{E}_{k_{0,E}}(p_B)$ with $k_{0,E} = \mathsf{E}_{k_0}(p_A)$ is random. Thus, to forge, an adversary has to break the PR−corpi-resistance of H.

The complete (long and cumbersome) proof can be found in App. I.4 for space reasons with the time complexity bounds.

Faults inside the atoms. Similarly to CONCRETE, there shouldn't be any problem if the adversary can inject faults in any atom in decryption, except for F_k^{-1}. On the other hand, for encryption queries, if we allow the adversary to inject a fault inside an atom, except for F_k, we have two challenges: First, we have to be sure that we do not set a value, but this is prevented using only differential faults. Second, we have to be sure that the outputs of E remain random. This is however a theoretical problem, but it is needed in the security proof: if the adversary can assert a differential fault in the computation of $\mathsf{E}_{k_0}(p_B)$, we do not know how to be sure that the real c_0 is random. But, in the PR−corpi security definition, we need to output the hash before obtaining the random prefix (and to output the hash, we need to perform the faulted computation of $\mathsf{E}_{k_0}(p_B)$). Thus, we need to be sure that $\mathsf{E}_{k_0}(p_B)$ is random given the faulted $\mathsf{E}_{k_0}(p_B)$, or to find another assumption (for example, perhaps it can be proven in the random oracle model). We strongly suspect that an adversary cannot exploit this case with an attack, we leave this as an open discussion and future challenge. There hash computation shouldn't be a problem, and we believe that, in practice, we have to protect against faults and leakage only a single TBC-call.

6 Conclusion

In this paper, we have given a model for integrity in the presence of leakage and faults expanding the atomic model. We have also shown how it is delicate to model, exposing a problem in the fault-resilience model.

We have also shown that it is possible to have an AE scheme providing integrity in the presence of leakage and faults (only in decryption) when the adversary can do a powerful leakage attack (unbounded leakage model) and fault attacks (unbounded faults between the atoms) only assuming that a single call to a TBC is fault- and leak-free. We have also proven that we can obtain a milder notion of security (hardening the requirement of freshness) allowing the adversary to inject only differential faults in encryption queries. Modifying CONCRETE, and proposing CONCRETE2 and CONCRETESponge we have proved that it is possible to provide integrity in the presence of leakage and faults in both encryption and decryption in a very generous fault model for the adversary, with a scheme that is more efficient than the previously proposed scheme, MEM (see App. F). The complexity of the proofs and the need to introduce new security assumptions show how difficult it is to prove security in such a generous framework for an adversary.

For future works, first, we think, there is the whole problem of privacy in the presence of faults and leakage, both to find good definitions (achievable and giving meaningful security) and schemes achieving them. Second, we would like to remove the leak-free requirement for the TBC replacing with strong-unpredictability with leakage, and to study if there is a similar security notion in the presence of leakage and faults. Third, to study the security when the adversary can inject faults inside the atoms, finding security notions on which we can rely.

Acknowledgments

Francesco Berti and Itamar Levi were founded by the Israel Science Foundation (ISF) grant 2569/21.

References

[1] D. F. Aranha, C. Orlandi, A. Takahashi, and G. Zaverucha. Security of hedged fiat-shamir signatures under fault attacks. In A. Canteaut and Y. Ishai, editors, *Advances in Cryptology - EUROCRYPT 2020 - 39th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Zagreb, Croatia, May 10-14, 2020, Proceedings, Part I*, volume 12105 of *Lecture Notes in Computer Science*, pages 644–674. Springer, 2020.

[2] H. Bar-El, H. Choukri, D. Naccache, M. Tunstall, and C. Whelan. The sorcerer's apprentice guide to fault attacks. *Proc. IEEE*, 94(2):370–382, 2006.

[3] G. Barwell, D. P. Martin, E. Oswald, and M. Stam. Authenticated encryption in the face of protocol and side channel leakage. In T. Takagi and T. Peyrin, editors, *Advances in Cryptology - ASIACRYPT 2017 - 23rd International Conference on the Theory and Applications of Cryptology and Information Security, Hong Kong, China, December 3-7, 2017, Proceedings, Part I*, volume 10624 of *Lecture Notes in Computer Science*, pages 693–723. Springer, 2017.

[4] D. Bellizia, F. Berti, O. Bronchain, G. Cassiers, S. Duval, C. Guo, G. Leander, G. Leurent, I. Levi, C. Momin, O. Pereira, T. Peters, F. Standaert, B. Udvarhelyi, and F. Wiemer. Spook: Sponge-based leakage-resistant authenticated encryption with a masked tweakable block cipher. *IACR Trans. Symmetric Cryptol.*, 2020(S1):295–349, 2020.

[5] D. Bellizia, O. Bronchain, G. Cassiers, V. Grosso, C. Guo, C. Momin, O. Pereira, T. Peters, and F. Standaert. Mode-level vs. implementation-level physical security in symmetric cryptography - A practical guide through the leakage-resistance jungle. In D. Micciancio and T. Ristenpart, editors, *Advances in Cryptology - CRYPTO 2020 - 40th Annual International Cryptology Conference, CRYPTO 2020, Santa Barbara, CA, USA, August 17-21, 2020, Proceedings, Part I*, volume 12170 of *Lecture Notes in Computer Science*, pages 369–400. Springer, 2020.

[6] S. Berndt, T. Eisenbarth, S. Faust, M. Gourjon, M. Orlt, and O. Seker. Combined fault and leakage resilience: Composability, constructions and compiler. In H. Handschuh and A. Lysyanskaya, editors, *Advances in Cryptology - CRYPTO 2023 - 43rd Annual International Cryptology Conference, CRYPTO 2023, Santa Barbara, CA, USA, August 20-24, 2023, Proceedings, Part III*, volume 14083 of *Lecture Notes in Computer Science*, pages 377–409. Springer, 2023.

[7] F. Berti, C. Guo, O. Pereira, T. Peters, and F. Standaert. Strong authenticity with leakage under weak and falsifiable physical assumptions. In Z. Liu and M. Yung, editors, *Information Security and Cryptology - 15th International Conference, Inscrypt 2019, Nanjing, China, December 6-8, 2019, Revised Selected Papers*, volume 12020 of *Lecture Notes in Computer Science*, pages 517–532. Springer, 2019.

[8] F. Berti, C. Guo, O. Pereira, T. Peters, and F. Standaert. Tedt, a leakage-resist AEAD mode for high physical security applications. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2020(1):256–320, 2020.

[9] F. Berti, C. Guo, T. Peters, Y. Shen, and F. Standaert. Secure message authentication in the presence of leakage and faults. *IACR Trans. Symmetric Cryptol.*, 2023(1):288–315, 2023.

[10] F. Berti, C. Guo, T. Peters, and F. Standaert. Efficient leakage-resilient macs without idealized assumptions. In M. Tibouchi and H. Wang, editors, *Advances in Cryptology - ASIACRYPT 2021 - 27th International Conference on the Theory and Application of Cryptology and Information Security, Singapore, December 6-10, 2021, Proceedings, Part II*, volume 13091 of *Lecture Notes in Computer Science*, pages 95–123. Springer, 2021.

[11] F. Berti, C. Hazay, and I. Levi. LR-OT: leakage-resilient oblivious transfer. *IACR Cryptol. ePrint Arch.*, page 1143, 2024.

[12] F. Berti, F. Koeune, O. Pereira, T. Peters, and F. Standaert. Ciphertext integrity with misuse and leakage: Definition and efficient constructions with symmetric primitives. In J. Kim, G. Ahn, S. Kim, Y. Kim, J. López, and T. Kim, editors, *Proceedings of the 2018 on Asia Conference on Computer and Communications Security, AsiaCCS 2018, Incheon, Republic of Korea, June 04-08, 2018*, pages 37–50. ACM, 2018.

[13] F. Berti, O. Pereira, T. Peters, and F. Standaert. On leakage-resilient authenticated encryption with decryption leakages. *IACR Trans. Symmetric Cryptol.*, 2017(3):271–293, 2017.

[14] F. Berti, O. Pereira, and F. Standaert. Reducing the cost of authenticity with leakages: a ciml2-secure AE scheme with one call to a strongly protected tweakable block cipher. *IACR Cryptol. ePrint Arch.*, page 451, 2019.

[15] F. Berti, O. Pereira, and F. Standaert. Reducing the cost of authenticity with leakages: a \mathsf CIML2 -secure \mathsf AE scheme with one call to a strongly protected tweakable block cipher. In J. Buchmann, A. Nitaj, and T. Rachidi, editors, *Progress in Cryptology - AFRICACRYPT 2019 - 11th International Conference on Cryptology in Africa, Rabat, Morocco, July 9-11, 2019, Proceedings*, volume 11627 of *Lecture Notes in Computer Science*, pages 229–249. Springer, 2019.

[16] G. Bertoni, J. Daemen, M. Peeters, and G. V. Assche. Duplexing the sponge: Single-pass authenticated encryption and other applications. In A. Miri and S. Vaudenay, editors, *Selected Areas in Cryptography - 18th International Workshop, SAC 2011, Toronto, ON, Canada, August 11-12, 2011, Revised Selected Papers*, volume 7118 of *Lecture Notes in Computer Science*, pages 320–337. Springer, 2011.

[17] E. Biham and A. Shamir. Differential fault analysis of secret key cryptosystems. In B. S. K. Jr., editor, *Advances in Cryptology - CRYPTO '97, 17th Annual International Cryptology Conference, Santa Barbara, California, USA, August 17-21, 1997, Proceedings*, volume 1294 of *Lecture Notes in Computer Science*, pages 513–525. Springer, 1997.

[18] D. Boneh, R. A. DeMillo, and R. J. Lipton. On the importance of checking cryptographic protocols for faults (extended abstract). In W. Fumy, editor, *Advances in Cryptology - EUROCRYPT '97, International Conference on the Theory and Application of Cryptographic Techniques, Konstanz, Germany, May 11-15, 1997, Proceeding*, volume 1233 of *Lecture Notes in Computer Science*, pages 37–51. Springer, 1997.

[19] G. Cassiers, B. Grégoire, I. Levi, and F. Standaert. Hardware private circuits: From trivial composition to full verification. *IEEE Trans. Computers*, 70(10):1677–1690, 2021.

[20] J. Coron, J. Patarin, and Y. Seurin. The random oracle model and the ideal cipher model are equivalent. In D. A. Wagner, editor, *Advances in Cryptology - CRYPTO 2008, 28th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 17-21, 2008. Proceedings*, volume 5157 of *Lecture Notes in Computer Science*, pages 1–20. Springer, 2008.

[21] E. Danieli, M. Goldzweig, M. Avital, and I. Levi. Revealing the secrets of radio embedded systems: Extraction of raw information via rf. *IEEE Transactions on Information Forensics and Security*, 2023.

[22] J. P. Degabriele, C. Janson, and P. Struck. Sponges resist leakage: The case of authenticated encryption. In S. D. Galbraith and S. Moriai, editors, *Advances in Cryptology - ASIACRYPT 2019 - 25th International Conference on the Theory and Application of Cryptology and Information Security, Kobe, Japan, December 8-12, 2019, Proceedings, Part II*, volume 11922 of *Lecture Notes in Computer Science*, pages 209–240. Springer, 2019.

[23] C. Dhar, J. Ethan, R. Jejurikar, M. Khairallah, E. List, and S. Mandal. Context-committing security of leveled leakage-resilient aead. *IACR Transactions on Symmetric Cryptology*, 2024(2):348–370, Jun. 2024.

[24] C. Dobraunig, M. Eichlseder, S. Mangard, F. Mendel, and T. Unterluggauer. ISAP - towards side-channel secure authenticated encryption. *IACR Trans. Symmetric Cryptol.*, 2017(1):80–105, 2017.

[25] C. Dobraunig, S. Mangard, F. Mendel, and R. Primas. Fault attacks on nonce-based authenticated encryption: Application to keyak and ketje. In C. Cid and M. J. J. Jr., editors, *Selected Areas in Cryptography - SAC 2018 - 25th International Conference, Calgary, AB, Canada, August 15-17, 2018, Revised Selected Papers*, volume 11349 of *Lecture Notes in Computer Science*, pages 257–277. Springer, 2018.

[26] C. Dobraunig, B. Mennink, and R. Primas. Leakage and tamper resilient permutation-based cryptography. In H. Yin, A. Stavrou, C. Cremers, and E. Shi, editors, *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security, CCS 2022, Los Angeles, CA, USA, November 7-11, 2022*, pages 859–873. ACM, 2022.

[27] M. Fischlin and F. Günther. Modeling memory faults in signature and authenticated encryption schemes. In S. Jarecki, editor, *Topics in Cryptology - CT-RSA 2020 - The Cryptographers' Track at the RSA Conference 2020, San Francisco, CA, USA, February 24-28, 2020, Proceedings*, volume 12006 of *Lecture Notes in Computer Science*, pages 56–84. Springer, 2020.

[28] D. Goudarzi and M. Rivain. How fast can higher-order masking be in software? In J. Coron and J. B. Nielsen, editors, *Advances in Cryptology - EUROCRYPT 2017 - 36th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Paris, France, April 30 - May 4, 2017, Proceedings, Part I*, volume 10210 of *Lecture Notes in Computer Science*, pages 567–597, 2017.

[29] C. Guo, O. Pereira, T. Peters, and F. Standaert. Authenticated encryption with nonce misuse and physical leakage: Definitions, separation results and first construction - (extended abstract). In P. Schwabe and N. Thériault, editors, *Progress in Cryptology - LATINCRYPT 2019 - 6th International Conference on Cryptology and Information Security in Latin America, Santiago de Chile, Chile, October 2-4, 2019, Proceedings*, volume 11774 of *Lecture Notes in Computer Science*, pages 150–172. Springer, 2019.

[30] A. Jana, A. Nath, G. Paul, and D. Saha. Differential fault analysis of NORX using variants of coupon collector problem. *J. Cryptogr. Eng.*, 12(4):433–459, 2022.

[31] A. Journault and F. Standaert. Very high order masking: Efficient implementation and security evaluation. In W. Fischer and N. Homma, editors, *Cryptographic Hardware and Embedded Systems - CHES 2017 - 19th International Conference, Taipei, Taiwan, September 25-28, 2017, Proceedings*, volume 10529 of *Lecture Notes in Computer Science*, pages 623–643. Springer, 2017.

[32] J. Katz and Y. Lindell. *Introduction to Modern Cryptography, Second Edition.* CRC Press, 2014.

[33] P. C. Kocher. Timing attacks on implementations of diffie-hellman, rsa, dss, and other systems. In N. Koblitz, editor, *Advances in Cryptology - CRYPTO '96, 16th Annual International Cryptology Conference, Santa Barbara, California, USA, August 18-22, 1996, Proceedings*, volume 1109 of *Lecture Notes in Computer Science*, pages 104–113. Springer, 1996.

[34] P. C. Kocher, J. Jaffe, and B. Jun. Differential power analysis. In M. J. Wiener, editor, *Advances in Cryptology - CRYPTO '99, 19th Annual International Cryptology Conference, Santa Barbara, California, USA, August 15-19, 1999, Proceedings*, volume 1666 of *Lecture Notes in Computer Science*, pages 388–397. Springer, 1999.

[35] Y. Lindell, editor. *Tutorials on the Foundations of Cryptography.* Springer International Publishing, 2017.

[36] M. D. Liskov, R. L. Rivest, and D. A. Wagner. Tweakable block ciphers. In M. Yung, editor, *Advances in Cryptology - CRYPTO 2002, 22nd Annual International Cryptology Conference, Santa Barbara, California, USA, August 18-22, 2002, Proceedings*, volume 2442 of *Lecture Notes in Computer Science*, pages 31–46. Springer, 2002.

[37] E. List. TEDT2 - highly secure leakage-resilient tbc-based authenticated encryption. In P. Longa and C. Ràfols, editors, *Progress in Cryptology - LATINCRYPT 2021 - 7th International Conference on Cryptology and Information Security in Latin America, Bogotá, Colombia, October 6-8, 2021, Proceedings*, volume 12912 of *Lecture Notes in Computer Science*, pages 275–295. Springer, 2021.

[38] J. Longo, D. P. Martin, E. Oswald, D. Page, M. Stam, and M. Tunstall. Simulatable leakage: Analysis, pitfalls, and new constructions. In *Advances in Cryptology - ASIACRYPT, Proceedings, Part I*, volume 8873 of *LNCS*, pages 223–242. Springer, 2014.

[39] C. Namprempre, P. Rogaway, and T. Shrimpton. Reconsidering generic composition. In P. Q. Nguyen and E. Oswald, editors, *Advances in Cryptology - EUROCRYPT 2014 - 33rd Annual International Conference on the Theory and Applications of Cryptographic Techniques, Copenhagen, Denmark, May 11-15, 2014. Proceedings*, volume 8441 of *Lecture Notes in Computer Science*, pages 257–274. Springer, 2014.

[40] O. Pereira, F. Standaert, and S. Vivek. Leakage-resilient authentication and encryption from symmetric cryptographic primitives. In I. Ray, N. Li, and C. Kruegel, editors, *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security, Denver, CO, USA, October 12-16, 2015*, pages 96–108. ACM, 2015.

[41] J. Quisquater and D. Samyde. Electromagnetic analysis (EMA): measures and counter-measures for smart cards. In I. Attali and T. P. Jensen, editors, *Smart Card Programming and Security, International Conference on Research in Smart Cards, E-smart 2001, Cannes, France, September 19-21, 2001, Proceedings*, volume 2140 of *Lecture Notes in Computer Science*, pages 200–210. Springer, 2001.

[42] P. Rogaway. Nonce-based symmetric encryption. In B. K. Roy and W. Meier, editors, *Fast Software Encryption, 11th International Workshop, FSE 2004, Delhi, India, February 5-7, 2004, Revised Papers*, volume 3017 of *Lecture Notes in Computer Science*, pages 348–359. Springer, 2004.

[43] P. Rogaway and T. Shrimpton. Cryptographic hash-function basics: Definitions, implications, and separations for preimage resistance, second-preimage resistance, and collision resistance. In B. K. Roy and W. Meier, editors, *Fast Software Encryption, 11th International Workshop, FSE 2004, Delhi, India, February 5-7, 2004, Revised Papers*, volume 3017 of *Lecture Notes in Computer Science*, pages 371–388. Springer, 2004.

[44] P. Rogaway and T. Shrimpton. Deterministic authenticated-encryption: A provable-security treatment of the key-wrap problem. *IACR Cryptol. ePrint Arch.*, page 221, 2006.

[45] S. Saha, M. Alam, A. Bag, D. Mukhopadhyay, and P. Dasgupta. Learn from your faults: Leakage assessment in fault attacks using deep learning. *J. Cryptol.*, 36(3):19, 2023.

[46] S. Saha, A. Bag, D. Jap, D. Mukhopadhyay, and S. Bhasin. Divided we stand, united we fall: Security analysis of some SCA+SIFA countermeasures against sca-enhanced fault template attacks. In M. Tibouchi and H. Wang, editors, *Advances in Cryptology - ASIACRYPT 2021 - 27th International Conference on the Theory and Application of Cryptology and Information Security, Singapore, December 6-10, 2021, Proceedings, Part II*, volume 13091 of *Lecture Notes in Computer Science*, pages 62–94. Springer, 2021.

[47] S. Saha, M. Khairallah, and T. Peyrin. Exploring integrity of aeads with faults: Definitions and constructions. *IACR Trans. Symmetric Cryptol.*, 2022(4):291–324, 2022.

[48] M. I. Salam, H. Q. A. Mahri, L. Simpson, H. Bartlett, E. Dawson, and K. K. Wong. Fault attacks on tiaoxin-346. In D. Abramson, editor, *Proceedings of the Australasian Computer Science Week Multiconference, ACSW 2018, Brisbane, QLD, Australia, January 29 - February 02, 2018*, pages 5:1–5:9. ACM, 2018.

[49] M. I. Salam, T. H. Ooi, L. Xue, W. Yau, J. Pieprzyk, and R. C. Phan. Random differential fault attacks on the lightweight authenticated encryption stream cipher grain-128aead. *IEEE Access*, 9:72568–72586, 2021.

[50] D. Salomon and I. Levi. Masksimd-lib: on the performance gap of a generic C optimized assembly and wide vector extensions for masked software with an ascon-p test case. *J. Cryptogr. Eng.*, 13(3):325–342, 2023.

A Additional definitions - fault resilient constructions

A.1 MAC

Here, we give the syntax definition for the cryptographic construction to authenticate: Message Auhtentication Codes (MAC)

Definition 12. A *Message Autentication Code (*MAC*)* is a triple of algorithms $\Pi = (\mathsf{Gen}, \mathsf{Mac}, \mathsf{Vrfy})$ where

- The key-generation algorithm Gen generates a key from the sets of keys, $\mathcal{K}$.
- The tag-generation algorithm Mac is a deterministic algorithm which takes as input a key $k \in \mathcal{K}$, and a message $m \in \mathcal{M}$, and outputs a tag τ. We denote this with $\tau \leftarrow \mathsf{Mac}_k(m)$.
- The verification algorithm Vrfy is a deterministic algorithm which takes as input a key $k \in \mathcal{K}$, a message $m \in \mathcal{M}$, and a tag τ, outputs either $\top$ ("valid") or $\bot$ ("invalid"). We denote this with $\top / \bot = \mathsf{Vrfy}_k(m, \tau)$.

We require *correctness*, that is $\forall (k, m) \in \mathcal{K} \times \mathcal{M}$, $\top = \mathsf{Vrfy}_k(m, \mathsf{Mac}_k(m))$.

Its security definition can be found easily, [32] (it can also be obtained from Def. 6 removing faults and leakage).

A.2 AE

Here, we give the standard syntax definition for AE.

Definition 13. An *authenticated encryption* (AE) scheme is a triple of algorithms $\Pi = (\mathsf{Gen}, \mathsf{AEnc}, \mathsf{ADec})$ where

- The key-generation algorithm Gen generates a key from the sets of keys, $\mathcal{K}$.
- The encryption algorithm AEnc is a probabilistic algorithm which takes as input a key $k \in \mathcal{K}$, an associated data $a \in \mathcal{AD}$ and a message $m \in \mathcal{M}$, and outputs a ciphertext $c \in \mathcal{C}$. We denote this with $c \leftarrow \mathsf{AEnc}_k(a, m)$.
- The decryption algorithm ADec is a deterministic algorithm which takes as input a key $k \in \mathcal{K}$, an associated data $a \in \mathcal{AD}$ and a ciphertext $c \in \mathcal{C}$, and outputs a message $m \in \mathcal{M}$ or $\bot$ ("invalid"). We denote this with $\bot / m = \mathsf{ADec}_k(a, c)$.

We require *correctness*, that is $\forall (k, a, m) \in \mathcal{K} \times \mathcal{AD} \times \mathcal{M}$, $m = \mathsf{ADec}_k(a, c)$ $\forall c \leftarrow \mathsf{AEnc}_k(a, m)$. A *nonce-based* AE *(*nAE*)* is an AE with an additional input called the nonce.

A.3 ivE - iv-based encryption schemes

For encryption, we use iv-based encryption schemes:

Definition 14. An iv*-based encryption scheme (*ivE*)* is a triple of algorithms $\Pi = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ where

- The key-generation algorithm Gen generates a key from the sets of keys, $\mathcal{K}$.
- The encryption algorithm Enc is a deterministic algorithm which takes as input a key $k \in \mathcal{K}$, an initialization vector $\mathsf{iv} \in \mathcal{IV}$, and a message $m \in \mathcal{M}$, and outputs a ciphertext $c \in \mathcal{C}$. We denote this with $c \leftarrow \mathsf{Enc}_k(\mathsf{iv}, m)$.
- The decryption algorithm Dec is a deterministic algorithm which takes as input a key $k \in \mathcal{K}$, an $\mathsf{iv} \in \mathcal{IV}$, and a ciphertext $c \in \mathcal{C}$, and outputs a message $m \in \mathcal{M}$ or $\bot$ ("invalid"). We denote this with $\bot / m = \mathsf{Dec}_k(\mathsf{iv}, c)$.

In the security definitions, we assume that the iv is picked uniformly at random from $\mathcal{IV}$. Thus, we denote with $\mathsf{Enc}^{\$}_k(m)$, the fact that we pick $\mathsf{iv} \xleftarrow{\$} \mathcal{IV}$, and, then, we compute $\mathsf{Enc}_k(\mathsf{iv}, m)$.

A.4 Black-box security for AE

In the black-box model (that is when the adversary can choose the inputs and only sees the outputs of their oracles), the standard security notion [39] for AE is the following:

Definition 15. An AE-scheme $\Pi = (\mathsf{Gen}, \mathsf{AEnc}, \mathsf{ADec})$ is (q_E, q_D, t, ϵ)-AE-*secure* if for any (q_E, q_D, t)-adversary A, the following advantage

$$|\Pr[1 \leftarrow \mathsf{A}^{\mathsf{AEnc}_k(\cdot,\cdot),\mathsf{ADec}_k(\cdot,\cdot)}] - \Pr[1 \leftarrow \mathsf{A}^{\$(\cdot,\cdot),\perp(\cdot,\cdot)}] \leq \epsilon, \qquad (2)$$

where $\$(\cdot,\cdot)$ is an oracle which on input (a, m) outputs $c \xleftarrow{\$} \{0,1\}^{|\mathsf{AEnc}_k(a,m)|}$ and $\perp$ is an oracle which always output $\perp$ ("invalid"). The adversary is not allowed to query his second oracle on input (a, c), if she has received c as an answer from the first oracle on input (a, m) for any $m \in \mathcal{M}$. For nAE-*security*, we adapt the previous definition to its syntax, and we add the requirement that the nonce is not repeated between different $\mathsf{AEnc}_k/\$$-queries.

This security definition gives both privacy and authenticity.

A.5 Privacy in the presence of leakage.

We now move to define privacy in the presence of leakage. As already discussed, modifying the standard real-or-ideal game, (that is, distinguishing AEnc_k from $\$$), by adding leakage to both members is very difficult, there are two possibilities:
Pereira et al. [40] started from the standard CPA-security (Chosen-Plaintext Attack) [32], adding leakage to this. We modify this definition to make it coherent with the AE-syntax [14]:

Definition 16. An AE-scheme $\Pi = (\mathsf{Gen}, \mathsf{AEnc}, \mathsf{ADec})$ is $(q_\mathsf{L}, q_E, t, \epsilon)$-$\mathsf{CPAL}$-*secure (*$\mathsf{CPA}$ *with leakage)*, if for any (q_L, q_E, t)-adversary $\mathsf{A} = (\mathsf{A}_1, \mathsf{A}_2)$

$$\Big|\Pr[b = b' \mid b' \leftarrow \mathsf{A}_2^{\mathsf{L}(\cdot,\cdot;\cdot),\mathsf{AEncL}_k(\cdot,\cdot)}(c^*, \ell^*, st),\ (c^*, \ell^*) = \mathsf{AEncL}_k(a, m_b),$$
$$(st, m_0, m_1) \leftarrow \mathsf{A}_1^{\mathsf{L}(\cdot,\cdot;\cdot),\mathsf{AEncL}_k(\cdot,\cdot)} \text{ s.t. } |m_0| = |m_1|, b \xleftarrow{\$} \{0,1\}] - \frac{1}{2}\Big| \leq \epsilon$$

where $\mathsf{L}(\cdot,\cdot;\cdot)$ is an oracle used to model the leakage, A_1 is a $(q_{1,L}, q_{1,E}, t_1)$-adversary, A_2 is a $(q_{2,L}, q_{2,E}, t_2)$-adversary, with $q_{1,L} + q_{2,L} \leq q_L,\ q_{1,E} + q_{2,E} \leq q_E$ and $t_1 + t_2 \leq t$. A scheme is $\mathsf{CPAL2}$ if in the previous definition the adversary receives not only ℓ^*, but also ℓ_D^* which is the leakage of $\mathsf{ADec}_k(a, c^*)$.

Note that the $\mathsf{L}(\cdot,\cdot;\cdot)$ is an oracle that A uses to understand and model the leakage of AEnc. For these queries, A chooses the inputs *and* the key, as suggested in [40].

Instead, Barwell et al. [3] started from the real-or-ideal game and gave the adversary the ability to have access to leaking queries in both worlds

Definition 17. An AE-scheme $\Pi = \mathsf{Gen}, \mathsf{AEnc}, \mathsf{ADec}$-scheme has $(q_E, q_{EL}, t, \epsilon)$-*encryption security with leakage* ($\mathsf{IND\text{-}CLPA}$) if for any (q_E, q_{EL}, t)-adversary A

$$|\Pr[1 \leftarrow \mathsf{A}^{\mathsf{AEnc}_k(\cdot,\cdot),\mathsf{AEncL}_k(\cdot,\cdot)}] - \Pr[1 \leftarrow \mathsf{A}^{\$(\cdot,\cdot),\mathsf{AEncL}_k(\cdot,\cdot)}]| \leq \epsilon,$$

where $\forall (a, m)$ $\$(a, m)$ picks $c \xleftarrow{\$} \{0,1\}^{|\mathsf{AEnc}_k(|a,m)|}$. [15]

Differences between these definitions. The $\mathsf{IND\text{-}CLPA}$ does not provide any security for the message encrypted when there is leakage, because an implementation of a scheme can be $\mathsf{IND\text{-}CLPA}$-secure even if the leakage function leaks the full message, that is, $\mathsf{L}_{\mathsf{AEnc}}(a, m; k) = m$. $\mathsf{IND\text{-}CLPA}$ provides privacy only for encryption when there is no leakage (thus, it provides at most *leakage-resilience*).

[14] The original definition [40] for encryption scheme, is obtained from ours replacing the AE scheme with an encryption scheme.

[15] The original definition [3] is not quantitative and it is for nonce-based AE, and can be obtained from ours, simply adding the nonce as input to AEnc_k. Moreover, the adversary is not allowed to repeat queries between his oracles, because they are deterministic, and he cannot repeat the nonce.

On the other hand, CPAL provides *leakage-resistance*, because it provides privacy for encryption when there is leakage. But, to have a CPAL-secure scheme, we must restrict more the possible leakage functions, since they cannot leak any bit of the message.

We will discuss the leakage functions for which we can give privacy when we prove the security of our schemes.

A.6 Fault resilient PRF (frPRF), MAC, encryption scheme and frAE

Saha et al. [47] introduced the notion of fault-resilient PRF (frPRF). They assume first, that every faulted query does not leak more than one correct couple (input/output), that is, $n^i_{\mathsf{pre}} \leq 1$ and after having done *all* faulted query, for any fresh query, the outputs from F_k are indistinguishable from random ones. Formally:

Definition 18. Let F be a PRF. It is a $(q_{Fa}, q_F, t, \epsilon)$-fault resilient PRF (frPRF) if $\forall$ (q_{Fa}, q_F, t)-adversary A

$$\Pr[\exists i \in [q_{Fa}] \text{ s.t. } n^i_{\mathsf{pre}}(\mathsf{A}) > 1] + |\Pr[\mathsf{A}^{\mathsf{FFa}_k,\mathsf{F}_k} \Rightarrow 1] - \Pr[\mathsf{A}^{\mathsf{FFa}_k,\$} \Rightarrow 1]| \leq \epsilon.$$

Similarly, they define a fault-resilient random oracle (which gives random outputs for fresh inputs even if the adversary has faulted previous queries).

When we pass from a PRF to an ivEv scheme, we have to consider that ivE schemes takes a random input for encryption queries. Thus, we have the frivE-security notion:

Definition 19. A $(q_{Fa}, q_E, q_D, t, \epsilon)$*-fault resistant* ivE*-scheme (*frivE*)* $\Pi = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ if $\forall$ (q_{Fa}, q_E, q_D, t)-adversary A

$$|\Pr[\mathsf{A}^{\mathsf{Enc}^\$\mathsf{Fa}_k,\mathsf{Enc}^\$_k} \Rightarrow 1] - \Pr[\mathsf{A}^{\mathsf{Enc}^\$\mathsf{Fa}_k,\$} \Rightarrow 1]| \leq \epsilon.$$

Combining together the frivE and frMAC security definitions, we have the security notion for AE-schemes, frAE:

Definition 20. A $(q_{Fa}, q_E, q_D, t, \epsilon)$*-fault resilient* AE*-scheme (*frAE*)* $\Pi = (\mathsf{Gen}, \mathsf{AEnc}, \mathsf{ADec})$ if $\forall$ (q_{Fa}, q_E, q_D, t)-adversary A

$$|\Pr[\mathsf{A}^{\mathsf{AEncFa}_k,\mathsf{AEnc}_k,\mathsf{ADec}_k} \Rightarrow 1] - \Pr[\mathsf{A}^{\mathsf{AEncFa}_k,\$,\perp} \Rightarrow 1]| \leq \epsilon,$$

where the adversary is not allowed to forward queries between different oracles. That is, there is a list $\mathcal{S}$ which for any query with input (a, m) contains the couple (a, c), with c the answer of the AEnc_k or \$ oracle, and a list $\mathcal{S}_{flt}$ of all queries to AEncFa_k which for every query on input (a, m), contains the couple message (a', c), with a' any value that a takes via fault, such that $\mathsf{ADec}_k(a', c) \neq \perp$; the adversary cannot ask to ADec_k any query in $\mathcal{S} \cup \mathcal{S}_{flt}$. We can give the same definition for nAE-scheme, simply modifying the definition for the nAE-syntax. In this latter case, the adversary cannot force to repeat nonces via faults.

Again, to AEncFa_k oracle the adversary can ask queries with no faults. All the queries to $\mathsf{AEnc}_k\mathsf{Fa}$ must be done *before* the first $\mathsf{AEnc}_k/\$$ oracle query.

A.6.1 A secure frMAC: Hash-then-frPRF

They start from the well-known *Hash-then-*MAC. As for LR−MACr, they use a random input r. They hash the message and r, to obtain $h = \mathsf{H}(r\|m)$, and, they compute $\tau = \mathsf{F}_k(h)$, where F is a frPRF. We depict it in Alg. 1.

For security, they assume that the H is a fault-resilient random oracle (frRO), which means that if the input is fresh and random, the internal states and outputs are unpredictable [47].

frMAC-security. They prove the frMAC-security of frMAC [47]:

Theorem 3. *If* $\mathsf{H} : \mathcal{HK} \times \{0,1\}^* \rightarrow \{0,1\}^n$ *is a* frRO*, and* F_k *is a* $(q_{Fa}, q_M + q_V, t, \epsilon_{\mathsf{frPRF}})$*-secure-*frPRF*, then,* frMAC *is a* $(q_{Fa}, q_M, q_V, q_H, t, \epsilon)$*-secure-*frMAC *against adversaries that performs differential faults, with*

$$\epsilon \leq \epsilon_{\mathsf{frPRF}} + \frac{\binom{q_E}{2} + q_E q_H}{2^{|r|}} + \frac{2(q_H + q_E + q_V)^2 + 2(q_E + q_V + 1) + 2q_{Fa}q_V}{2^{|h|}} + \frac{q_V}{2^{|\tau|}},$$

with $q_E = 2q_{Fa} + q_M$ *and* q_H *is the number of queries made to the random oracle* H.

Algorithm 1 The frMAC algorithm.

It uses a frPRF $\mathsf{F} : \mathcal{K} \times \{0,1\}^n \rightarrow \{0,1\}^n$ and a hash function $\mathsf{H} : \mathcal{HK} \times \{0,1\}^n \rightarrow \mathcal{TW}$.

```
• Gen
    – $k \xleftarrow{\$} \mathcal{K}$
• $\mathsf{Mac}_k(m)$:
    – $r \xleftarrow{\$} \{0,1\}^n$
    – $h = \mathsf{H}(r\|m)$
    – $\tau = \mathsf{F}_k(h)$
    – Return $(r, \tau)$
• $\mathsf{Vrfy}_k(m, r, \tau)$:
    – $h = \mathsf{H}(r\|m)$
    – If $\tau \overset{?}{=} \mathsf{F}_k(h)$ Return $\top$
    – Else Return $\bot$
```

A.6.2 A secure frAE: MAC-then-Encrypt-then-MAC (MEM)

Saha et al. [47] observe that using a composition of an encryption scheme and a frMAC (e.g, Enc-then-MAC) is insecure because the adversary can fault the input of frMAC and enable the *decoupling attack*. Let us suppose that Enc outputs a ciphertext c which is the input of frMAC. We inject a fault in c before frMAC processes it, thus, modifying it to c'. Then (c', τ) is a valid forgery, with τ the output of frMAC for the previous call.

