IACR Transactions on Symmetric Cryptology
ISSN 2519-173X, Vol. 2024, No. 2, pp. 1–34. DOI:10.46586/tosc.v2024.i2.1-34

Multiplex: TBC-Based Authenticated Encryption with Sponge-Like Rate

Yaobin Shen[1(✉)], Thomas Peters[2] and François-Xavier Standaert[2]

[1] School of Informatics, Xiamen University, Xiamen, China
[2] UCLouvain, ICTEAM, Crypto Group, Louvain-la-Neuve, Belgium
yaobin.shen@xmu.edu.cn, thomas.peters@uclouvain.be, fstandae@uclouvain.be

Abstract. Authenticated Encryption (AE) modes of operation based on Tweakable Block Ciphers (TBC) usually measure efficiency in the number of calls to the underlying primitive per message block. On the one hand, many existing solutions reach a *primitive-rate* of 1, meaning that each n-bit block of message asymptotically needs a single call to the TBC with output length n. On the other hand, while these modes look optimal in a blackbox setting, they become less attractive when leakage comes into play, since all these calls must then be equally well protected to maintain security. Leakage-resistant modes improve this situation, by generating ephemeral keys every constant number of calls. However, rekeying is inherently suboptimal in primitive-rate, since a TBC call can only be used either to refresh a key or to encrypt a block. Even worse, existing solutions achieving almost n bits of security for n-bit secret keys have at most a primitive-rate 2/3. Hence the question: *Can we design a highly-secure TBC-based rekeying mode with "nearly optimal" primitive-rate?* We answer this question positively with Multiplex, a new mode that has primitive-rate $d/(d+1)$ given a TBC with a dn-bit tweak. Multiplex achieves $n - \log_2(dn)$ bits of security for both (i) misuse-resilience CCA confidentiality security in the blackbox setting and (ii) Ciphertext Integrity with Misuse-resistant and unbounded Leakage in encryption and decryption (CIML2). It also provides (iii) confidentiality with leakage up to the birthday bound. Furthermore, Multiplex can run $d+1$ calls in parallel in each iteration. The combination of these features gives a mode of operation that inherits most of the good implementation features and flexibility of a sponge construction – therefore paving the way towards sound comparisons between TBC-based and permutation-based AE.

Keywords: Leakage-Resistance · Authenticated Encryption · Tweakable Block Cipher

1 Introduction

Leakage-resistant modes of operation aim to better balance the burden of preventing side-channel attacks between cryptographic designers and hardware engineers. They allow so-called leveled implementations, where different parts of the implemented designs require different (more or less expensive) countermeasures. In general, adding leakage-resistant features to a mode implies moderate overheads when side-channel attacks are not a concern and can lead to significant gains when they are a concern. In the context of lightweight cryptography where single-pass modes are preferred, it is for example possible to reach the best possible integrity with leakage guarantees so far, coined Ciphertext Integrity with Misuse-resistance and Leakage in encryption and decryption (CIML2) [GPPS19], in a liberal model where only a key-derivation function (KDF) and a tag-generation function (TGF) have to be protected against leakage, while the rest of the computations can leak in

Received: 2024-03-01 Accepted: 2024-05-01 Published: 2024-06-18

full (so require no countermeasure). By contrast, it is not possible to reach the best possible confidentiality with leakage guarantees in a leveled manner (which requires two passes) in this case: only CCA security with misuse-resilience and Leakage in encryption, coined CCAmL1 [GPPS19], can then be obtained. CCAmL1 designs ensure that confidentiality of plaintexts is preserved as long as the nonces used to encrypt are fresh (even if other plaintexts are compromised in the past [ADL17]). They require strong protections against leakage for their KDF and TGF, and mild protections for the message-processing part. As discussed in [BBC+20], a necessary condition for the strongly protected parts of these modes is that they resist Differential Power Analysis (DPA), which are side-channel attacks where the adversary can observe the leakage of many different inputs for the same key, and a necessary condition for the mildly protected parts is that they resist Simple Power Analysis (SPA), which are side-channel attacks where the adversary can only observe the leakage of a small number of inputs for the same key (e.g., fixed by the cryptographic design thanks to a re-keying process in the case of leakage-resistant encryption). Leveled implementations that ensure CIML2 and CCAmL1 security are denoted Grade-2 by Bellizia et al. in [BBC+20].

The leveled implementation of (e.g., grade-2) designs can lead to significant performance gains over uniformly protected ones. Furthermore, these gains increase when the target physical security level increases, due to increased cost of the countermeasures needed in this case. This conclusion has now been consolidated both for sponge-based designs like Ascon [DEMS21, VCS22] or Spook [BBB+20, MCS22] and for designs based on block ciphers or Tweakable Block Ciphers (TBC) like LR-BC [BMPS21] or Triplex [SPS+22]. Triplex (which is the starting point of our work) is a single-pass and leakage-resistant mode of operation. In each iteration, it encrypts $2n$ bits of message with 3 calls to a TBC with $2n$-bit tweaks where n is the block size. It has primitive-rate $2/3$, but its 3 TBC calls cannot be performed in parallel.

The main contribution of this work is to observe that by improving Triplex to make it parallelizable for each iteration, we can reach a flexible design, denoted as Multiplex and illustrated in Figure 5. It leads to two important outcomes.

On the one hand, Multiplex can encrypt with a flexible rate given a TBC with arbitrary tweak length and fixed block and key size of n bits. For this purpose, Multiplex uses a TBC with dn-bit tweak as underlying primitive and it is able to encrypt dn bits of message with $d+1$ calls to this TBC.[1] So whereas Triplex has a primitive-rate of $2/3$ and is partially serial, the primitive-rate of Multiplex approaches 1 when d increases and its $d+1$ TBC calls can be processed in parallel. Therefore, Multiplex has a primitive-rate of $d/(d+1)$. A more detailed comparison between Multiplex and Triplex is provided in Section 4.