To prevent such an attack, they propose an AE scheme MEM (MAC-then-Encrypt-then-MAC) [47]. The idea is to first authenticate the message m (and the AD, a) along with a random input r, with frMAC, obtaining the first tag τ_1 Then, we use PSV (Sec. 2.5) to encrypt both r and m, obtaining c, using τ_1 as the iv. Finally, we authenticate τ_1, c with frMAC, obtaining τ_2.
The idea is that the decoupling attack is prevented because if the adversary changes the input of the second MAC, then, she changes either τ_1 or c. In the first case, τ_1' is no more correct, while if we change c into c', the decryption of c', (r', m') is no more ok with τ_1 We depict MEM in Alg. 2.

frAE-security. The security relies on the security in the presence of faults and leakage of MAC and ivE [47].

Theorem 4. *If* MAC_1 *and* MAC_3 *are two independent* $(q_{Fa}, q_M, q_V, q_p, t, \epsilon_{\mathsf{frMAC}})$*-fault resilient* MAC*s, and* $\Pi_2 = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ *be a* $(q_{Fa}, q_E, t, \epsilon_{\mathsf{frivE}})$*-fault-resilient* ivE*-scheme, which is built using a secure* frPRF F*, a pseudorandom key generator* KG *s.t.* $\mathsf{Enc}_{k_2} = \mathsf{KG}(\mathsf{F}_{k_2}(\mathsf{iv})) \oplus m$*, then, randomized* MAC*-then-*MAC *(*MEM*) is a randomized* $(q_{Fa}, q_M, q_V, t, \epsilon)$*-*$\mathsf{frAE}$ *against adversaries which perform differential faults, with*

$$\epsilon \leq 2\epsilon_{\mathsf{frMAC}} + \epsilon_{\mathsf{frivE}} + \frac{q_E q_V}{2^{|r|}} + \frac{\binom{q_E}{2}}{2^{|\tau_1|}} + \frac{\binom{q_E}{2}}{2^{|r}},$$

with $q_E = 2q_{Fa} + q_M$ *and* q_p *is the maximum number of queries made to either* H_1 *and* H_2*.*

B Previous security results for CONCRETE

AE-security. For simplicity, in all the proofs, we omit the quantitative bounds for the time (which can be found in [14]).

Theorem 5. *Let* F *be a* $(q_{\mathsf{F}}, \epsilon_{\mathsf{sTPRP}})$*-*$\mathsf{sTPRP}$ *with* n*-bit blocks, let* E *be a* $(2, \epsilon_{\mathsf{PRF}})$*-*$\mathsf{PRF}$*, and let* H *be a* (ϵ_{CR})*-collision resistant hash function, then, the mode* $\mathsf{CONCRETE}$*, which encrypts at most* L*-block long messages is* (q_E, q_D, ϵ)*-*AE*-secure with*

$$\epsilon \leq \epsilon_{\mathsf{sTPRP}} + \epsilon_{\mathsf{CR}} + \frac{(q_D+1)(L+1)(q+q_E)}{2^{n+1}} + \frac{q_D}{2^n} + (q_E(L+1)+q_D)\epsilon_{\mathsf{PRF}} + \frac{q_E(L+1)[q_E(L+1)-1]}{2^{n+1}} + \frac{(q_D+q_E)(q_D+q_E-1)}{2^{n+1}}.$$

We briefly explain the terms of the bound:

- $\epsilon_{\mathsf{sTPRP}} + \epsilon_{\mathsf{CR}} + \frac{(q_D+1)(L+1)(q+q_E)}{2^{n+1}} + \frac{q_D}{2^n} + (q_D+1)\epsilon_{\mathsf{PRF}}$ is needed for authenticity. In particular,
  - $\epsilon_{\mathsf{sTPRP}} + \frac{(q_D+1)(q_E)}{2^{n+1}}$ because we need that the output of F and F^{-1} are random;
  - $q_D\epsilon_{\mathsf{PRF}}$ because we are checking c_0 and not k_0. We assume c_0 is random because E is a PRF;
  - $\frac{(q_D+1)(L+1)(q+q_E)}{2^{n+1}}$ because we need that in every decryption query the k_0 we have obtained has never been used before (to use the PRF-security of E).
- $(q_E(L+1)+q_D)\epsilon_{\mathsf{PRF}} + \frac{q_E(L+1)[q_E(L+1)-1]}{2^{n+1}} + \frac{(q_D+q_E)(q_D+q_E-1)}{2^{n+1}}$ is for confidentiality for PSV. In particular,
  - $\frac{q_E(L+1)[q_E(L+1)-1]}{2^{n+1}}$ to ensure that no keys used in the PSV part collides.

The idea of the proof is first to prove the integrity (see the following theorem for more details, then, for an encryption query if k_0 is random, all ciphertext blocks $c_0, \ldots, c_l$ are random due to the PRF-security of E (except if there are two ephemeral keys which collide in the whole story of the game). The last ciphertext block, c_l is random due to the sTPRP security of F and the collision resistance of H.

Integrity in the presence of leakage - CIL2 Now, we move to authenticity in the presence of leakage ($\mathsf{CIML2}$). In addition to the hypothesis for $\mathsf{CIL2}$, we assume that the adversary has taken control of the random source, that is, she can choose k_0 for encryption queries. In the unbounded leakage model, assuming that F is implemented in a leak-free way, the leakage functions for $\mathsf{CONCRETE}$ are the following: for encryption the leakage $\mathsf{L}_{\mathsf{AEnc}}(a, m, k_0; r) :=$ does not return anything, since choosing k_0, she can determine all the ephemeral values, except for $\mathsf{F}_k^h(k_0)$, but this is c_{l+1}, while $\mathsf{L}_{\mathsf{AEnc}}(a, c; r) := k_0$ since from k_0 the adversary can recompute all ephemeral values and the output.

Theorem 6. *Let* F *be a leak-free* $(q_{\mathsf{F}}, \epsilon_{\mathsf{sTPRP}})$*-*$\mathsf{sTPRP}$ *with* n*-bit blocks, let* E *be a* $(2, \epsilon_{\mathsf{PRF}})$*-*$\mathsf{PRF}$*, and let* H *be a* (ϵ_{CR})*-collision resistant hash function, then, in the unbounded leakage model, the mode* $\mathsf{CONCRETE}$*, which encrypts at most* L*-block long messages is* (q_E, q_D, ϵ)*-*$\mathsf{CIML2}$*-secure with*

$$\epsilon \leq \epsilon_{\mathsf{sTPRP}} + \epsilon_{\mathsf{CR}} + \frac{(q_D+1)(L+1)(q+q_E)}{2^{n+1}} + \frac{q_D+1}{2^n} + (q_D+1)\epsilon_{\mathsf{PRF}}.$$

Algorithm 2 The MEM algorithm [47].

It uses a frPRF $\mathsf{F} : \mathcal{K} \times \{0,1\}^n \to \{0,1\}^n$ with a strongly protected implementation, a BC $\mathsf{E} : \{0,1\}^n \times \{0,1\}^n \to \{0,1\}^n$ with a weakly protected implementation and a hash function $\mathsf{H} : \{0,1\}^* \to \mathcal{TW}$ which is modelled as a random oracle.

```
Gen:
  • $k_1, k_2, k_3 \xleftarrow{\$} \mathcal{K}$
  • $p_A, p_B \xleftarrow{\$} \{0,1\}^n$
  • Return $k = (k_1, k_2, k_3)$
$\mathsf{MEMEnc}_k(a, m)$:
  • Parse $m$ in $m_1, ..., m_l$
  • Parse $k$ in $k_1, k_2, k_3$
  • $(\tau_1, r) \leftarrow \mathsf{Mac}_{k_1}(a\|m)$:
      – $r \xleftarrow{\$} \{0,1\}^n$
      – $h_1 = \mathsf{H}(r\|a\|m)$
      – $\tau_1 = \mathsf{F}_{k_1}(h_1)$
  • $c = \mathsf{Enc}_{k_2}(r\|m, \tau_1)$:
      – $k_0\mathsf{F}_{k_2}(\tau_1)$
      – $c_0 = \mathsf{E}_{k_0}(p_B) \oplus r$
      – $k_1 = \mathsf{E}_{k_0}(p_A)$
      – For $i = 1, ..., l-1$
          ∗ $y_i = \mathsf{E}_{k_i}(p_B)$
          ∗ $c_i = y_i \oplus m_i$
          ∗ $k_{i+1} = \mathsf{E}_{k_i}(p_A)$
      – $y_l = \mathsf{E}_{k_l}(p_B)$
      – $c_l = \pi_{|m_l|}(y_l) \oplus m_l$
      – $C = (c_0, c_1, ..., c_l)$
  • $\tau_2 \leftarrow \mathsf{Mac}_{k_3}(\tau_1\|C)$:
      – $h_2 = \mathsf{H}(\tau_1\|C)$
      – $\tau_2 = \mathsf{F}_{k_3}(h_2)$
  • Return $c = (\tau_1, C, \tau_2)$
```

```
$\mathsf{MEMDec}_k(a, c)$:
  • Parse $c$ in $\tau_1, C, \tau_2$
  • Parse $k$ in $k_1, k_2, k_3$
  • $\mathsf{Vrfy}_{k_3}(\tau_1\|C, \tau_2)$:
      – $h_2 = \mathsf{H}(\tau_1\|C)$
      – $\tilde{\tau_2} = \mathsf{F}_{k_3}(h_2)$
  • If $\tau_2 \overset{?}{=} \tilde{\tau_2}$
      – $r\|m = \mathsf{Dec}_{k_2}(C, \tau_1)$:
          ∗ Parse $C$ in $c_0, c_1, ..., c_l$
          ∗ $k_0\mathsf{F}_{k_2}(\tau_1)$
          ∗ $r = \mathsf{E}_{k_0}(p_B) \oplus c_0$
          ∗ $k_1 = \mathsf{E}_{k_0}(p_A)$
          ∗ For $i = 1, ..., l-1$
              · $y_i = \mathsf{E}_{k_i}(p_B)$
              · $m_i = y_i \oplus c_i$
              · $k_{i+1} = \mathsf{E}_{k_i}(p_A)$
          ∗ $y_l = \mathsf{E}_{k_l}(p_B)$
          ∗ $m_l = \pi_{|c_l|}(y_l) \oplus c_l$
          ∗ $m = (m_1, ..., m_l)$
      – $\mathsf{Vrfy}_{k_3}(a\|m, \tau_1)$:
          ∗ $h_2 = \mathsf{H}(\tau_1\|C)$
          ∗ $\tilde{\tau_2} = \mathsf{F}_{k_3}(h_2)$
      – If $\tau_2 \overset{?}{=} \tilde{\tau_2}$
          ∗ Return $m$
  • Return $\perp$
```

The idea of the proof is that if the ciphertext is fresh, then, the key obtained by $\mathsf{F}_k^{-1,h}$ is fresh (if there are no collisions for the hash), thus the probability that c_0 is a correct commitment for the key is $\leq \epsilon_{\mathsf{PRF}} + 2^{-n}$ because E is a good BC.

Privacy in the presence of leakage - CPAL2 When we move to privacy in the presence of leakage, we start observing that we cannot allow the adversary to choose k_0 and/or to leak without bounds the outputs of E. Thus, we can give a security proof giving some hypothesis on the leakage, which can be found in [15].

Theorem 7. *Let* F *be a* $(q_E+1, \epsilon_{\mathsf{sTPRP}})$-$\mathsf{sTPRP}$ *whose implementation is leak-free, let* E *be a* $(2, \epsilon_{\mathsf{PRF}})$-$\mathsf{PRF}$ *whose implementation has some leakage property, let* PSVs^I *is* $(q_{\mathsf{L}}, \epsilon_{\mathsf{EavL2}})$-$\mathsf{EavL2}$*-secure, then, the mode* $\mathsf{CONCRETE}$*, if it encrypts at most* L*-blocks messages, is* (q_E, ϵ)-$\mathsf{CPAL2}$*-secure with*

$$\epsilon \leq \epsilon_{\mathsf{sTPRP}} + \frac{q_E}{2^n} + \epsilon_{\mathsf{2\text{-}sim'}} + (L+1)\epsilon_{\mathsf{PRF}} + L(\epsilon_{\mathsf{2\text{-}sim}} + \epsilon_{\mathsf{EavL2}}).$$

where PSVs^I is a single-message block idealized version of PSV, where the random value y and the new ephemeral key are picked uniformly at random, $\mathsf{EavL2}$ is an eavesdropper game where the adversary has to distinguish the encryption with leakage of two different plaintexts of her choice (in addition as for $\mathsf{CPAL2}$ she receives the leakage of the decryption of the ciphertext obtained). We can see $\mathsf{EavL2}$ as the $\mathsf{CPAL2}$-game where $q_E = 0$, that is, with no encryption query. The definition of $\epsilon_{\mathsf{2\text{-}sim}}$, $\epsilon_{\mathsf{2\text{-}sim'}}$ and the leakage property of the implementation of E are left to the original paper [14].

Roughly speaking, we have reduced the $\mathsf{CPAL2}$-security of the full scheme to the $\mathsf{EavL2}$ (a simpler definition) of a much simpler scheme that encrypts only n-bit long messages.

C CONCRETESponge

To have a more efficient scheme, we can use a duplex-construction based on a sponge to do both the encryption and the pass to obtain c_{l+1}.

Sponges. The sponge construction is based on a permutation $\mathsf{P} : \{0,1\}^N \rightarrow \{0,1\}^N$, where N is the *width*, thus, it has a *state* st of N bits [16].

Now, we explain how we can construct a *sponge* Sponge to hash a message m [16]. Let $r \leq N$, this is the *rate* and $c = N - r$ is the *capacity.* At the start the state st_0 of the sponge state is set to a known value, and m is parsed in m-bit blocks (for simplicity, we assume all of them are full) then, we proceed in two phases:

- *absorbing phase:* Sponge **absorbs** the message, that is, $\forall i = 1, \dots, l$ $\mathsf{st}'_{i-1} = \mathsf{st}_{i-1} \oplus m_i \| 0^c$, and then it applies the permutation P to compute the new state, that is, $\mathsf{st}_i = \mathsf{P}(\mathsf{st}'_{i-1})$.
- *squeezing phase:* Sponge **squeezes** the state to obtain a λr-bit long string h, that is, $h = h_1 \| \dots h_\lambda$, where $\forall j = 1, \dots, \lambda$ $h_j = \pi_r(\mathsf{st}_{l+j-1})$, with $\mathsf{st}_{l+j} = \mathsf{P}(\mathsf{st}_{l+j-1})$.

It has been proved that this construction cannot be differentiated from a random oracle [16].

We can also use a sponge for an authenticated encryption scheme using the *duplex construction* where the absorbing and squeezing are done at the same time [16]:

At the start, the state st_0 of the sponge state is set to a certain value (for example the key can be embedded here), and m is parsed in m-bit blocks (for simplicity, we assume all of them are full) then, we proceed as follows:

- $\forall i = 1, \dots, l$, we *absorb* $\mathsf{st}'_{i-1} = \mathsf{st}_{i-1} \oplus m_i \| 0^c$, $\mathsf{st}_i = \mathsf{P}(\mathsf{st}'_{i-1})$ and we *squeeze* $y_i = \pi_r(\mathsf{st}_i)$.

The output $y = (y_1, \dots, y_r)$ is a random string because the security of the sponge construction is related to the security of the standard sponge construction.

The sponge construction has been used many times for leakage-resilient cryptography [26, 24, 4].

Design of CONCRETESponge. The idea is to use the duplex construction, where we put $k_{0,A}$ in the initial state, to use the value squeezed from the sponge as ephemeral keys (the keys k_i) and to absorb $\mathsf{E}_{k_{i,A}}(m_i)$. The full details can be found in Alg. 9 in App. J.8.

D Collision resistance, pre-image resistance and pre-image resistant with chosen target and random n-bit prefix input

Here, we present some additional properties for hash functions, and we discuss the implications between them and the two new security definitions for hash functions (PR−corpi, Def. 10 and PR−coirv, Def. 11).

D.1 Multi-input collision resistance

In CONCRETE we use a hash function that takes two inputs. We need to define collision resistance for it, following [23]:

Definition 21. A hash function $\mathsf{H} : \mathcal{HK} \times \mathcal{X}_1 \times \mathcal{X}_2 \to \{0,1\}^n$ is (t, ϵ)-*multi-input collision resistant* (CR) if $\forall$ t-adversaries A

$$\Pr[\, ((x_0, x_1), (x_0', x_1')) \leftarrow \mathsf{A}(s) \text{ s.t. } \mathsf{H}_s(x_0, x_1) = \mathsf{H}_s(x_0', x_1'),\ (x_0, x_1) \neq (x_0', x_1') \mid s \xleftarrow{\$} \mathcal{HK}] \leq \epsilon.$$

The need for a multi-input collision resistance. In the hash we use for CONCRETE, we need to distinguish the different inputs, and not trivially $\mathsf{H}'_s(c, a) := \mathsf{H}_s(c\|a)$. In fact, if $c = c_1\|c_2, c' = c_1, a' = c_2\|a$, then $\mathsf{H}'(c, a) = \mathsf{H}(c_1\|c_2\|a) = \mathsf{H}'(c', a')$.

Ideally, we would like that from a collision for H'_s that is, $\mathsf{H}'(c, a) = \mathsf{H}'(c', a')$, to find a collision for H. One such function, has been suggested with the original description of CONCRETE [15]:

$$\mathsf{H}'(c, a) := (c_1\|0\|c_2\|0\|...\|c_l\|0\|a_1\|1\|a_2\|1...\|a_{l_a}\|1\|a_{l_a}\|1),$$

where $a_1, ..., a_{l_a}$ is a parsing of a and $c_1, ..., c_l$ one of c in n-bit blocks.

D.2 Relations between PR−corpi, PR−coirv and the standard security definitions for hash functions

First, we discuss the relation between PR−corpi and the well-known definition of collision-resistance, then we move to PR−coirv and show its relations to the well-known definitions of pre-image resistance.

PR−corpi and collision-resistance. We can prove that if a hash function is collision-resistant (Def. 1) then, it is PR−corpi (Def. 10), with n-bit prefix, using the rewinding technique [35]:

let $\mathsf{A} = (\mathsf{A}_1, \mathsf{A}_2)$ a PR−corpi adversary, we build a collision resistant adversary B as follows: B sends s, the hash key she has received to A_1. When A_1 outputs (st, y), B picks $x' \xleftarrow{\$} \{0,1\}^n$ and gives x' and st to A_2 who outputs x s.t. $\pi_n(x) = x'$ and $\mathsf{H}_s(x) = y$. Then, B rewinds A_2 and she picks $z' \xleftarrow{\$} \{0,1\}^n$ and she sends z', st to A_2 who outputs z s.t. $\pi_n(z) = z'$ and $\mathsf{H}_s(z) = y$. If $x' \neq z'$ then, A has found a collision for the hash function. A wins with probability less than $\epsilon^2_{\mathsf{PR-corpi}}$, thus

$$\epsilon^2_{\mathsf{PR-corpi}} \leq \epsilon_{\mathsf{CR}} \text{ and } \epsilon_{\mathsf{PR-corpi}} \leq \sqrt{\epsilon_{\mathsf{CR}}}.$$

We believe that for a good hash function the adversary can win with a much lower success rate (for example, it is easy to see that in the random oracle model, PR−corpi $= q/\mathcal{Y}$ where Y is target space for H and q the number of queries.

PR-coirv and range-oriented pre-image resistance. First, we introduce one of the well-known security definitions for hash functions: the range-oriented pre-image resistance. The adversary has to find a pre-image for a value that is picked uniformly at random [43].

Definition 22. A hash function $\mathsf{H} : \mathcal{HK} \times \{0,1\}^* \to \{0,1\}^N$ is (t,ϵ)-*range oriented pre-image resistant* (PR) if $\forall$ t-adversaries A

$$\Pr[m \leftarrow \mathsf{A}(s,y) \text{ s.t. } \mathsf{H}_s(m) = y \mid s \xleftarrow{\$} \mathcal{HK}, y \xleftarrow{\$} \{0,1\}^N] \leq \epsilon.$$

Now, we look to the PR-coirv definition, Def. 11.

In the random oracle model, with probability $\leq q2^{-n}$, the value picked by $\mathcal{D}$ has not been previously evaluated by H (q is the number of evaluations the adversary is allowed). Thus, the target h' is random (if the previous bad event does not happen). We expect that a similar situation will happen for reasonably good hash functions. Thus, if the target obtained in the PR-coirv experiment is indistinguishable from a random one, the probability that a PR-coirv adversary wins is the same as the probability a PR adversary wins.

E Dependency matrices and division into atoms

In this section, we give the division into atoms of some MAC and AE schemes. First, we give one of the MAC introduced at TosC 2023 [9], its division in atoms, and the security result. Then, we divide into atoms CONCRETE, CONCRETE2.

E.1 Example of atomic model - LR-MACr - proving security in the presence of faults

Berti et al. [9] proposed a scheme, LR-MACr (Alg. 3 and Fig. 2). This MAC is probabilistic. It takes as input a random value which is used *both* in the hash and as input of the TBC. In this way, the adversary cannot predict h.

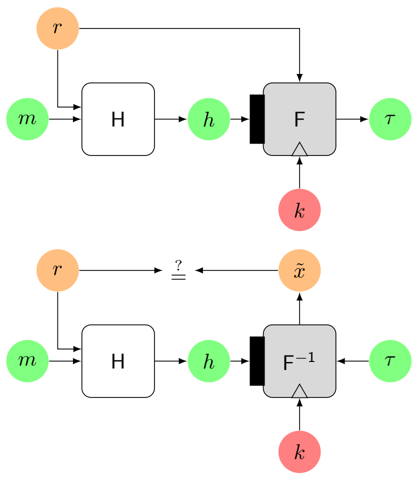

Figure 2: LR-MACr [9]

Now, we move to its leakage functions in unbounded leakage model, and then, to the divisions into atoms of the tag-generation and verification algorithms.

Algorithm 3 The $\mathsf{LR\text{-}MACr}$ algorithm [9].

It uses a strongly protected TBC $\mathsf{F} : \mathcal{K} \times \mathcal{TW} \times \{0,1\}^n \to \{0,1\}^n$ and a hash function $\mathsf{H} : \mathcal{HK} \times \{0,1\}^n \to \mathcal{TW}$.

- Gen
  - $k \xleftarrow{\$} \mathcal{K}$
  - $s \xleftarrow{\$} \mathcal{HK}$
- $\mathsf{Mac}_k(m)$:
  - $r \xleftarrow{\$} \{0,1\}^n$
  - $h = \mathsf{H}_s(r\|m)$
  - $\tau = \mathsf{F}_k^h(r)$
  - Return (r, τ)
- $\mathsf{Vrfy}_k(m, r, \tau)$:
  - $h = \mathsf{H}_s(r\|m)$
  - $\tilde{x} = \mathsf{F}_k^{h,-1}(\tau)$
  - If $\tilde{x} \overset{?}{=} r$ Return $\top$
  - Else Return $\bot$

Leakage functions. Regarding the leakage functions, we have $\mathsf{L}_{\mathsf{Mac}} = \mathsf{L}_F$ (the leakage of the computation of F), while $\mathsf{L}_{\mathsf{Vrfy}} = (\tilde{x}, \mathsf{L}_{\mathsf{F}^{-1}})$.

Division into atoms of Mac. Since there is a random input r, we assume an atom, denoted with f_0, is dedicated to this picking. From now on, we consider the randomness r as an additional input, thus, it cannot always be replaced with the same value. As the key is protected against faults, we use as atoms the picking $r \xleftarrow{\$} \{0,1\}^n$, the hash function $\mathsf{H}_s()$, and the TBC F_k [16]. In this way, we do not need to protect any input of our atoms, thus, $\mathcal{I}_{\mathsf{Mac}} = \emptyset$.

For Mac, we have as inputs $x_1 = m$, then f_0 takes no input and outputs $x_2 = r$ picked uniformly at random, $y_1 = h = \mathsf{H}_s(r\|m) = \mathsf{H}_s(x_2\|x_1)$, $\tau = y_2 = \mathsf{F}_k^h(r) = \mathsf{F}_k^{y_1}(x_2)$. We assume that the adversary can only insert differential faults. Thus, we can define the dependency matrix for Mac and characterize the fault matrices:
$\begin{pmatrix} \epsilon & \epsilon & \epsilon \\ x_1 & x_2 & \epsilon \\ \epsilon & x_2 & y_1 \end{pmatrix}$ and $\mathsf{Fa}_{\mathsf{Mac}} = \begin{pmatrix} \epsilon & \epsilon & \epsilon \\ \cdot & \oplus z_2 & \epsilon \\ \epsilon & \oplus z_3 & \oplus z_4 \end{pmatrix}$.
Note that since x_1 is used only once, then we cannot fault it (otherwise, we are computing the Mac of $m \oplus z_1$). Finally, $z_2 \neq z_3$ (if both are not equal to 0^n, that is, not injecting a fault), because otherwise, we are always replacing the random input with another.

Division into atoms of Vrfy. For Vrfy_k, the inputs are $(x_1, x_2, x_3) = (m, r, \tau)$, the atoms are $\mathsf{H}_s()$ and F_k^{-1}, and the comparison: $h = y_1 = \mathsf{H}_s(r\|m) = \mathsf{H}_s(x_2\|x_1)$, $\tilde{x} = y_2 = \mathsf{F}_k^{-1,h}(\tau) = \mathsf{F}_k^{-1,y_1}(x_3)$ and the output is the result of the comparison $\tilde{x} \overset{?}{=} r$, that is,

[16]Since the atom is F_k the protection of k is implicit, while if we use F as an atom we would have to protect the k input for F.

$y_2 \overset{?}{=} x_2$. As before, $\mathcal{I}_{\mathsf{Vrfy}} = \emptyset$. Thus, we can define the dependency matrix for Vrfy and characterize the faults, the adversary can inject (as before only differential faults): $\begin{pmatrix} x_1 & x_2 & \epsilon & \epsilon & \epsilon \\ \epsilon & \epsilon & x_3 & y_1 & \epsilon \\ \epsilon & x_2 & \epsilon & \epsilon & y_2 \end{pmatrix}$ and $\mathsf{Fa}_{\mathsf{Vrfy}} = \begin{pmatrix} \cdot & \oplus z_2 & \epsilon & \epsilon & \epsilon \\ \epsilon & \epsilon & \cdot & \cdot & \epsilon \\ \epsilon & \oplus z_5 & \epsilon & \epsilon & \oplus z_6 \end{pmatrix}$. Note that x_1 and and x_3 are used only twice, moreover $z_2 \neq z_5$ (if there is a fault).

SUF-FL2-security of LR-MACr. Finally, we move to the proof of the SUF-FL2-security of LR-MACr.

To prove the SUF-FL2-security of LR-MACr, we model the TBC as strongly unpredictable in the presence of leakage[17] We assume that the key of the TBC is protected against faults Now, we can state the SUF-FL-security of LR-MACr [18] in the unbounded leakage model (with also leakage for the TBC):

Theorem 8. *Let hash function* H *be* $(t_1, \epsilon_{\mathsf{CR}})$*-collision resistant and* $(t_2, \epsilon_{\mathsf{PR}})$*-preimage resistant in the hash oracle model. Let* F *be a* $(q_{FL}, q_M, q_V, t_3), \epsilon_{\mathsf{SUP\text{-}L2}})$-SUP-L2 TBC *with fault immune long-term key. Then, for any* (q_{FL}, q_M, q_V, t)*-adversary* $\mathsf{A}^{\mathsf{L},\mathsf{Fa}}$ *with leaking function pair* $\mathsf{L} = (\mathsf{L}_{\mathsf{Mac}}, \mathsf{L}_{\mathsf{Vrfy}})$ *and fault-injection pair* $\mathsf{Fa} = (\mathsf{Fa}_{\mathsf{Mac}}, \mathsf{Fa}_{\mathsf{Vrfy}})$. LR-MACr *is* $(q_{FL}, q_M, q_V, t, \epsilon)$*-strongly existentially unforgeable against unbounded differential fault-then-leak attacks in tag-generation and verification with*

$$\epsilon \leq \epsilon_{\mathsf{CR}} + (q_V + 1)\epsilon_{\mathsf{SUP\text{-}L2}} + \epsilon_{\mathsf{PR}} + \frac{q_M^2}{2^{n+1}} + \frac{q_M}{2^n}.$$

For technical reasons, the SUF-FL2-security of LR-MACr has been proved in the *hash oracle model*[19].

E.2 Divisions into atoms of CONCRETE

Encryption. Here, we give the dependency matrix of the encryption of CONCRETE:

$$\begin{pmatrix}
\epsilon & \cdots & \epsilon & \cdots & \epsilon & \epsilon & \epsilon & \cdots & \epsilon & \cdots & \epsilon & \epsilon & \epsilon & \cdots & \epsilon \\
\epsilon & \cdots & \epsilon & \cdots & \epsilon & z_{0.1} & \epsilon & \cdots & \epsilon & \cdots & \epsilon & \epsilon & \epsilon & \cdots & \epsilon \\
\vdots & & & & & \vdots & \ddots & \ddots & & \vdots & & \vdots & \vdots & & \vdots \\
\epsilon & \cdots & \epsilon & \cdots & \epsilon & \epsilon & \epsilon & \cdots & z_{i-1.1} & \cdots & \epsilon & \epsilon & \epsilon & \cdots & \epsilon \\
\epsilon & \cdots & \epsilon & \cdots & \epsilon & \epsilon & \epsilon & \cdots & \epsilon & \cdots & z_{i.1} & \epsilon & \epsilon & \cdots & \epsilon \\
\epsilon & \cdots & x_i & \cdots & \epsilon & \epsilon & \epsilon & \cdots & \epsilon & \cdots & \epsilon & z_{i.2} & \epsilon & \cdots & \epsilon \\
\vdots & & & & \vdots & \vdots & \vdots & & \vdots & & \vdots & \ddots & \ddots & & \vdots \\
\epsilon & \cdots & \epsilon & \cdots & x_{l+1} & \epsilon & z_{0.2} & \cdots & \epsilon & \cdots & \epsilon & \epsilon & z_{i.3} & \cdots & \epsilon \\
\epsilon & \cdots & \epsilon & \cdots & \epsilon & z_{0.1} & \epsilon & \cdots & \epsilon & \cdots & \epsilon & \epsilon & \epsilon & \cdots & \epsilon
\end{pmatrix} \tag{3}$$

where we have represented the lines corresponding to the computations 0.1,0.2,i.1,i.2,i.3,$l+1.1$, and $l + 1.2$ (the columns correspond to the variables m_1, m_i, a, k_0, c_0, k_i, y_i, c_i, and h).

Decryption. We proceed similarly for decryption. For the query on input (a, c), we define $(x_0, x_1, \ldots, x_{l+1}) = (c_0, \ldots, c_{l+1})$, the parsing of c in n-bit long blocks, $x_{l+2} = a$. Moreover, as atoms, we consider $\mathsf{F}_k^{-1,\cdot}(\cdot)$, $\mathsf{E}.(p_A)$, $\mathsf{E}.(p_B)$, $\mathsf{H}'_s(\cdot,\cdot)$ and the XOR $\cdot \oplus \cdot$. Additionally, as

[17] *Strong unpredictability in the presence of leakage* (SUP-L2), introduced in [7], models the security of strongly protected (T)BC. It implies that finding a fresh and valid couple (input/output) for the BC even with leakage is difficult. It is a falsifiable notion by an evaluation lab. It is less demanding than asking that a BC is leak-free.

[18] For simplicity, we do not state the time bounds. They can be found in [9].

[19] In this model, we allow the adversary to choose any target for the hash H_s she wants (to find the pre-image), as long as she has never seen y as output of a previous evaluation of H.

in [9], we do not consider the comparison $c_0 \overset{?}{=} \tilde{c}_0$ as an atom, because we assume that both values are known by the adversary (one via leakage, and the other is part of the input).

We define $f_{0.0} = h = \mathsf{H}'(c_0\|...\|c_l, a)$, $f_{0.1} = \mathsf{F}_k^{-1,h}(c_{l+1})$, $f_{0.2} = \mathsf{E}_{k_0}(p_B)$, then we iterate these three blocks for $i = 1, ..., l-1$: $f_{i.1} = k_i = \mathsf{E}_{k_{i-1}}(p_A)$, $f_{i.2} = y_i = \mathsf{E}_{k_i}(p_B)$, $f_{i.3} = m_i = y_i \oplus c_i$.

We observe that $f_{0.0}$ uses $(x_0, \dots, x_{l+1})$, and x_{l+2} as input (that is, $(c_0, \dots, c_l)$, and a), while $f_{0.1}$ only $z_{0.0} = k_0$. For all $i = 1, ..., l$, $f_{i.1}$ uses only $z_{i-1.1} = k_{i-1}$, $f_{i.2}$ uses only $z_{i.1} = k_i$, and $f_{i.3}$ only $z_{i.2} = y_i$ and $x_i = c_i$. The output is $(z_{1.3}, ..., z_{l.3}) = (m_1, ..., m_l)$, for a valid query, $\perp$ otherwise. Thus, we can give the dependency matrix:

$$\begin{pmatrix}
x_0 & \cdots & x_i & \cdots & x_{l+1} & x_{l+2} & \epsilon & \epsilon & \cdots & \epsilon & \cdots & \epsilon & \epsilon & \epsilon & \cdots \\
\epsilon & \cdots & \epsilon & \cdots & \epsilon & \epsilon & z_{0.0} & \epsilon & \cdots & \epsilon & \cdots & \epsilon & \epsilon & \epsilon & \cdots \\
\vdots & & & & \vdots & \ddots & \ddots & \vdots & & \vdots & & \vdots & \vdots & \vdots & \\
\epsilon & \cdots & \epsilon & \cdots & \epsilon & \epsilon & \epsilon & \epsilon & \cdots & z_{i-1.1} & \cdots & \epsilon & \epsilon & \epsilon & \cdots \\
\epsilon & \cdots & \epsilon & \cdots & \epsilon & \epsilon & \epsilon & \epsilon & \cdots & \epsilon & \cdots & z_{i.1} & \epsilon & \epsilon & \cdots \\
\epsilon & \cdots & x_i & \cdots & \epsilon & \epsilon & \epsilon & \epsilon & \cdots & \epsilon & \cdots & \epsilon & z_{i.2} & \epsilon & \cdots \\
\vdots & & & & \vdots & \vdots & \vdots & \vdots & & \vdots & & \ddots & \ddots & \ddots &
\end{pmatrix} \tag{4}$$

where we have represented the lines corresponding to the computations 0.0,0.1,i.1,i.2,i.3 (the columns correspond to the variables c_0, c_i, c_{l+1}, a, k, k_0, k_i, y_i, m_i).

E.3 Divisions into atoms of CONCRETE2

Encryption. First, we observe that the associated data, a is used only once by CONCRETE2, thus, according to the atomic model (Sec. 2.6.1), it cannot be faulted.

For the query on input (a, m), we define $(x_1, \dots, x_l) = (m_1, \dots, m_l)$, the parsing of m in n-bit long blocks, $x_{l+1} = a$. Similar to what done in [9], we assume that the master key is hard coded into the implementation, thus, we consider $\mathsf{F}_k^{\cdot}(\cdot)$ as an atom; moreover, we consider the production of randomness as an additional atom. So, as atoms, we consider $\mathsf{F}_k^{\cdot}(\cdot)$, $\mathsf{E}_{\cdot}(p_A)$, $\mathsf{E}_{\cdot}(p_B)$, $\mathsf{E}_{\cdot}(\cdot)$, $\mathsf{H}'_s(\cdot, \cdot)$, and the random picking $\cdot \xleftarrow{\$} \{0,1\}^n$.