On the other hand, and besides its interest as an efficient leakage-resistant mode of operation, the similarity of Multiplex with the Duplex construction is striking [BDPA11]. In order to reach a similar security level (namely $\approx n - \log_2(dn)$ bits of CIML2 and CCA security, $\approx n/2$ bits of CCAmL1 security [BBB+20]), a Duplex construction needs a $2n$-bit capacity. It means that in order to encrypt dn bits of message in each iteration, it requires a permutation operating on a $(d+2)n$-bit state. Multiplex requires a TBC with an input space of the same size in this case: the dn-bit tweak, the n-bit plaintext and the n-bit key, for a total of $(d+2)n$ bits. The $d+1$ TBCs it uses in parallel have the same input space as well (i.e., these TBCs use the same inputs organized differently). Hence, Multiplex and the Duplex construction require primitives with identical input spaces to encrypt messages of the same size with the same security. They only differ in their output space: Multiplex uses $d+1$ calls to a TBC with n-bit output space for this purpose, while the Duplex construction uses 1 call to a $(d+2)n$-bit permutation.[2]

[1] The TBCs needed by Multiplex can be designed based on the tweakey framework [JNP14]. We refer to the Skinny family for exemplary instances of such designs [BJK+16].

[2] We remark that the Duplex construction may be superior in a different aspect, and here we focus on

The interest of Multiplex actually goes beyond the context of leakage-resistance. How to compare the efficiency of TBC-based and permutation-based designs has been a long-standing open question in symmetric cryptography [Pey20]. One recurrent challenge to answer it is the better flexibility of permutation-based designs, which can increase their rate by increasing the size of their underlying permutation. Multiplex provides such a flexibility and improves over Skinny-Hash [BJK+20] which, to the best of our knowledge, is the only TBC-based design with similar flexibility. As a result, it allows nailing down an interesting research challenge for symmetric (crypt)analysists. Namely, since the flexibility of Multiplex and the Duplex construction respectively come from increasing the tweak size of a TBC or the permutation size of a sponge, it would be interesting to understand how the number of rounds of these two primitives must increase in function of d for the corresponding authenticated encryption schemes to maintain their security guarantees. The latter may differ since, for example, generating pseudorandom n-bit blocks (with $d+1$ blocks per iteration) is not the same as generating pseudorandom $(d+2)n$-bit states. Equipped with such an understanding, it would then become possible to evaluate the implementation figures of the corresponding authenticated encryption schemes according to different metrics (e.g., speed, area, energy per bit, memory requirements, ...).

Note that we give multi-user security proofs and consider schemes with n bits of public key to have good bounds in this setting [BT16]. Our bounds are maintained if we ignore this public parameter but are then only valid in the single-user setting. A similar tradeoff can be applied to permutation-based designs [BBB+20], and it is therefore not discriminant for the TBC vs. Duplex/sponge discussion.

Related works. Leakage-resistant authentication, encryption and authenticated encryption has been active research topics over the recent years, with many theoretical treatments and proposals of concrete schemes. Theoretical treatments of block cipher based or TBC-based designs include [BKP+18, BPPS17, PSV15] and [BGPS21]. Theoretical treatments of permutation-based designs include [DJS19, DM19, GPPS20] and [DM21]. Examples of grade-3 designs (which enable leveled implementations with CIML2 and CCAmL2 guarantees) include TEDT and TEDT2 which are TBC-based [BGP+20, Lis21] and ISAP which is permutation-based [DEM+20]. Example of Grade-2 designs were given before (Ascon, Spook, LR-BC, Triplex). These works generally suggest that a possible advantage of TBCs in the context of leakage is that they are more easily amenable to proofs without idealized assumptions. They are built on the seed results of Micali and Reyzin [MR04] and Dziembowski and Pietrzak [DP08] who laid the foundations of leakage-resilient primitives such as pseudorandom generators, functions and permutations, with numerous follow-ups [Pie09, YSPY10, DP10, FPS12, YS13].

As for analyses without leakage, we first mention the important result of Jovanovic et al. [JLM+19]. It shows that some sponge designs offer n bits of security with a capacity lower than $2n$ bits, which renders the output space of "standard" sponge constructions (i.e., $(d+2)n$ bits) closer to that of Multiplex (i.e., $(d+1)n$ bits). Yet, this only holds without leakage and while ensuring $\approx n - \log_2(dn)$ bits of CIML2 security requires a $2n$-bit capacity (see [SPS+22, Appendix C]). More generally, TBC-based and permutation-based designs are the focus of a rich literature. We refer to [PS16] for a state-of-the-art TBC-based authenticated encryption and to [DMA17, MRV15] for developments of the Duplex construction.

We finally refer to Section 4.1 for a detailed model-level comparison with Triplex and Skinny-Hash, which are Multiplex's main competitors.

Future work. In order to further demonstrate the generality of Multiplex, we initiate the study of adapting the mode to short-block TBCs in Appendix B. Such TBCs have

the comparison between the underlying TBC calls in Multiplex and the underlying permutation call in Duplex.

been considered in a recent work aiming at low-memory authenticated encryption [NSS22], which the authors argue is relevant for higher-order masking against side-channel attacks. It considers a different model than ours and, in particular, cannot lead to leveled implementations. We leave open the question to fully address the adaptation of Multiplex to this case and assess its security, as it requires different techniques to achieve CIML2 (for instance, concatenating shorter tags does not directly prevent divide-and-conquer attacks on each shorter tag). Nevertheless, our design principle shows that such a variant could enjoy less time to proceed the data thanks to parallelism in each iteration, which motivates an interesting direction for future research.

We also provide a multi-block hash function based on TBCs in Appendix C, which does not have a direct application for our (leakage-resistant) authenticated encryption application but could be useful in other contexts.

Organization. We first give notations and security notions in Section 2. Next, in Section 3, we introduce a new hash function called Multihash that is the main component of Multiplex. In Section 4, we formally give the design rationale and specification of Multiplex. Then, in Section 5 and Section 6, we give the security proofs for Multiplex. We finally conclude the paper in Section 7.

2 Preliminaries

Notations Let ε denote the empty string. Let $\{0,1\}^*$ be the set of all finite bit strings including the empty string ε. For a finite set S, let $x \xleftarrow{\$} S$ denote the uniform sampling from S assigning a value to x. Let $|x|$ denote the length of the string x. Let $x[i:j]$ denote the substring from the i-th bit to the j-th bit (inclusive) of x. Concatenation of strings x and y is written as $x \,\|\, y$ or simply xy. If A is an algorithm, let $y \leftarrow A(x_1, \ldots; r)$ denote running A with randomness r on inputs $x_1, \ldots$ and assigning the output to y. Let $y \xleftarrow{\$} A(x_1, \ldots)$ be the result of picking r at random and letting $y \leftarrow A(x_1, \ldots; r)$. Let $\mathrm{Perm}(n)$ denote the set of all permutations over $\{0,1\}^n$, and let finally $\mathrm{Func}(*, n)$ denote the set of all functions from $\{0,1\}^*$ to $\{0,1\}^n$.