We define $f_{0.1} = k_0 \xleftarrow{\$} \{0,1\}^n$, $f_{0.2} := k_{0,E} = \mathsf{E}_{k_0}(p_B)$, $f_{0.3} := k_{0,A} = \mathsf{E}_{k_0}(p_A)$, $f_{0.4} := c_0 = \mathsf{E}_{k_{0,E}}(p_B)$, $f_{0.5} := k_1 = \mathsf{E}_{k_{0,E}}(p_A)$; then we iterate these three blocks for $i = 1, ..., l$: $f_{i.1} := k_{i,E} = \mathsf{E}_{k_i}(p_B)$, $f_{i.2} := c_i = \mathsf{E}_{k_{i,E}}(m_B)$, $f_{i.3} := k_{i+1} = \mathsf{E}_{k_i}(p_A)$, and $f_{i.4} := k_{i,A} = \mathsf{E}_{k_{i-1,A}}(m_i)$; after that, $f_{l+1.1} := k_{l+1,E} = \mathsf{E}_{k_{l+1}}(p_B)$, $f_{l+1.2} := c_{l+1} = \mathsf{E}_{k_{l+1,E}}(k_{l,A})$, $f_{l+2.1} := h = \mathsf{H}'(c_0\| \dots \|c_{l+1}, a)$, and $f_{l+2.2} := c_{l+2} = \mathsf{F}_k^h(k_0)$.

We observe that $f_{0.1}$ uses no input, m_i is used only in the $f_{i.2}$ and $f_{i.4}$ atoms. The output is $(z_{0.4}, z_{1.2}, ..., z_{l.2}, z_{l+1.2}, z_{l+2.2}) = (c_0, c_1, ..., c_l, c_{l+1}, c_{l+2})$, that is, $\mathsf{Select}(z_{0.0}, \dots, z_{l+2.2}) = (z_{0.4}, z_{1.2}, ..., z_{l.2}, z_{l+1.2}, z_{l+2.2})$.

Here, we give the dependency matrix of the encryption of CONCRETE2:

$$\begin{pmatrix}
\cdots & \epsilon & \cdots & \epsilon & \epsilon & \epsilon & \epsilon & \epsilon & \cdots & \epsilon & \epsilon & \epsilon & \epsilon & \cdots & \epsilon & \epsilon & \epsilon & \epsilon \\
\cdots & \epsilon & \cdots & \epsilon & z_{0.1} & \epsilon & \epsilon & \epsilon & \cdots & \epsilon & \epsilon & \epsilon & \epsilon & \cdots & \epsilon & \epsilon & \epsilon & \epsilon \\
\cdots & \epsilon & \cdots & \epsilon & z_{0.1} & \epsilon & \epsilon & \epsilon & \cdots & \epsilon & \epsilon & \epsilon & \epsilon & \cdots & \epsilon & \epsilon & \epsilon & \epsilon \\
\cdots & \epsilon & \cdots & \epsilon & \epsilon & z_{0.2} & \epsilon & \epsilon & \cdots & \epsilon & \epsilon & \epsilon & \epsilon & \cdots & \epsilon & \epsilon & \epsilon & \epsilon \\
\cdots & \epsilon & \cdots & \epsilon & \epsilon & \epsilon & z_{0.3} & \epsilon & \cdots & \epsilon & \epsilon & \epsilon & \epsilon & \cdots & \epsilon & \epsilon & \epsilon & \epsilon \\
\vdots & & & & & & & \vdots & \ddots & \ddots & & \ddots & & \ddots & \vdots & & \vdots & \vdots \\
\cdots & \epsilon & \cdots & \epsilon & \epsilon & \epsilon & \epsilon & \epsilon & \cdots & z_{i-1.3} & \epsilon & \epsilon & \epsilon & \cdots & \epsilon & \epsilon & \epsilon & \epsilon \\
\cdots & x_i & \cdots & \epsilon & \epsilon & \epsilon & \epsilon & \epsilon & \cdots & \epsilon & \epsilon & z_{i.1} & \epsilon & \cdots & \epsilon & \epsilon & \epsilon & \epsilon \\
\cdots & \epsilon & \cdots & \epsilon & \epsilon & \epsilon & \epsilon & \epsilon & \cdots & z_{i-1.3} & \epsilon & \epsilon & \epsilon & \cdots & \epsilon & \epsilon & \epsilon & \epsilon \\
\cdots & x_i & \cdots & \epsilon & \epsilon & \epsilon & \epsilon & \epsilon & \cdots & \epsilon & z_{i-1.4} & \epsilon & \epsilon & \cdots & \epsilon & \epsilon & \epsilon & \epsilon \\
\vdots & & & & & \vdots & & \vdots & \ddots & & \ddots & & & \ddots & \vdots & & \vdots & \vdots \\
\cdots & \epsilon & \cdots & \epsilon & \epsilon & \epsilon & \epsilon & \epsilon & \cdots & \epsilon & \epsilon & \epsilon & \epsilon & \cdots & z_{l.3} & \epsilon & \epsilon & \epsilon \\
\cdots & \epsilon & \cdots & \epsilon & \epsilon & \epsilon & \epsilon & \epsilon & \cdots & \epsilon & \epsilon & \epsilon & \epsilon & \cdots & \epsilon & \epsilon & z_{l+1.1} & \epsilon \\
\cdots & \epsilon & \cdots & \epsilon & \epsilon & \epsilon & \epsilon & \epsilon & \cdots & \epsilon & \epsilon & \epsilon & \epsilon & \cdots & \epsilon & z_{l.4} & z_{l+1.1} & \epsilon \\
\cdots & \epsilon & \cdots & x_{l+1} & \epsilon & \epsilon & \epsilon & z_{0.4} & \cdots & \epsilon & \epsilon & \epsilon & z_{i.2} & \cdots & \epsilon & \epsilon & z_{l+1.1} & \epsilon \\
\cdots & \epsilon & \cdots & \epsilon & z_{0.1} & \epsilon & \epsilon & \epsilon & \cdots & \epsilon & \epsilon & \epsilon & \epsilon & \cdots & \epsilon & \epsilon & \epsilon & z_{l+2.1}
\end{pmatrix} \quad (5)$$

where we have represented the lines corresponding to the computations 0.1,0.2,0.3,0.4,0.5,i.1,i.2,i.3,i.4,$l+1.1$,$l+1.2$,$l+2.1$ and $l+2.2$ (the columns correspond to the variables m_i, a, k_0, $k_{0,E}$, $k_{0,A}$, c_0, k_i, $k_{i-1,A}$, $k_{i,E}$, c_i, k_{l+1}, $k_{l,A}$, c_{l+1}, and h).

Decryption. We proceed similarly for decryption. For the query on input (a, c), we define $(x_0, x_1, \dots, x_{l+2}) = (c_0, \dots, c_{l+1}, c_{l+2})$, the parsing of c in n-bit long blocks, $x_{l+3} = a$. Moreover, as atoms, we consider $\mathsf{F}_k^{-1,\cdot}(\cdot)$, $\mathsf{E}_\cdot(p_A)$, $\mathsf{E}_\cdot(p_B)$, $\mathsf{E}_\cdot^{-1}(\cdot)$, $\mathsf{E}_\cdot(\cdot)$, and $\mathsf{H}'_s(\cdot,\cdot)$. Additionally, as in [9], we do not consider the comparisons $c_0 \overset{?}{=} \tilde{c}_0$, $k_{l,A} \overset{?}{=} \tilde{k}_{l,A}$ as an atom, because we assume that both values are known by the adversary (one via leakage, and the other is part of the input, or both via leakage).

We define $f_{0.1} := h = \mathsf{H}'(c_0\|...\|c_{l+1}, a)$, $f_{0.2} := k_0 = \mathsf{F}_k^{-1,h}(c_{l+2})$, $f_{0.3} := k_{0,E} = \mathsf{E}_{k_0}(p_A)$, $f_{0.4} := k_{0,A} = \mathsf{E}_{k_0}(p_B)$, $f_{0.5} := \tilde{c}_0 = \mathsf{E}_{k_{0,E}}(p_B)$, $f_{0.6} := k_1 = \mathsf{E}_{k_{0,E}}(p_A)$. Then we iterate these four blocks for $i = 1, ..., l$: $f_{i.1} := k_{i,E} = \mathsf{E}_{k_i}(p_B)$, $f_{i.2} := m_i = \mathsf{E}_{k_{i,E}}^{-1}(c_i)$, $f_{i.3} := k_{i+1} = \mathsf{E}_{k_i}(p_A)$, $f_{i.4} := k_{i+1} = \mathsf{E}_{k_{i-1,A}}(m_i)$. Finally, we compute $f_{l+1.1} := k_{l+1,E} = \mathsf{E}_{k_{l+1}}(p_B)$, and $f_{l+1.2} := \tilde{k}_{l,A} = \mathsf{E}_{k_{l+1,E}}^{-1}(c_{l+1})$.

The output is $(z_{1.2}, ..., z_{l.2}) = (m_1, ..., m_l)$, for a valid query, $\perp$ otherwise. The dependency matrix can be found in the full version of the paper.

F Comparison between CONCRETE2 and MEM

With respect to CONCRETE, MEM (Alg. 2, described in App. A.6.2) is generally a 3-pass scheme, however in terms of performance it is quasi-2-pass, as the pass producing the ciphertext blocks and the computation of the commitment of the message can be performed in parallel. Moreover, per each message block, we use 4 calls to the block cipher E, while CONCRETE uses only two. But, similarly to CONCRETE, we still use the master key only once in the leak-free and fault-free component (which is used once). Since CONCRETE2 uses twice the message, for an adversary, via leakage, one may think it is easier to have information about the message encrypted with respect to CONCRETE, but this, in our opinion is inevitable since we believe that there must be a commitment on the plaintext to avoid attacks that are not covered in the weak fault security model (this result is coherent with [45]).

With respect to MEM [47], one of the main strengths is that we use significantly less the strongly protected component (once as compared to three times), and we provide authenticity even if the adversary can inject any fault in decryption. On the other hand, CONCRETE2 is a slightly more complex using 4 queries to E and one hash evaluation, with respect to two queries to E and two hash evaluations. Moreover, CONCRETE2 can be executed faster as the two passes can be done in parallel (quasi-1-pass), while for MEM, we need to execute one pass after the previous is finished.

G Attack on CONCRETE2 setting more than one value

The attack represented in Sec. 5.2 is related to a theoretical result which we informally state here:

Theorem 9. *In the unbonded leakage model, if the adversary can set all the values of an atom in the direct queries (i.e.,* Enc *or* Mac*), then no* CIL2F2 *(or* SUF−FL2*) security is achievable.*

Proof. (*Idea:*) We consider only the atom when the key of the scheme is used (or part of the key). Using a fault, an adversary can choose the inputs of this atom. Then, via leakage, she obtains the output. Thus, she has oracle access to all atoms using the key. Having this is easy to compute a forgery because she can compute all other atoms having this oracle access (we have only to check that these faults are admissible). □

This result cannot (luckily for us) be extended when the adversary can inject any fault in the atom of inverse queries (that is, Dec and Vrfy) because, as Berti et al. do for LR−MACr, in these queries it is possible to avoid using the atoms (using the master key) which are used in the direct queries.

For simplicity, we have not used the dependency matrix and the faulty matrix since they can be obtained straightforwardly.

G.0.1 Setting one value per atom.

In the previous attack, the adversary can forge simply by setting the two inputs of the atom $\mathsf{F}^{\cdot}_{k}(\cdot)$, so it is natural to see if it is possible to attack CONCRETE2 if the adversary can set a single value per atom (thus, preventing the previous attack which gives to the adversary oracle access to $\mathsf{F}^{\cdot}_{k}(\cdot)$).

Unfortunately, an attack is possible. The adversary can proceed as follows: she chooses k'_0, then she computes the ephemeral keys $k'_{0,A}, k'_1$ and the ciphertext block c'_0 as for an encryption query where the key picked is k'_0. Then, she does an encryption query on input (a, m), where she inject these faults: in the computation of $k_{1,A}$, she replaces $k_{0,A}$ with $k'_{0,A}$, every time k_1 is used, we replace it with k'_1, in the computation of h we replace c_0 with c'_0. The output of this faulted query is $c = (c_0, \dots, c_{l+1}, c_{l+2})$. The adversary outputs (a, c^*) with $c^* = (c'_0, c_1, \dots, c_{l+2})$.

Clearly, c^* is fresh. Moreover, c^* is valid since hashing $c'_0, c_1, \dots, c'_{l+1}$ and a, we obtain the same h used in the encryption query, and $\mathsf{F}^{-1,h}_{k}(c_{l+2}) = k'_0$. c'_0 is the correct committing for k'_0, and $c'_1, \dots, c'_l$ is the correct encryption for m using k'_0 as via faults we have forced to encrypt using k'_0 and c'_{l+1} is the correct committing for m and k'_0 (due to the faults when c'_{l+1}).

Thus, CONCRETE2 cannot be CIL2F2 secure when the adversary can set a value per atom via a fault.

Note that in this attack we set 4 values (once c_0, twice k_1 and once $k_{0,A}$).

H CONCRETE2 if the message is not full

In this section, we explain how we can modify CONCRETE2 if the last message block is not full. The idea is to make this last block full appending as many 1 as needed. Moreover, together with the ciphertext we send the length of the last message block. Moreover, this length is sent together with the ciphertext and is one of the input of the hash function. Formally we proceed as for $\mathsf{CONCRETE2}$ with these differences:

- instead of using m_l, we use $m'_l = m_l \| 1^{n-|m_l|}$
- $h = \mathsf{H}(c, a, \lambda)$ with $\lambda = |m_l|$
- The output is (c, λ)

In most of the security definitions, for example CPA, AE, CCA, the adversary is allowed to know the length of the plaintext [32, 44, 39].

I Proofs

I.1 Proof of Thm. 1

Theorem 10. *Let* F *be a strongly protected* $(q, t_1, \epsilon_{\mathsf{sTPRP}})$-$\mathsf{sTPRP}$*, let* E *be an ideal block-cipher, let* A *be allowed* q_I *queries to* E*, let* H *be a* $(t_2, \epsilon_{\mathsf{CR}})$*-collision resistant and* $(t_3, \epsilon_{\mathsf{PR\text{-}corpi}})$-$n$-$\mathsf{PR\text{-}corpi}$ *hash function. Then,* $(\mathcal{I}_{\mathsf{AEnc}}, \mathcal{I}_{\mathsf{ADec}})$*-protected implementation of the* AE*-scheme* $\mathsf{CONCRETE}\Pi = (\mathsf{Gen}, \mathsf{AEnc}, \mathsf{ADec})$*, encoding messages at most* Ln *bits long, with leaking function pair* $\mathsf{L} = (\mathsf{L}_{\mathsf{AEnc}}, \mathsf{L}_{\mathsf{ADec}})$ *and fault admissible sets* $\mathcal{F} = (\mathcal{F}_E, \mathcal{F}_D)$ *has* $(q_I, q_E, q_D, t, \epsilon)$-ciphertext integrity in the presence of leakage and faults against $\mathcal{F}$-fault-then-leakage attacks in encryption and decryption ($\mathsf{CIL2Fd}$)*, with*

$$\epsilon \leq \epsilon_{\mathsf{sTPRP}} + \epsilon_{\mathsf{CR}} + (q_D + 1)\epsilon_{\mathsf{PR\text{-}corpi}} + \frac{(q_D + 2)(q_D + 1)}{2^{n+1}} + \frac{q_D[(q_E(L + 1) + 1)(q_D + 1)/2 + q_I]}{2^n},$$

with q_I *the number of queries the adversary* A *is allowed to do to the ideal block-cipher* E*, with* $q = q_E + q_D + 1$*,* $q_I = t_1 = t + qt_{\mathsf{H}} + (q(2L + 1) + 1)t_{\mathsf{E}}$*,* $t_2 = t + q(t_{\mathsf{H}}) + (q_E + q_D)[(2L + 2) + 1)t_{\mathsf{E}}]$ $t_3 = t + q(t_{\mathsf{H}}) + (q_E + q_D + 1)[(2L + 2) + 1)t_{\mathsf{E}} + t_{\mathsf{F}}] - t_{\mathsf{F}} - t_{\mathsf{E}}$*.*

We assume that picking a value uniformly at random does not take time.

Proof. We use a sequence of games for the proofs. For simplicity, we consider the decryption of the challenge associated data and ciphertext as the $q_D + 1$th decryption query.

Game 0. It is the standard $\mathsf{CIL2Fd}_{\mathcal{F},\mathsf{L},\mathsf{A}^{\mathsf{L}}}$ game. Let E_0 be the event that A wins this game.

Game 1. It is Game 0 except that we have replaced F with a random tweakable permutation f. Let E_1 be the event that A wins this game.

Transition between Game 0 and Game 1. Since Game 0 and Game 1 are the same except for the replacement of F with f, we can build an adversary B to bound the difference. She plays the sTPRP game, having to distinguish F_k from f, and simulates A's oracles using its own.

At the start of the game, B picks s in $\mathcal{HK}$ and gives it to A, and an ideal block-cipher E. Moreover, he has a list $\mathcal{S}$ which is empty.

When A does an encryption query on input (a, m), B simply picks k_0 uniformly at random from $\{0, 1\}^n$ and after having parsed m in $(m_1, \ldots, m_l)$, computes $c_0, \ldots, c_l$ as follows: $c_0 = \mathsf{E}_{k_0}(p_B)$, and then, for all $i = 1, \ldots, l$, $k_i = \mathsf{E}_{k_{i-1}}(p_A)$, $y_i = \mathsf{E}_{k_i}(p_B)$ and $c_i = y_i \oplus m_i$ (for $i = l$ $c_l = \pi_{|m_l|}(y_l) \oplus m_l$. Then, B computes $h = \mathsf{H}'_s(a, c_0 \| \ldots \| c_l)$ and she queries her oracle on input (h, k_0) receiving as answer c_{l+1}. So, she can answer A $c = (c_0 \| \ldots \| c_{l+1})$ and the leakage k_0. Moreover, B adds (a, c) to the list $\mathcal{S}$.

For this, B does one oracle query and needs time $t_{\mathsf{H}} + (2l + 1)t_{\mathsf{E}} \leq t_{\mathsf{H}} + (2L + 1)t_{\mathsf{E}}$.

When A does a decryption query on input (a, c, Fa), B proceeds as follows: 1) she computes $h = \mathsf{H}'_s(a, c_0 \| \ldots \| c_l)$ where a and $c_0, \ldots, c_l$ are modified according to the 0.0 row of Fa, 2) she queries her inverse oracle on input (h, c_{l+1}) which are modified according to 0.1 row of Fa, obtaining k_0 as answer, 3) she computes $\tilde{c_0} = f_{k_0}(p_A)$ with k_0 modified according the 0.2 line of Fa, if $\tilde{c_0} \neq c_0$, she answers $\perp$ and the leakage k_0 to A; otherwise 4) for $i = 1, \ldots, l$, a) computes $k_i = \mathsf{E}_{k_{i-1}}(p_A)$ with k_{i-1} modified according to the i.1 line of Fa, b) computes $y_i = \mathsf{E}_{k_i}(p_B)$ with k_i modified according to the i.2 line of Fa c) computes

$m_i = y_i \oplus c_i$ (if $i = l$ $\pi_{|c_l|}(y_l) \oplus c_l$) with y_i and c_i modified according to the i.3 line of Fa, 5) finally, B answers $m = (m_1, \dots, m_l)$ and the leakage k_0 to A.

For this, B does one oracle query and time $t_{\mathsf{H}} + (2l+1)t_{\mathsf{E}} \leq t_{\mathsf{H}} + (2L+1)t_{\mathsf{E}}$.

When A outputs her output (a^*, c^*), B proceeds as follows: 1) she computes $h^* = \mathsf{H}'_s(a^*, c_0^* \| \dots \| c_l^*)$, 2) she queries her inverse oracle on input (h^*, c_{l+1}^*), obtaining k_0^* as answer, 3) she computes $\tilde{c_0^*} = f_{k_0^*}(p_A)$, if $\tilde{c*}_0 = c*_0$ and $(a^*, c^*) \notin \mathcal{S}$, she outputs 1; otherwise 0.

For this, B does one oracle query and time $t_{\mathsf{H}} + t_{\mathsf{E}}$.

Thus, in total B does at most $q_E + q_D + 1 = q$ queries to her oracle and runs in time bounded by $t + q(t_{\mathsf{H}}) + (q(2L+1)+1)t_{\mathsf{E}} = t_1$.

If her oracle is implemented with F, B correctly simulates Game 0 for A, otherwise Game 1. Since F is a $(q, t_1, \epsilon_{\mathsf{sTPRP}})$-$\mathsf{sTPRP}$,

$$|\Pr[E_1] - \Pr[E_0]| \leq \epsilon_{\mathsf{sTPRP}}.$$

Game 2. It is Game 1, where in decryption instead of using f^{-1}, where f is a random tweakable permutation, we pick its answers uniformly at random (and we keep a list of all queries already done to make the answers coherent). Let E_2 be the event that A wins this game.

Transition between Game 1 and Game 2. In Game 2, for decryption queries, we are using a sTPRP, while in Game 3 a PRF. Due to the well-known birthday bound, we have that

$$|\Pr[E_2] - \Pr[E_1]| \leq \frac{(q_D+2)(q_D+1)}{2^{n+1}}.$$

Game 3. It is Game 3, where during decryption queries (not for the challenge one), immediately after k_0 is computed (via f^{-1}) the decryption oracle computes $c_{l+2} := \mathsf{E}_{k_0}(p_B)$, without any faults inserted, and appends this to its normal answer. Let E_3 be the event that the adversary wins this game.

Transition between Game 2 and Game 3. We build an adversary C. C behaves as A for encryption queries. For decryption queries, C proceeds as follows: when A does a decryption query on input (a, c), C simply forwards this query to her oracle. After having received the answer and the leakage k_0, C simply takes k_0, calls the ideal cipher E on input (k_0, p_B), obtains c_{l+2}, and answer A with the oracle's answer appended with c_{l+2}. Thus, C needs time $t + q_D t_{\mathsf{E}}$ and does $q_I + q_D$ queries to the ideal cipher E. Thus, since

$$\Pr[E_2] = \Pr[E_3].$$

Game 4. It is Game 3, with the condition that the following event does not happen: during a decryption query, when a new key k_0 is picked, there exists no query to the ideal cipher on input (k_0, p_B). Let E_4 be the event that the adversary wins this game.

Transition between Game 3 and Game 4. The two games are identic except if the following Bad_1 event happens: there exists a decryption query s.t. it requires to pick a new key k_0^i and there exists a previous query to the ideal cipher E on input (k_0^i, p_B). We observe that during an encryption query there at most $L+1$ queries to $(\cdot, p_B)$ with at most $L+1$ different keys, while during a decryption query there at most $L+2$ different queries to the ideal block-cipher with input (k_i, p_B). The adversary can do at most q_I queries of its choice to the ideal block-cipher. Thus, during the jth decryption query there at most $q_E(L+1) + q_I + (j-1)(L+2)$ different keys that, if picked, would trigger the Bad_1 event. Thus,

$$\Pr[\mathsf{Bad}_1] \leq \sum_{j=1}^{q_D} \frac{q_E(L+1) + q_I + (j-1)(L+2)}{2^n} \leq$$

$$\frac{q_D[q_E(L+1) + q_I] + q_D(\frac{q_D+1}{2})}{2^n} = q_D[q_E(L+1)\frac{q_D+1}{2} + q_I + \frac{q_D+1}{2}]2^{-n}.$$

Thus, $|\Pr[E_4] - \Pr[E_3]| \leq q_D[q_E(L+1)(q_D+1)/2 + q_I + (q_D+1)/2]2^{-n}$.

Game 5. It is Game 4, where the following event Bad_2 does not happen: the hash queries induced during encryption and decryption queries induce a collision for the hash function. Let E_5 be the event that the adversary wins this game.
Transition between Game 4 and Game 5. Since Game 4 and Game 5 are the same except if event Bad_2 happens, it is enough to bound $\Pr[\mathsf{Bad}_2]$ to bound the difference $|\Pr[E_4] - \Pr[E_5]|$. To bound $\Pr[\mathsf{Bad}_2]$, we build an adversary D who wants to find a collision for the hash function. At the start of the game, D receives a key for the hash function s in $\mathcal{HK}$ and gives it to A. Moreover, she picks a random tweakable permutation f. Finally, he has a list $\mathcal{H}$ which is empty.

When A does an encryption query on input (a, m), D simply picks k_0 uniformly at random from $\{0,1\}^n$ and after having parsed m in $(m_1, \dots, m_l)$, computes $c_0, \dots, c_l$ as follows: $c_0 = \mathsf{E}_{k_0}(p_B)$, and then, for all $i = 1, \dots, l$, $k_i = \mathsf{E}_{k_{i-1}}(p_A)$, $y_i = \mathsf{E}_{k_i}(p_B)$ and $c_i = y_i \oplus m_i$ (for $i = l$ $c_l = \pi_{|m_l|}(y_l) \oplus m_l$. Then, D computes $h = \mathsf{H}'_s(a, c_0\| \dots \|c_l)$, she adds $((a, c_0\| \dots \|c_l), h)$ to $\mathcal{H}$ and she computes $c_{l+1} = \mathsf{f}(h, k_0)$. So, she can answer A $c = (c_0\| \dots \|c_{l+1})$ and the leakage k_0.(

For this, D needs time $t_\mathsf{H} + (2l+1)t_\mathsf{E} + t_\mathsf{F} \leq t_\mathsf{F} + t_\mathsf{H} + (2L+1)t_\mathsf{E}$.

When A does a decryption query on input (a, c, Fa), D proceeds as follows: 1) she computes $h = \mathsf{H}'_s(a, c_0\| \dots \|c_l)$ where a and $c_0, \dots, c_l$ are modified according to the 0.0 row of Fa, and she adds $((a, c_0\| \dots \|c_l), h)$ to $\mathcal{H}$, 2) she computes $k_0 = \mathsf{f}^{-1}(h, c_{l+1})$ where the inputs are modified according to 0.1 row of Fa, obtaining k_0 as answer, 3) she computes $\tilde{c_0} = f_{k_0}(p_A)$ with k_0 modified according the 0.2 line of Fa and $c_{l+2} = \mathsf{E}_{k_0}(p_B)$, if $\tilde{c_0} \neq c_0$, she answers $\perp$ and the leakage k_0 to A; otherwise 4) for $i = 1, \dots, l$, a) computes $k_i = \mathsf{E}_{k_{i-1}}(p_A)$ with k_{i-1} modified according to the i.1 line of Fa, b) computes $y_i = \mathsf{E}_{k_i}(p_B)$ with k_i modified according to the i.2 line of Fa c) computes $m_i = y_i \oplus c_i$ (if $i = l$ $\pi_{|c_l|}(y_l) \oplus c_l$) with y_i and c_i modified according to the i.3 line of Fa, 5) finally, D answers $m = (m_1, \dots, m_l)$ and the leakage k_0 to A.

For this, D needs time $t_\mathsf{F} + t_\mathsf{H} + (2l+2)t_\mathsf{E} \leq t_\mathsf{F} + t_\mathsf{H} + (2L+2)t_\mathsf{E}$.

When A outputs her output (a^*, c^*), D proceeds as follows: 1) she computes $h^* = \mathsf{H}'_s(a^*, c^*_0\| \dots \|c^*_l)$ and she adds $((a^*, c^*_0\| \dots \|c^*_l), h)$ to $\mathcal{H}$, 2) she looks through $\mathcal{H}$ to see if there is a collision. If it is the case, she outputs it, otherwise $0^n, 1^n$. For this, D needs time t_H.

Thus, in total D runs in time bounded by $t + q(t_\mathsf{H}) + (q_E + q_D)[(2L+2)+1)t_\mathsf{E} + t_\mathsf{F}] = t_2$.

D correctly simulates Game 4 for A. Moreover, D finds a collision for the hash function H if event Bad_2 happens. Thus, since H is a $(t_2, \epsilon_\mathsf{CR})$-collision resistant hash function,

$$|\Pr[E_5] - \Pr[E_4]| = \Pr[\mathsf{Bad}_2] \leq \epsilon_\mathsf{CR}.$$

Game 6. It is Game 5, where the following event Bad_3 does not happen: for any tweak h for f used only in decryption queries, (that is, used only in f^{-1}) there exist two decryption queries, which we call the ith and the jth s.t. $h' = \mathsf{H}(a^j, c^j_0\| \dots \|c^j_l)$ (where a^j and $c^j_0, \dots, c^j_l$ are modified according to the 0.0 line of Fa^j, s.t. the modified c^j_0 is equal to c^i_{l+2} obtained in the ith decryption query, in which $\mathsf{f}^{-1}(h^i, \tau^i)$ is computed (modified according to the 0.1 row of Fa^i) and the h^i modified is equal to h'. Let E_6 be the event that the adversary wins this game.
Transition between Game 5 and Game 6. Since Game 5 and Game 6 are the same except if event Bad_3 happens, it is enough to bound $\Pr[\mathsf{Bad}_3]$ to bound the difference $|\Pr[E_5] - \Pr[E_6]|$. To bound $\Pr[\mathsf{Bad}_3]$, we divide it in $q_D + 1$ different events:$\mathsf{Bad}_{3.1}, \dots, \mathsf{Bad}_{3.q_D+1}$, where $\mathsf{Bad}_{3.\iota}$ is event Bad_3 where $\iota = i$. Clearly, $\mathsf{Bad}_3 = \bigcup_{\iota=1}^{q_D+1} \mathsf{Bad}_{3.\iota}$. To bound $\Pr[\mathsf{Bad}_{3.\iota}]$ we build an adversary EE^ι against the n-PR-corpi-security of the hash function H.
The n-PR-corpi-adversary EE^ι. At the start of the game, EE^ι receives a key for the hash function s in $\mathcal{HK}$ and gives it to A. Moreover, she picks a random tweakable permutation f. Finally, she has two lists $\mathcal{H}$ and $\mathcal{CH}$ which are empty.

When A does an encryption query on input (a, m), EE^ι (we use EE^ι) to identify both EE^ι_1 and EE^ι_2) simply picks k_0 uniformly at random from $\{0,1\}^n$ and after having parsed m in $(m_1, \dots, m_l)$, computes $c_0, \dots, c_l$ as follows: $c_0 = \mathsf{E}_{k_0}(p_B)$, and then, for all

$i = 1, \ldots, l$, $k_i = \mathsf{E}_{k_{i-1}}(p_A)$, $y_i = \mathsf{E}_{k_i}(p_B)$ and $c_i = y_i \oplus m_i$ (for $i = l$ $c_l = \pi_{|m_l|}(y_l) \oplus m_l$. Then, EE^ι computes $h = \mathsf{H}'_s(a, c_0\| \ldots \|c_l)$, she adds $((a, c_0, \ldots, c_l), h)$ to $\mathcal{H}$, she computes $c_{l+1} = \mathsf{f}(h, k_0)$, and she adds (h, c_{l+1}, c_0) to $\mathcal{CH}$. So, she can answer A $c = (c_0\| \ldots \|c_{l+1})$ and the leakage k_0.(

For this, EE^ι needs time $t_\mathsf{H} + (2l+1)t_\mathsf{E} \leq t_\mathsf{H} + (2L+1)t_\mathsf{E}$.

When A does the ith decryption query, $i < \iota$ on input (a, c, Fa), EE_1^ι proceeds as follows: 1) she computes $h = \mathsf{H}'_s(c_0\| \ldots \|c_l, a)$ where $c_0, \ldots, c_l$ and a are modified according to the 0.0 row of Fa, and she adds $((c_0, \ldots, c_l, a), h)$ to $\mathcal{H}$ (with $c_0, \ldots, c_l$ and a modified according to the 0.0 row of Fa), 2) she computes $k_0 = \mathsf{f}^{-1}(h, c_{l+1})$ where the inputs are modified according to 0.1 row of Fa, obtaining k_0, as answer; she sets $c_{l+1}^i := c_{l+1}$, $h^i = h$ (where c_{l+1} and h are modified according to the 0.1 row of Fa), 3) she computes $\tilde{c_0} = f_{k_0}(p_A)$ with k_0 modified according the 0.2 line of Fa, and $c_{l+2} = \mathsf{E}_{k_0}(p_B)$, and she adds $(h^i, c_{l+1}^i, c_{l+2})$ to $\mathcal{CH}$, 4) she sees if $\tilde{c}_0 \neq c_0$, she answers $\perp$ and the leakage k_0 to A; otherwise 5) for $i = 1, \ldots, l$, a) computes $k_i = \mathsf{E}_{k_{i-1}}(p_A)$ with k_{i-1} modified according to the i.1 line of Fa, b) computes $y_i = \mathsf{E}_{k_i}(p_B)$ with k_i modified according to the i.2 line of Fa c) computes $m_i = y_i \oplus c_i$ (if $i = l$ $\pi_{|c_l|}(y_l) \oplus c_l$) with y_i and c_i modified according to the i.3 line of Fa, 6) finally, EE_1^ι answers $m = (m_1, \ldots, m_l)$ and the leakage k_0 to A.

For this, EE_1^ι needs time $t_\mathsf{H} + (2l+2)t_\mathsf{E} \leq t_\mathsf{H} + (2L+2)t_\mathsf{E}$.

When A does the ι decryption query on input (a, c, Fa), EE_1^ι proceeds as follows: 1) she computes $h = \mathsf{H}'_s(a, c_0\| \ldots \|c_l)$ where a and $c_0, \ldots, c_l$ are modified according to the 0.0 row of Fa, and she adds $((c_0, \ldots, c_l, a), h)$ to $\mathcal{H}$ (with $c_0, \ldots, c_l$ and a modified according to the 0.0 row of Fa), 2) she sets h^ι and c_{l+1}^ι, which are h and c_{l+1} modified according to the 0.1 row of Fa and she checks if there is an entry in $\mathcal{CH}$ (h^ι, c_{l+1}^ι). If this is the case EE_1^ι aborts. Otherwise, EE_1^ι outputs h^ι as her challenge and $\mathsf{st} = (h^\iota, c_{l+1}^\iota, \mathcal{CH}, \mathcal{H}, s, \mathsf{f})$. Then, $\mathsf{EE}_2^\iota(\mathsf{st})$, parses st in $h^\iota, c_{l+1}^\iota, \mathcal{CH}, \mathcal{H}, s, \mathsf{f})$, and she computes $k_0 = \mathsf{f}^{-1}(h^\iota, c_{l+1}^\iota)$. 3) after that, EE_2^ι computes $\tilde{c_0} = f_{k_0}(p_A)$ with k_0 modified according the 0.2 line of Fa, and $c_{l+2} = \mathsf{E}_{k_0}(p_B)$, and she adds $(h^i, c_{l+1}^i, c_{l+2})$ to $\mathcal{CH}$, 4) she sees if $\tilde{c}_0 \neq c_0$, she answers $\perp$ and the leakage k_0 to A; otherwise 5) for $i = 1, \ldots, l$, a) computes $k_i = \mathsf{E}_{k_{i-1}}(p_A)$ with k_{i-1} modified according to the i.1 line of Fa, b) computes $y_i = \mathsf{E}_{k_i}(p_B)$ with k_i modified according to the i.2 line of Fa c) computes $m_i = y_i \oplus c_i$ (if $i = l$ $\pi_{|c_l|}(y_l) \oplus c_l$) with y_i and c_i modified according to the i.3 line of Fa, 6) finally, EE_2^ι answers $m = (m_1, \ldots, m_l)$ and the leakage k_0 to A. (If $\iota = q_D + 1$ the decryption query associated to A output, EE proceeds as as normal decryption query, simply assuming that there is no fault injected).

For this, EE_1^ι needs time t_H, while EE_2^ι needs time $(2l+2)t_\mathsf{E} \leq (2L+2)t_\mathsf{E}$. To be polynomial, we assume that f is simulated via lazy sampling and EE_1 puts in st the samples she has already done.

When A does the ith decryption query, $i > \iota$ on input (a, c, Fa), EE_2^ι proceeds as follows: 1) she computes $h = \mathsf{H}'_s(c_0\| \ldots \|c_l, a)$ where $c_0, \ldots, c_l$ and a are modified according to the 0.0 row of Fa, and she adds $((c_0, \ldots, c_l, a), h)$ to $\mathcal{H}$ (with $c_0, \ldots, c_l$ and a modified according to the 0.0 row of Fa), 2) she computes $k_0 = \mathsf{f}^{-1}(h, c_{l+1})$ where the inputs are modified according to 0.1 row of Fa, obtaining k_0, as answer; she sets $c_{l+1}^i := c_{l+1}$, $h^i = h$ (where c_{l+1} and h are modified according to the 0.1 row of Fa), 3) she computes $\tilde{c_0} = f_{k_0}(p_A)$ with k_0 modified according the 0.2 line of Fa, and $c_{l+2} = \mathsf{E}_{k_0}(p_B)$, 4) she sees if $\tilde{c}_0 \neq c_0$, she answers $\perp$ and the leakage k_0 to A; otherwise 5) for $i = 1, \ldots, l$, a) computes $k_i = \mathsf{E}_{k_{i-1}}(p_A)$ with k_{i-1} modified according to the i.1 line of Fa, b) computes $y_i = \mathsf{E}_{k_i}(p_B)$ with k_i modified according to the i.2 line of Fa c) computes $m_i = y_i \oplus c_i$ (if $i = l$ $\pi_{|c_l|}(y_l) \oplus c_l$) with y_i and c_i modified according to the i.3 line of Fa, 6) finally, EE_2^ι answers $m = (m_1, \ldots, m_l)$ and the leakage k_0 to A.

For this, EE_2^ι needs time $t_\mathsf{H} + (2l+2)t_\mathsf{E} \leq t_\mathsf{H} + (2L+2)t_\mathsf{E}$.

When A outputs her output (a^*, c^*), EE_2^ι proceeds as follows: 1) she computes $h^* = \mathsf{H}'_s(c_0^*\| \ldots \|c_l^*, a^*)$ and she adds $((c_0^*\| \ldots \|c_l^*, a^*), h)$ to $\mathcal{H}$, 2) she computes $k_0^* = \mathsf{f}^{-1}(h, c_{l+1})$ 3) she computes $\tilde{c_0} = f_{k_0}(p_A)$ with k_0 modified according the 0.2 line of Fa, and $c_{l+2} = \mathsf{E}_{k_0}(p_B)$, 4) she sees if $\tilde{c^*}_0 \neq c_0^*$, she answers $\perp$; otherwise 5) for $i = 1, \ldots, l$, a) computes $k_i^* = \mathsf{E}_{k_{i-1}^*}(p_A)$, b) computes $y_i^* = \mathsf{E}_{k_i^*}(p_B)$, c) computes $m_i^* = y_i^* \oplus c_i^*$ (if $i = l$ $\pi_{|c_l^*|}(y_l^*) \oplus c_l^*$,

6) finally, EE_2^ι observe that A has given a correct decryption query, and she looks through $\mathcal{H}$ to see if there is an entry $(c_0\|\dots\|c_l, a), h)$ s.t $h = h^\iota$ and $c_0 = c^\iota_{l+2}$. If it is the case, she outputs it, otherwise 0^n.

Thus, in total EE runs in time bounded by $t + q(t_\mathsf{H}) + (q_E + q_D + 1)(2L + 2) + 1)t_\mathsf{E} = t'_3$. EE correctly simulates Game 5 for A.

Bounding $\Pr[\mathsf{Bad}_{3.\iota}|\neg\mathsf{Bad}_{3.1},\dots,\mathsf{Bad}_{3.\iota-1}]$**.** First, we observe that if EE_1^ι aborts, it means that there is a previous query for which $h^i = h^\iota$ and $c^i_{l+2} = c^\iota_{l+2}$, thus event $\mathsf{Bad}_{3.\iota}$ is event $\mathsf{Bad}_{3.i}$, thus,

$$\Pr[\mathsf{Bad}_{3.\iota}|\neg\mathsf{Bad}_{3.1},\dots,\mathsf{Bad}_{3.\iota-1}] = 0.$$

Second, we observe that since k_0^ι is picked randomly (and (k_0^ι, p_B) has never been queried before the ι decryption query to E), to pick the random prefix uniformly at random or to pick a k_0 uniformly at random and as a random prefix $\mathsf{E}(k_0, p_B)$ is indistinguishable for the n-$\mathsf{PR\text{-}corpi}$ game. We are doing the latter case for EE^ι. Thus, if event $\mathsf{Bad}_{3.\iota}\neg\mathsf{Bad}_{3.1},\dots,\mathsf{Bad}_{3.\iota-1}$ happens, then, EE wins the n-$\mathsf{PR\text{-}corpi}$ game. Since EE is a t'_3-adversary and H is $(t_3, \epsilon_{\mathsf{PR\text{-}corpi}})$-$n$-$\mathsf{PR\text{-}corpi}$-secure (because we need to subtract the time needed to compute the random pre-fix), then,

$$\Pr[\mathsf{Bad}_{3.\iota}|\neg\mathsf{Bad}_{3.1},\dots,\mathsf{Bad}_{3.\iota-1}] \le \epsilon_{\mathsf{PR\text{-}corpi}}.$$

Concluding the proof. First, we observe that

$$\Pr[\mathsf{Bad}_3] \le \sum_{\iota=1}^{q_D+1} \Pr[\mathsf{Bad}_{3.\iota}|\neg\mathsf{Bad}_{3.1},\dots,\mathsf{Bad}_{3.\iota-1}] \le \epsilon_{\mathsf{PR\text{-}corpi}} \le (q_D+1)\epsilon_{\mathsf{PR\text{-}corpi}}.$$

Second, we observe that $\Pr[E_6] = 0$, since the adversary in the final verification query cannot have h^* equal to an h of an encryption query, and if $h^* = h$ of a decryption query, it cannot be the case that $\tilde{c^*}_0 = c_0^*$ (EE^ι covers also the case of k_0^ι random and c^ι_{l+2} (which is c_0^ι correctly computed) equal to c_0 which is computed to obtain h^ι). Thus,

$$\Pr[E_0] \le \sum_{i=1}^{5} |\Pr[E_{i+1}] - \Pr[E_i]| \le$$

$$\epsilon_{\mathsf{sTPRP}} + \frac{(q_D+2)(q_D+1)}{2^{n+1}} + q_D[q_E(L+1)(q_D+1)/2 + q_I + (q_D+1)/2]2^{-n} +$$

$$\epsilon_{\mathsf{CR}} + (q_D+1)\epsilon_{\mathsf{PR\text{-}corpi}} = \epsilon.$$

□

I.2 CONCRETE vs. any fault except in F in decryption

Formalizing faults inside the components. Before to state the theorem, we need to formalize faults into the component. We will do this with the *atom faults vector* $\mathsf{Fa}^{\mathsf{atom}}$. First, we enumerate all the calls to the atoms (and we consider only those for which a fault is interesting, i.e., not the XOR). Let n_{atom} be the number of such calls. $\mathsf{Fa}^{\mathsf{atom}}$ is a vector of n_{atom} faults: $(\mathsf{Fa}_1,\dots,\mathsf{Fa}_{n_{\mathsf{atom}}})$ where Fa_i is the description of the faults to be inserted in the call of the atom corresponding to index i. For example, if the ith call correspond to a block-cipher Fa_i explain how, when and which wires of E are modified during that computation. If no fault is injected, we denote this with $\mathsf{Fa}_i = \epsilon$. For the decryption of $\mathsf{CONCRETE}$, the atoms whose fault are described by $\mathsf{Fa}^{\mathsf{atom}}$ are H, F, and E, where the index are: 1 for $\mathsf{H}'_s(c_0\|\dots\|c_l, a)$, 2 for $\mathsf{F}_k^{-1,h}(c_{l+1})$, 3 for $\mathsf{E}_{k_0}(p_B)$, $2+2i$ $(i = 1,\dots,l)$ for $\mathsf{E}_{k_{i-1}}(p_A)$, $2i+3$ $(i = 1,\dots,l)$ for $\mathsf{E}_{k_{i-1}}(p_B)$. Similarly to what is done to define $\mathcal{I}$ (Sec. 2.6.1), we define $\mathcal{I}^{\mathsf{atom}}$ with the set of protected calls and we denote with $\perp$ in $\mathsf{Fa}^{\mathsf{atom}}$ that this call cannot be faulted. Let $\mathcal{I}^c = (\mathcal{I}, \mathcal{I}^{\mathsf{atom}})$ be the set of protected inputs and atoms. In the next theorem, $\mathcal{I}^c_{\mathsf{AEnc}} = (\mathcal{I}_{\mathsf{AEnc}}, \mathcal{I}^{\mathsf{atom}}_{\mathsf{AEnc}})$ where all the entries of $\mathcal{I}_{\mathsf{AEnc}}$ are ϵ, while all the entries of $\mathcal{I}^{\mathsf{atom}}_{\mathsf{AEnc}}$ are $\perp$ (i.e., no fault is allowed); instead for decryption, we use $\mathcal{I}^c_{\mathsf{ADec}} = (\mathcal{I}_{\mathsf{ADec}}, \mathcal{I}^{\mathsf{atom}}_{\mathsf{ADec}})$ where all the entries of $\mathcal{I}_{\mathsf{ADec}}$ are free (except those forced to be equal to ϵ by the matrix defined in Eq. 4, while all the adversary can choose all the entries of $\mathcal{I}^{\mathsf{atom}}_{\mathsf{ADec}}$ except for the second one (corresponding to $k_0 = \mathsf{F}^{-1,h}_k(c_{l+1})$ which is $\perp$ (i.e., no fault is allowed). Note that we are using the same $\mathcal{I}_{\mathsf{AEnc}}$, and $\mathcal{I}_{\mathsf{ADec}}$ as those used for Thm. 1. Now, we can formally state this result:

Theorem 11. *Let* F *be a strongly protected* $(q, t_1, \epsilon_{\mathsf{sTPRP}})$-$\mathsf{sTPRP}$*, let* E *be an ideal block-cipher, let* A *be allowed* q_I *queries to* E*, let* H *be a* $(t_2, \epsilon_{\mathsf{CR}})$*-collision resistant and* $(t_3, \epsilon_{\mathsf{PR\text{-}corpi}})$-$n$-$\mathsf{PR\text{-}corpi}$ *hash function. Then,* $(\mathcal{I}^c_{\mathsf{AEnc}}, \mathcal{I}^c_{\mathsf{ADec}})$*-protected implementation of the* AE*-scheme* $\mathsf{CONCRETE}\Pi = (\mathsf{Gen}, \mathsf{AEnc}, \mathsf{ADec})$*, encoding messages at most* Ln *bits long, with leaking function pair* $\mathsf{L} = (\mathsf{L}_{\mathsf{AEnc}}, \mathsf{L}_{\mathsf{ADec}})$ *and fault admissible sets* $\mathcal{F} = (\mathcal{F}_E, \mathcal{F}_D)$ *has* $(q_I, q_{FL}, q_E, q_D, t, \epsilon)$-ciphertext integrity in the presence of leakage and faults against $\mathcal{F}$-fault-then-leakage attacks in encryption and decryption ($\mathsf{CIL2Fd}$)*, with*

$$\epsilon = \epsilon_{\mathsf{sTPRP}} + \frac{(q_D+2)(q_D+1)}{2^{n+1}} + (q_D+1)[q_E(L+1)+q_I+(j-1)(L+2)]2^{-n} + \epsilon_{\mathsf{CR}} + (q_D+1)\epsilon_{\mathsf{PR\text{-}corpi}}$$

with q_I *the number of queries the adversary* A *is allowed to do to the ideal block-cipher* E*, with* $q = q_E + q_D + 1$*,* $q_I = t_1 = t + qt_{\mathsf{H}} + (q(2L+1)+1)t_{\mathsf{E}}$*,* $t_2 = t + q(t_{\mathsf{H}}) + (q_E + q_D)[(2L+2)+1)t_{\mathsf{E}} + t_{\mathsf{F}}]$ $t_3 = t + q(t_{\mathsf{H}}) + (q_E + q_D + 1)[(2L+2)+1)t_{\mathsf{E}} + t_{\mathsf{F}}] - t_{\mathsf{F}} - t_{\mathsf{E}}$*. (we assume that picking a value uniformly at random does not take time).*

The proof is the same as for Thm. 1 and can be found in the full version of the paper. Substantially, we have only to modify the description of all adversaries to apply the faults described in $\mathcal{I}^{\mathsf{atom}}_{\mathsf{ADec}}$.

I.3 wCILF2-security of CONCRETE with only differential faults in encryption

Leakage function and faulty matrix. For the leakage function, we use the same leakage function as for Thm. 1: $\mathsf{L}_{\mathsf{AEnc}}(a, m, k_0; k) := \emptyset$, and $\mathsf{L}_{\mathsf{ADec}}(a, c; k) := k_0$. Regarding faults, the adversary is allowed to set any fault she wants between the atoms in decryption, without any restriction and only differential faults in encryption, that is all entries of $\mathcal{I}_{\mathsf{AEnc}}$ are either ϵ (when the corresponding value in the dependency matrix is ϵ) or $\perp/\oplus$; while $\mathcal{I}_{\mathsf{ADec}}$ is empty, that is, we allow the adversary to choose any entry different from ϵ of the matrix defined in Eq. 4 as long as they are compliant with Sec. 2.6.1. $\mathcal{F}_E$ and $\mathcal{F}_D$ are defined accordingly.

Theorem 12. *Let* F *be a strongly protected* $(q, t_1, \epsilon_{\mathsf{sTPRP}})$-$\mathsf{sTPRP}$*, let* E *be an ideal block-cipher, let* A *be allowed* q_I *queries to* E*, let* H *be a* $(t_2, \epsilon_{\mathsf{CR}})$*-collision resistant and* $(t_3, \epsilon_{\mathsf{PR\text{-}corpi}})$-$n$-$\mathsf{PR\text{-}corpi}$ *hash function. Then,* $(\mathcal{I}_{\mathsf{AEnc}}, \mathcal{I}_{\mathsf{ADec}})$*-protected implementation of the* AE*-scheme* $\mathsf{CONCRETE}\Pi = (\mathsf{Gen}, \mathsf{AEnc}, \mathsf{ADec})$*, encoding messages at most* Ln *bits long, with leaking function pair* $\mathsf{L} = (\mathsf{L}_{\mathsf{AEnc}}, \mathsf{L}_{\mathsf{ADec}})$ *and fault admissible sets* $\mathcal{F} = (\mathcal{F}_E, \mathcal{F}_D)$ *has* $(q_I, q_{FL}, q_E, q_D, t, \epsilon)$ weakly ciphertext integrity in the presence of leakage and faults against $\mathcal{F}$-fault-then-leakage attacks in encryption and decryption ($\mathsf{wCILF2} - (\mathsf{de})$) *with*

$$\epsilon_{\mathsf{sTPRP}} + \frac{(q_D+2)(q_D+1)}{2^{n+1}} + \frac{(q_D+1)[q_E(L+1)+q_I+(j-1)(L+2)]}{2^n} + \\ \epsilon_{\mathsf{CR}} + (q_D+1)\epsilon_{\mathsf{PR\text{-}corpi}}$$

with q_I *the number of queries the adversary* A *is allowed to do to the ideal block-cipher* E*, with* $q = q_E + q_D + 1$*,* $q_I = t_1 = t + qt_{\mathsf{H}} + (q(2L+1)+1)t_{\mathsf{E}}$*,* $t_2 = t + q(t_{\mathsf{H}}) + (q_E + q_D)[(2L+2)+1)t_{\mathsf{E}} + t_{\mathsf{F}}]$ $t_3 = t + q(t_{\mathsf{H}}) + (q_E + q_D + 1)[(2L+2)+1)t_{\mathsf{E}} + t_{\mathsf{F}}] - t_{\mathsf{F}} - t_{\mathsf{E}}$*. (we assume that picking a value uniformly at random does not take time).*

Proof. We use a sequence of games for the proofs. For simplicity, we consider the decryption of the challenge associated data and ciphertext as the $q_D + 1$th decryption query.
Game 0. It is the standard $\mathsf{wCILF2}_{\mathsf{CONCRETE},\mathcal{F},\mathsf{L},\mathsf{A}^{\mathsf{L}}}$ game. Let E_0 be the event that A wins this game.
Game 1. It is Game 0, where we simply put (a', c', a, m, r) in $\mathcal{S}$ where a' is a modified according to the $l+1.1$ row of Fa and $c' = (c'_0, \dots, c'_l, c_{l+1})$ with c'_i $(i = 0, \dots, l)$ is c_i

modified according to the $l+1.1$ row of Fa. (That is, we take the actual input of H). Let E_1 be the event that A wins this game. So $\Pr[E_0] \leq \Pr[E_1]$.
Game 2. It is Game 1 except that we have replaced F with a random tweakable permutation f. Let E_2 be the event that A wins this game.
Transition between Game 1 and Game 2. Since Game 1 and Game 2 are the same except for the replacement of F with f, we can build an adversary B to bound the difference. She plays the sTPRP game, having to distinguish F_k from f, and simulates A's oracles using its own.

At the start of the game, B picks s in $\mathcal{HK}$ and gives it to A. Moreover, he has a list $\mathcal{S}$ which is empty.

When A does an encryption query on input (a,m), B simply picks k_0 uniformly at random from $\{0,1\}^n$. Moreover, she has $L+1$ lists $\mathcal{S}_0, \mathcal{S}_1, \dots, \mathcal{S}_L$ which are empty. After having parsed m in $(m_1, \dots, m_l)$, she proceeds as follows: 1) she computes $c_0 = \mathsf{E}_{k_0}(p_B)$, where k_0 is modified according to the 0.2 row of Fa, and she adds c_0 to $\mathcal{S}_0$; 2) for $i = 1, \dots, l$, a) computes $k_i = \mathsf{E}_{k_{i-1}}(p_A)$ with k_{i-1} modified according to the $i.1$ line of Fa, b) computes $y_i = \mathsf{E}_{k_i}(p_B)$ with k_i modified according to the $i.2$ line of Fa c) computes $c_i = y_i \oplus c_i$ (if $i = l$ $\pi_{|c_l|}(y_l) \oplus c_l$) with y_i and c_i modified according to the $i.3$ line of Fa, and adds c_i to $\mathcal{S}_i$; 3) she computes $h = \mathsf{H}'_s(c_0\| \dots \|c_l, a)$ where $c_0, \dots, c_l$, and a are modified according to the $l+1.1$ row of Fa; moreover, she adds c'_i to $\mathcal{C}_i$, for $i = 0, \dots, l$, where c'_i is c_i modified according to the element of the $l+1.1$th row corresponding to c_i. 4) she queries her oracle on input (h, k_0) which are modified according to $l+1.2$ row of Fa, obtaining c_{l+1} as answer; 5) finally, B answers $c = (c_0, \dots, c_l)$ and the leakage k_0 to A. Moreover, if there is no fault inserted and in the 0.2 row and in the $l+1.2$ row of fault (or the same fault for k_0 in both rows), she adds $c' = (c'_0, \dots, c'_l, c_{l+1}, a')$ to $\mathcal{S}$ (where a' is a modified according to the $l+1.1$th row of Fa), otherwise nothing. Moreover, B adds (a', c') to the list $\mathcal{S}$, where a' is a modified according to the $l+1.1$ row of Fa and $c' = (c'_0, \dots, c'_l, c_{l+1})$ with c'_i $(i = 0, \dots, l)$ is c_i modified according to the $l+1.1$ row of Fa.

For this, B does one oracle query and needs time $t_\mathsf{H} + (2l+1)t_\mathsf{E} \leq t_\mathsf{H} + (2L+1)t_\mathsf{E}$.

When A does a decryption query on input (a, c, Fa), B proceeds as follows: 1) she computes $h = \mathsf{H}'_s(a, c_0\| \dots \|c_l)$ where a and $c_0, \dots, c_l$ are modified according to the 0.0 row of Fa, 2) she queries her inverse oracle on input (h, c_{l+1}) which are modified according to 0.1 row of Fa, obtaining k_0 as answer, 3) she computes $\tilde{c_0} = f_{k_0}(p_A)$ with k_0 modified according the 0.2 line of Fa, if $\tilde{c}_0 \neq c_0$, she answers $\perp$ and the leakage k_0 to A; otherwise 4) for $i = 1, \dots, l$, a) computes $k_i = \mathsf{E}_{k_{i-1}}(p_A)$ with k_{i-1} modified according to the $i.1$ line of Fa, b) computes $y_i = \mathsf{E}_{k_i}(p_B)$ with k_i modified according to the $i.2$ line of Fa c) computes $m_i = y_i \oplus c_i$ (if $i = l$ $\pi_{|c_l|}(y_l) \oplus c_l$) with y_i and c_i modified according to the $i.3$ line of Fa, 5) finally, B answers $m = (m_1, \dots, m_l)$ and the leakage k_0 to A.

For this, B does one oracle query and time $t_\mathsf{H} + (2l+1)t_\mathsf{E} \leq t_\mathsf{H} + (2L+1)t_\mathsf{E}$.

When A outputs her output (a^*, c^*), B proceeds as follows: 1) she computes $h^* = \mathsf{H}'_s(a^*, c^*_0\| \dots \|c^*_l)$, 2) she queries her inverse oracle on input (h^*, c^*_{l+1}), obtaining k^*_0 as answer, 3) she computes $\tilde{c^*_0} = f_{k^*_0}(p_A)$, if $\tilde{c*}_0 = c*_0$ and $(a^*, c^*) \notin \mathcal{S}$, she outputs 1; otherwise 0.

For this, B does one oracle query and time $t_\mathsf{H} + t_\mathsf{E}$.

Thus, in total B does at most $q_E + q_D + 1 = q$ queries to her oracle and runs in time bounded by $t + q(t_\mathsf{H}) + (q(2L+1)+1)t_\mathsf{E} = t_1$.

If her oracle is implemented with F, B correctly simulates Game 1 for A, otherwise Game 2. Since F is a $(q, t_1, \epsilon_\mathsf{sTPRP})$-$\mathsf{sTPRP}$,

$$|\Pr[E_2] - \Pr[E_1]| \leq \epsilon_\mathsf{sTPRP}.$$

Game 3. It is Game 3, where in decryption instead of using f^{-1}, where f is a random tweakable permutation, we pick its answers uniformly at random (and we keep a list of all queries already done to make the answer coherent). Let E_3 be the event that A wins this game.
Transition between Game 2 and Game 3. In Game 3, for decryption queries, we are using a sTPRP, while in Game 3 a PRF. Due to the well-known birthday bound, we have

that

$$|\Pr[E_3] - \Pr[E_2]| \le \frac{(q_D+2)(q_D+1)}{2^{n+1}}.$$

Game 4. It is Game 3, where during decryption queries (not for the challenge one), immediately after k_0 is computed (via f^{-1}) the decryption oracle computes $c_{l+2} := \mathsf{E}_{k_0}(p_B)$, without any faults inserted, and appends this to its normal answer. Let E_4 be the event that the adversary wins this game.

Transition between Game 3 and Game 4. We build an adversary C. C behaves as A for encryption queries. For decryption queries, C proceeds as follows: when A does a decryption query on input (a, c), C simply forwards this query to her oracle. After having received the answer and the leakage k_0, C simply takes k_0, calls the ideal cipher E on input (k_0, p_B), obtains c_{l+2}, and answer A with the oracle's answer appended with c_{l+2}. Thus, C needs time $t + q_D t_{\mathsf{E}}$ and does $q_I + q_D$ queries to the ideal cipher E. Thus, since

$$\Pr[E_3] = \Pr[E_4].$$

Game 5. It is Game 4, with the condition that the following event does not happen: during a decryption query, when a new key k_0 is picked, there exists no query to the ideal cipher on input (k_0, p_B). Let E_5 be the event that the adversary wins this game.

Transition between Game 4 and Game 5. The two games are identical except if the following Bad_1 event happens: there exists a decryption query s.t. it requires to pick a new key k_0^i and there exists a previous query to the ideal cipher E on input (k_0^i, p_B). We observe that during an encryption query there at most $L+1$ queries to (k_i, p_B) with at most $L+1$ different keys, while during a decryption query there at most $L+2$ different query to the ideal block-cipher with input (k_i, p_B). The adversary can do at most q_I queries of its choice to the ideal block-cipher. Thus, during the jth decryption query there at most $q_E(L+1) + q_I + (j-1)(L+2)$ different keys that, if picked, would trigger the Bad_1 event. Thus,

$$\Pr[\mathsf{Bad}_1] \le \sum_{i=1}^{q_D} \frac{q_E(L+1) + q_I + (j-1)(L+2)}{2^n} \le \frac{(q_D+1)[q_E(L+1) + q_I + (j-1)(L+2)]}{2^n}.$$

Thus,

$$|\Pr[E_5] - \Pr[E_4]| \le (q_D+1)[q_E(L+1) + q_I + (j-1)(L+2)]2^{-n}.$$

Game 6. It is Game 6, where the following event Bad_2 does not happen: the hash queries induced during encryption and decryption queries induce a collision for the hash function. Let E_5 be the event that the adversary wins this game.

Transition between Game 5 and Game 6. Since Game 5 and Game 6 are the same except if event Bad_2 happens, it is enough to bound $\Pr[\mathsf{Bad}_2]$ to bound the difference $|\Pr[E_5] - \Pr[E_6]|$. To bound $\Pr[\mathsf{Bad}_2]$, we build an adversary D who wants to find a collision for the hash function. At the start of the game, D receives a key for the hash function s in $\mathcal{HK}$ and gives it to A. Moreover, she picks a random tweakable permutation f. Finally, he has a list $\mathcal{H}$ which is empty.

When A does an encryption query on input (a, m), D simply picks k_0 uniformly at random from $\{0,1\}^n$. Moreover, she has $L+1$ lists $\mathcal{S}_0, \mathcal{S}_1, \dots, \mathcal{S}_L$ which are empty. After having parsed m in $(m_1, \dots, m_l)$, she proceeds as follows: 1) she computes $c_0 = \mathsf{E}_{k_0}(p_B)$, where k_0 is modified according to the 0.2 row of Fa, and she adds c_0 to $\mathcal{S}_0$; 2) for $i = 1, \dots, l$, a) computes $k_i = \mathsf{E}_{k_{i-1}}(p_A)$ with k_{i-1} modified according to the i.1 line of Fa, b) computes $y_i = \mathsf{E}_{k_i}(p_B)$ with k_i modified according to the i.2 line of Fa c) computes $c_i = y_i \oplus c_i$ (if $i = l$ $\pi_{|c_l|}(y_l) \oplus c_l$) with y_i and c_i modified according to the i.3 line of Fa, and adds c_i to $\mathcal{S}_i$; 3) she computes $h = \mathsf{H}'_s(c_0\| \dots \|c_l, a)$ where $c_0, \dots, c_l$, and a are modified according to the $l+1.1$ row of Fa; moreover, she adds c'_i to $\mathcal{C}_i$, for $i = 0, \dots, l$, where c'_i is c_i modified according to the element of the $l+1.1$th row corresponding to c_i, and she adds $((c'_0\| \dots \|c'_l, a), h)$ to $\mathcal{H}$. 4) she queries her oracle on input (h, k_0) which are modified according to $l+1.2$ row of Fa, obtaining c_{l+1} as answer; 5) finally, D answers $c = (c_0, \dots, c_l)$ and the leakage k_0 to A. Moreover, if there is no fault inserted and in the 0.2 row and in the $l+1.2$ row of fault (or the same fault for k_0 in both rows), she adds $c' = (c'_0, \ldots, c'_l, c_{l+1}, a')$ to $\mathcal{S}$ (where a' is a modified according to the $l+1.1$th row of Fa), otherwise nothing. Moreover, D adds (a', c') to the list $\mathcal{S}$, where a' is a modified according to the $l+1.1$ row of Fa and $c' = (c'_0, \ldots, c'_l, c_{l+1})$ with c'_i $(i = 0, \ldots, l)$ is c_i modified according to the $l+1.1$ row of Fa.

For this, D needs time $t_{\mathsf{H}} + (2l+1)t_{\mathsf{E}} + t_{\mathsf{F}} \leq t_{\mathsf{F}} + t_{\mathsf{H}} + (2L+1)t_{\mathsf{E}}$.

When A does a decryption query on input (a, c, Fa), D proceeds as follows: 1) she computes $h = \mathsf{H}'_s(a, c_0 \| \ldots \| c_l)$ where a and $c_0, \ldots, c_l$ are modified according to the 0.0 row of Fa, and she adds $((a, c_0 \| \ldots \| c_l), h)$ to $\mathcal{H}$, 2) she computes $k_0 = \mathsf{f}^{-1}(h, c_{l+1})$ where the inputs are modified according to 0.1 row of Fa, obtaining k_0 as answer, 3) she computes $\tilde{c_0} = f_{k_0}(p_A)$ with k_0 modified according the 0.2 line of Fa and $c_{l+2} = \mathsf{E}_{k_0}(p_B)$, if $\tilde{c}_0 \neq c_0$, she answers $\perp$ and the leakage k_0 to A; otherwise 4) for $i = 1, \ldots, l$, a) computes $k_i = \mathsf{E}_{k_{i-1}}(p_A)$ with k_{i-1} modified according to the $i.1$ line of Fa, b) computes $y_i = \mathsf{E}_{k_i}(p_B)$ with k_i modified according to the $i.2$ line of Fa c) computes $m_i = y_i \oplus c_i$ (if $i = l$ $\pi_{|c_l|}(y_l) \oplus c_l$) with y_i and c_i modified according to the $i.3$ line of Fa, 5) finally, D answers $m = (m_1, \ldots, m_l)$ and the leakage k_0 to A.

For this, D needs time $t_{\mathsf{F}} + t_{\mathsf{H}} + (2l+2)t_{\mathsf{E}} \leq t_{\mathsf{F}} + t_{\mathsf{H}} + (2L+2)t_{\mathsf{E}}$.

When A outputs her output (a^*, c^*), D proceeds as follows: 1) she computes $h^* = \mathsf{H}'_s(a^*, c^*_0 \| \ldots \| c^*_l)$ and she adds $((a^*, c^*_0 \| \ldots \| c^*_l), h)$ to $\mathcal{H}$, 2) she looks through $\mathcal{H}$ to see if there is a collision. If it is the case, she outputs it, otherwise $0^n, 1^n$. For this, D needs time t_{H}.

Thus, in total, D runs in time bounded by $t + q(t_{\mathsf{H}}) + (q_E + q_D)[(2L+2)+1)t_{\mathsf{E}} + t_{\mathsf{F}}] = t_2$.

D correctly simulates Game 6 for A. D finds a collision for the hash function H if event Bad_2 happens. Thus, since H is a $(t_2, \epsilon_{\mathsf{CR}})$-collision resistant hash function,

$$|\Pr[E_6] - \Pr[E_5]| = \Pr[\mathsf{Bad}_2] \leq \epsilon_{\mathsf{CR}}.$$

Game 7. It is Game 6, where the following event Bad_3 does not happen: for any tweak h for f used only in decryption queries, (that is, used only in f^{-1}) there exist two decryption queries, which we call the ith and the jth s.t. $h' = \mathsf{H}(a^j, c^j_0 \| \ldots \| c^j_l)$ (where a^j and $c^j_0, \ldots, c^j_l$ are modified according to the 0.0 line of Fa^j), s.t. the modified c^j_0 is equal to c^i_{l+2} obtained in the ith decryption query, in which $\mathsf{f}^{-1}(h^i, \tau^i)$ is computed (modified according to the 0.1 row of Fa^i) and the h^i modified is equal to h'. Let E_7 be the event that the adversary wins this game.

Transition between Game 6 and Game 7. Since Game 6 and Game 7 are the same except if event Bad_3 happens, it is enough to bound $\Pr[\mathsf{Bad}_3]$ to bound the difference $|\Pr[E_6] - \Pr[E_7]|$. To bound $\Pr[\mathsf{Bad}_3]$, we divide it in $q_D + 1$ different events:$\mathsf{Bad}_{3.1}, \ldots, \mathsf{Bad}_{3.q_D+1}$, where $\mathsf{Bad}_{3.\iota}$ is event Bad_3 where $\iota = i$. Clearly, $\mathsf{Bad}_3 = \bigcup_{\iota=1}^{q_D+1} \mathsf{Bad}_{3.\iota}$. To bound $\Pr[\mathsf{Bad}_{3.\iota}]$ we build an adversary EE^ι against the n-PR-corpi-security of the hash function H.

The n-PR-corpi-adversary EE^ι. At the start of the game, EE^ι receives a key for the hash function s in $\mathcal{HK}$ and gives it to A. Moreover, she picks a random tweakable permutation f. Finally, she has two lists $\mathcal{H}$ and $\mathcal{CH}$ which are empty.

When A does an encryption query on input (a, m), EE^ι (we use EE^ι) to identify both EE^ι_1 and EE^ι_2) simply picks k_0 uniformly at random from $\{0,1\}^n$. Moreover, she has $L+1$ lists $\mathcal{S}_0, \mathcal{S}_1, \ldots, \mathcal{S}_L$ which are empty. After having parsed m in $(m_1, \ldots, m_l)$, she proceeds as follows: 1) she computes $c_0 = \mathsf{E}_{k_0}(p_B)$, where k_0 is modified according to the 0.2 row of Fa, and she adds c_0 to $\mathcal{S}_0$; 2) for $i = 1, \ldots, l$, a) computes $k_i = \mathsf{E}_{k_{i-1}}(p_A)$ with k_{i-1} modified according to the $i.1$ line of Fa, b) computes $y_i = \mathsf{E}_{k_i}(p_B)$ with k_i modified according to the $i.2$ line of Fa c) computes $c_i = y_i \oplus c_i$ (if $i = l$ $\pi_{|c_l|}(y_l) \oplus c_l$) with y_i and c_i modified according to the $i.3$ line of Fa, and adds c_i to $\mathcal{S}_i$; 3) she computes $h = \mathsf{H}'_s(c_0 \| \ldots \| c_l, a)$ where $c_0, \ldots, c_l$, and a are modified according to the $l+1.1$ row of Fa; moreover, she adds c'_i to $\mathcal{C}_i$, for $i = 0, \ldots, l$, where c'_i is c_i modified according to the element of the $l+1.1$th row corresponding to c_i, and she adds $((c'_0 \| \ldots \| c'_l, a), h)$ to $\mathcal{H}$. 4) she queries her oracle on input (h, k_0) which are modified according to $l+1.2$ row of Fa, obtaining c_{l+1} as answer; 5) finally, EE^ι answers $c = (c_0, \ldots, c_l)$ and the leakage k_0 to A. Moreover, if there is no fault inserted and in the 0.2 row and in the $l+1.2$ row of fault (or the same fault for k_0 in both rows), she adds $c' = (c'_0, \dots, c'_l, c_{l+1}, a')$ to $\mathcal{S}$ (where a' is a modified according to the $l + 1.1$th row of Fa), otherwise nothing. Moreover, EE^ι adds (a', c') to the list $\mathcal{S}$, where a' is a modified according to the $l + 1.1$ row of Fa and $c' = (c'_0, \dots, c'_l, c_{l+1})$ with c'_i $(i = 0, \dots, l)$ is c_i modified according to the $l + 1.1$ row of Fa.

For this, EE^ι needs time $t_\mathsf{H} + (2l + 1)t_\mathsf{E} + t_\mathsf{F} \leq t_\mathsf{F} + t_\mathsf{H} + (2L + 1)t_\mathsf{E}$.