Tweakable block cipher [LRW11]. A block cipher $E : \mathcal{K} \times \mathcal{M} \to \mathcal{M}$ is a family of permutations, where $E_K(\cdot) = E(K, \cdot)$ is a permutation over $\mathcal{M}$. A tweakable block cipher $E : \mathcal{K} \times \mathcal{T} \times \mathcal{M} \to \mathcal{M}$ (slightly abusing notation for simplicity) is a family of permutations over $\mathcal{M}$, indexed by two functionally distinct parameters: a key $K \in \mathcal{K}$ that is secret and used to provide the security, and a tweak $T \in \mathcal{T}$ that is public and used to provide variability. We therefore write $E_K(T, \cdot) = E(K, T, \cdot)$, a permutation over $\mathcal{M}$.

Nonce-based authenticated encryption [Rog02]. An AE scheme Π is a triplet of algorithms $(\mathcal{K}, \mathcal{E}, \mathcal{D})$, where $\mathcal{K}$ is the key-generation algorithm, $\mathcal{E}$ the encryption algorithm and $\mathcal{D}$ the decryption algorithm. The key-generation algorithm $\mathcal{K}$ samples a key K uniformly at random from the key space. The encryption algorithm $\mathcal{E}$ takes as input a key K, a nonce N, Associated Data (AD) A, a message M, and returns a ciphertext and tag $C \,\|\, \mathsf{tag} \leftarrow \mathcal{E}_K(N, A, M)$. The decryption algorithm $\mathcal{D}$ takes as input a key K, a nonce N, an AD A, a ciphertext and tag $C \,\|\, \mathsf{tag}$, and returns either a message M or a symbol $\perp$ indicating invalidity. For correctness, we assume that if $C \,\|\, \mathsf{tag} \leftarrow \mathcal{E}_K(N, A, M)$ then $M \leftarrow \mathcal{D}_K(N, A, C \,\|\, \mathsf{tag})$. In this paper, the tag is always of fixed length.

Privacy security. We define the privacy security with respect to nonce-misuse resilience as introduced in [ADL17]. The privacy security game $\mathsf{G}^{\mathrm{priv}}_{\Pi}$ is detailed in Figure 1. We consider the security in the multi-user setting. For queries to the same user, the adversary may repeat the nonce in the first encryption oracle Enc_1, but the nonce in the second encryption oracle should be unique and fresh. For queries to different users, the adversary

Game $\mathsf{G}_{\Pi}^{\text{priv}}(\mathcal{A})$

$K_1, K_2, \ldots, \xleftarrow{\$} \mathcal{K}$; $b \xleftarrow{\$} \{0,1\}$
$b' \leftarrow \mathcal{A}^{\text{PRIM},\text{ENC}_1,\text{ENC}_2}$; **return** $(b' = b)$

Procedure PRIM(J, T, X)
if $X = (+, x)$ **then return** $E_J(T, x)$
if $X = (-, y)$ **then return** $E_J^{-1}(T, y)$

Procedure ENC$_1(i, N, A, M)$
$C \,\|\, \mathsf{tag} \leftarrow \mathcal{E}(K_i, N, A, M)$
return $C \,\|\, \mathsf{tag}$

Procedure ENC$_2(i, N, A, M)$
$C_1 \,\|\, \mathsf{tag}_1 \leftarrow \mathcal{E}(K_i, N, A, M)$
$C_0 \,\|\, \mathsf{tag}_0 \xleftarrow{\$} \{0,1\}^{|C_1|+|\mathsf{tag}|}$
return $C_b \,\|\, \mathsf{tag}_b$

Figure 1: Game $\mathsf{G}_{\Pi}^{\text{priv}}$: multi-user privacy security of an AE Π.

Game $\mathsf{G}_{\Pi}^{\text{CIML2}}(\mathcal{A})$

$K_1, K_2, \ldots, \xleftarrow{\$} \mathcal{K}$; $b \xleftarrow{\$} \{0,1\}$
$\mathcal{Q} \leftarrow \emptyset$; $b' \xleftarrow{\$} \mathcal{A}^{\text{PRIM},\text{ENC},\text{DEC}}$
return $(b' = b)$

Procedure ENC(i, N, A, M)
$L_e \leftarrow \mathcal{L}_E(K_i, N, A, M)$
$C \,\|\, \mathsf{tag} \leftarrow \mathcal{E}(K_i, N, A, M)$
$\mathcal{Q} \leftarrow \mathcal{Q} \cup \{(K_i, N, A, C \,\|\, \mathsf{tag})\}$
return $(C \,\|\, \mathsf{tag}, L_e)$

Procedure DEC$(i, N, A, C \,\|\, \mathsf{tag})$
$L_d \leftarrow \mathcal{L}_D(K_i, N, A, C \,\|\, \mathsf{tag})$
$M \leftarrow \mathcal{D}(K_i, N, A, C \,\|\, \mathsf{tag})$
if $(K_i, N, A, C \,\|\, \mathsf{tag}) \in \mathcal{Q}$ **then**
  return (M, L_d)
if $b = 0$ **then return** $(\perp, L_d)$
else return (M, L_d)

Procedure PRIM(J, T, X)
if $X = (+, x)$ **then return** $E_J(T, x)$
if $X = (-, y)$ **then return** $E_J^{-1}(T, y)$

Figure 2: Game $\mathsf{G}_{\Pi}^{\text{CIML2}}$: multi-user CIML2 security of an AE Π.

may repeat the nonce in both oracles. With access to oracles PRIM, ENC$_1$ and ENC$_2$, the goal of the adversary is to distinguish the second encryption oracle of an AE scheme from a random function. Formally, given an adversary $\mathcal{A}$, we define

$$\mathsf{Adv}_{\Pi}^{\text{priv}}(\mathcal{A}) = 2\Pr\left[\,\mathsf{G}_{\Pi}^{\text{priv}}(\mathcal{A})\,\right] - 1$$

as adversary's advantage against the privacy security of an AE Π in the nonce misuse-resilience setting, with $\mathsf{G}_{\Pi}^{\text{priv}}(\mathcal{A})$ the abbreviation of $\mathsf{G}_{\Pi}^{\text{priv}}(\mathcal{A}) = \mathsf{true}$.