When A does the ith decryption query, $i < \iota$ on input (a, c, Fa), EE^ι_1 proceeds as follows: 1) she computes $h = \mathsf{H}'_s(c_0\| \dots \|c_l, a)$ where $c_0, \dots, c_l$ and a are modified according to the 0.0 row of Fa, and she adds $((c_0, \dots, c_l, a), h)$ to $\mathcal{H}$ (with $c_0, \dots, c_l$ and a modified according to the 0.0 row of Fa), 2) she computes $k_0 = \mathsf{f}^{-1}(h, c_{l+1})$ where the inputs are modified according to 0.1 row of Fa, obtaining k_0, as answer; she sets $c^i_{l+1} := c_{l+1}$, $h^i = h$ (where c_{l+1} and h are modified according to the 0.1 row of Fa), 3) she computes $\tilde{c_0} = f_{k_0}(p_A)$ with k_0 modified according the 0.2 line of Fa, and $c_{l+2} = \mathsf{E}_{k_0}(p_B)$, and she adds $(h^i, c^i_{l+1}, c_{l+2})$ to $\mathcal{CH}$, 4) she sees if $\tilde{c}_0 \neq c_0$, she answers $\perp$ and the leakage k_0 to A; otherwise 5) for $i = 1, \dots, l$, a) computes $k_i = \mathsf{E}_{k_{i-1}}(p_A)$ with k_{i-1} modified according to the $i.1$ line of Fa, b) computes $y_i = \mathsf{E}_{k_i}(p_B)$ with k_i modified according to the $i.2$ line of Fa c) computes $m_i = y_i \oplus c_i$ (if $i = l$ $\pi_{|c_l|}(y_l) \oplus c_l$) with y_i and c_i modified according to the $i.3$ line of Fa, 6) finally, EE^ι_1 answers $m = (m_1, \dots, m_l)$ and the leakage k_0 to A.

For this, EE^ι_1 needs time $t_\mathsf{F} + t_\mathsf{H} + (2l + 2)t_\mathsf{E} \leq t_\mathsf{F} + t_\mathsf{H} + (2L + 2)t_\mathsf{E}$.

When A does the ι decryption query on input (a, c, Fa), EE^ι_1 proceeds as follows: 1) she computes $h = \mathsf{H}'_s(a, c_0\| \dots \|c_l)$ where a and $c_0, \dots, c_l$ are modified according to the 0.0 row of Fa, and she adds $((c_0, \dots, c_l, a), h)$ to $\mathcal{H}$ (with $c_0, \dots, c_l$ and a modified according to the 0.0 row of Fa), 2) she sets h^ι and c^ι_{l+1}, which are h and c_{l+1} modified according to the 0.1 row of Fa and she checks if there is an entry in $\mathcal{CH}$ (h^ι, c^ι_{l+1}). If this is the case EE^ι_1 aborts. Otherwise, EE^ι_1 outputs h^ι as her challenge and $\mathsf{st} = (h^\iota, c^\iota_{l+1}, \mathcal{CH}, \mathcal{H}, s, \mathsf{f})$. Then, $\mathsf{EE}^\iota_2(\mathsf{st})$, parses st in $h^\iota, c^\iota_{l+1}, \mathcal{CH}, \mathcal{H}, s, \mathsf{f})$, and she computes $k_0 = \mathsf{f}^{-1}(h^\iota, c^\iota_{l+1})$. 3) after that, EE^ι_2 computes $\tilde{c_0} = f_{k_0}(p_A)$ with k_0 modified according the 0.2 line of Fa, and $c_{l+2} = \mathsf{E}_{k_0}(p_B)$, and she adds $(h^i, c^i_{l+1}, c_{l+2})$ to $\mathcal{CH}$, 4) she sees if $\tilde{c}_0 \neq c_0$, she answers $\perp$ and the leakage k_0 to A; otherwise 5) for $i = 1, \dots, l$, a) computes $k_i = \mathsf{E}_{k_{i-1}}(p_A)$ with k_{i-1} modified according to the $i.1$ line of Fa, b) computes $y_i = \mathsf{E}_{k_i}(p_B)$ with k_i modified according to the $i.2$ line of Fa c) computes $m_i = y_i \oplus c_i$ (if $i = l$ $\pi_{|c_l|}(y_l) \oplus c_l$) with y_i and c_i modified according to the $i.3$ line of Fa, 6) finally, EE^ι_2 answers $m = (m_1, \dots, m_l)$ and the leakage k_0 to A. (If $\iota = q_D + 1$ the decryption query associated to A output, EE proceeds as as normal decryption query, simply assuming that there is no fault injected).

For this, EE^ι_1 needs time t_H, while EE^ι_2 needs time $t_\mathsf{F} + (2l + 2)t_\mathsf{E} \leq t_\mathsf{F} + (2L + 2)t_\mathsf{E}$. To be polynomial, we assume that f is simulated via lazy sampling and EE_1 puts in st the samples she has already done.

When A does the ith decryption query, $i > \iota$ on input (a, c, Fa), EE^ι_2 proceeds as follows: 1) she computes $h = \mathsf{H}'_s(c_0\| \dots \|c_l, a)$ where $c_0, \dots, c_l$ and a are modified according to the 0.0 row of Fa, and she adds $((c_0, \dots, c_l, a), h)$ to $\mathcal{H}$ (with $c_0, \dots, c_l$ and a modified according to the 0.0 row of Fa), 2) she computes $k_0 = \mathsf{f}^{-1}(h, c_{l+1})$ where the inputs are modified according to 0.1 row of Fa, obtaining k_0, as answer; she sets $c^i_{l+1} := c_{l+1}$, $h^i = h$ (where c_{l+1} and h are modified according to the 0.1 row of Fa), 3) she computes $\tilde{c_0} = f_{k_0}(p_A)$ with k_0 modified according the 0.2 line of Fa, and $c_{l+2} = \mathsf{E}_{k_0}(p_B)$, 4) she sees if $\tilde{c}_0 \neq c_0$, she answers $\perp$ and the leakage k_0 to A; otherwise 5) for $i = 1, \dots, l$, a) computes $k_i = \mathsf{E}_{k_{i-1}}(p_A)$ with k_{i-1} modified according to the $i.1$ line of Fa, b) computes $y_i = \mathsf{E}_{k_i}(p_B)$ with k_i modified according to the $i.2$ line of Fa c) computes $m_i = y_i \oplus c_i$ (if $i = l$ $\pi_{|c_l|}(y_l) \oplus c_l$) with y_i and c_i modified according to the $i.3$ line of Fa, 6) finally, EE^ι_2 answers $m = (m_1, \dots, m_l)$ and the leakage k_0 to A.

For this, EE^ι_2 needs time $t_\mathsf{F} + t_\mathsf{H} + (2l + 2)t_\mathsf{E} \leq t_\mathsf{F} + t_\mathsf{H} + (2L + 2)t_\mathsf{E}$.

When A outputs her output (a^*, c^*), EE^ι_2 proceeds as follows: 1) she computes $h^* = \mathsf{H}'_s(c^*_0\| \dots \|c^*_l, a^*)$ and she adds $((c^*_0\| \dots \|c^*_l, a^*), h)$ to $\mathcal{H}$, 2) she computes $k_0^{=}\mathsf{f}^{-1}(h, c_{l+1})$ 3) she computes $\tilde{c_0} = f_{k_0}(p_A)$ with k_0 modified according the 0.2 line of Fa, and $c_{l+2} = \mathsf{E}_{k_0}(p_B)$, 4) she sees if $\tilde{c^*}_0 \neq c^*_0$, she answers $\perp$; otherwise 5) for $i = 1, \dots, l$, a) computes $k^*_i = \mathsf{E}_{k^*_{i-1}}(p_A)$, b) computes $y^*_i = \mathsf{E}_{k^*_i}(p_B)$, c) computes $m^*_i = y^*_i \oplus c^*_i$ (if $i = l$ $\pi_{|c^*_l|}(y^*_l) \oplus c^*_l$,

6) finally, EE_2^ι observe that A has given a correct decryption query, and she looks through $\mathcal{H}$ to see if there is an entry $(c_0\|\dots\|c_l, a), h)$ s.t $h = h^\iota$ and $c_0 = c^\iota_{l+2}$. If it is the case, she outputs it, otherwise 0^n.

Thus, in total EE runs in time bounded by $t+q(t_\mathsf{H})+(q_E+q_D+1)[(2L+2)+1)t_\mathsf{E}+t_\mathsf{F}] = t_3'$.

EE correctly simulates Game 6 for A, except for the computation of Fa.

Bounding $\Pr[\mathsf{Bad}_{3.\iota}|\neg\mathsf{Bad}_{3.1},\dots,\mathsf{Bad}_{3.\iota-1}]$**.** First, we observe that if EE_1^ι aborts, it means that there is a previous query for which $h^i = h^\iota$ and $c^i_{l+2} = c^\iota_{l+2}$, thus event $\mathsf{Bad}_{3.\iota}$ is event 3.i, thus,

$$\Pr[\mathsf{Bad}_{3.\iota}|\neg\mathsf{Bad}_{3.1},\dots,\mathsf{Bad}_{3.\iota-1}] = 0.$$

Second, we observe that since k_0^ι is picked randomly (and (k_0^ι, p_B) has never been queried before the ι decryption query to E), to pick the random prefix uniformly at random or to pick a k_0 uniformly at random and as a random prefix $\mathsf{E}(k_0, p_B)$ is indistinguishable for the n-PR-corpi game. We are doing the latter case for EE^ι. Thus, if event $\mathsf{Bad}_{3.\iota}\neg\mathsf{Bad}_{3.1},\dots,\mathsf{Bad}_{3.\iota-1}$ happens, then, EE wins the n-PR-corpi game. Since EE is a t_3'-adversary and H is $(t_3, \epsilon_{\mathsf{PR\text{-}corpi}})$-$n$-$\mathsf{PR}$-$\mathsf{corpi}$-secure (because we need to subtract the time needed to compute the random pre-fix), then,

$$\Pr[\mathsf{Bad}_{3.\iota}|\neg\mathsf{Bad}_{3.1},\dots,\mathsf{Bad}_{3.\iota-1}] \leq \epsilon_{\mathsf{PR\text{-}corpi}}.$$

Note that as it happened for D the fact that she does not compute correctly Fa it is not a problem, because the correct computation of Fa is only needed to compute the output of the game and the output of EE^ι is produced before the effect of Fa enters into play.

Concluding the proof First, we observe that

$$\Pr[\mathsf{Bad}_3] \leq \sum_{\iota=1}^{q_D+1} \Pr[\mathsf{Bad}_{3.\iota}|\neg\mathsf{Bad}_{3.1},\dots,\mathsf{Bad}_{3.\iota-1}] \leq \epsilon_{\mathsf{PR\text{-}corpi}} \leq (q_D+1)\epsilon_{\mathsf{PR\text{-}corpi}}.$$

Second, we observe that $\Pr[E_6] = 0$, since the adversary in the final verification query cannot have h^* equal to an h of an encryption query, and if $h^* = h$ of a decryption query, it cannot be the case that $\tilde{c^*}_0 = c_0^*$ (EE^ι covers also the case of k_0^ι random and c^ι_{l+2} (which is c_0^ι correctly computed) equal to c_0 which is computed to obtain h^ι). Moreover, if during the ith encryption query, there is another possible combination of the ciphertext blocks $c_0,\dots,c_l$, $(c_0'',\dots,c_l'')$ different from $c_0',\dots,c_l'$ (which are the inputs of Hf during the encryption query due to faults) s.t. $h = \mathsf{H}_s'((c_0'',\dots,c_l''), a'')$, a collision for H would have been found. But we have ruled out this possibility with event Bad_2. Thus,

$$\Pr[E_0] \leq \Pr[E_1] \leq \sum_{i=1}^{7} |\Pr[E_{i+1}] - \Pr[E_i]| \leq$$
$$\epsilon_{\mathsf{sTPRP}} + \frac{(q_D+2)(q_D+1)}{2^{n+1}} + \frac{(q_D+1)[q_E(L+1)+q_I+(j-1)(L+2)]}{2^n} +$$
$$\epsilon_{\mathsf{CR}} + (q_D+1)\epsilon_{\mathsf{PR\text{-}corpi}}.$$

□

If in the $\mathsf{wCILF2}$ definition we had asked that for each encryption queries there exists only one couple (a', c') s.t. $\mathsf{ADec}_k(a'c') \neq \perp$ (with a' and c' one of the various values that a and c takes during the proof), the proof would have been much more complex because we would have to check all these couples. Instead, with our definition, we make the adversary "do" all the work, because to win she can output 2 such couples, thus, she has to find them (and, thus, she has to use her time). We could have proved the existential version of wCILF, but we would have to check that all hash does not collide and every ciphertext (except at most one), when decrypted are deemed to be invalid (thus, we would have to compute all possible k_0, check if the hash is equal to the hash of another call, etc).

CONCRETE is not wCILF2 if the adversary can set 2 values. It is clear that the decoupling attack does not affect the wCILF2 security, thus, we may wonder if CONCRETE is wCILF2 secure if the adversary can set values. Unfortunately, this is not the case. Consider the following attack: 1) the adversary chooses k_0, m, and a. Then, she computes $c_0, \ldots, c_l$ as follows: $c_0 = \mathsf{E}_{k_0}(p_B)$, and then, for all $i = 1, \ldots, l$, $k_i = \mathsf{E}_{k_{i-1}}(p_A)$, $y_i = \mathsf{E}_{k_i}(p_B)$ and $c_i = y_i \oplus m_i$ (for $i = l$ $c_l = \pi_{|m_l|}(y_l) \oplus m_l$. Finally, she computes $h = \mathsf{H}'_s(a, c_0\| \ldots \|c_l)$. 2) The adversary does an encryption query on input (a', m'). In the computation she injects the following fault: in the computation of c_{l+1} she replaces the k_0 and h obtained in during the computation with the k_0 and h obtained in 1). She obtains a ciphertext c'. 3) The forgery is $(a, (c_0, \ldots, c_l, \ldots c'_{l+1}))$.

The reason why we cannot have security in this model is that with faults we can force the strongly protected TBC to output what we want. On the other hand, we have devised the decryption algorithm in a way that even if the adversary has unlimited access to F_k^{-1}, she cannot forge. We conjecture that in the unbounded leakage model, we can have no security if the adversary is allowed to set all the inputs of a strongly protected component in encryption.

But if the adversary is only allowed to set one value (and put all the differential faults she wants), CONCRETE remains wCILF.

Theorem 13. *Let* F *be a strongly protected* $(q, t_1, \epsilon_{\mathsf{sTPRP}})$-sTPRP*, let* E *be an ideal block-cipher, let* A *be allowed* q_I *queries to* E*, let* H *be a* $(t_2, \epsilon_{\mathsf{CR}})$*-collision resistant and* $(t_3, \epsilon_{\mathsf{PR\text{-}corpi}})$*-*$n$*-*PR−corpi *hash function. Then,* $(\mathcal{I}_{\mathsf{AEnc}}, \mathcal{I}_{\mathsf{ADec}})$*-protected implementation of the* AE*-scheme* CONCRETEΠ = (Gen, AEnc, ADec)*, encoding messages at most* Ln *bits long, with leaking function pair* $\mathsf{L} = (\mathsf{L}_{\mathsf{AEnc}}, \mathsf{L}_{\mathsf{ADec}})$ *and fault admissible sets* $\mathcal{F} = (\mathcal{F}_E, \mathcal{F}_D)$ *has* $(q_I, q_{FL}, q_E, q_D, t, \epsilon)$ weakly ciphertext integrity in the presence of leakage and faults against $\mathcal{F}$-fault-then-leakage attacks in encryption and decryption (wCILF2)) as long the adversary sets a single fault per encryption query *with*

$$\epsilon_{\mathsf{sTPRP}} + \frac{(q_D + 2)(q_D + 1)}{2^{n+1}} + \frac{(q_D + 1)[q_E(L + 1) + q_I + (j - 1)(L + 2)]}{2^n} +$$

$$\epsilon_{\mathsf{CR}} + (q_D + 1)\epsilon_{\mathsf{PR\text{-}corpi}}$$

with q_I *the number of queries the adversary* A *is allowed to do to the ideal block-cipher* E*, with* $q = q_E + q_D + 1$*,* $q_I = t_1 = t + qt_{\mathsf{H}} + (q(2L + 1) + 1)t_{\mathsf{E}}$*,* $t_2 = t + q(t_{\mathsf{H}}) + (q_E + q_D)[(2L + 2) + 1)t_{\mathsf{E}} + t_{\mathsf{F}}]$ $t_3 = t + q(t_{\mathsf{H}}) + (q_E + q_D + 1)[(2L + 2) + 1)t_{\mathsf{E}} + t_{\mathsf{F}}] - t_{\mathsf{F}} - t_{\mathsf{E}}$*. (we assume that picking a value uniformly at random does not take time).*

The proof which is the same as the previous proof can be found in the full version of the paper.

I.4 CIL2F2-security of CONCRETE2

First, we give the statement of Thm. 2 complete with the quantitative bounds:

Theorem 14. *Let* F *be a strongly protected* $(q, t_1, \epsilon_{\mathsf{sTPRP}})$-sTPRP*, let* E *be an ideal block-cipher, let* A *be allowed* q_I *queries to* E*, let* H *be a* $(t_1, \epsilon_{\mathsf{CR}})$*-collision resistant,* $(t_2, \epsilon_{\mathsf{PR\text{-}coirv}})$ *pre-image resistant for a chosen offset of the image of a random value, and* $(t_3, \epsilon_{\mathsf{PR\text{-}corpi}})$ *pre-image resistant with chosen target and random* n*-bit prefix input. Then, the* $(\mathcal{I}_{\mathsf{AEnc}}, \mathcal{I}_{\mathsf{ADec}})$*-protected implementation of the* AE*-scheme* CONCRETE2 Π = (Gen, AEnc, ADec)*, encrypting messages at most* Ln *bits long, with leaking function pair* $\mathsf{L} = (\mathsf{L}_{\mathsf{AEnc}}, \mathsf{L}_{\mathsf{ADec}})$ *and fault admissible sets* $\mathcal{F} = (\mathcal{F}_E, \mathcal{F}_D)$ *has* $(q_I, q_E, q_D, t, \epsilon)$ *provides* ciphertext integrity in the presence of leakage and faults against $\mathcal{F}$-fault-then-leakage attacks in encryption and decryption (CIL2F2 − (de)) *with*

$$\epsilon \leq \epsilon_{\mathsf{sTPRP}} + \epsilon_{\mathsf{CR}} + q\epsilon_{\mathsf{PR\text{-}corpi}} + 2q_E\epsilon_{\mathsf{PR\text{-}coirv}} +$$

$$\{4q_E + q(q + 1) + 1 + q_D + q_E(\frac{q_E - 1}{2})q_I(2 + 2q_D + 5q_E) +$$

$$(3L + 4)[6q_E q_D + 2q_D + 2q_D\frac{q_D + 1}{2} + 2q_E + 5q_E\frac{q_E - 1}{2})]\}2^{-n},$$

with q_I the number of queries the adversary A *is allowed to do to the ideal block-cipher* E*, with $q = q_E + q_D + 1$, $t_1 = t + q[t_\mathsf{H}) + (4(L+1)+2)t_\mathsf{E}]$, $t_2 = t + qt_\mathsf{H} + [(q-1)4(L+4)]t_\mathsf{E}$, $t_3 = t + qt_\mathsf{H} + [(q-1)(4L+6)+2q_D]t_\mathsf{E}$, t_E the time needed to compute once* E*, and t_H to compute once* H *(we assume that picking a value uniformly at random does not take time).*

In the proof, to simplify it, we use the following lemma:

Lemma 1. *Let* E *be an ideal cipher. Let (a, m, Fa) be an encryption query to* $\mathsf{CONCRETE2}$ *where the fault inserted is* Fa*. For any possible adversarial choice of a, m and fault* Fa*, provided that the adversary does not set k_0 in both the computation of $k_{0,E}$ and $k_{0,A}$ and she sets at most a single value (and use any differential fault she wants), then there exists at least a ciphertext block $c_0, \dots, c_{l+1}$ which is uniformly random.*

Proof. It is enough to prove that $\forall(a, m, \mathsf{Fa})$ with $m = (c_0, \dots, c_{l+1})$ there exists i s.t. c_i is uniformly at random, that is, for any $x \in \{0,1\}^n$,

$$\Pr[c_i = x | (c_0, \dots, c_i, \dots, c_{k+1} \leftarrow \mathsf{AEnc}_k(a, m, \mathsf{Fa}, k_0),\ k_0 \xleftarrow{\$} \{0,1\}^n] = 2^{-n},$$

where with $(c_0, \dots, c_i, \dots, c_{k+1} \leftarrow \mathsf{AEnc}_k(a, m, \mathsf{Fa}, k_0)$ we denote that we computing $\mathsf{AEnc}_k(a, m)$ picking k_0 and with the faults injected specified by Fa.

We start assuming that there is no set fault injected in the computation of $k_{0,E} = \mathsf{E}_{k_0}(p_A)$. Thus, k_0', the key used in the previous computation is uniformly at random because $k_0' = k_0 \oplus \Delta_{0,1}$ with $\Delta_{0.1}$ a differential fault chosen by the adversary before k_0 is picked uniformly at random. Thus, since E is an ideal cipher and k_0' is uniformly random, the output of the computation $k_{0,E} = \mathsf{E}_{k_0'}(p_A)$ is uniformly random.

Now let us assume that no set fault is injected in the computation $c_0 = \mathsf{E}_{k_{0,E}}(p_B)$, thus, in the computation we use $k_{0,E}' = k_{0,E} \oplus \Delta_{0.3}$ a differential fault chosen by the adversary before k_0 is picked uniformly at random. Thus, $c_0 = \mathsf{E}_{k_0'}(p_B)$ is uniformly random.

Now, we have to consider the missing cases. Let us suppose that the adversary set k_0 to k_0' in the computation of $k_{0,E} = \mathsf{E}_{k_0}(p_A)$ (or she sets $k_{0,E}$ in the computation of $c_0 = \mathsf{E}_{k_{0,E}}(p_B)$), thus in the computation of $k_{0,A} = \mathsf{E}_{k_0}(p_B)$ we use $k_0'' = k_0 \oplus \Delta_{0.2}$ with $\Delta_{0.2}$ a differential fault chosen by the adversary before k_0 is picked uniformly at random (the adversary has already used her only set fault). Thus, since E is an ideal cipher and k_0'' is uniformly random, the output of the computation $k_{0,A} = \mathsf{E}_{k_0''}(p_B)$ is uniformly random.

Then, we observe that for all $i = 1, \dots, l$, in the computation of $k_{i,A} = \mathsf{E}_{k_{i-1,A}}(m_i)$ we use $k_{i-1,A}' = k_{i,A} \oplus \Delta_{i.4,0}$ and $m_i' = m_i \oplus \Delta_{i.4,1}$ with both differential faults chosen before k_0 is picked. By induction we assume that $k_{i-1,A}$ is uniformly random (we have already proved that for $k_{0,A}$), thus $k_{i-1,A}'$ is uniformly random. Consequently, since E is an ideal cipher the output of that computation is random, thus $k_{i,A}$ is random.

Iterating we arrive to prove that $k_{l+1,A}$ is uniformly random. Now, we consider the computation of $c_{l+1} = \mathsf{E}_{k_{l+1,E}}(k_{l,A})$. Since the adversary has already used her set fault, then $k_{l,A}' = k_{l,A} \oplus \Delta_{l+1.2,1}$ is uniformly random since $\Delta_{l+1.2,1}$ has been chosen before k_0 is picked. Thus, since E is an ideal block-cipher, c_{l+1} is uniformly random.

We end by observing that it is irrelevant if the adversary has done any previous query to E that is the same as one query that is done in the execution of the proof because we do not claim that at least one block indistinguishable from one randomly picked, but only that given a k_0 picked uniformly at random at least one block is uniformly at random. □

Now, we give the formal proof:

Proof. As usual, we use a sequence of games. Let E_i be the event that the adversary wins Game i, that she outputs a fresh and valid forgery.

Game 0. It is the standard $\mathsf{CIL2F2}$ against $\mathsf{CONCRETE2}$.

Game 1. It is Game 0 where we replace F with a tweakable random permutation f.

Transition between Game 0 and Game 1. Since Game 0 and Game 1 are the same except for the replacement of F with f, we can build an adversary B to bound the difference.

She plays the sTPRP game, having to distinguish F_k from f, and simulates A's oracles using its own.

At the start of the game, B picks s in $\mathcal{HK}$ and gives it to A, and an ideal block-cipher E. Moreover, he has a list $\mathcal{S}$ which is empty.

When A does an encryption query on input (a, m, Fa), B simply picks k_0 uniformly at random from $\{0,1\}^n$ and after having parsed m in $(m_1, \ldots, m_l)$, computes $c_0, \ldots, c_{l+1}$ as follows: $1i)$ [20] $k_{0,E} = \mathsf{E}_{k'_0}(p_A)$, where k'_0 is how k_0 is modified according to the fault Fa for this computation [21], $1ii)$ $k_{0,A} = \mathsf{E}_{k''_0}(p_B)$, where k''_0 is how k_0 is modified according to the fault Fa for this computation, $1iii)$ $c_0 = \mathsf{E}_{k'_{0,E}}(p_B)$, where $k'_{0,E}$ is how $k_{0,E}$ is modified according to the fault Fa for this computation, and $1iv)$ $k_1 = \mathsf{E}_{k''_{0,E}}(p_B)$, where $k''_{0,E}$ is how $k_{0,E}$ is modified according to the fault Fa for this computation. Then, for all $i = 1, \ldots, l$, $2i)$ B computes $k_{i,E} = \mathsf{E}_{k'_i}(p_B)$, where k'_i is how k_i is modified according to the fault Fa for this computation, $2ii)$ $c_i = \mathsf{E}_{k''_{i,E}}(m'_i)$, where $k''_{i,E}$ and m'_i are how $k_{i,E}$ and m_i are modified according to the fault Fa for this computation, $2iii)$ $k_{i+1} = \mathsf{E}_{k''_i}(p_A)$, where k''_i is how k_i is modified according to the fault Fa for this computation, and $2iv)$ $k_{i,A} = \mathsf{E}_{k'_{i-1,A}}(m''_i)$, where $k'_{i-1,A}$ and m''_i are how $k_{i-1,A}$ and m_i are modified according to the fault Fa for this computation. After that, $3i)$ B computes $k_{l+1,E} = \mathsf{E}_{k'_{l+1}}(p_B)$, where k'_{l+1} is how k_{l+1} is modified according to the fault Fa for this computation, and $3ii)$ $c_{l+1} = \mathsf{E}_{k'_{l+1,E}}(k'_{l,A})$, where $k'_{l+1,E}$ and $k'_{l,A}$ are how $k_{l+1,E}$ and $k_{l,A}$ are modified according to the fault Fa for this computation. Finally, $4i)$ B computes $h = \mathsf{H}'_s(c'_0 \| \ldots \| c'_{l+1}, a)$, where c'_i is how c_i is modified according to fault Fa, and $4ii)$ she queries her oracle on input (h', k'''_0), where h' and k'''_0 are how h and k_0 are modified according to Fa in this computation, receiving as answer c_{l+2}. So, she can answer A $c = (c_0 \| \ldots \| c_{l+2})$ and the leakage k_0. Moreover, B adds (a, c) to the list $\mathcal{S}$.

For this, B does one oracle query and needs time $t_{\mathsf{H}} + (4(l+1)+2)t_{\mathsf{E}} \leq t_{\mathsf{H}} + (4(L+1)+2)t_{\mathsf{E}}$.

When A does a decryption query on input (a, c, Fa), B parses $c = (c_0, \ldots, c_{l+2})$, and she proceeds as follows: $1i)$ B computes $h = \mathsf{H}'_s(c'_0 \| \ldots \| c'_{l+1}, a)$, where c'_i is how c_i is modified according to fault Fa, and $1ii)$ she queries her inverse oracle on input (h', c_{l+2}), where h' is how h is modified according to Fa in this computation [22], receiving as answer k_0. Then, $2i)$ $k_{0,E} = \mathsf{E}_{k'_0}(p_A)$, where k'_0 is how k_0 is modified according to the fault Fa for this computation, $2ii)$ $k_{0,A} = \mathsf{E}_{k''_0}(p_B)$, where k''_0 is how k_0 is modified according to the fault Fa for this computation, $2iii)$ $\tilde{c}_0 = \mathsf{E}_{k'_{0,E}}(p_B)$, where $k'_{0,E}$ is how $k_{0,E}$ is modified according to the fault Fa for this computation, and she checks if $c_0 \overset{?}{=} \tilde{c}_0$, if it is not the case she answers $\perp$ to A and the leakage k_0; otherwise $2iv)$ she computes $k_1 = \mathsf{E}_{k''_{0,E}}(p_B)$, where $k''_{0,E}$ is how $k_{0,E}$ is modified according to the fault Fa for this computation. After that B computes for all $i = 1, \ldots, l$, $3i)$ B computes $k_{i,E} = \mathsf{E}_{k'_i}(p_B)$, where k'_i is how k_i is modified according to the fault Fa for this computation, $3ii)$ $m_i = \mathsf{E}^{-1}_{k''_{i,E}}(c''_i)$, where $k''_{i,E}$ and c''_i are how $k_{i,E}$ and c_i are modified according to the fault Fa for this computation, $3iii)$ $k_{i+1} = \mathsf{E}_{k''_i}(p_A)$, where k''_i is how k_i is modified according to the fault Fa for this computation, and $3iv)$ $k_{i,A} = \mathsf{E}_{k'_{i-1,A}}(m'_i)$, where $k'_{i-1,A}$ and m'_i are how $k_{i-1,A}$ and m_i are modified according to the fault Fa for this computation. Finally, B computes $4i)$ B computes $k_{l+1,E} = \mathsf{E}_{k'_{l+1}}(p_B)$, where k'_{l+1} is how k_{l+1} is modified according to the fault Fa for this computation, and $4ii)$ $\tilde{c}_{l+1} = \mathsf{E}_{k'_{l+1,E}}(k'_{l,A})$, where $k'_{l+1,E}$ and $k'_{l,A}$ are how $k_{l+1,E}$ and $k_{l,A}$ are modified according to the fault Fa for this computation. $4iii)$ if $\tilde{c}_{l+1}$ is equal to c_{l+1}, B answers $m = (m_1, \ldots, m_l)$ and the leakage k_0 to A; otherwise $\perp$ and the leakage k_0.

For this, B does one oracle query and needs time $t_{\mathsf{H}} + (4(l+1)+2)t_{\mathsf{E}} \leq t_{\mathsf{H}} + (4(L+1)+2)t_{\mathsf{E}}$.

[20] Note that each of the steps, which we divide this computation in, corresponds to an atom (see Sec. 2.6).

[21] That is, she calls the ideal cipher E on input (k'_0, p_A), we use *compute* to simplify the writing. Moreover, in this transition it is irrelevant the fact that E is an ideal block-cipher.

[22] Since this is the only time, we use c_{l+2}, the adversary cannot modify it via a fault.

When A outputs her output (a^*, c^*), B proceeds as for a normal decryption query with the following differences: 1) there are no faults involved, 2) in step $4iii)$ if $\tilde{c}_{l+1}$ is equal to c_{l+1} and $(a^*, c^*) \notin \mathcal{S}$, B outputs 1, otherwise 0.

For this, B does one oracle query and needs time $t_{\mathsf{H}}+(4(l+1)+2)t_{\mathsf{E}} \leq t_{\mathsf{H}}+(4(L+1)+2)t_{\mathsf{E}}$.

Thus, in total B does at most $q_E + q_D + 1 = q$ queries to her oracle and runs in time bounded by $t + q[t_{\mathsf{H}}) + (4(L+1)+2)t_{\mathsf{E}}] = t_1$.

If her oracle is implemented with F, B correctly simulates Game 0 for A, otherwise Game 1. Since F is a $(q, t_1, \epsilon_{\mathsf{sTPRP}})$-$\mathsf{sTPRP}$,

$$|\Pr[E_1] - \Pr[E_0]| \leq \epsilon_{\mathsf{sTPRP}}.$$

Game 2. It is Game 1, where instead of using f and f^{-1}, we use two tweakable random functions f and g (but, if we have obtained $c_{l+2} = \mathsf{f}(h, k_0)$ we define $\mathsf{g}_k(h, c_{l+2}) := k_0$ (and vice-versa).

Transition between Game 1 and Game 2. In Game 2, for decryption queries, we are using a sTPRP, while in Game 3 a PRF. Due to the well-known birthday bound, we have that

$$|\Pr[E_1] - \Pr[E_2]| \leq 2\frac{q(q+1)}{2^{n+1}},$$

because in addition to switching two PRPs with two PRFs (but since these functions are correlated we do a single switch), we have to consider the probability that we cannot do the simulation correctly, that is, when there exists a tweak h and two inputs x, x' s.t. $\mathsf{f}(h, x) = \mathsf{f}(h, x')$ (in this case, we cannot compute the inverse correctly) and vice-versa.

New sampling of f and g. From now on, when we say that an adversary *computes* f and g we mean that we have a list $\mathcal{L}$, and every time we have to compute $\mathsf{f}(x, y)$ (resp. $g(x, z)$) we look into $\mathcal{L}$ to see if there is a triple (x, y, z). If it is the case, we output z (resp.y); otherwise, we pick $z \xleftarrow{\$} \{0,1\}^n$ (resp. $y \xleftarrow{\$} \{0,1\}^n$) and we output z (resp. x) and we add (x, y, z) to $\mathcal{L}$.

We have already covered the case that we cannot correctly simulate the game.

Game 3. It is Game 2 where we assume that during the execution of the game, there are no collisions in the hash function.

Transition between Game 2 and Game 3. Since Game 2 and Game 3 are the same except if the following event Bad_1 happens: there is a collision for the hash function.

To bound $\Pr[\mathsf{Bad}_1]$ we build adversary C.

She plays the CR game against the hash function H, using the adversary A.

At the start of the game, C receives a key s in $\mathcal{HK}$ and gives it to A, and an ideal block-cipher E. Moreover, he has two lists $\mathcal{S}$ and $\mathcal{H}$ which are empty.

When A does an encryption query on input (a, m, Fa), C simply picks k_0 uniformly at random from $\{0,1\}^n$ and after having parsed m in $(m_1, \dots, m_l)$, computes $c_0, \dots, c_{l+1}$ as follows: $1i)$ $k_{0,E} = \mathsf{E}_{k_0'}(p_A)$, where k_0' is how k_0 is modified according to the fault Fa for this computation [23], $1ii)$ $k_{0,A} = \mathsf{E}_{k_0''}(p_B)$, where k_0'' is how k_0 is modified according to the fault Fa for this computation, $1iii)$ $c_0 = \mathsf{E}_{k_{0,E}'}(p_A)$, where $k_{0,E}'$ is how $k_{0,E}$ is modified according to the fault Fa for this computation, and $1iv)$ $k_1 = \mathsf{E}_{k_{0,E}''}(p_B)$, where $k_{0,E}''$ is how $k_{0,E}$ is modified according to the fault Fa for this computation. Then, for all $i = 1, \dots, l$, $2i)$ C computes $k_{i,E} = \mathsf{E}_{k_i'}(p_B)$, where k_i' is how k_i is modified according to the fault Fa for this computation, $2ii)$ $c_i = \mathsf{E}_{k_{i,E}''}(m_i')$, where $k_{i,E}''$ and m_i' are how $k_{i,E}$ and m_i are modified according to the fault Fa for this computation, $2iii)$ $k_{i+1} = \mathsf{E}_{k_i''}(p_A)$, where k_i'' is how k_i is modified according to the fault Fa for this computation, and $2iv)$ $k_{i,A} = \mathsf{E}_{k_{i-1,A}'}(m_i'')$, where $k_{i-1,A}'$ and m_i'' are how $k_{i-1,A}$ and m_i are modified according to the fault Fa for this computation. After that, $3i)$ C computes $k_{l+1,E} = \mathsf{E}_{k_{l+1}'}(p_B)$, where k_{l+1}' is how k_{l+1}

[23]That is, she calls the ideal cipher E on input (k_0', p_A), we use *compute* to simplify the writing. Moreover, in this transition it is irrelevant the fact that E is an ideal block-cipher.

is modified according to the fault Fa for this computation, and $3ii)$ $c_{l+1} = \mathsf{E}_{k'_{l+1,E}}(k'_{l,A})$, where $k'_{l+1,E}$ and $k'_{l,A}$ are how $k_{l+1,E}$ and $k_{l,A}$ are modified according to the fault Fa for this computation. Finally, $4i)$ C computes $h = \mathsf{H}'_s(c'_0\| \dots \|c'_{l+1}, a)$, where c'_i is how c_i is modified according to fault Fa, and she adds $((c'_0\| \dots \|c'_{l+1}, a), h)$ to $\mathcal{H}$, and $4ii)$ she queries her oracle on input (h', k'''_0), where h' and k'''_0 are how h and k_0 are modified according to Fa in this computation, receiving as answer c_{l+2}. So, she can answer A $c = (c_0\| \dots \|c_{l+2})$ and the leakage k_0. Moreover, C adds (a, c) to the list $\mathcal{S}$.