AUTHENTICITY SECURITY. We consider the authenticity security in the leakage setting, and follow the notion of Ciphertext Integrity with Misuse-resistance and encryption and decryption Leakage (CIML2) by Berti et al. [BPPS17]. Here, the adversary not only has access to the encryption oracle $\mathcal{E}$ and decryption oracle $\mathcal{D}$, but also to their corresponding leakage functions $\mathcal{L}_E$ and $\mathcal{L}_D$. We consider it in the multi-user setting. Given an adversary $\mathcal{A}$, we define

$$\mathsf{Adv}_{\Pi}^{\text{CIML2}}(\mathcal{A}) = 2\Pr\left[\,\mathsf{G}_{\Pi}^{\text{CIML2}}(\mathcal{A})\,\right] - 1$$

as the advantage of the adversary against the CIML2 security of an AE scheme Π, where game $\mathsf{G}_{\Pi}^{\text{CIML2}}$ is illustrated in Fig. 2 and $\mathsf{G}_{\Pi}^{\text{CIML2}}(\mathcal{A})$ is the abbreviation that $\mathsf{G}_{\Pi}^{\text{CIML2}}(\mathcal{A}) = \mathsf{true}$. The adversary is given encryption and decryption oracles, both of which contain the corresponding leakage function. She can repeat nonces in encryption and decryption queries. She may also make a decryption query $(i, N, A, C \,\|\, \mathsf{tag})$ even if $(i, N, A, C \,\|\, \mathsf{tag})$ has appeared in previous encryption queries. This kind of decryption query lets her obtain additional leakage during decryption. The goal of the adversary is to output a valid and new tuple $(i, N, A, C \,\|\, \mathsf{tag})$ that passes the decryption oracle of the real AE scheme, while in the ideal world she will always receive a rejection symbol $\perp$.

Some useful Lemmas. The next lemmas will be useful in our analyzes.

Lemma 1. *[HT17, Lemma 15] Suppose that we throw u balls uniformly at random into 2^n bins. Then, with probability at most 2^{-n}, there exists some bin of more than $\max\{4n, 4u/2^n\}$ balls.*

Lemma 2. *[BHT18, Lemma 10] Suppose that we throw u balls into 2^n bins and, conditioning on the result of prior throws, the probability that each ball falls into any particular bin is at most 2^{1-n}. Fix $0 < \epsilon < 1$, and let $u \leq 2^{(1-\epsilon)n-1}$. Then with probabilty at most $2^{-n/2}$, there exists some bin of more than $\lceil 1.5/\epsilon \rceil$ balls.*

3 Multihash: Extending Hirose's Hash Function

In this section, we introduce a new hash function called Multihash which relies on a tweakable block cipher with n-bit key, dn-bit tweak and n-bit block. This hash function can be seen as an extension of Hirose's double-block length hash function [Hir06] that further outputs auxiliary blocks. Multihash is the main component of Multiplex whose specification is given in the next section, but we already give the intuition below of how these auxiliary blocks can be used to encrypt message blocks. We prove the collision resistance up to $O(2^n)$ queries in the ideal TBC model, where we take into account the computation of these auxiliary blocks for better modularity, which will be helpful in the analysis of Multiplex.

Compression function. We first recall the definition of Hirose's compression function based on an underlying TBC with dn-bit tweak.

Definition 1. Let $E : \{0,1\}^n \times \{0,1\}^{dn} \times \{0,1\}^n \to \{0,1\}^n$ be a tweakable block cipher. Then $\mathsf{Hirose} : \{0,1\}^{2n} \times \{0,1\}^{dn} \to \{0,1\}^{2n}$ is a compression function such that $(h_i, k_i) = \mathsf{Hirose}(h_{i-1}, k_{i-1}, t_i)$ where $h_{i-1}, k_{i-1} \in \{0,1\}^n$ is the state, $t_i \in \{0,1\}^{dn}$ is being absorbed, and

$$\begin{cases} h_i = E_{k_{i-1}}(t_i, h_{i-1}) \oplus h_{i-1}\,, \\ k_i = E_{k_{i-1}}(t_i, h_{i-1} \oplus \theta_1) \oplus h_{i-1} \oplus \theta_1\,, \end{cases}$$

where $\theta_1 \in \{0,1\}^n$ is a non-zero constant, e.g., $1 = 0^{n-1}1$.

Next, we describe the extension of Hirose's compression function that also outputs $d-1$ auxiliary blocks $e^1, \ldots, e^{d-1}$. In general, these blocks are optional and they are not directly involved in the collision bound in the sense that (h_i, k_i) remains the actual compressed output. Still, in the security proof in the ideal TBC model, we take all these computations into account and show that they do not decrease the expected high security.

Definition 2. Let $E : \{0,1\}^n \times \{0,1\}^{dn} \times \{0,1\}^n \to \{0,1\}^n$ be a tweakable block cipher. Then $F : \{0,1\}^{2n} \times \{0,1\}^{dn} \to \{0,1\}^{(d+1)n}$ is an extended Hirose's compression function such that $(h_i, k_i, e_i^1, \ldots, e_i^{d-1}) = F(h_{i-1}, k_{i-1}, t_i)$ where $h_{i-1}, k_{i-1} \in \{0,1\}^n$ is the state, $t_i \in \{0,1\}^{dn}$ is being absorbed, $h_i, k_i \in \{0,1\}^n$ is the main output (the refreshed state), and $e_i^1, \ldots, e_i^{d-1}$ the auxiliary blocks that are computed as follows:

$$\begin{cases} h_i = E_{k_{i-1}}(t_i, h_{i-1}) \oplus h_{i-1}, \\ k_i = E_{k_{i-1}}(t_i, h_{i-1} \oplus \theta_1) \oplus h_{i-1} \oplus \theta_1, \\ e_i^1 = E_{k_{i-1}}(t_i, h_{i-1} \oplus \theta_2) \oplus h_{i-1} \oplus \theta_2, \\ \qquad\qquad \vdots \\ e_i^{d-1} = E_{k_{i-1}}(t_i, h_{i-1} \oplus \theta_d) \oplus h_{i-1} \oplus \theta_d. \end{cases}$$

Here $\theta_1, \ldots, \theta_d \in \{0,1\}^n$ are distinct non-zero constants (e.g., $1, \ldots, d$).