For this, C needs time $t_\mathsf{H} + (4(l+1)+2)t_\mathsf{E} \leq t_\mathsf{H} + (4(L+1)+2)t_\mathsf{E}$.

When A does a decryption query on input (a, c, Fa), C parses $c = (c_0, \dots, c_{l+2})$, and she proceeds as follows: $1i)$ C computes $h = \mathsf{H}'_s(a, c'_0\| \dots \|c'_{l+1})$, where c'_i is how c_i is modified according to fault Fa, and she adds $((c'_0\| \dots \|c'_{l+1}, a), h)$ to $\mathcal{H}$, and $1ii)$ she queries her inverse oracle on input (h', c_{l+2}), where h' is how h is modified according to Fa in this computation, receiving as answer k_0. Then, $2i)$ $k_{0,E} = \mathsf{E}_{k'_0}(p_A)$, where k'_0 is how k_0 is modified according to the fault Fa for this computation, $2ii)$ $k_{0,A} = \mathsf{E}_{k''_0}(p_B)$, where k''_0 is how k_0 is modified according to the fault Fa for this computation, $2iii)$ $\tilde{c}_0 = \mathsf{E}_{k'_{0,E}}(p_B)$, where $k'_{0,E}$ is how $k_{0,E}$ is modified according to the fault Fa for this computation, and she checks if $c_0 \overset{?}{=} \tilde{c}_0$, if it is not the case she answers $\perp$ to A and the leakage k_0; otherwise $2iv)$ she computes $k_1 = \mathsf{E}_{k''_{0,E}}(p_B)$, where $k''_{0,E}$ is how $k_{0,E}$ is modified according to the fault Fa for this computation. After that C computes for all $i = 1, \dots, l$, $3i)$ C computes $k_{i,E} = \mathsf{E}_{k'_i}(p_B)$, where k'_i is how k_i is modified according to the fault Fa for this computation, $3ii)$ $m_i = \mathsf{E}^{-1}_{k''_{i,E}}(c''_i)$, where $k''_{i,E}$ and c''_i are how $k_{i,E}$ and c_i are modified according to the fault Fa for this computation, $3iii)$ $k_{i+1} = \mathsf{E}_{k''_i}(p_A)$, where k''_i is how k_i is modified according to the fault Fa for this computation, and $3iv)$ $k_{i,A} = \mathsf{E}_{k'_{i-1,A}}(m'_i)$, where $k'_{i-1,A}$ and m'_i are how $k_{i-1,A}$ and m_i are modified according to the fault Fa for this computation. Finally, C computes $4i)$ C computes $k_{l+1,E} = \mathsf{E}_{k'_{l+1}}(p_B)$, where k'_{l+1} is how k_{l+1} is modified according to the fault Fa for this computation, and $4ii)$ $\tilde{c}_{l+1} = \mathsf{E}_{k'_{l+1,E}}(k'_{l,A})$, where $k'_{l+1,E}$ and $k'_{l,A}$ are how $k_{l+1,E}$ and $k_{l,A}$ are modified according to the fault Fa for this computation. $4iii)$ if $\tilde{c}_{l+1}$ is equal to c_{l+1}, C answers $m = (m_1, \dots, m_l)$ and the leakage k_0 to A; otherwise $\perp$ and the leakage k_0.

For this, C needs time $t_\mathsf{H} + (4(l+1)+2)t_\mathsf{E} \leq t_\mathsf{H} + (4(L+1)+2)t_\mathsf{E}$.

When A outputs her output (a^*, c^*), C proceeds as for a normal decryption query with the following differences: 1) there are no faults involved, 2) C does not answer A anything 3) after having computed this decryption query, C looks into her list $\mathcal{H}$ if she finds a collision: if it is the case, she outputs it, otherwise $(0^n, 1^n)$.

For this, C needs time $t_\mathsf{H} + (4(l+1)+2)t_\mathsf{E} \leq t_\mathsf{H} + (4(L+1)+2)t_\mathsf{E}$.

Thus, in total C runs in time bounded by $t + q[t_\mathsf{H}) + (4(L+1)+2)t_\mathsf{E}] = t_1$.

C correctly simulate Game 2 for A. Since Game 2 and Game 3 are the same, except if event Bad_1 happens, and C wins her CR-game if event Bad_1 happens, and C is a t_1-adversary and H is a $(t_1, \epsilon_\mathsf{CR})$-collision-resistant hash function,

$$|\Pr[E_2] - \Pr[E_3]| = \Pr[\mathsf{Bad}_1] \leq \epsilon_\mathsf{CR}.$$

Game 4. It is Game 3, where in every decryption query (not the challenge one), immediately after k_0 is computed (via g), the decryption oracle computes $k_{l+3,E} = \mathsf{E}_{k_0}(p_A)$ and $c_{l+3} = \mathsf{E}_{k_{l+3,E}}(p_B)$ and it is appended to the leakage.

Transition between Game 3 and Game 4. These games are the same except for these two additional calls per decryption query that any adversary can do. Thus, it is easy to build an adversary A' which when she receives the answer of a decryption query computes $k_{l+3,E} = \mathsf{E}_{k_0}(p_A)$ and $c_{l+3} = \mathsf{E}_{k_0,E}(p_B)$, appends them to the leakage and forwards them to the adversary.

Such an adversary needs $2q_D$ calls to the ideal blockcipher. Thus,

$$\Pr[E_3] = \Pr[E_4].$$

Game 5. It is Game 4, with the condition that the following event does not happen: during a decryption query, when a new key k_0 is picked by g [24], there exists no query to the ideal cipher on input (k_0, p_A).

Transition between Game 4 and Game 5. The two games are the same except if the following Bad_2 event happens: there exists a decryption query s.t. it requires to pick a new key k_0^i and there exists a previous query to the ideal cipher E on input (k_0^i, p_A). We observe that during an encryption query there at most $L+2$ queries to $\mathsf{E}(\cdot, p_A)$ with at most $L+2$ different keys, and $2L+1$ queries with $(\cdot,\cdot)$ where the key is either $k_{j,E}$ or $k_{j,A}$ for a certain j and the input is a message m_j or $k_{l,A}$ (which can be equal to p_A [25]), while during a decryption query there at most $L+2$ different queries to the ideal block-cipher with input $\mathsf{E}(\cdot, p_A)$ and $2L+1$ to $\mathsf{E}(\cdot,\cdot)$. In addition, due to the change we have introduced in Game 4, there are two additional queries, one of which is to $\mathsf{E}(\cdot, p_A)$ The adversary can do at most q_I queries of its choice to the ideal block-cipher. Thus, during the jth decryption query there at most $q_E(3L+4) + q_I + (j-1)(3L+4)$ different keys that, if picked, would trigger the Bad_2 event. Thus,

$$\Pr[\mathsf{Bad}_2] \leq \sum_{j=1}^{q_D} \frac{q_E(3L+4) + q_I + (j-1)(3L+4)]}{2^n} \leq$$

$$\frac{q_D[q_E(3L+4) + q_I] + q_D(\frac{q_D+1}{2})(3L+4)]}{2^n} = q_D[(3L+4)(q_E + \frac{q_D+1}{2}) + q_I]2^{-n}.$$

$$\text{Thus, } |\Pr[E_4] - \Pr[E_5]| \leq \Pr[\mathsf{Bad}_2] = q_D[(3L+4)(q_E + \frac{q_D+1}{2}) + q_I]2^{-n}.$$

Game 6. It is Game 5, with the condition that the following event does not happen: during a decryption query, when a new key k_0 is picked by g, and the associated $k_{0,E} = \mathsf{E}_{k_0}(p_B)$ is computed, there exists no query to the ideal cipher on input $(k_{0,E}, p_B)$.

Transition between Game 5 and Game 6. The two games are the same except if the following Bad_3 event happens: there exists a decryption query s.t. it requires to pick a new key k_0^i, and s.t., given $k_{0,E}^i = \mathsf{E}_{k_0^i}(p_A)$ its associated key (which is randomly picked due to the fact that event Bad_2 has not happened), there exists a previous query to the ideal cipher E on input $(k_{0,E}^i, p_B)$. As before, we have to consider $3L+3$ possible problematic ideal cipher queries per encryption query, and $3L+5$ per previous decryption query (considering the additional query due to the change introduced in Game 4) [26], and the q_I adversarial query to the ideal block-cipher. Thus, during the jth decryption query there at most $q_E(3L+4) + q_I + (j-1)[3(L+1)+2]$ different keys that, if picked, would trigger the Bad_3 event. Thus,

$$\Pr[\mathsf{Bad}_3] \leq \sum_{j=1}^{q_D} \frac{q_E(3L+4) + q_I + (j-1)[(3L+4)+1]}{2^n} \leq$$

$$\frac{q_D[q_E(3L+4) + q_I] + q_D(\frac{q_D+1}{2})[(3L+4)+1]}{2^n} = q_D[(3L+4)(q_E + \frac{q_D+1}{2}) + q_I + 1]2^{-n}.$$

$$\text{Thus, } |\Pr[E_5] - \Pr[E_6]| \leq \Pr[\mathsf{Bad}_3] = q_D[(3L+4)(q_E + \frac{q_D+1}{2}) + q_I + 1]2^{-n}.$$

Game 7. It is Game 6, where the following event Bad_4 does not happen: for any tweak h used by g only in decryption queries, there exist two decryption queries, which we call the ith and the jth s.t. $h' = \mathsf{H}(c_0^j \| \dots \| c_{l+1}^j, a^j)$ (where $c_0^j, \dots, c_l^j$ and a^j are modified

[24]that is, there is no entry in the list used for f and g containing (h', k_0, c_{l+2}), with h' hash h obtained in step $1i$) and modified according to what fault Fa specifies for the step $1ii$).

[25]If instead of using a block-cipher E, we use a tweakable blockcipher, we can avoid this situation and significantly improve the number of possible calls to E, see Alg. 8 in the appendix.

[26]Since here we consider queries $\mathsf{E}(\cdot, p_B)$, while in the previous transition we consider queries $\mathsf{E}(\cdot, p_A)$, it is enough to observe that for every query $\mathsf{E}(k_j, p_A)$ (or $\mathsf{E}(k_{0,E}, p_A)$ there exists one query $\mathsf{E}(k_j, p_B)$ (or $\mathsf{E}(k_{0,E}, p_B)$). In addition, there is a final query $\mathsf{E}(k_{l+1}, p_B)$ which does not correspond to any $\mathsf{E}(k_{l+1}, P_A)$.

according to the fault Fa^j for the $1i)$ step, s.t. the modified c_0^j is equal to c_{l+2}^i obtained in the ith decryption query, in which $\mathsf{f}^{-1}(h^i, \tau^i)$ is computed (modified according to the Fa^i for step $1ii)$) and the h^i modified is equal to h'.

Transition between Game 6 and Game 7. Since Game 6 and Game 7 are the same except if event Bad_4 happens, it is enough to bound $\Pr[\mathsf{Bad}_4]$ to bound the difference $|\Pr[E_6] - \Pr[E_7]|$. To bound $\Pr[\mathsf{Bad}_4]$, we divide it in $q_D + 1$ different events:$\mathsf{Bad}_{4.1}, \dots, \mathsf{Bad}_{4.q_D+1}$, where $\mathsf{Bad}_{4.\iota}$ is event Bad_4 where $\iota = i$. Clearly, $\mathsf{Bad}_4 = \bigcup_{\iota=1}^{q_D+1} \mathsf{Bad}_{4.\iota}$.

To bound $\Pr[\mathsf{Bad}_{4.\iota}]$ we build an adversary D^ι against the n-PR-corpi-security of the hash function H.

The n-PR-corpi-adversary D^ι. At the start of the game, D^ι receives a key for the hash function s in $\mathcal{HK}$ and gives it to A. Moreover, she picks two random tweakable functions f and g as discussed before. Finally, she has a list $\mathcal{H}$ which is empty.

When A does an encryption query on input (a, m), D^ι (we use D^ι) to identify both D_1^ι and D_2^ι) behaves as C. For this, D^ι needs time $t_\mathsf{H} + [4(l+1)+2]t_\mathsf{E} \le t_\mathsf{H} + [4(L+1)+2]t_\mathsf{E}$.

When A does the ith decryption query, $i < \iota$ on input (a, c, Fa), D_1^ι proceeds as C with the following differences: after step $1ii)$, and before to step $2i)$, she $1iii)$ computes $k_{l+3,E} = \mathsf{E}_{k_0}(p_A)$ and $1iv)$ $c_{l+3} = \mathsf{E}_{k_{0,E}}(p_B)$.

For this, D_1^ι needs time $t_\mathsf{H} + 4(l+2)t_\mathsf{E} \le t_\mathsf{H} + 4(L+2)t_\mathsf{E}$.

When A does the ι decryption query on input (a, c, Fa), D_1^ι proceeds as follows: she performs $1i)$, then she checks if she has to sample g or not, that is, if there is an entry in the list $\mathcal{L}$ containing $(\cdot, h^\iota, c_{l+2}^\iota)$ with h the actual input used. If this is the case she aborts, otherwise, she outputs h^ι as her challenge, and the state $\mathsf{st} = (h^\iota, c_{l+2}^\iota, \mathcal{H}, s, \mathsf{f}, \mathsf{g})$. D_2^ι receives c_{l+3} which is a random value. D_2^ι parses st in $(h^\iota, c_{l+2}^\iota, \mathcal{H}, s, \mathsf{f}, \mathsf{g})$, she picks $k_{0,A}$ and k_1. From then, she proceeds as C from step $2i)$. For this D_1^ι needs time t_H, while D_2^ι needs time $(4l+2)t_\mathsf{E} \le (4L+2)t_\mathsf{E}$.

When A does the ith decryption query, $i > \iota$ on input (a, c, Fa), D_2^ι proceeds as D_1^ι for the ith decryption query $(i > \iota)$, thus she needs time $t_\mathsf{H} + 4(l+2)t_\mathsf{E} \le t_\mathsf{H} + 4(L+2)t_\mathsf{E}$.

When A outputs her output (a^*, c^*), D_2^ι proceeds as follows: she computes $h^* = \mathsf{H}'_s(c_0^* \| \dots \| c_{l+1}^*, a^*)$ and she adds $((c_0^* \| \dots \| c_{l+1}^*, a^*), h)$ to $\mathcal{H}$, and then, she looks through $\mathcal{H}$ to see if there is an entry $(c_0 \| \dots \| c_{l+1}, a), h)$ s.t $h = h^\iota$ and $c_0 = c_{l+3}^\iota$. If it is the case, she outputs it, otherwise 0^n. This takes time t_H.

Thus, in total D runs in time bounded by

$$\begin{aligned} t + q_E[t_\mathsf{H} + (4(L+1)+2)t_\mathsf{E}] + (\iota - 1)[t_\mathsf{H} + (4(L+2))t_\mathsf{E}] + \\ t_\mathsf{H} + (4L+2)t_\mathsf{E} + (q_D - \iota)[t_\mathsf{H} + (4(L+2))t_\mathsf{E}] + t_\mathsf{H} \\ t + qt_\mathsf{H} + [(q-1)(4L+6) + 2q_D]t_\mathsf{E} \le t_3. \end{aligned}$$

Bounding $\Pr[\mathsf{Bad}_{4.\iota} | \neg \mathsf{Bad}_{4.1}, \dots, \mathsf{Bad}_{4.\iota-1}]$**.** First, we observe that if D_1^ι aborts, it means that there is a previous query for which $h^i = h^\iota$ and $c_{l+3}^i = c_{l+3}^\iota$, thus event $\mathsf{Bad}_{4.\iota}$ is event $\mathsf{Bad}_{4.i}$, thus,

$$\Pr[\mathsf{Bad}_{4.\iota} | \neg \mathsf{Bad}_{4.1}, \dots, \mathsf{Bad}_{4.\iota-1}] = 0.$$

Second, we observe that since k_0^ι, and $k_{l+3,E}^\iota$ are picked randomly (and (k_0^ι, p_A) and $(k_{l+3,E}^\iota, p_B)$ have never been queried before the ι decryption query to E due to the events Bad_2 and Bad_3 not happening), to pick the random prefix uniformly at random or to pick a k_0 uniformly at random, to compute $k_{l+3,E} = \mathsf{E}_{k_0}(p_A)$ and as a random prefix $\mathsf{E}(k_{l+3}, p_B)$ is indistinguishable for the n-PR-corpi game. We are doing the latter case for D^ι. Thus, if event $\mathsf{Bad}_{4.\iota} \neg \mathsf{Bad}_{4.1}, \dots, \mathsf{Bad}_{4.\iota-1}$ happens, then, D wins the n-PR-corpi game. Since D is a t_3-adversary and H is $(t_3, \epsilon_{\mathsf{PR\text{-}corpi}})$-$n$-$\mathsf{PR}$-$\mathsf{corpi}$-secure, then,

$$\Pr[\mathsf{Bad}_{4.\iota} | \neg \mathsf{Bad}_{4.1}, \dots, \mathsf{Bad}_{4.\iota-1}] \le \epsilon_{\mathsf{PR\text{-}corpi}}.$$

Bounding $|\Pr[E_6] - \Pr[E_7]|$. We observe that

$$\Pr[\mathsf{Bad}_4] \le \sum_{\iota=1}^{q_D+1} \Pr[\mathsf{Bad}_{4.\iota} | \neg \mathsf{Bad}_{4.1}, \dots, \mathsf{Bad}_{4.\iota-1}] \le \epsilon_{\mathsf{PR\text{-}corpi}} \le (q_D+1)\epsilon_{\mathsf{PR\text{-}corpi}}.$$

$$\text{Thus}, \Pr[E_6] - \Pr[E_7] \le \Pr[\mathsf{Bad}_4] \le (q_D + 1)\epsilon_{\mathsf{PR\text{-}corpi}}.$$

Having proved that no triple (h, k_0, c_{l+2}) associated to a decryption query can be used to forge, we pass to the triples associated to encryption queries.

Game 8. It is Game 7, where the following event Bad_5 does not happen: for every encryption query, if the adversary has not set k_0 in the input of the f computation, then, there is no previous query to $\mathsf{E}(k_0', p_A)$, where k_0' is how k_0 is modified for step $4ii)$ according to the fault Fa.

Transition between Game 7 and Game 8. The two games are the same except if the following Bad_5 event happens: there exists an encryption query $(a^j, m^j, \mathsf{Fa}^j)$, the j^{th}, s.t. 1) Fa^j does not set the k_0 input for the computation considered in Step $4ii)$ (the call to f), 2) let k_0^j the key picked, and let $k_0^{\dagger,j}$ be how k_0^j is modified for the computation of Step $4ii$, then there is no previous query to the ideal cipher on input $(k_0^{\dagger,j}, p_A)$.

Now, note that since the fault is predetermined, a challenger can compute $k_0^{\dagger,j}$ before having to do any computation for the j^{th} encryption query.

We observe that during an encryption query there at most $L+2$ queries to $\mathsf{E}(\cdot, p_A)$ with at most $L+2$ different keys, and $2L+1$ queries with $(\cdot,\cdot)$ where the key is either $k_{\iota,E}$ or $k_{\iota,A}$ for a certain ι and the input is a message m_ι (which can be equal to p_A), while during a decryption query there at most $L+2$ different queries to the ideal block-cipher with input $\mathsf{E}(\cdot, p_A)$ and $2L+1$ to $\mathsf{E}(\cdot,\cdot)$. In addition, due to the change we have introduced in Game 4, there are two additional queries, one of which is to $\mathsf{E}(\cdot, p_A)$ The adversary can do at most q_I queries of its choice to the ideal block-cipher. Thus, during the j^{th} encryption query there at most $(j-1)(3L+3) + q_D(3L+4) + q_I$ different keys that, if picked, would trigger the Bad_5 event. Thus,

$$\Pr[\mathsf{Bad}_5] \le \sum_{j=1}^{q_E} \frac{(j-1)(3L+3) + q_D(3L+4) + q_I}{2^n} \le$$
$$[q_E(3L+3)(q_E-1)/2 + q_E q_I + q_E q_D(3L+4)]2^{-n} =$$
$$q_E[3(q_E-1)(L+1)/2 + q_I + q_D(3L+4)]2^{-n}.$$

$$\text{Thus}, |\Pr[E_7] - \Pr[E_8]| \le \Pr[\mathsf{Bad}_5] = q_E[3(q_E-1)(L+1)/2 + q_I + q_D(3L+4)]2^{-n}.$$

Game 9. It is Game 8, where the following event Bad_6 does not happen: for every encryption query, if the adversary has not set k_0 in the input of the f computation, then, there is no previous query to $\mathsf{E}(k_{0,E}', p_B)$, where $k_{0,E}' = \mathsf{E}_{k_0'}(p_B)$, and k_0' is how k_0 is modified for step $4ii)$ according to the fault Fa.

Transition between Game 8 and Game 9. The two games are the same except if the following Bad_6 event happens: there exists an encryption query $(a^j, m^j, \mathsf{Fa}^j)$, the j^{th}, s.t. 1) Fa^j does not set the k_0 input for the computation considered in Step $4ii)$ (the call to f), 2) let k_0^j the key picked, and let $k_0^{\dagger,j}$ be how k_0^j is modified for the computation of Step $4ii$, and $k_{0,E}^{\dagger,j}$ then there is no previous query to the ideal cipher on input $(k_{0,E}^{\dagger,j}, p_B)$.

Now, note that since the fault is predetermined, a challenger can compute $k_0^{\dagger,j}$ and $k_{0,E}^{\dagger,j}$ (which is random due to the previous transition), before having to do any computation for the j^{th} encryption query.

We observe that during an encryption query there at most $L+3$ queries to $\mathsf{E}(\cdot, p_B)$ with at most $L+3$ different keys, and $2L+1$ queries with $(\cdot,\cdot)$ where the key is either $k_{\iota,E}$ or $k_{\iota,A}$ for a certain ι and the input is a message m_ι (which can be equal to p_B), while during a decryption query there at most $L+3$ different queries to the ideal block-cipher with input $\mathsf{E}(\cdot, p_B)$ and $2L+1$ to $\mathsf{E}(\cdot,\cdot)$. In addition, due to the change we have introduced in Game 4, there are two additional queries, one of which is to $\mathsf{E}(\cdot, p_B)$ The adversary can do at most q_I queries of its choice to the ideal block-cipher. Thus, during the j^{th}

encryption query there at most $(j-1)(3L+4)+q_D(3L+4)+q_I$ different keys that, if picked, would trigger the Bad_6 event. Thus,

$$\Pr[\mathsf{Bad}_6] \le \sum_{j=1}^{q_E} \frac{(j-1)(3L+4)+q_D(3L+4)+q_I}{2^n} \le$$
$$[q_E(3L+4)(q_E-1)/2+q_Eq_I+q_Eq_D(3L+4)]2^{-n} =$$
$$q_E[(q_E-1)(3L+4)/2+q_I+q_D(3L+4)]2^{-n}.$$

Thus, $|\Pr[E_8]-\Pr[E_9]| \le \Pr[\mathsf{Bad}_6] = q_E[(q_E-1)(3L+4)/2+q_I+q_D(3L+4)]2^{-n}$.

Game 10. It is Game 9, where in every encryption query, immediately after Step $4ii$), after $k_0^\dagger$ is computed (where $k_0^\dagger$ is k_0 modified for the computation of Step $4ii$ according to fault Fa), the encryption oracle computes $k_{l+3,E} = \mathsf{E}_{k_0^\dagger}(p_A)$ and $c_{l+3} = \mathsf{E}_{k_{l+3},E}(p_B)$ and it is appended to the leakage.

Transition between Game 9 and Game 10. These games are the same except for these two additional calls per encryption query that any adversary can do. Thus, it is easy to build an adversary A' which when she receives the answer of a encryption query computes $k_{l+3,E} = \mathsf{E}_{k_0}(p_A)$ and $c_{l+3} = \mathsf{E}_{k_0,E}(p_B)$, appends them to the leakage and forwards them to the adversary.

Such an adversary needs $2q_E$ calls to the ideal blockcipher. Thus,

$$\Pr[E_9] = \Pr[E_{10}].$$

Game 11. It is Game 10, here the following event Bad_7 does not happen: there is an encryption query, the j^{th}, s.t. all these conditions holds: 1) the adversary does not use a set fault for k_0^j in Step $1i$) during the the j^{th} encryption query 2) there exists a previous ideal cipher query on input $(k_0^{j,\dagger}, p_A)$ (where $k_0^{j,\dagger}$ is how k_0^j is modified according to the fault Fa^j for the $1i$) step).

Transition between Game 10 and Game 11. The two games are the same except if even Bad_7 happens. Since A is not setting k_0^j in Step $1i$) via a fault, thus, $k_0^{j,\dagger}$ is random, where $k_0^{j,\dagger} = k_0^j \oplus \Delta$ with k_0^j randomly picked and Δ, the offset specified by the fault Fa^j which is chosen before k_0^j is picked.

Similar to what we have done before, we observe that during an encryption query there at most $L+3$ queries to $\mathsf{E}(\cdot, p_A)$ with at most $L+3$ different keys (we have an additional query due to Game 10), and $2L+1$ queries with $(\cdot,\cdot)$ where the key is either $k_{\iota,E}$ or $k_{\iota,A}$ for a certain ι and the input is a message m_ι (which can be equal to p_A), while during a decryption query there at most $L+2$ different queries to the ideal block-cipher with input $\mathsf{E}(\cdot, p_A)$ and $2L+1$ to $\mathsf{E}(\cdot,\cdot)$. In addition, due to the change we have introduced in Game 4, there are two additional queries, one of which is to $\mathsf{E}(\cdot, p_A)$ The adversary can do at most q_I queries of its choice to the ideal block-cipher. Thus, during the j^{th} encryption query there at most $(j-1)(3L+4)+q_D(3L+4)+q_I$ different keys that, if picked, would trigger the Bad_5 event. Thus,

$$\Pr[\mathsf{Bad}_7] \le \sum_{j=1}^{q_E} \frac{(j-1)(3L+4)+q_D(3L+4)+q_I}{2^n} \le$$
$$[q_E(3L+4)(q_E-1)/2+q_Eq_I+q_Eq_D(3L+4)]2^{-n} =$$
$$q_E[(3L+4)[q_D+(q_E-1)/2]+q_I]2^{-n}.$$

Thus, $|\Pr[E_{10}]-\Pr[E_{11}]| \le \Pr[\mathsf{Bad}_7] = q_E[(3L+4)[q_D+(q_E-1)/2]+q_I]2^{-n}$.

Game 12. It is Game 11, here the following event Bad_8 does not happen: there is an encryption query, the j^{th}, s.t. all these conditions holds: 1) the adversary does not use a set fault for k_0^j in Step $1i$) during the the j^{th} encryption query, 2) the adversary does not use a set fault for $k_{0,E}^j$ in Step $1iii$) during the the j^{th} encryption query 3) there exists a previous ideal cipher query on input $(k_{0,E}^{j,\dagger}, p_A)$ (where $k_{0,E}^{j,\dagger}$ is how $k_{0,E}^{j}$ is modified according to the fault Fa^j for the $1iii)$ step).

Transition between Game 11 and Game 12. The two games are the same except if even Bad_8 happens. The previous transition has assured that $k_{0,E}^{j}$ is random, if the condition 1) of event Bad_8 happens. Since A is not setting both k_0^j in Step $1i)$ and $k_{0,E}^j$ in Step $1iii)$ via a fault, thus, $k_{0,E}^{j,\dagger}$ is random, where $k_{0,E}^{j,\dagger} = k_{0,E}^{j} \oplus \Delta$ with $k_{0,E}^{j}$ randomly picked and Δ, the offset specified by the fault Fa^j which is chosen before $k_{0,E}^{j}$ is picked.

Similar to what we have done before, we observe that during an encryption query there at most $L+4$ queries to $\mathsf{E}(\cdot, p_B)$ with at most $L+4$ different keys (we have an additional query due to Game 10), and $2L+1$ queries with $(\cdot,\cdot)$ where the key is either $k_{\iota,E}$ or $k_{\iota,A}$ for a certain ι and the input is a message m_ι (which can be equal to p_B), while during a decryption query there at most $L+3$ different queries to the ideal block-cipher with input $\mathsf{E}(\cdot, p_A)$ and $2L+1$ to $\mathsf{E}(\cdot,\cdot)$. In addition, due to the change we have introduced in Game 4, there are two additional queries, one of which is to $\mathsf{E}(\cdot, p_B)$ The adversary can do at most q_I queries of its choice to the ideal block-cipher. Thus, during the j^{th} encryption query there at most $(j-1)(3L+5) + q_D(3L+4) + q_I$ different keys that, if picked, would trigger the Bad_8 event. Thus,

$$\Pr[\mathsf{Bad}_8] \leq \sum_{j=1}^{q_E} \frac{(j-1)(3L+5) + q_D(3L+4) + q_I}{2^n} \leq$$

$$[q_E(3L+5)(q_E-1)/2 + q_E q_I + q_E q_D(3L+4)]2^{-n} =$$

$$q_E[(3L+4)[q_D + (q_E-1)/2] + (q_E-1)/2 + q_I]2^{-n}.$$

$$\text{Thus, } |\Pr[E_{11}] - \Pr[E_{12}]| \leq \Pr[\mathsf{Bad}_8] =$$

$$q_E[(3L+4)[q_D + (q_E-1)/2] + (q_E-1)/2 + q_I]2^{-n}.$$

Game 13. It is Game 12, where the following event Bad_9 does not happen: the adversarial output $(a^*, (c_0^*, \ldots, c_{l^*+2}^*))$ is fresh and valid (thus, we call it a forgery), and let (h^*, k_0^*, c_{l+2}^*) be the triple associated to the forgery, (that is, the inputs and output of the call to g used in the decryption), there exists an encryption query, the j^{th} for which all these conditions hold: 1) (h^*, k_0^*, c_{l+2}^*) is the triple associated to the call to f done during the j^{th} encryption query, 2) the adversary sets k_0^* in the j^{th} encryption query (that is, she modifies k_0^j in k_0^* with a set fault) for the computation of step $4ii)$, 3) h^* is the actual output of the H atom of the j^{th} encryption query, 4) $((c_0^* \ldots, c_{l+1}^*, a^*)$ is the actual input of the H atom the j^{th} encryption query.

Transition between Game 12 and Game 13. Since Game 12 and 13 are the same except if event Bad_9 happens, to bound the difference between these two games, we need to bound Bad_9. Let $\mathsf{Bad}_{9.j}$ be the event that Bad_9 happens and the 4 conditions holds for the jth encryption query. Clearly $\mathsf{Bad}_9 = \underset{j=1}{\overset{\cup}{q_E}} \mathsf{Bad}_{9.j}$. Now, we bound $\Pr[\mathsf{Bad}_{9.j}]$. Since the ciphertext is valid, this means that $\mathsf{E}_{k_{0,E}^*}(p_B) = c_0^* = \mathsf{E}_{k_{0,E}^{j,\dagger}}(p_B)$, where $k_{0,E}^* = \mathsf{E}_{k_0^*}(p_A)$, $k_{0.E}^{j,\dagger}$ is how $k_{0,E}^j$ is modified according to the fault Fa^j for the $1iii)$ step, and $k_{0,E}^j = \mathsf{E}_{k_0^{j,\dagger}}(p_A)$ (with $k_0^{j,\dagger}$ is how k_0^j is modified according to the fault Fa^j for the $1i)$ step. Now, due to the fact that events Bad_7 and Bad_8 do not happen, then, $k_{0,E}^{j,\dagger}$ is random, and also $c_{0,E}^j$ is random. Moreover, the actual input of the H atom (step $4i)$) of the j^{th} encryption uses $c_{0,E}^j$ and can only modify with a fault. Thus, in step $4i)$ we use $c_{0,E}^{j,\dagger} = c_{0,E}^j \oplus \Delta$ with Δ the offset specified by the fault Fa^j (which is chosen before all k_0^j, $k_{0,E}^j$, $c_{0,E}^j$ are picked. So, the probability that $c_{0,E}^{j,\dagger} = \mathsf{E}_{k_{0,E}^*}(p_B)$ is 2^{-n} which gives the bound for event Bad_9:

$$\Pr[\mathsf{Bad}_9] \le \sum_{j=1}^{q_E} \Pr[\mathsf{Bad}_{9.j}] = q_E 2^{-n}.$$

Game 14. It is Game 13, where the following event Bad_{10} does not happen: the adversarial output $(a^*, (c_0^*, \dots, c_{l^*+2}^*))$ is fresh and valid (thus, we call it a forgery), and let (h^*, k_0^*, c_{l+2}^*) be the triple associated to the forgery, (that is, the inputs and output of the call to g used in the decryption), there exists an encryption query, the j^{th} for which all these conditions hold: 1) (h^*, k_0^*, c_{l+2}^*) is the triple associated to the call to f done during the j^{th} encryption query, 2) the adversary sets k_0^j to k_0^* with fault Fa^j in the j^{th} encryption query for Step $4ii)$, 3) h^* is the actual output of the H atom (Step $4i)$) in the j^{th} encryption query, 4) $((c_0^* \dots, c_{l+1}^*, a^*)$ is not the actual input of the H atom (Step $4i)$) in the j^{th} encryption query.

Transition between Game 13 and Game 14. Since Game 13 and 14 are the same except if event Bad_{10} happens, to bound the difference between these two games, we need to bound Bad_{10}. Let $\mathsf{Bad}_{10.j}$ be the event that Bad_{10} happens and the 4 conditions holds for the jth encryption query. Clearly $\mathsf{Bad}_{10} = \bigcup_{j=1}^{q_E} \mathsf{Bad}_{10.j}$. Now, we bound $\Pr[\mathsf{Bad}_{10.j}]$. We observe that in the j^{th}, it is computed $h^j = \mathsf{H}_s(c_0^{j,\dagger}\| \dots \|c_{l_j+1}^{j,\dagger}, a^j)$ (where $c_i^{j,\dagger}$ is how c_i^j is modified according to fault Fa^j for step $4i)$, while for the forgery, it is computed $h^* = \mathsf{H}_s(c_0^*\| \dots c_{l^*+1}^*, a^*)$. Due to condition 3) $h^j = h^*$, while due to the condition 4) $(c_0^{j,\dagger}\| \dots \|c_{l_j+1}^{j,\dagger}, a^j) \neq ((c_0^* \dots, c_{l+1}^*, a^*)$, thus, a collision for the hash function has occurred, which is impossible since we have assumed that event Bad_1 (that is, no collision), thus $\Pr[\mathsf{Bad}_{10.j}] = 0$ and

$$\Pr[\mathsf{Bad}_{10}] = \sum_{j=1}^{q_E} \Pr[\mathsf{Bad}_{10.j}] = 0. \text{ So, } \Pr[E_{13}] = \Pr[E_{14}].$$

Game 15. It is Game 14, where the following event Bad_{11} does not happen: the adversarial output $(a^*, (c_0^*, \dots, c_{l+2}^*))$ is fresh and valid (thus, we call it a forgery), and let (h^*, k_0^*, c_{l+2}^*) be the triple associated to the forgery, (that is, the inputs and output of the call to g used in the decryption), there exists an encryption query, the j^{th}, for which all these conditions hold: 1) (h^*, k_0^*, c_{l+2}^*) is the triple associated to the call to f done during the j^{th} encryption query, 2) the adversary sets k_0^j to k_0^* with fault Fa^j in the j^{th} encryption query for Step $4ii)$, 3) h^* is not the actual output of the H atom (Step $4i)$) in the j^{th} encryption query.

Transition between Game 14 and Game 15. Since Game 14 and 15 are the same except if event Bad_{11} happens, to bound the difference between these two games, we need to bound Bad_{11}. Let $\mathsf{Bad}_{11.\iota}$ be the event that Bad_{11} happens and the 3 conditions holds for the ιth encryption query. Clearly $\mathsf{Bad}_{11} = \bigcup_{\iota=1}^{q_E} \mathsf{Bad}_{11.\iota}$. Now, we bound $\Pr[\mathsf{Bad}_{11.\iota}]$. For this, we build a t_2-PR-coirv adversary EE^ι.