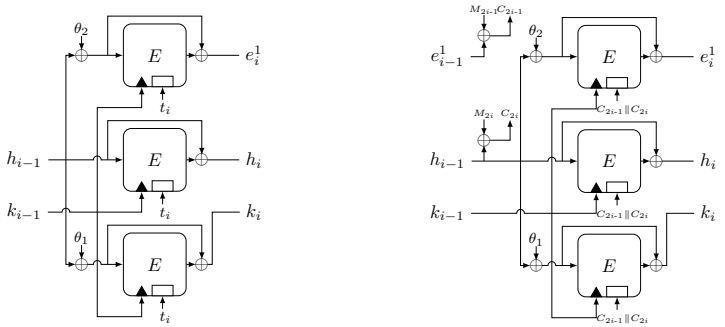

Figure 3: The compression function F based on a tweakable block cipher E with n-bit key and $2n$-bit tweak (left). The intuitive idea to turn the compression function F into encryption and authentication (right).

Hash function $\mathsf{Multihash}(t)$
$t_1 \dots t_\ell \leftarrow \mathrm{pad}(t)$; $(h_0, k_0) \leftarrow (0^n, 0^n)$
for $i \leftarrow 1$ **to** ℓ **do**
  $(h_i, k_i, e_i^1, \dots, e_i^{d-1}) \leftarrow F(h_{i-1}, k_{i-1}, t_i)$
return (h_ℓ, k_ℓ)

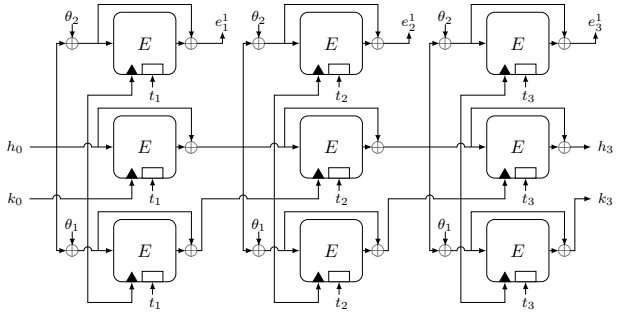

Figure 4: The hash function $\mathsf{Multihash}$ built from the compression function F of Figure 3.

It is easy to see that if $(h_i, k_i, e_i^1, \dots, e_i^{d-1}) = F(h_{i-1}, k_{i-1}, t_i)$ then we have $(h_i, k_i) = \mathsf{Hirose}(h_{i-1}, k_{i-1}, t_i)$. A pictorial illustration of the extended compression function F when $d = 2$ is given in the left of Figure 3. On the right-hand side of the figure, the reader can already have an intuition about how the iteration of the compression function F can be turned into an encryption and how the resulting ciphertext blocks can be digested.

We now turn to $\mathsf{Multihash} : \{0,1\}^* \to \{0,1\}^{2n}$, the hash function defined as the iteration of the compression function F. Let $\mathrm{pad} : \{0,1\}^* \to (\{0,1\}^{dn})^+$ be an injective padding function, e.g., the one-zero padding $\mathrm{pad}(t) = t \,\|\, 10^a$ where $a = dn - 1 - (|t| \mod dn)$. The $\mathsf{Multihash}$ specification is given in Figure 4. Note that after each iteration, only the two blocks h_i and k_i are forwarded to the next iteration. The remaining $d - 1$ blocks are output as auxiliary blocks and will later be used for encrypting messages.

From hash to encryption and authentication. We first explain the way to turn $\mathsf{Multihash}$ into encryption and authentication that motivates the collision resistance achieved by $\mathsf{Multihash}$. The collision resistance mainly focuses on the two-block (h, k)-output but we also take into account another type of internal collisions that will be crucial to avoid in the security analysis of $\mathsf{Multiplex}$.

The encryption is based on the observation that the (full) output of the compression function F (see the right of Figure 3 for an example with $d = 2$) is a $(d + 1)n$-bit random

string as long as the ephemeral key k_{i-1} of the TBC is secret and random. Therefore, assuming the current $(d+1)$ n-bit blocks are random, we can use the top d blocks to XOR-encrypt d blocks of a message while keeping the single bottom n-bit block k_{i-1} as the secret ephemeral key to produce the next bunch of $(d+1)$ random n-bit blocks during the current iteration. Regarding authentication, it follows from the fact that the tweak of a TBC is public but can significantly influence the output. Therefore, we can put the ciphertext blocks (or blocks of associated data) as the tweak to be authenticated. Yet, this process requires a key-derivation function to produce the initial random full state $(h_1, k_1, e_1^1, \ldots, e_1^{d-1})$ to begin with[3] as well as a tag-generation function to encapsulate the digest, i.e., the final (short) output state "(h_{last}, k_{last})", and produce a tag. The details of these two components and more design considerations are postponed to the next section.

While for the AE the iterations manipulate a (full) state of $(d+1)$ blocks in addition to the message blocks, the underlying (short) state of the extended compression function F remains of 2 blocks. Indeed, in terms of collision resistance, only (h_i, k_i) carries the security to the next iteration. This is important to reach the CIML2 notion since the adversary has the full control on the ciphertext blocks in the tweaks both in encryption and decryption (due to leakage and nonce reuse). Moreover, for secrecy concerns we also want to avoid that any pair (h_i, k_i) appears in another iteration at any position in a single *full* state. In summary, and back to Multihash, we need for technical reason to capture the following event called *two-block collision*. By definition, a two-block collision occurs if there are $(x_1, x_2) \in \{(h_\ell, k_\ell), (h_\ell \oplus \theta_1, k_\ell), \ldots, (h_\ell \oplus \theta_d, k_\ell)\}$ and $(y_1, y_2) \in \{(h_{\ell'}, k_{\ell'}), (h_{\ell'} \oplus \theta_1, k_{\ell'}), \ldots, (h_{\ell'} \oplus \theta_d, k_{\ell'})\}$ such that $(x_1, x_2) = (y_1, y_2)$, where (h_ℓ, k_ℓ) and $(h_{\ell'}, k_{\ell'})$ are the $2n$-bit outputs of Multihash for two different messages t and t' respectively. In that case, we say that one finds a two-block collision. For an adversary $\mathcal{A}$ with oracle access to E and E^{-1}, let $\mathsf{Adv}^{\mathsf{tcoll}}_{\mathsf{Multihash}}(\mathcal{A})$ be the probability that $\mathcal{A}$ finds a two-block collision of Multihash. The following lemma shows the high two-block-collision resistance of Multihash.

Lemma 3. *Let* Multihash *be a hash function specified in Figure 4. Then for any adversary $\mathcal{A}$ making at most q queries to E and E^{-1} where $q \leq 2^{n-2}$, we have*

$$\mathsf{Adv}^{\mathsf{tcoll}}_{\mathsf{Multihash}}(\mathcal{A}) \leq \frac{8dq^2}{2^{2n}}$$

by assuming $\theta_1, \ldots, \theta_d = 1, \ldots, d$.