The PR-coirv-adversary EE^ι. At the start of the game, EE^ι receives a key for the hash function s in $\mathcal{HK}$ and gives it to A. Moreover, she picks two random tweakable functions f and g as discussed before.

When A does the i^{th} encryption query, $i < \iota$, on input $(a^i, m^i, \mathsf{Fa}^i)$, EE_1^ι proceeds as C with the following exceptions: after step $4iii)$ she computes $k_{l+3}^i = \mathsf{E}_{k_0^{i,\dagger}}(p_A)$ where $k_0^{i,\dagger}$ is how k_0^i is modified in step $4ii)$ according to fault Fa^i, and $4iv)$ $c_{l+3} = \mathsf{E}_{k_{l+3}}(p_B)$. For this, EE_1^ι needs time $t_{\mathsf{H}} + [4(l+1)+4]t_{\mathsf{E}} \le t_{\mathsf{H}} + [4(L+2)]t_{\mathsf{E}}$.

When A does the ι encryption query on input $(a^\iota, c^\iota, \mathsf{Fa}^\iota)$, EE_1^ι proceeds as follows: she outputs $(\mathsf{st}, \mathcal{D}, \Delta)$ where $\mathsf{st} = (s, \mathcal{H}, \mathcal{L}, \mathcal{D})$, Δ is the offset specified by the fault Fa^ι for step $4ii)$ for the h input (since we are considering event $\mathsf{Bad}_{11.\iota}$ it means that the adversary has already been spent her only set fault for the k_0^ι for step $4ii$), and she uses a fault here, thus, it is a differential fault, and we call Δ the difference XORed to h in this computation, and $\mathcal{D}$ is this distribution for the input of the hash function: 1) pick $k_0^\iota \xleftarrow{\$} \{0,1\}^n$ $1i$) compute $k_{0,E} = \mathsf{E}_{k_0^{\iota,\dagger}}(p_B)$, where $k_0^{\iota,\dagger}$ is how k_0^ι is modified according to the fault Fa^ι, for this computation, $1ii$) compute $k_{0,A}^\iota = \mathsf{E}_{k_0^{\iota,\ddagger}}(p_B)$, where $k_0^{\iota,\ddagger}$ is how k_0 is modified according to the fault Fa^ι for this computation, $1iii$) compute $c_0 = \mathsf{E}_{k_{0,E}^{\iota,\dagger}}(p_A)$, where $k_{0,E}^{\iota,\dagger}$ is how $k_{0,E}^\iota$ is modified according to the fault Fa^ι for this computation, and $1iv$) compute $k_1 = \mathsf{E}_{k_{0,E}^{\iota,\ddagger}}(p_B)$, where $k_{0,E}^{\iota,\ddagger}$ is how $k_{0,E}^\iota$ is modified according to the fault Fa^ι for this computation. Then, for all $i = 1, \dots, l^\iota$, $2i$) compute $k_{i,E}^\iota = \mathsf{E}_{k_i^{\iota,\dagger}}(p_B)$, where $k_i^{\iota,\dagger}$ is how k_i^ι is modified according to the fault Fa^ι for this computation, $2ii$) compute $c_i^\iota = \mathsf{E}_{k_{i,E}^{\iota,\ddagger}}(m_i^{\iota,\dagger})$, where $k_{i,E}^{\iota,\ddagger}$ and $m_i^{\iota,\dagger}$ are how $k_{i,E}^\iota$ and m_i^ι are modified according to the fault Fa^ι for this computation, $2iii$) compute $k_{i+1}^\iota = \mathsf{E}_{k_i^{\iota,\ddagger}}(p_A)$, where $k_i^{\iota,\ddagger}$ is how k_i^ι is modified according to the fault Fa^ι for this computation, and $2iv$) compute $k_{i,A}^\iota = \mathsf{E}_{k_{i-1,A}^{\iota,\dagger}}(m_i^{\iota,\ddagger})$, where $k_{i-1,A}^{\iota,\dagger}$ and $m_i^{\iota,\ddagger}$ are how $k_{i-1,A}^\iota$ and m_i^ι are modified according to the fault Fa^ι for this computation. After that, $3i$) compute $k_{l^\iota+1,E}^\iota = \mathsf{E}_{k_{l^\iota+1}^{\iota,\dagger}}(p_B)$, where $k_{l^\iota+1}^{\iota,\dagger}$ is how $k_{l^\iota+1}$ is modified according to the fault Fa^ι for this computation, and $3ii$) compute $c_{l^\iota+1}^\iota = \mathsf{E}_{k_{l^\iota+1,E}^{\iota,\dagger}}(k_{l^\iota,A}^{\iota,\dagger})$, where $k_{l^\iota+1,E}^{\iota,\dagger}$ and $k_{l^\iota,A}^{\iota,\dagger}$ are how $k_{l^\iota+1,E}^\iota$ and $k_{l^\iota,A}^\iota$ are modified according to the fault Fa^ι for this computation. The inputs of H are $(c_0^{\iota,\dagger}\| \dots \|c_{l^{\iota,\dagger}+1}, a^\iota)$, where $c_i^{\iota,\dagger}$ is how c_i^ι is modified according to fault Fa^ι for step $4i$).

Now, EE_2^ι receiving st, samples $(c_0^\iota\| \dots \|c_{l^\iota+1}, a^\iota)$ according to the distribution $\mathcal{D}$, then she computes $4i$) $h^\iota = \mathsf{H}_s(c_0^\iota\| \dots \|c_{l^\iota+1}, a^\iota)$, where $c_i^{\iota,\dagger}$ is how c_i^ι is modified according to fault Fa^ι for this computation, $4ii$) $c_{l^\iota+2}^\iota = \mathsf{f}(h^{\iota,\dagger}, k_0^{\iota,\dagger\dagger})$, where $h_i^{\iota,\dagger}$ and $k_0^{\iota,\dagger\dagger}$ are how h_i^ι and k_0^ι are modified according to fault Fa^ι for this computation (note that if Event $\mathsf{Bad}.\iota_{11}$ happens, by the condition 2) the adversary sets k_0^ι to k_0^*). $4iii$) she computes $k_{l^\iota+3}^\iota = \mathsf{E}_{k_0^{\iota,\dagger\dagger}}(p_A)$ where $k_0^{\iota,\dagger\dagger}$ is how k_0^ι is modified in step $4ii$) according to fault Fa^ι, and $4iv$) $c_{l^\iota+3} = \mathsf{E}_{k_{l+3}^\iota}(p_B)$. For EE_2 $x = (c_0^\iota\| \dots \|c_{l^\iota+1}, a^\iota)$ and $h' = h_i^\iota = h^\iota \oplus \Delta$.

Thus, EE_2^ι needs time $t_\mathsf{H} + [4(l+1)+4]t_\mathsf{E} \leq t_\mathsf{H} + [4(L+2)]t_\mathsf{E}$.

When A does the i^{th} encryption query, $i > \iota$, on input $(a^i, m^i, \mathsf{Fa}^i)$, EE_2^ι proceeds as EE_1^ι for the j^{th} encryption query with $j < \iota$. Thus, EE_2^ι needs time $t_\mathsf{H} + [4(l+1)+4]t_\mathsf{E} \leq t_\mathsf{H} + [4(L+2)]t_\mathsf{E}$.

When A does the ι decryption query on input (a, c, Fa), EE^ι (we use EE^ι) to identify both EE_1^ι and EE_2^ι) behaves as C with the following exceptions: after step $1ii$), before doing step $2i$, EE^ι computes $1iii$) $k_{l^\iota+3}^\iota = \mathsf{E}_{k_0^\iota}(p_A)$, and $1iv$) $c_{l^\iota+3}^\iota = \mathsf{E}_{k_{l^\iota+3}}(p_B)$.

Thus, EE^ι needs time $t_\mathsf{H} + [4(l+1)+4]t_\mathsf{E} \leq t_\mathsf{H} + [4(L+2)]t_\mathsf{E}$.

When A outputs her output (a^*, c^*), EE_2^ι proceeds as follows: she computes $h^* = \mathsf{H}'_s(c_0^*\| \dots \|c_{l+1}^*, a^*)$ and she adds $((c_0^*\| \dots \|c_{l+1}^*, a^*), h)$ to $\mathcal{H}$, and then, she looks through $\mathcal{H}$ to see if there is an entry $(c_0\| \dots \|c_{l+1}, a), h)$ s.t $h = h'$. If it is the case, she outputs it, otherwise 0^n. This takes time t_H.

Thus, in total EE runs in time bounded by

$$t + q_E[t_\mathsf{H} + (4(L+2))t_\mathsf{E}] + (q_D)[t_\mathsf{H} + (4(L+2))t_\mathsf{E}] + t_\mathsf{H}$$

$$t + qt_\mathsf{H} + [(q-1)4(L+4)]t_\mathsf{E} \leq t_2.$$

Bounding $\Pr[\mathsf{Bad}_{11}]$. First, we observe that EE^ι simulates perfectly Game 14 for A. As already discussed, to bound $\Pr[\mathsf{Bad}_{11}]$, it is enough to bound $\Pr[\mathsf{Bad}_{11.\iota}$ $\forall \iota$. First, we observe that EE^ι samples correctly according to $\mathcal{D}$ the input for the hash function (in the $\mathsf{PR\text{-}coirv}$ game (Def. 11) it is irrelevant who samples the input for H, as long as, it is sampled according to $\mathcal{D}$). Moreover, due to the fact that if event $\mathsf{Bad}_{11.\iota}$ happens, in the ι encryption query the adversary A uses her set fault for k_0 in Step $4ii$), we can use Lemma 1 which assures that $\mathcal{D}$ has the property required by the $\mathsf{PR\text{-}coirv}$ definition (for the randomness). Clearly, if event $\mathsf{Bad}_{11.\iota}$ happens, the adversary has chosen an offset for h. Thus, if event $\mathsf{Bad}_{11.\iota}$ happens, then EE^{ι} can find a pre-image for h', where h' is obtained by adding an offset chosen in advance to the hash of a "random enough" input. Since H is a $(t_2, \epsilon_{\mathsf{PR\text{-}coirv}})$-PR-coirv-secure hash function and EE^{ι} is a t_2-adversary, then

$$\Pr[\mathsf{Bad}_{11.\iota}] \leq \epsilon_{\mathsf{PR\text{-}coirv}}, \text{ and}$$

$$|\Pr[E_{14}] - \Pr[E_{15}] \leq \Pr[\mathsf{Bad}_{11}] \leq \sum_{\iota=1}^{q_E} \Pr[\mathsf{Bad}_{11.\iota}] = q_E \epsilon_{\mathsf{PR\text{-}coirv}}.$$

Game 16. It is Game 15, where the following event Bad_{12} does not happen: the adversarial output $(a^*, (c_0^*, \dots, c_{l+2}^*))$ is fresh and valid (thus, we call it a forgery), and let (h^*, k_0^*, c_{l+2}^*) be the triple associated to the forgery, (that is, the inputs and output of the call to g used in the decryption), there exists an encryption query, the j^{th}, for which all these conditions hold: 1) (h^*, k_0^*, c_{l+2}^*) is the triple associated to the call to f done during the j^{th} encryption query, 2) the adversary sets h^j to h^* with fault Fa^j in the j^{th} encryption query for Step $4ii$).

Transition between Game 15 and Game 16. Since Game 15 and 16 are the same except if event Bad_{12} happens, to bound the difference between these two games, we need to bound Bad_{12}. Let $\mathsf{Bad}_{12.\iota}$ be the event that Bad_{12} happens and the 2 conditions holds for the ιth encryption query. Clearly $\mathsf{Bad}_{12} = \overset{\cup}{q_E}_{\iota=1} \mathsf{Bad}_{12.\iota}$. Now, we bound $\Pr[\mathsf{Bad}_{12.\iota}]$. For this, we build a t_3-PR-corpi adversary GG^{ι}.

The n-PR-corpi-adversary GG^{ι}. At the start of the game, GG^{ι} receives a key for the hash function s in $\mathcal{HK}$ and gives it to A. Moreover, she picks two random tweakable functions f and g as discussed before.

When A does the i^{th} encryption query, $i < \iota$, on input $(a^i, m^i, \mathsf{Fa}^i)$, GG_1^{ι} proceeds as C with the following exceptions: after step $4iii$) she computes $k_{l^i+3}^i = \mathsf{E}_{k_0^{i,\dagger}}(p_A)$ where $k_0^{i,\dagger}$ is how k_0^i is modified in step $4ii$) according to fault Fa^i, and $4iv$) $c_{l^i+3} = \mathsf{E}_{k_{l^i+3}}(p_B)$. For this, GG_1^{ι} needs time $t_{\mathsf{H}} + [4(l^i + 1) + 4]t_{\mathsf{E}} \leq t_{\mathsf{H}} + [4(L + 2)]t_{\mathsf{E}}$.

When A does the ι encryption query on input $(a^{\iota}, m^{\iota}, \mathsf{Fa}^{\iota})$, GG_1^{ι} proceeds as follows: from Fa^{ι} she sees if the adversary sets h in step $4ii$) to $h^{\iota,\dagger}$: if it is the case she outputs $h^{\iota,\dagger}$ as her target; in addition, she outputs $\mathsf{st} = (s, \mathcal{H}, \mathcal{L}, a^{\iota,}, m^{\iota}, \mathsf{Fa}^{\iota})$; otherwise, she aborts. GG_2^{ι} receiving st, she proceeds as C for a encryption query on input $(a^{\iota}, m^{\iota}, \mathsf{Fa}^{\iota})$ with the following exceptions: after step $4iii$) she computes $k_{l^{\iota}+3}^{\iota} = \mathsf{E}_{k_0^{\iota,\dagger}}(p_A)$ where $k_0^{i,\dagger}$ is how k_0^{ι} is modified in step $4ii$) according to fault Fa^i, and $4iv$) $c_{l^{\iota}+3} = \mathsf{E}_{k_{l^{\iota}+3}}(p_B)$. For this, GG_1^{ι} and GG_2^{ι} need in total time $t_{\mathsf{H}} + [4(l^{\iota} + 1) + 4]t_{\mathsf{E}} \leq t_{\mathsf{H}} + [4(L + 2)]t_{\mathsf{E}}$. Finally GG_2^{ι} takes $c_{l^{\iota}+3}^{\iota}$ as her random prefix for the pre-image of $h^{\iota,\dagger}$.

When A does the i^{th} encryption query, $i > \iota$, on input $(a^i, m^i, \mathsf{Fa}^i)$, GG_2^{ι} proceeds as C with the following exceptions: after step $4iii$) she computes $k_{l^i+3}^i = \mathsf{E}_{k_0^{i,\dagger}}(p_A)$ where $k_0^{i,\dagger}$ is how k_0^i is modified in step $4ii$) according to fault Fa^i, and $4iv$) $c_{l^i+3} = \mathsf{E}_{k_{l^i+3}}(p_B)$.

For this, GG_2^{ι} needs time $t_{\mathsf{H}} + [4(l^i + 1) + 4]t_{\mathsf{E}} \leq t_{\mathsf{H}} + [4(L + 2)]t_{\mathsf{E}}$.

When A does the ι decryption query on input (a, c, Fa), GG^{ι} (we use GG^{ι}) to identify both GG_1^{ι} and GG_2^{ι}) behaves as C with the following exceptions: after step $1ii$), before doing step $2i$, GG^{ι} computes $1iii$) $k_{l^{\iota}+3}^{\iota} = \mathsf{E}_{k_0^{\iota}}(p_A)$, and $1iv$) $c_{l^{\iota}+3}^{\iota} = \mathsf{E}_{k_{l^{\iota}+3}}(p_B)$.

Thus, GG^{ι} needs time $t_{\mathsf{H}} + [4(l + 1) + 4]t_{\mathsf{E}} \leq t_{\mathsf{H}} + [4(L + 2)]t_{\mathsf{E}}$.

When A outputs her output (a^*, c^*), GG_2^{ι} proceeds as follows: she computes $h^* = \mathsf{H}'_s(c_0^* \| \dots \| c_{l+1}^*, a^*)$ and she adds $((c_0^* \| \dots \| c_{l+1}^*, a^*), h)$ to $\mathcal{H}$, and then, she looks through $\mathcal{H}$ to see if there is an entry $(c_0 \| \dots \| c_{l+1}, a), h)$ s.t $h = h^{\iota,\dagger}$ and $c_0 = c_{l^{\iota}+3}^{\iota}$. If this is the case, she outputs it, otherwise 0^n. This takes time t_{H}.

Thus, in total GG runs in time bounded by

$$\begin{aligned} t + q_E[t_{\mathsf{H}} + (4(L + 2))t_{\mathsf{E}}] + (q_D)[t_{\mathsf{H}} + (4(L + 2))t_{\mathsf{E}}] + t_{\mathsf{H}} \\ t + qt_{\mathsf{H}} + [(q - 1)4(L + 4)]t_{\mathsf{E}} \leq t_3'. \end{aligned}$$

Bounding $\Pr[\mathsf{Bad}_{12}]$. First, we observe that GG^ι simulates perfectly Game 15 for A. Thus, it is equivalent to pick the random prefix as we do or uniformly at random from $\{0,1\}^*$. As already discussed, to bound $\Pr[\mathsf{Bad}_{12}]$, it is enough to bound $\Pr[\mathsf{Bad}_{12.\iota}\ \forall\iota$. First, if event $\mathsf{Bad}_{12.\iota}$ happens, in the ι encryption query the adversary A uses her set fault for h in Step $4ii)$ setting it in $h^{\iota,\dagger}$; moreover, we observe that $c^\iota_{l^\iota+3}$ is random due to the fact that we have assumed that both event Bad_7 and Bad_8 do not happen. Thus, if event $\mathsf{Bad}_{12.\iota}$ happens, then GG^ι can find a pre-image for $h^{\iota,\dagger}$, of her choice whose prefix is random, and it is equal to $c^\iota_{l^\iota+3}$. Since H is a $(t_3,\epsilon_{\mathsf{PR\text{-}corpi}})$-$\mathsf{PR\text{-}corpi}$-secure hash function and GG^ι is a t_3-adversary, then

$$\Pr[\mathsf{Bad}_{12.\iota}] \le \epsilon_{\mathsf{PR\text{-}corpi}}, \text{ and}$$

$$|\Pr[E_{15}] - \Pr[E_{16}] \le \Pr[\mathsf{Bad}_{12}] \le \sum_{\iota=1}^{q_E} \Pr[\mathsf{Bad}_{12.\iota}] = q_E \epsilon_{\mathsf{PR\text{-}corpi}}.$$

Game 17. It is Game 16, where the following event Bad_{13} does not happen: the adversarial output $(a^*,(c^*_0,\dots,c^*_{l+2}))$ is fresh and valid (thus, we call it a forgery), and let (h^*,k^*_0,c^*_{l+2}) be the triple associated to the forgery, (that is, the inputs and output of the call to g used in the decryption), there exists an encryption query, the j^{th}, for which all these conditions hold: 1) (h^*,k^*_0,c^*_{l+2}) is the triple associated to the call to f done during the j^{th} encryption query, 2) in step $4ii)$ of the j^{th} encryption query there is no set fault, 3) h^j the output of step $4i)$ is the h input for step $4ii)$, that is, there is no differential faultput applied to it. 4) in the input of step $4i)$, $c^{j,\dagger}_0 \neq c^j_{l^j+3}$, where $c^{j,\dagger}_0$ is how c_0 is modified according to fault Fa^j for step $4i)$ and $c^j_{l^j+3} = \mathsf{E}_{k^*_{0,E}}(p_B)$, with $k^*_{0,E} = \mathsf{E}_{k^*_0}(p_A)$.

Transition between Game 16 and Game 17. Since Game 16 and 17 are the same except if event Bad_{13} happens, to bound the difference between these two games, we need to bound Bad_{13}. Let $\mathsf{Bad}_{13.\iota}$ be the event that Bad_{13} happens and the 2 conditions holds for the ιth encryption query. Clearly $\mathsf{Bad}_{13} = \overset{\cup}{q_E}_{\iota=1} \mathsf{Bad}_{13.\iota}$. Now, we bound $\Pr[\mathsf{Bad}_{13.\iota}]$.

We observe that if event $\mathsf{Bad}_{13.\iota}$ happens, it means that for h^ι there are two pre-images, one computed in the j^{th} encryption query and the other in the forgery. These two pre-images are different because one uses $c^{j,\dagger}_0$, the other is associated with the forgery, thus, $c^*_0 = c^\iota_{l^\iota+3}$ because in the forgery we use k^*_0 (since we use the triple mentioned in the condition 1) of event Bad_{13}), thus, we need that $c^*_0 = c^\iota_{l^\iota+3}$ (due to how we have defined $c^\iota_{l^\iota+3}$ in Game 10). But, because event Bad_1 does not happen, it is impossible that there are two pre-images for the same value of the hash functions. Thus, $\Pr[\mathsf{Bad}_{13.\iota}] = 0$, and

$$|\Pr[E_{16}] - \Pr[E_{17}]| = \Pr[\mathsf{Bad}_{13}] \le \sum_{\iota=1}^{q_E} \Pr[\mathsf{Bad}_{13.\iota}] = 0.$$

Game 18. It is Game 17, where the following event Bad_{14} does not happen: the adversarial output $(a^*,(c^*_0,\dots,c^*_{l+2}))$ is fresh and valid (thus, we call it a forgery), and let (h^*,k^*_0,c^*_{l+2}) be the triple associated to the forgery, (that is, the inputs and output of the call to g used in the decryption), there exists an encryption query, the j^{th}, for which all these conditions hold: 1) (h^*,k^*_0,c^*_{l+2}) is the triple associated to the call to f done during the j^{th} encryption query, 2) in step $4ii)$ of the j^{th} encryption query there is no set fault, 3) h^j the output of step $4i)$ is the h input for step $4ii)$, that is, there is no differential fault applied to it. 4) in the input of step $4i)$, $c^{j,\dagger}_0 = c^j_{l^j+3}$, where $c^{j,\dagger}_0$ is how c_0 is modified according to fault Fa^j for step $4i)$ and $c^j_{l^j+3} = \mathsf{E}_{k^*_{0,E}}(p_B)$, with $k^*_{0,E} = \mathsf{E}_{k^*_0}(p_A)$ (so $c^{j,\dagger}_0 = c^*_0$), 5) the input for Step $4i)$ in the j^{th} encryption query, $(c^{j,\dagger}_0 \| \dots \| c^{j,\dagger}_{l^j+1}, a^j)$, where $c^{j,\dagger}_i$ is how c^j_i is modified according to the fault Fa^j for step $4i)$, is a " valid encryption" for k^*_0, that is, there exists a message $m^{j,\ddagger}$ s.t. given k^*_0 and $m^{j,\ddagger}$ applying all the encryption steps from $1i)$ till $3ii)$ without any fault, the result is $c^{j,\dagger}_0,\dots,c^{j,\dagger}_{l^j+1}$.

Transition between Game 17 and Game 18. Since Game 17 and 18 are the same except if event Bad_{14} happens, to bound the difference between these two games, we need to bound Bad_{14}. Let $\mathsf{Bad}_{14.\iota}$ be the event that Bad_{14} happens and the 5 conditions holds for the ιth encryption query. Clearly $\mathsf{Bad}_{14} = \overset{\cup}{\underset{\iota=1}{q_E}} \mathsf{Bad}_{14.\iota}$. Now, we bound $\Pr[\mathsf{Bad}_{14.\iota}]$. To bound it, we divide event $\mathsf{Bad}_{14.\iota}$ in two:

- $\mathsf{Bad}_{14.\iota.1}$: the adversary injects any effective fault (that is, any fault which modifies the output) in the j^{th} encryption query.
- $\mathsf{Bad}_{14.\iota.2}$: the adversary injects no effective fault in the j^{th} encryption query.

Bounding $\Pr[\mathsf{Bad}_{14.\iota.1}]$**.** We have two possibilities: the adversary has injected a fault in step $4i$) or in step $1ii$) or not. If this is the case, since due to events Bad_6 and Bad_7 not happening $c^j_{l^j+3}$ is random, thus, the probability that $c_0^{j,\dagger} = c^j_{l^j+3}$ is $\leq 2^{-n}$ if there is an effective fault in step $1i$) or $1ii$) or $4ii$) (due to conditions 2 and 3 of event Bad_{14}, in step $4ii$), A can only inject a differential fault the k_0 input). First, we assume that the adversary injects a fault in step $1i$), or in the k_0 input of step $4ii$): thus, if the fault is effective, that is, step $1i$) is not $\mathsf{E}_{k_0^*}(p_A)$, the probability that the output of step $1i$) is k_{l^j+3} is 2^{-n}. Again, if in step $1ii$), we do not inject an effective fault (or we have put an effective fault in $1i$) not obtaining $k^j_{l^j+3}$, if we are not computing $\mathsf{E}_{k_{l^j+3}}(p_B)$, the probability that c_0^*, which is computed after due to event Bad_7 not happening, is equal to this output is c_0^* is 2^{-n}. Now, we can use Lemma 1, to cover the case that the adversary has not injected a fault in Step $1i$), $1ii$) and $4i$). Thus, we can observe that the last block is uniformly random, thus, the probability it is the correct one is 2^{-n}, because there is only one correct [27]. Thus, $\Pr[\mathsf{Bad}_{14.\iota.1}] \leq 3/2^n$.

Bounding $\Pr[\mathsf{Bad}_{14.\iota.2}]$**.** Due to the condition 1) of event Bad_{14}, in the forgery we use the triple (h^*, k_0^*, c^*_{l+2}) associated to the j^{th} encryption query. Moreover, since we consider event $\mathsf{Bad}_{14.\iota.2}$, in the j^{th} encryption query there are no faults. Thus, the adversary has to find another pre-image for h^j, which means that the adversary has found a collision for the hash function. But this is impossible because we have assumed that event Bad_1 does not happen. Thus, $\Pr[\mathsf{Bad}_{14.\iota.2}] = 0$ and

$$\Pr[\mathsf{Bad}_{14.\iota}] \leq \Pr[\mathsf{Bad}_{14.\iota.1}] + \Pr[\mathsf{Bad}_{14.\iota.2}] = 3/2^{-n}.$$

Finally, we can bound

$$|\Pr[E_{17}] - \Pr[E_{18}]| = \Pr[\mathsf{Bad}_{14}] \leq \sum_{\iota=1}^{q_E} \Pr[\mathsf{Bad}_{14.\iota}] = 3q_E 2^{-n}.$$

Game 19. It is Game 18, where the following event Bad_{15} does not happen: the adversarial output $(a^*, (c_0^*, \dots, c^*_{l+2}))$ is fresh and valid (thus, we call it a forgery), and let (h^*, k_0^*, c^*_{l+2}) be the triple associated to the forgery, (that is, the inputs and output of the call to g used in the decryption), there exists an encryption query, the j^{th}, for which all these conditions hold: 1) (h^*, k_0^*, c^*_{l+2}) is the triple associated to the call to f done during the j^{th} encryption query, 2) in step $4ii$) of the j^{th} encryption query there is no set fault, 3) h^j the output of step $4i$) is the h input for step $4ii$), that is, there is no differential fault applied to it. 4) in the input of step $4i$), $c_0^{j,\dagger} = c^j_{l^j+3}$, where $c_0^{j,\dagger}$ is how c_0 is modified according to fault Fa^j for step $4i$) and $c^j_{l^j+3} = \mathsf{E}_{k^*_{0,E}}(p_B)$, with $k^*_{0,E} = \mathsf{E}_{k_0^*}(p_A)$ (so $c_0^{j,\dagger} = c_0^*$), 5) the input for Step $4i$) in the j^{th} encryption query, $(c_0^{j,\dagger}\| \dots \|c^{j,\dagger}_{l^j+1}, a^j)$, where $c_i^{j,\dagger}$ is how c_i^j is modified according to the fault Fa^j for step $4i$), is not " valid encryption" for k_0^*, that is, there exists no message $m^{j,\ddagger}$ s.t. given k_0^* and $m^{j,\ddagger}$ applying all the encryption steps from $1i$) till $3ii$) without any fault, the result is $c_0^{j,\dagger}, \dots, c^{j,\dagger}_{l^j+1}$.

[27]For $\mathsf{CONCRETE}$ any ciphertext with the correct c_0 is valid. This is one reason why $\mathsf{CONCRETE2}$ achieves this security, differently from $\mathsf{CONCRETE}$.

Transition between Game 18 and Game 19. Since Game 18 and 19 are the same except if event Bad_{15} happens, to bound the difference between these two games, we need to bound Bad_{15}. Let $\mathsf{Bad}_{15.\iota}$ be the event that Bad_{15} happens and the 5 conditions holds for the ιth encryption query. Clearly $\mathsf{Bad}_{15} = \bigcup_{\iota=1}^{q_E} \mathsf{Bad}_{15.\iota}$. Now, we bound $\Pr[\mathsf{Bad}_{15.\iota}]$.

Due to the condition 1) of event Bad_{15}, in the forgery we use the triple (h^*, k_0^*, c_{l+2}^*) associated to the j^{th} encryption query. Moreover, since we consider event Bad_{15}, in the j^{th} the input of the hash does not correspond to a "valid" encryption for k_0^*. Thus, the adversary has to find another pre-image for h^j, which means that the adversary has found a collision for the hash function. But this is impossible because we have assumed that event Bad_1 does not happen. Thus, $\Pr[\mathsf{Bad}_{15.\iota}] = 0$ and

$$|\Pr[E_{18}] - \Pr[E_{19}]| = \Pr[\mathsf{Bad}_{15}] \leq \sum_{\iota=1}^{q_E} \Pr[\mathsf{Bad}_{15.\iota}] = 0.$$

Game 20. It is Game 19, where the following event Bad_{16} does not happen: the adversarial output $(a^*, (c_0^*, \dots, c_{l+2}^*))$ is fresh and valid (thus, we call it a forgery), and let (h^*, k_0^*, c_{l+2}^*) be the triple associated to the forgery, (that is, the inputs and output of the call to g used in the decryption), there exists an encryption query, the j^{th}, for which all these conditions hold: 1) (h^*, k_0^*, c_{l+2}^*) is the triple associated to the call to f done during the j^{th} encryption query, 2) in step $4ii)$ of the j^{th} encryption query there is no set fault, 3) h^j, the output of step $4i)$, is not the h input for step $4ii)$. Note that we have ruled out the possibility that A injects a set fault in $4ii)$ because this case was covered in Game 16, thus, the only possibility is that A injects a differential fault in h in step $4ii)$.

Transition between Game 19 and Game 20. Since Game 19 and 20 are the same except if event Bad_{16} happens, to bound the difference between these two games, we need to bound Bad_{16}. Let $\mathsf{Bad}_{16.\iota}$ be the event that Bad_{16} happens and the 3 conditions holds for the ι^{th} encryption query. Clearly $\mathsf{Bad}_{16} = \bigcup_{\iota=1}^{q_E} \mathsf{Bad}_{16.\iota}$. Now, we bound $\Pr[\mathsf{Bad}_{16.\iota}]$. For this, we build a t_2-$\mathsf{PR\text{-}coirv}$ adversary HH^ι.

The $\mathsf{PR\text{-}coirv}$-adversary HH^ι. At the start of the game, HH^ι receives a key for the hash function s in $\mathcal{HK}$ and gives it to A. Moreover, she picks two random tweakable functions f and g as discussed before.

When A does the i^{th} encryption query, $i < \iota$, on input $(a^i, m^i, \mathsf{Fa}^i)$, HH_1^ι proceeds as C with the following exceptions: after step $4iii)$ she computes $k_{l+3}^i = \mathsf{E}_{k_0^{i,\dagger}}(p_A)$ where $k_0^{i,\dagger}$ is how k_0^i is modified in step $4ii)$ according to fault Fa^i, and $4iv)$ $c_{l+3} = \mathsf{E}_{k_{l+3}}(p_B)$. For this, HH_1^ι needs time $t_\mathsf{H} + [4(l+1)+4]t_\mathsf{E} \leq t_\mathsf{H} + [4(L+2)]t_\mathsf{E}$.

When A does the ι encryption query on input $(a^\iota, c^\iota, \mathsf{Fa}^\iota)$, HH_1^ι proceeds as follows: she outputs $(\mathsf{st}, \mathcal{D}, \Delta)$ where $\mathsf{st} = (s, \mathcal{H}, \mathcal{L}, \mathcal{D})$, Δ is the offset specified by the fault Fa^ι for step $4ii)$ for the h input (since we are considering event $\mathsf{Bad}_{16.\iota}$ it means that in step $4ii)$ the adversary uses no set fault, and she uses a differential fault for the h input. Let Δ be the difference XORed to h in this computation, and $\mathcal{D}$ is this distribution for the input of the hash function: 1) pick $k_0^\iota \xleftarrow{\$} \{0,1\}^n$ $1i)$ compute $k_{0,E} = \mathsf{E}_{k_0^{\iota,\dagger}}(p_B)$, where $k_0^{\iota,\dagger}$ is how k_0^ι is modified according to the fault Fa^ι, for this computation, $1ii)$ compute $k_{0,A}^\iota = \mathsf{E}_{k_0^{\iota,\ddagger}}(p_B)$, where $k_0^{\iota,\ddagger}$ is how k_0 is modified according to the fault Fa^ι for this computation, $1iii)$ compute $c_0 = \mathsf{E}_{k_{0,E}^{\iota,\dagger}}(p_A)$, where $k_{0,E}^{\iota,\dagger}$ is how $k_{0,E}^\iota$ is modified according to the fault Fa^ι for this computation, and $1iv)$ compute $k_1 = \mathsf{E}_{k_{0,E}^{\iota,\ddagger}}(p_B)$, where $k_{0,E}^{\iota,\ddagger}$ is how $k_{0,E}^\iota$ is modified according to the fault Fa^ι for this computation. Then, for all $i = 1, \dots, l^\iota$, $2i)$ compute $k_{i,E}^\iota = \mathsf{E}_{k_i^{\iota,\dagger}}(p_B)$, where $k_i^{\iota,\dagger}$ is how k_i^ι is modified according to the fault Fa^ι for this computation, $2ii)$ compute $c_i^\iota = \mathsf{E}_{k_{i,E}^{\iota,\ddagger}}(m_i^{\iota,\dagger})$, where $k_{i,E}^{\iota,\ddagger}$ and $m_i^{\iota,\dagger}$

are how $k^{\iota}_{i,E}$ and m^{ι}_i are modified according to the fault Fa^{ι} for this computation, $2iii$) compute $k^{\iota}_{i+1} = \mathsf{E}_{k^{\iota,\ddagger}_i}(p_A)$, where $k^{\iota,\ddagger}_i$ is how k^{ι}_i is modified according to the fault Fa^{ι} for this computation, and $2iv$) compute $k^{\iota}_{i,A} = \mathsf{E}_{k^{\iota,\dagger}_{i-1,A}}(m^{\iota,\ddagger}_i)$, where $k^{\iota,\dagger}_{i-1,A}$ and $m^{\iota,\ddagger}_i$ are how $k^{\iota}_{i-1,A}$ and m^{ι}_i are modified according to the fault Fa^{ι} for this computation. After that, $3i$) compute $k^{\iota}_{l^{\iota}+1,E} = \mathsf{E}_{k^{\iota,\dagger}_{l^{\iota}+1}}(p_B)$, where $k^{\iota,\dagger}_{l^{\iota}+1}$ is how $k_{l^{\iota}+1}$ is modified according to the fault Fa^{ι} for this computation, and $3ii$) compute $c^{\iota}_{l^{\iota}+1} = \mathsf{E}_{k^{\iota,\dagger}_{l^{\iota}+1,E}}(k^{\iota,\dagger}_{l^{\iota},A})$, where $k^{\iota,\dagger}_{l^{\iota}+1,E}$ and $k^{\iota,\dagger}_{l^{\iota},A}$ are how $k^{\iota}_{l^{\iota}+1,E}$ and $k^{\iota}_{l^{\iota},A}$ are modified according to the fault Fa^{ι} for this computation. The inputs of H are $(c^{\iota,\dagger}_0\|\dots\|c_{l^{\iota,\dagger}+1}, a^{\iota})$, where $c^{\iota,\dagger}_i$ is how c^{ι}_i is modified according to fault Fa^{ι} for step $4i$).