Proof. Let $\mathcal{A}$ be an adversary that aims at finding a two-block collision for the hash function Multihash. In total, $\mathcal{A}$ asks at most q queries to E and E^{-1}. Suppose that $\mathcal{A}$ finds a two-block colliding pair t and t' for the hash function Multihash. Then it is easy to see that $\mathcal{A}$ also finds a two-block colliding pair for the compression function F. Note that each output block of F is determined by both the plaintext and the ciphertext of the TBC. Hence, regardless of the adversary asking forward queries or backward queries to E, $h_i = E_{k_{i-1}}(t_i, h_{i-1}) \oplus h_{i-1}$ is always determined randomly. Similar arguments hold for $k_i, e_i^1, \ldots, e_i^{d-1}$.

Let (K, T, X, Y) be the entry that records the query and response of E, where K is the key, T the tweak, X the plaintext, Y the ciphertext. Without loss of generality, we assume that the adversary does not make repeated queries to E since otherwise she will receive the same responses. To obtain blocks h_i and k_i, it requires a pair of entries $(k_{i-1}, t_i, h_{i-1}, h_{i-1} \oplus h_i)$ and $(k_{i-1}, t_i, h_{i-1} \oplus \theta_1, h_{i-1} \oplus h_i \oplus \theta_1)$. We now consider the probability that a two-block collision happens. For $1 \leq j \leq q$, let C_j be the event that a two-block collision occurs for F at the j-th pair of queries. It implies that for some $0 \leq j' < j$,

$$(x_1, x_2) = (y_1, y_2)$$

[3] This is when there is no AD, otherwise (h_1, k_1) is enough and the first auxiliary values will be generated when processing the last blocks of AD.

where $(x_1, x_2) \in \{(h_j, k_j), (h_j \oplus \theta_1, k_j), \ldots, (h_j \oplus \theta_d, k_j)\}$ and $(y_1, y_2) \in \{(h_{j'}, k_{j'}), (h_{j'} \oplus \theta_1, k_{j'}), \ldots, (h_{\ell'} \oplus \theta_d, k_{\ell'})\}$. This is the same as (h_j, k_j) belonging to

$$\{(h_{j'}, k_{j'})\} \cup \{(h_{j'} \oplus \theta_i, k_{j'}) : 1 \le i \le d\} \cup \{(h_{j'} \oplus \theta_{i_1} \oplus \theta_{i_2}, k_{j'}) : 1 \le i_1 < i_2 \le d\}.$$

From the assumption $\theta_1, \ldots, \theta_d = 1, \ldots, d$, it holds $3 \le \theta_{i_1} + \theta_{i_2} \le 2d - 1$ for any $1 \le i_1 < i_2 \le d$. Hence the total number of elements in above three sets is at most $2d$ that counts from $(h_{j'}, k_{j'})$ to $(h_{j'} \oplus (2d-1), k_{j'})$. Hence

$$\Pr[C_j] \le \frac{2dj}{(2^n - (2j-2))(2^n - (2j-1))} \le \frac{2dj}{(2^n - (2j-1))^2}$$

since h_j and k_j are chosen uniformly at random without replacement from a set of size at least $2^n - (2j-2)$. Finally, we have

$$\mathsf{Adv}_H^{\mathsf{tcoll}}(\mathcal{A}) \le \sum_{j=1}^{q} \Pr[C_j] \le \sum_{j=1}^{q} \frac{2dj}{(2^n - (2j-1))^2} \le \frac{8dq^2}{2^{2n}}$$

by assuming $q \le 2^{n-2}$ which concludes the proof. □

Remark 1. In CHES 2022 [SPS+22], Shen et al. proved the collision resistance of Hirose's double-block-length hash function based on a TBC, which can be seen as a special case of our lemma when $d = 2$. Their construction can only output a $2n$-bit state since it inherits from Hirose's construction.

Remark 2. Since Multihash only forwards a $2n$-bit state to the next iteration, it is collision-resistant up to $O(2^n)$ queries. While it is enough in many cases, we introduce another multi-block hash function called MBLhash as a side contribution in Appendix C. It achieves a better collision-resistance security up to $O(2^{(d+1)n/2})$ queries. Its internal compression function forwards a $(d+1)n$-bit state to the next iteration and absorbs a single block at a time. This provides an interesting tradeoff with respect to Multihash. While the absorption has a smaller rate, the output length has been increased logarithmically in the total amount of queries it can support. It is also based on a TBC with n-bit key and dn-bit tweak, aims at hashing a message with a high collision-resistance security and cannot be simply turned into an AE scheme.

4 Design and Specification of Multiplex

In this section, we give the design rationale and specification of Multiplex. In general, Multiplex enjoys $n - \log_2(dn)$ bits of CIML2 security in the unbounded leakage model with a leveled implementation and $n - \log_2(dn)$ bits of confidentiality without leakage in the nonce misuse-resilient setting. It also provides $n/2$-bit CCAmL1 security, a standard confidentiality guarantee with leakage due to a re-keying process for the bulk of the computations. Moreover, the rate (here defined as the number of n-bit message blocks processed per number of TBC calls) gets closer to 1 as the tweak size increases.

High-level structure. Multiplex follows the general 3-step framework of leakage-resistant AE modes suggested in [BBC+20]. It is based on a TBC with a dn-bit tweak, and consists of three parts, including key-derivation function (KDF), message-processing function (MPF) and tag-generation function (TGF). KDF is used to generate an initial $2n$-bit state (when AD is not the empty string) from a long-term master key K and a nonce N. MPF makes several iterations of Hirose's compression function Hirose in order to absorb the associated data, ending with a single call to its extended version F to create a

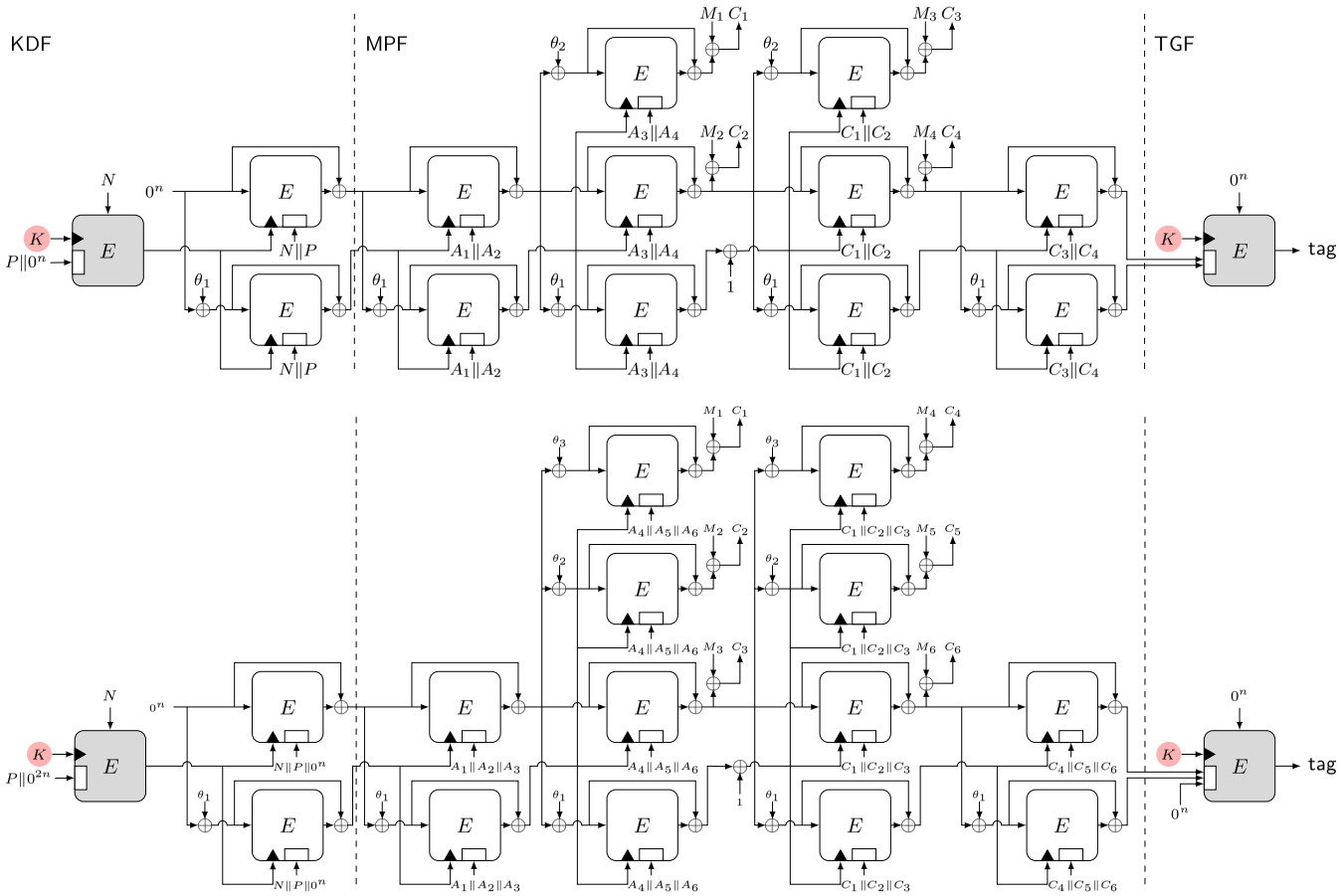

Figure 5: Multiplex's encryption when the tweak of a TBC is $2n$ bits (top) and when the tweak of a TBC is $3n$ bits (bottom). In both cases, the first 3 TBC calls represent the key-derivation function (KDF) that produces a $2n$-bit initial state. The middle part is the message-processing function (MPF) that encrypts and authenticates message. The last TBC call is the tag-generation function (TGF) that generates the tag. Only the two TBC calls colored in gray must be protected against DPA.

first full state of $d+1$ blocks to start processing the message, as sketched in the previous section. More precisely, MPF goes on with Multihash using the last random short state (h_{v+1}, h_{v+1}) of this first full state as IV (with an extra separation bit). This process encrypts message blocks and hashes ciphertext blocks. Finally, TGF encapsulates the final short state, i.e., the digest of Multihash, to compute a tag for the associated data and message. The pictorial illustration of Multiplex can be found in Figure 5. Below we give a more detailed description of these three parts.

Key-derivation function. The KDF takes as input a long-term master key K, a nonce N, and a public key P if we target multi-user security (otherwise, $P = 0^n$ in the single-key setting). As illustrated on the left of Figure 5, it first invokes a (DPA-protected) TBC to produce an ephemeral key k_0 and expands it into the $2n$-bit state $(h_1, k_1) = \mathsf{Hirose}(0^n, k_0, N||P||0^*)$ by two TBC calls. The public key P can be a public random string or simply a unique ID of each user, which is used to resist key-collision attack in the multi-user scenario.

Message-processing function. The MPF (illustrated on the middle of Figure 5) can be regarded as the hash function Multihash introduced in Section 3. It uses the $2n$-bit state (h_1, k_1) from KDF as the initial value, and continues generating a $2n$-bit state (h_i, k_i) by absorbing data where h_i is used as an input and k_i as a key to a TBC. Note that for each iteration that may require invoking a TBC several times, distinct constants $\theta_1, \ldots, \theta_d$ are XORed to h_i in order to force inputs being distinct while the key k_i remains the same.

The MPF first handles the associated data A and then the message M. To authenticate the associate data A, it simply puts them as the tweak to a TBC, following the intuition that the tweak is public but can significantly influence the output of a TBC. For each iteration, it requires two TBC calls to authenticate d blocks of associated data. In the last iteration of associated data, besides two TBC calls for authentication, $(d-1)$ additional TBC calls are invoked to generate the first *full* state, leading to a $(d+1)n$-bit random state $(h_i, k_i, e_i^1, \ldots, e_i^{d-1})$. A constant '1' is then XORed with k_i for domain separation between associated data and message. To encrypt and authenticate, it first uses the previous dn-bit state $(h_i, e_i^1, \ldots, e_i^{d-1})$ to XOR-encrypt d blocks of message. Then it authenticates the resulting d blocks of ciphertext and generates a $(d+1)n$-bit state for the next iteration. Note that for the $(d+1)n$-bit state $(h_i, k_i, e_i^1, \ldots, e_i^{d-1})$, only k_i as a key of a TBC should always be kept secret and cannot be used to XOR-encrypt the message, while the rest of the dn-bit state can XOR-encrypt the message since the disclosure of these values will not affect the output randomness of a TBC. Hence, it requires $d+1$ TBC calls to authenticate and encrypt d blocks of message. The additional $(d-1)$ TBC calls in the last iteration of associated data are compensated in the last iteration of message that only requires two TBC calls for authentication and outputs a $2n$-bit state.