Now, HH^{ι}_2 receiving st, samples $(c^{\iota}_0\|\dots\|c_{l^{\iota}+1}, a^{\iota})$ according to the distribution $\mathcal{D}$, then she computes $4i$) $h^{\iota} = \mathsf{H}_s(c^{\iota}_0\|\dots\|c_{l^{\iota}+1}, a^{\iota})$, where $c^{\iota,\dagger}_i$ is how c^{ι}_i is modified according to fault Fa^{ι} for this computation, $4ii$) $c^{\iota}_{l^{\iota}+2} = \mathsf{f}(h^{\iota,\dagger}, k^{\iota,\dagger\dagger}_0)$, where $h^{\iota,\dagger}_i$ and $k^{\iota,\dagger\dagger}_0$ are how h^{ι}_i and k^{ι}_0 are modified according to fault Fa^{ι} for this computation (note that if Event $\mathsf{Bad}.\iota_{11}$ happens, by the condition 2) the adversary sets k^{ι}_0 to k^*_0). $4iii$) she computes $k^{\iota}_{l^{\iota}+3} = \mathsf{E}_{k^{\iota,\dagger\dagger}_0}(p_A)$ where $k^{\iota,\dagger\dagger}_0$ is how k^{ι}_0 is modified in step $4ii$) according to fault Fa^{ι}, and $4iv$) $c_{l^{\iota}+3} = \mathsf{E}_{k^{\iota}_{l+3}}(p_B)$. For HH_2 $x = (c^{\iota}_0\|\dots\|c_{l^{\iota}+1}, a^{\iota})$ and $h' = h^{\iota}_i = h^{\iota} \oplus \Delta$.

Thus, HH^{ι}_2 needs time $t_{\mathsf{H}} + [4(l+1)+4]t_{\mathsf{E}} \leq t_{\mathsf{H}} + [4(L+2)]t_{\mathsf{E}}$.

When A does the i^{th} encryption query, $i > \iota$, on input $(a^i, m^i, \mathsf{Fa}^i)$, HH^{ι}_2 proceeds as HH^{ι}_1 for the j^{th} encryption query with $j < \iota$. Thus, HH^{ι}_2 needs time $t_{\mathsf{H}} + [4(l+1)+4]t_{\mathsf{E}} \leq t_{\mathsf{H}} + [4(L+2)]t_{\mathsf{E}}$.

When A does the ι decryption query on input (a, c, Fa), HH^{ι} (we use HH^{ι}) to identify both HH^{ι}_1 and HH^{ι}_2) behaves as C with the following exceptions: after step $1ii$), before doing step $2i$, HH^{ι} computes $1iii$) $k^{\iota}_{l^{\iota}+3} = \mathsf{E}_{k^{\iota}_0}(p_A)$, and $1iv$) $c^{\iota}_{l^{\iota}+3} = \mathsf{E}_{k_{l^{\iota}+3}}(p_B)$.

Thus, HH^{ι} needs time $t_{\mathsf{H}} + [4(l+1)+4]t_{\mathsf{E}} \leq t_{\mathsf{H}} + [4(L+2)]t_{\mathsf{E}}$.

When A outputs her output (a^*, c^*), HH^{ι}_2 proceeds as follows: she computes $h^* = \mathsf{H}'_s(c^*_0\|\dots\|c^*_{l+1}, a^*)$ and she adds $((c^*_0\|\dots\|c^*_{l+1}, a^*), h)$ to $\mathcal{H}$, and then, she looks through $\mathcal{H}$ to see if there is an entry $(c_0\|\dots\|c_{l+1}, a), h)$ s.t $h = h'$. If this is the case, she outputs it, otherwise 0^n. This takes time t_{H}.

Thus, in total HH runs in time bounded by

$$t + q_E[t_{\mathsf{H}} + (4(L+2))t_{\mathsf{E}}] + (q_D)[t_{\mathsf{H}} + (4(L+2))t_{\mathsf{E}}] + t_{\mathsf{H}}$$
$$t + qt_{\mathsf{H}} + [(q-1)4(L+4)]t_{\mathsf{E}} \leq t_2.$$

Bounding $\Pr[\mathsf{Bad}_{16}]$. First, we observe that HH^{ι} simulates perfectly Game 19 for A. As already discussed, to bound $\Pr[\mathsf{Bad}_{16}]$, it is enough to bound $\Pr[\mathsf{Bad}_{11.\iota}$ $\forall \iota$. First, we observe that HH^{ι} samples correctly according to $\mathcal{D}$ the input for the hash function (in the $\mathsf{PR\text{-}coirv}$ game (Def. 11) it is irrelevant who samples the input for H, as long as, it is sampled according to $\mathcal{D}$). Moreover, due to the fact that if event $\mathsf{Bad}_{16.\iota}$ happens, in the ι encryption query the adversary A uses her set fault for k_0 in Step $4ii$), we can use Lemma 1 which assures that $\mathcal{D}$ has the property required by the $\mathsf{PR\text{-}coirv}$ definition (for the randomness). Clearly, if event $\mathsf{Bad}_{16.\iota}$ happens, the adversary has chosen an offset for h. Thus, if event $\mathsf{Bad}_{16.\iota}$ happens, then HH^{ι} can find a pre-image for h', where h' is obtained by adding an offset chosen in advance to the hash of a “random enough” input. Since H is a $(t_2, \epsilon_{\mathsf{PR\text{-}coirv}})$-$\mathsf{PR\text{-}coirv}$-secure hash function and HH^{ι} is a t_2-adversary, then

$$\Pr[\mathsf{Bad}_{16.\iota}] \leq \epsilon_{\mathsf{PR\text{-}coirv}}, \text{ and}$$

$$|\Pr[E_{19}] - \Pr[E_{20}] \leq \Pr[\mathsf{Bad}_{16}] \leq \sum_{\iota=1}^{q_E} \Pr[\mathsf{Bad}_{16.\iota}] = q_E \epsilon_{\mathsf{PR\text{-}coirv}}.$$

Studying event E_{20}**.** Since we have covered all cases where the adversary has already seen the triple $(h^*, k^*_0, c^*_{l^*}+2)$ for f and g, the only possibility is that $\mathsf{g}(h^*, c^*_{l^*+2})$ has never been computed before. So, we define event Bad_{17}: 1) $\mathsf{g}(h^*, c^*_{l^*+2})$ has never been defined before (thus k^*_0 is random), 2) there exists a previous query to $\mathsf{E}(k^*_0, p_A)$. Additionally, we define event Bad_{18}: 1) $\mathsf{g}(h^*, c^*_{l^*+2})$ has never been defined before, 2) there exists no previous query to $\mathsf{E}(k^*_0, p_A)$ (thus, $k^*_{0,E} = \mathsf{E}_{k^*_0}(p_A)$ is random). 3) there exists a previous query to $\mathsf{E}(k^*_{0,E}, p_B)$.

Bounding $\Pr[\mathsf{Bad}_{17}]$**.** Since $\mathsf{g}(h^*, c^*_{l^*+2})$ has never been computed before, so k^*_0 is picked uniformly at random.

Now, we want to compute how many "bad keys" are there, that is, keys for which $\mathsf{E}(\cdot, p_A)$ has been computed before in the execution of the game. Similar to what we have done before, we observe that during an encryption query there at most $L+3$ queries to $\mathsf{E}(\cdot, p_A)$ with at most $L+3$ different keys (we have an additional query due to Game 10), and $2L+1$ queries with $(\cdot, \cdot)$ where the key is either $k_{\iota,E}$ or $k_{\iota,A}$ for a certain ι and the input is a message m_ι (which can be equal to p_A), while during a decryption query there at most $L+2$ different queries to the ideal block-cipher with input $\mathsf{E}(\cdot, p_A)$ and $2L+1$ to $\mathsf{E}(\cdot, \cdot)$. In addition, due to the change we have introduced in Game 4, there are two additional queries, one of which is to $\mathsf{E}(\cdot, p_A)$ The adversary can do at most q_I queries of its choice to the ideal block-cipher. Thus, there are $(q_E + q_D)(3L+4) + q_I$ different keys that, if picked, would trigger the Bad_{17} event. Thus,

$$\Pr[\mathsf{Bad}_{17}] \leq [(q_E + q_D)(3L+4) + q_I]2^{-n}.$$

Bounding $\Pr[\mathsf{Bad}_{17}]$**.** Since $\mathsf{g}(h^*, c^*_{l^*+2})$ has never been computed before, so k^*_0 is picked uniformly at random.

Now, we want to compute how many "bad keys" are there, that is, keys for which $\mathsf{E}(\cdot, p_B)$ has been computed before in the execution of the game. Similar to what we have done before, we observe that during an encryption query there at most $L+3$ queries to $\mathsf{E}(\cdot, p_B)$ with at most $L+3$ different keys (we have an additional query due to Game 11), and $2L+1$ queries with $(\cdot, \cdot)$ where the key is either $k_{\iota,E}$ or $k_{\iota,A}$ for a certain ι and the input is a message m_ι (which can be equal to p_B), while during a decryption query there at most $L+2$ different queries to the ideal block-cipher with input $\mathsf{E}(\cdot, p_B)$ and $2L+1$ to $\mathsf{E}(\cdot, \cdot)$. In addition, due to the change we have introduced in Game 4, there are two additional queries, one of which is to $\mathsf{E}(\cdot, p_B)$ The adversary can do at most q_I queries of its choice to the ideal block-cipher. Thus, there are $(q_E + q_D)(3L+4) + q_I$ different keys that, if picked, would trigger the Bad_{18} event. Thus,

$$\Pr[\mathsf{Bad}_{18}] \leq [(q_E + q_D)(3L+4) + q_I]2^{-n}.$$

Bounding $\Pr[E_{20}]$**.** If both event Bad_{17} and Bad_{18} don not happen, $\tilde{c}^*_0$ is uniformly at random, thus, the probability that $c^*_0 = \tilde{c}^*_0$ is 2^{-n}. Putting everything together, we obtain that

$$\Pr[E_{20}] \leq \Pr[\mathsf{Bad}_{17}] + \Pr[\mathsf{Bad}_{18}] + \Pr[E_{20} | (\neg\mathsf{Bad}_{17}) \wedge (\neg\mathsf{Bad}_{18})] \leq \\ 2[(q_E + q_D)(3L+4) + q_I] + 2^{-n}.$$

Table 3: The $\mathsf{wCILF2}$ experiment. If AEnc is a probabilistic scheme, $c = \mathsf{AEnc}_k(a, m; r)$ denotes that the randomness r is used. $\mathsf{Faulted}$ returns all possible values that a and c can take during the encryption query (according to the atomic model, Sec. 3.2). We denote with $(a^*, c^*) \notin \mathcal{S}$ that there exists no entry $(x, y, z, w, u) \in \mathcal{S}$ s.t. $(a^*, c^*) = (x, y)$

The $\mathsf{wCILF2}_{\Pi,\mathcal{F},\mathsf{L},\mathcal{F},\mathsf{A}^{\mathsf{LFa}}}$ experiment

```
Initialization:
    $k \leftarrow \mathsf{Gen}$
    $\mathcal{S} \leftarrow \emptyset$

Finalization:
    $(a^*, c^*, a^\dagger, c^\dagger) \leftarrow \mathsf{A}^{\mathsf{L},\mathsf{Fa},\mathsf{AEncLFa}_k,\mathsf{ADecLFa}_k}$
    If $\mathsf{ADec}_k(a^*, c^*) \neq \perp \wedge (a^*, c^*) \notin \mathcal{S}$
        Return 1
    If $\mathsf{ADec}_k(a^*, c^*) \neq \perp \wedge \mathsf{ADec}_k(a^\dagger, c^\dagger) \neq \perp$
        If $\exists (a, m, r)$ s.t.
            $(a^*, c^*, a, m, r), (a^\dagger, c^\dagger, a, m, r) \in \mathcal{S}$
            Return 0
        Return 1
    Return 0
```

```
Oracle $\mathsf{AEncLFa}_k(m, \mathsf{Fa})$:
    $c = \mathsf{AEnc}_k(a, m; r)$
    For $(a', c') \leftarrow \mathsf{Faulted}(a, m, \mathsf{Fa}; r)$
    $\mathcal{S} \leftarrow \{(a', c', a, m, r)\} \cup \mathcal{S}$
    Return $(c, \mathsf{L}_E(\mathsf{AEnc}_k(a, m, \mathsf{Fa})))$

Oracle $\mathsf{ADecLFa}_k(a, c, \mathsf{Fa})$:
    $\mathsf{ans} = \mathsf{ADec}_k(a, c, \mathsf{Fa})$
    $\ell = \mathsf{L}_V(\mathsf{Vrfy}_k(m, \tau, \mathsf{Fa}))$
Return $(\mathsf{ans}, \ell)$
```

Completing the proof. Now, putting all our bounds together we can conclude the proof:

$$\begin{aligned}
\Pr[E_0] \le \Pr[E_{20}] + \sum_{i=0}^{19} |\Pr[E_i] - \Pr[E_{i+1}]| = \\
2[(q_E + q_D)(3L+4) + q_I]2^{-n} + 2^{-n} + \epsilon_{\mathsf{sTPRP}} + 2\frac{q(q+1)}{2^{n+1}} + \\
\epsilon_{\mathsf{CR}} + 0 + q_D[(3L+4)(q_E + \frac{q_D+1}{2}) + q_I]2^{-n} + \\
q_D[(3L+4)(q_E + \frac{q_D+1}{2}) + q_I + 1]2^{-n} + (q_D+1)\epsilon_{\mathsf{PR\text{-}corpi}} + \\
q_E[3(q_E-1)(L+1)/2 + q_I + q_D(3L+4)]2^{-n} + \\
q_E[(q_E-1)(3L+4)/2 + q_I + q_D(3L+4)]2^{-n} + \\
q_E[(q_E-1)(3L+4)/2 + q_I + q_D(3L+4)]2^{-n} + \\
0 + q_E[(3L+4)[q_D + (q_E-1)/2] + q_I]2^{-n} + \\
q_E[(3L+4)[q_D + (q_E-1)/2] + (q_E-1)/2 + q_I]2^{-n} + q_E 2^{-n} + 0 + q_E\epsilon_{\mathsf{PR\text{-}coirv}} + \\
q_E\epsilon_{\mathsf{PR\text{-}corpi}} + 0 + 3q_E 2^{-n} + 0 + q_E\epsilon_{\mathsf{PR\text{-}coirv}} \le \\
\epsilon_{\mathsf{sTPRP}} + \epsilon_{\mathsf{CR}} + q\epsilon_{\mathsf{PR\text{-}corpi}} + 2q_E\epsilon_{\mathsf{PR\text{-}coirv}} + \\
\{4q_E + q(q+1) + 1 + q_D + q_E(\frac{q_E-1}{2})q_I(2 + 2q_D + 5q_E) + \\
(3L+4)[6q_Eq_D + 2q_D + 2q_D\frac{q_D+1}{2} + 2q_E + 5q_E\frac{q_E-1}{2})]\}2^{-n},
\end{aligned}$$

which concludes the proof. □

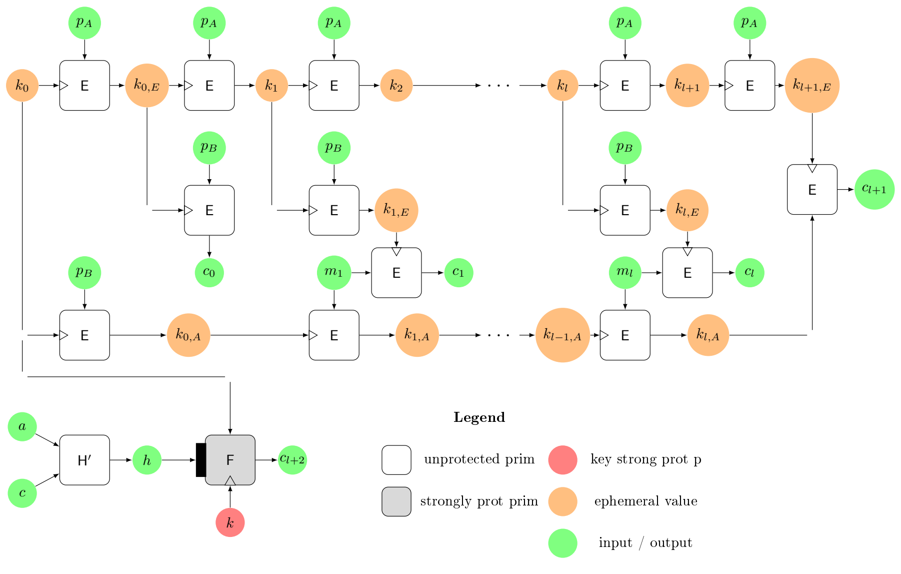

Figure 3: CONCRETE2.

J Algorithm, Tables and Figures

J.1 The wCILF experiment

J.2 Figure CONCRETE2

J.3 The algorithm for PSV.

J.4 The algorithm for PSV-MAC.

J.5 The algorithm for CONCRETE.

J.6 The algorithm for CONCRETE2

J.7 The algorithm for CONCRETE2-TBC

J.8 The algorithm for CONCRETESponge

Algorithm 4 The leakage-resistant encryption scheme PSV [40].

It uses a leak-free BC $\mathsf{F} : \mathcal{K} \times \{0,1\}^n \rightarrow \{0,1\}^n$, and a weakly protected BC $\mathsf{E} : \{0,1\}^n \times \{0,1\}^n \rightarrow \{0,1\}^n$.

```
Gen:
  • $k \xleftarrow{\$} \mathcal{K}$
  • $p_A, p_B \xleftarrow{\$} \{0,1\}^n$
Enc_k(m):
  • Parse $m$ in $m_1, ..., m_l$
  • $\mathsf{IV} \xleftarrow{\$} \{0,1\}^n$
  • $k_1 = \mathsf{F}_k(\mathsf{iv})$
  • For $i = 1, ..., l$
      – $y_i = \mathsf{E}_{k_i}(p_B)$
      – $c_i = \pi_{|m_i|}(y_i) \oplus m_i$
      – $k_{i+1} = \mathsf{E}_{k_i}(p_A)$
  • $c = (c_1, \ldots, c_l)$
  • Return $(\mathsf{iv}, c)$
```

```
Dec_k(m, (iv, c)):
  • Parse $c$ in $c_1, ..., c_l$
  • $k_1 = \mathsf{F}_k(\mathsf{iv})$
  • For $i = 1, ..., l$
      – $y_i = \mathsf{E}_{k_i}(p_B)$
      – $m_i = \pi_{|c_i|}(y_i) \oplus m_i$
      – $k_{i+1} = \mathsf{E}_{k_i}(p_A)$
  • Return $m = (m_1, \ldots, m_l)$
```

Algorithm 5 The leakage-resistant PSV-MAC [40].

It uses a leak-free BC $\mathsf{F} : \mathcal{K} \times \{0,1\}^n \rightarrow \{0,1\}^n$, and a weakly protected BC $\mathsf{E} : \{0,1\}^n \times \{0,1\}^n \rightarrow \{0,1\}^n$.

```
Gen:
  • $k \xleftarrow{\$} \mathcal{K}$
Mac_k(m):
  • Parse $m$ in $m_1, ..., m_l$
  • $\mathsf{IV} \xleftarrow{\$} \{0,1\}^n$
  • $k_1 = \mathsf{F}_k(\mathsf{iv})$
  • For $i = 1, ..., l$
      – $k_{i+1} = \mathsf{E}_{k_i}(m_i)$
  • $\tau = k_{l+1}$
  • Return $(\mathsf{iv}, \tau)$
```

```
Vrfy_k(m, (iv, τ)):
  • Parse $m$ in $m_1, ..., m_l$
  • $k_1 = \mathsf{F}_k(\mathsf{iv})$
  • For $i = 1, ..., l$
      – $k_{i+1} = \mathsf{E}_{k_i}(m_i)$
  • $\tilde{\tau} = k_{l+1}$
  • If $\tilde{\tau} = \tau$
      – Return $\top$
  • Return $\bot$
```

Algorithm 6 The $\mathsf{CONCRETE}$ algorithm [15, 14].

It uses a TBC $\mathsf{F} : \mathcal{K} \times \mathcal{TW} \times \{0,1\}^n \rightarrow \{0,1\}^n$ with a strongly protected implementation, a BC $\mathsf{E} : \{0,1\}^n \times \{0,1\}^n \rightarrow \{0,1\}^n$ with a weakly protected implementation and a hash function $\mathsf{H} : \mathcal{HK} \times \{0,1\}^* \rightarrow \mathcal{TW}$. $\mathsf{H}' : \mathcal{HK} \times \{0,1\}^* \rightarrow \mathcal{TW}$ is a multi-input collision resistant hash function based on H.

```
Gen:
  • $k \xleftarrow{\$} \mathcal{K}$, $s \xleftarrow{\$} \mathcal{HK}$
  • $p_A, p_B \xleftarrow{\$} \{0,1\}^n$
AEnc$_k(a, m)$:
  • Parse $m$ in $m_1, ..., m_l$
  • $k_0 \xleftarrow{\$} \{0,1\}^n$
  • $c_0 = \mathsf{E}_{k_0}(p_B)$
  • $k_1 = \mathsf{E}_{k_0}(p_A)$
  • For $i = 1, ..., l-1$
      – $y_i = \mathsf{E}_{k_i}(p_B)$
      – $c_i = y_i \oplus m_i$
      – $k_{i+1} = \mathsf{E}_{k_i}(p_A)$
  • $y_l = \mathsf{E}_{k_l}(p_B)$
  • $c_l = \pi_{|m_l|}(y_l) \oplus m_l$
  • $h = \mathsf{H}'_s(c_0 \| ... \| c_l, a)$
  • $c_{l+1} = \mathsf{F}_k^h(k_0)$
  • Return $c = c_0 \| ... \| c_l \| c_{l+1}$
ADec$_k(a, c)$:
  • Special Parse $(l)$ $c$ in $c_0, ..., c_l, c_{l+1}$
  • $h = \mathsf{H}'_s(c_0 \| ... \| c_l, a)$
  • $k_0 = \mathsf{F}_k^{-1,h}(c_{l+1})$
  • $\tilde{c_0} = \mathsf{E}_{k_0}(p_B)$
  • If $c_0 \neq \tilde{c_0}$
      – Return $\perp$
  • Else $k_1 = \mathsf{E}_{k_0}(p_A)$
  • For $i = 1, ..., l-1$
      – $y_i = \mathsf{E}_{k_i}(p_B)$
      – $m_i = y_i \oplus c_i$
      – $k_{i+1} = \mathsf{E}_{k_i}(p_A)$
  • $y_l = \mathsf{E}_{k_l}(p_B)$
  • $m_l = \pi_{|c_l|}(y_l) \oplus c_l$
  • Return $m = m_1 \| ... \| m_l$
H'$_s(c, a)$
  • Parse $a$ and $c$
  • $a' = a_1 \| 1 \| a_2 \| 1 \| ... \| 1 \| a_{l_a} \| 1^{n-l_a-1}$
  • $c' = c_1 \| 0 \| c_2 \| 0 \| ... \| 0 \| c_{l_c} \| 0^{n-l_c-1}$
  • $h = \mathsf{H}_s(c' \| a')$
```

Algorithm 7 The CONCRETE2 algorithm. The changes from CONCRETE (Alg. 6) are highlighted.

It uses a TBC $\mathsf{F} : \mathcal{K} \times \mathcal{TW} \times \{0,1\}^n \rightarrow \{0,1\}^n$ with a strongly protected implementation, a BC $\mathsf{E} : \{0,1\}^n \times \{0,1\}^n \rightarrow \{0,1\}^n$ with a weakly protected implementation and a hash function $\mathsf{H} : \mathcal{HK} \times \{0,1\}^* \rightarrow \mathcal{TW}$. $\mathsf{H}' : \mathcal{HK} \times \{0,1\}^* \rightarrow \mathcal{TW}$ is a multi-input collision resistant hash function based on H. For simplicity, we assume that all message blocks are *full*, that is, $|m_l| = n$. If they are not full, see App. H.

```
Gen:
  • $k \xleftarrow{\$} \mathcal{K}$, $s \xleftarrow{\$} \mathcal{HK}$
  • $p_A, p_B \xleftarrow{\$} \{0,1\}^n$

AEnc$_k(a, m)$:
  • Parse $m$ in $m_1, ..., m_l$
  • $k_0 \xleftarrow{\$} \{0,1\}^n$
  • $k_{0,E} = \mathsf{E}_{k_0}(p_A)$
  • $k_{0,A} = \mathsf{E}_{k_0}(p_B)$
  • $c_0 = \mathsf{E}_{k_{0,E}}(p_B)$
  • $k_1 = \mathsf{E}_{k_{0,E}}(p_A)$
  • For $i = 1, ..., l$
      – $k_{i,E} = \mathsf{E}_{k_i}(p_B)$
      – $c_i = \mathsf{E}_{k_{i,E}}(m_i)$
      – $k_{i+1} = \mathsf{E}_{k_i}(p_A)$
      – $k_{i,A} = \mathsf{E}_{k_{i-1,A}}(m_i)$
  • $k_{l+1,E} = \mathsf{E}_{k_{l+1}}(p_B)$
  • $c_{l+1} = \mathsf{E}_{k_{l+1,E}}(k_{l,A})$
  • $h = \mathsf{H}'_s(c_0\|...\|c_{l+1}, a)$
  • $c_{l+2} = \mathsf{F}^h_k(k_0)$
  • Return $c = c_0\|...\|c_l\|c_{l+1}\|c_{l+2}$

ADec$_k(a, c)$:
  • Special parse $(l)$ $c$ in $c_0, ..., c_l, c_{l+1}, c_{l+2}$
  • $h = \mathsf{H}'_s(c_0\|...\|c_l\|c_{l+1}, a)$
  • $k_0 = \mathsf{F}^{-1,h}_k(c_{l+1})$
  • $k_{0,E} = \mathsf{E}_{k_0}(p_A)$
  • $k_{0,A} = \mathsf{E}_{k_0}(p_B)$
  • $\tilde{c_0} = \mathsf{E}_{k_{0,E}}(p_B)$
  • If $c_0 \neq \tilde{c_0}$
      – Return $\perp$
  • Else $k_1 = \mathsf{E}_{k_{0,E}}(p_A)$
  • For $i = 1, ..., l$
      – $k_{i,E} = \mathsf{E}_{k_i}(p_B)$
      – $m_i = \mathsf{E}^{-1}_{k_{i,E}}(c_i)$
      – $k_{i+1} = \mathsf{E}_{k_i}(p_A)$
      – $k_{i,A} = \mathsf{E}_{k_{i-1,A}}(m_i)$
  • $k_{l+1,E} = \mathsf{E}_{k_{l+1}}(p_B)$
  • $\tilde{k}_{l,A} = \mathsf{E}^{-1}_{k_{l+1,E}}(c_{l+1})$
  • If $k_{l,A} \neq \tilde{k}_{l,A}$
      – Return $\perp$
  • Return $m = m_1\|...\|m_l$
```

Algorithm 8 The CONCRETE2-TBC algorithm. The changes from CONCRETE2 (Alg. 7) are highlighted.

It uses a TBC $\mathsf{F} : \mathcal{K} \times \mathcal{TW} \times \{0,1\}^n \rightarrow \{0,1\}^n$ with a strongly protected implementation, a TBC $\mathsf{E} : \{0,1\}^n \times \{0,1\}^n \times \{0,1\}^n \rightarrow \{0,1\}^n$ with a weakly protected implementation and a hash function $\mathsf{H} : \mathcal{HK} \times \{0,1\}^* \rightarrow \mathcal{TW}$. $\mathsf{H}' : \mathcal{HK} \times \{0,1\}^* \rightarrow \mathcal{TW}$ is a a multi-input collision resistant hash function based on H. For simplicity, we assume that all message blocks are *full*, that is, $|m_l| = n$, otherwise, see App. H. With $[\![i]\!]$ we denote that we write the number i with $n-2$ bits.

```
Gen:
  • $k \xleftarrow{\$} \mathcal{K}$, $s \xleftarrow{\$} \mathcal{HK}$
  • $p_A, p_B \xleftarrow{\$} \{0,1\}^n$

AEnc$_k(a, m)$:
  • Parse $m$ in $m_1, ..., m_l$
  • $k_0 \xleftarrow{\$} \{0,1\}^n$
  • $k_{0,E} = \mathsf{E}_{k_0}^{[\![0]\!]\|00}(p_A)$
  • $k_{0,A} = \mathsf{E}_{k_0}^{[\![0]\!]\|01}(p_B)$
  • $c_0 = \mathsf{E}_{k_{0,E}}^{[\![0]\!]\|10}(p_B)$
  • $k_1 = \mathsf{E}_{k_{0,E}}^{[\![0]\!]\|11}(p_A)$
  • For $i = 1, ..., l$
      – $k_{i,E} = \mathsf{E}_{k_i}^{[\![i]\!]\|00}(p_B)$
      – $c_i = \mathsf{E}_{k_{i,E}}^{[\![i]\!]\|01}(m_i)$
      – $k_{i+1} = \mathsf{E}_{k_i}^{[\![i]\!]\|10}(p_A)$
      – $k_{i,A} = \mathsf{E}_{k_{i-1,A}(m_i)}^{[\![i]\!]\|11}$
  • $k_{l+1,E} = \mathsf{E}_{k_{l+1}}^{[\![l+1]\!]\|00}(p_B)$
  • $c_{l+1} = \mathsf{E}_{k_{l+1,E}}^{[\![l+1]\!]\|01}(k_{l,A})$
  • $h = \mathsf{H}'_s(c_0\|...\|c_{l+1}, a)$
  • $c_{l+2} = \mathsf{F}_k^h(k_0)$
  • Return $c = c_0\|...\|c_l\|c_{l+1}\|c_{l+2}$

ADec$_k(a, c)$:
  • Special parse $(l)$ $c$ in $c_0, ..., c_l, c_{l+1}, c_{l+2}$
  • $h = \mathsf{H}'_s(c_0\|...\|c_l\|c_{l+1}, a)$
  • $k_0 = \mathsf{F}_k^{-1,h}(c_{l+1})$
  • $k_{0,E} = \mathsf{E}_{k_0}^{[\![0]\!]\|00}(p_A)$
  • $k_{0,A} = \mathsf{E}_{k_0}^{[\![0]\!]\|01}(p_B)$
  • $\tilde{c_0} = \mathsf{E}_{k_{0,E}}^{[\![0]\!]\|10}(p_B)$
  • If $c_0 \neq \tilde{c_0}$
      – Return $\perp$
  • Else $k_1 = \mathsf{E}_{k_{0,E}}^{[\![i]\!]\|11}(p_A)$
  • For $i = 1, ..., l$
      – $k_{i,E} = \mathsf{E}_{k_i}^{[\![i]\!]\|00}(p_B)$
      – $m_i = \mathsf{E}_{k_{i,E}}^{-1,[\![i]\!]\|01}(c_i)$
      – $k_{i+1} = \mathsf{E}_{k_i}^{[\![i]\!]\|10}(p_A)$
      – $k_{i,A} = \mathsf{E}_{k_{i-1,A}}^{[\![i]\!]\|11}(m_i)$
  • $k_{l+1,E} = \mathsf{E}_{k_{l+1}}^{[\![l+1]\!]\|00}(p_B)$
  • $\tilde{k}_{l,A} = \mathsf{E}_{k_{l+1,E}}^{-1,[\![l+1]\!]\|01}(c_{l+1})$
  • If $k_{l,A} \neq \tilde{k}_{l,A}$
      – Return $\perp$
  • Return $m = m_1\|...\|m_l$
```

Algorithm 9 The CONCRETESponge algorithm. The changes from CONCRETE2 (Alg. 7) are highlighted.

It uses a TBC $\mathsf{F} : \mathcal{K} \times \mathcal{TW} \times \{0,1\}^n \to \{0,1\}^n$ with a strongly protected implementation, a BC $\mathsf{E} : \{0,1\}^n \times \{0,1\}^n \to \{0,1\}^n$ with a weakly protected implementation, a sponge Sp based on a random permutation $\mathsf{P} : \{0,1\}^P \to \{0,1\}^P$ with $P \gg n$, and a hash function $\mathsf{H} : \mathcal{HK} \times \{0,1\}^* \to \mathcal{TW}$. $\mathsf{H}' : \mathcal{HK} \times \{0,1\}^* \to \mathcal{TW}$ is a multi-input collision resistant hash function based on H. For simplicity, we assume that all message blocks are *full*, that is, $|m_l| = n$. If they are not full, see App. H.

```
Gen:
  • $k \xleftarrow{\$} \mathcal{K}$, $s \xleftarrow{\$} \mathcal{HK}$
  • $p_A, p_B \xleftarrow{\$} \{0,1\}^n$
AEnc$_k(a,m)$:
  • Parse $m$ in $m_1, ..., m_l$
  • $k_0 \xleftarrow{\$} \{0,1\}^n$
  • $k_{0,E} = \mathsf{E}_{k_0}(p_A)$
  • $k_{0,A} = \mathsf{E}_{k_0}(p_B)$
  • $c_0 = \mathsf{E}_{k_{0,E}}(p_B)$
  • $\mathsf{st}_0 = k_{0,A} \| c_0 \| 0^{P-2n}$
  • $\mathsf{st}_1^o = \mathsf{P}(\mathsf{st}_0)$
  • $k_1 = \pi_{n,l}(\mathsf{st}_1^o)$
  • For $i = 1, ..., l$
      – $k_{i,E} = \mathsf{E}_{k_i}(p_B)$
      – $c_i = \mathsf{E}_{k_{i,E}}(m_i)$
      – $k_{i,A} = \mathsf{E}_{k_i}(p_A)$
      – $x_i = \mathsf{E}_{k_{i,A}}(m_i)$
      – $\mathsf{st}_i^i = x_i \| 0^{P-N} \oplus \mathsf{st}_{i-1}^o$
      – $\mathsf{st}_{i+1}^o = \mathsf{P}(\mathsf{st}_i^i)$
      – $k_{i+1} = \pi_{n,l}(\mathsf{st}_{i+1}^o)$
  • $c_{l+1} = k_{l+1}$
  • $h = \mathsf{H}'_s(c_0 \| ... \| c_{l+1}, a)$
  • $c_{l+2} = \mathsf{F}_k^h(k_0)$
  • Return $c = c_0 \| ... \| c_l \| c_{l+1} \| c_{l+2}$

ADec$_k(a,c)$:
  • Parse $c$ in $c_0, ..., c_l, c_{l+1}, c_{l+2}$
  • $h = \mathsf{H}'_s(c_0 \| ... \| c_l, c_{l+1}, a)$
  • $k_0 = \mathsf{F}_k^{-1,h}(c_{l+1})$
  • $k_{0,E} = \mathsf{E}_{k_0}(p_A)$
  • $k_{0,A} = \mathsf{E}_{k_0}(p_B)$
  • $\tilde{c_0} = \mathsf{E}_{k_{0,E}}(p_B)$
  • If $c_0 \neq \tilde{c_0}$
      – Return $\perp$
  • Else $\mathsf{st}_0 = k_{0,A} \| 0^{P-n}$
  • $\mathsf{st}_1^o = \mathsf{P}(\mathsf{st}_0)$
  • $k_1 = \pi_{n,l}(\mathsf{st}_1^o)$
  • For $i = 1, ..., l$
      – $k_{i,E} = \mathsf{E}_{k_i}(p_B)$
      – $m_i = \mathsf{E}_{k_{i,E}}^{-1}(c_i)$
      – $k_{i,A} = \mathsf{E}_{k_i}(p_A)$
      – $x_i = \mathsf{E}_{k_{i,A}}(m_i)$
      – $\mathsf{st}_i^i = x_i \| 0^{P-N} \oplus \mathsf{st}_{i-1}^o$
      – $\mathsf{st}_{i+1}^o = \mathsf{P}(\mathsf{st}_i^i)$
      – $k_{i+1} = \pi_{n,l}(\mathsf{st}_{i+1}^o)$
  • If $k_{l+1} \neq c_{l+1}$
      – Return $\perp$
  • Return $m = m_1 \| ... \| m_l$
```