Tag-generation function. Finally, the TGF (illustrated on the right of Figure 5) encapsulates the $2n$-bit state as a tweak of a TBC following the recent LR-MAC in [BGPS21] to produce a tag. It requires a TBC call with a long-term master key and a fixed-constant input 0^n. To check the validity of the tag in decryption, it is recommended to inverse this TBC call and check whether the output equals to the constant 0^n. This inversion avoids leaking information on the right tag given any adversarially chosen invalid ciphertexts, as formalized in [BPPS17].

4.1 Specification of Multiplex

The code description of Multiplex is detailed in Figure 6 (Appendix A) and illustrated in Figure 5 (above). Its parameter configurations are in Table 1.

Table 1: Parameters of Multiplex based on a TBC with n-bit key and dn-bit tweak. For example, the underlying TBC can be instantiated with Skinny-384 [BJK+16] for 121-bit security. The rate can be further improved when instantiated with larger tweak TBCs, e.g., Deoxys-TBC-512 for rate 3/4 and Deoxys-TBC-640 [CJPS22] for rate 4/5.

Parameters	General n	$n = 128$ (e.g., Skinny-384)
Secret key	n bits	128 bits
Public key[4]	n bits	128 bits
Tweak size	dn bits	256 bits
Nonce size	n bits	128 bits
Block length	max $2^n/n$	2^{95} GB
Tag size	n bits	128 bits
Security level	$n - \log_2(dn)$ bits	120 bits
Rate	$d/(d+1)$	2/3

Padding Method. For a TBC with dn-bit tweak and n-bit key, the padding function first appends a single 1 and then the smallest number of 0s to the plaintext M such that the length of the padded plaintext is a multiple of dn bits (since in each iteration, it can handle a dn-bit string). The padded plaintext is parsed into $d\ell$ blocks of n bits where $\ell = \lceil (|M| + 1)/dn \rceil$, namely $M[1] \parallel \ldots \parallel M[d\ell]$ where $|M[i]| = n$. The same padding function is applied to parse the associated data A into dv blocks of n bits where $v = \lceil (|A| + 1)/dn \rceil$, namely $A[1] \parallel \ldots \parallel A[dv]$ where $|A[i]| = n$, except if the associated data A is empty. In this case, no padding is required and no associated data is processed. Formally, for any $M \in \{0,1\}^*$ and $A \in \{0,1\}^*$, we have

$$
\begin{aligned}
M[1] \parallel \ldots \parallel M[d\ell] \leftarrow \mathrm{pad}(M) &= M \parallel 1 \parallel 0^{dn-1-(|M| \bmod dn)}, \\
A[1] \parallel \ldots \parallel A[dv] \leftarrow \mathrm{pad}(A) &= \begin{cases} A \parallel 1 \parallel 0^{dn-1-(|M| \bmod dn)} & \text{if } |A| > 0, \\ \varepsilon & \text{if} |A| = 0. \end{cases}
\end{aligned}
$$

Comparison with Triplex. As a non-trivial extension of Triplex [SPS+22], Multiplex is a more flexible design with a rate that can approach 1. Moreover, for each iteration of message-processing function, the $(d + 1)$ TBC calls can be implemented in parallel. These features cannot be achieved by Triplex or a naive generalization of Triplex. Here we provide a comparison between Multiplex and Triplex, from the perspective of both design and security analysis.

Regarding the design, Multiplex uses the same key-derivation function and tag-generation function as Triplex, due to the need of a $2n$-bit initial value and good leakage-resistance properties. Yet, the difference emerges in the message processing function that is the main part of an AEAD scheme. Given a TBC with $2n$-bit tweak, for each iteration of the message-processing function in Triplex, a TBC call should be first invoked with the tweak $N \parallel P$ to encrypt a message block M_i to a ciphertext block C_i. The rest of the two TBC calls can then be invoked with the tweak $C_{i-1} \parallel C_i$, where the left half tweak C_{i-1} comes from the previous iteration. By contrast, for Multiplex, these three TBC calls can be invoked at the same time with the same tweak $C_{i-1} \parallel C_i$ that both come from the previous iteration. Hence, these $(d + 1)$ TBC calls in Multiplex can be implemented in parallel and are more suitable for a general tweak size. We note that a naive generalization of Triplex with dn-bit tweaks cannot achieve this goal, and Triplex is initially designed to have a rate 2/3. Besides, the different tweak $N \parallel P$ used in the first TBC call of such a naive generalization would require additional memory, which is also in contrast with

[4]Note that this public key is optional and is only used in order to improve the security against key-collision attacks in the multi-user scenario.

Multiplex that deals with N only in the initial phase of its KDF, and can send and erase the nonce before the message processing starts. Last but not least, while the better rate of Multiplex than 2/3 of Triplex may come with the compensation on the increased number of rounds in the underlying larger tweak TBC, the flexible rate of Multiplex indeed cannot be achieved by the original Triplex, and may become more helpful when more efficient TBC with larger tweak is designed in the future.

On the other hand, due to construction differences, Multiplex requires different security analyses than Triplex for n-bit security. We highlight some differences here and refer to the next sections for detailed analyses. Firstly, it requires an extended collision lemma (shown in Section 3) to capture two-block collisions among $(d+1)$ TBC calls. Secondly, since the first TBC call in each iteration of the message-processing function uses the same tweak as the other d TBC calls instead of $N \| P$, we cannot simply rely on the uniqueness of nonce N and randomness of P as in the case of Triplex. Instead, we consider the influence of collisions among these tweaks. Eventually, the proof of Multiplex is more general and works for any $d \geq 1$. The proof of Triplex only focuses on the case $d = 2$.

Comparison with Skinny-Hash. Skinny-Hash [BJK+20] also allows a flexible rate by instantiating the permutation of sponge mode with several TBC calls. Hence, we next discuss the better rate of Multiplex compared to Skinny-Hash.

Firstly, as a hash function, Skinny-Hash (here we focus on Skinny-tk3-Hash and its generalization since it has better rate than Skinny-tk2-Hash), requires $(d+1)$ TBC calls where the outputs of $(d-1)$ TBC calls are used to XOR $(d-1)$ n-bit blocks of message and the outputs of 2 TBC calls are kept secret to maintain n-bit security. Hence, the rate of Skinny-Hash is $(d-1)/(d+1)$ when instantiated with a TBC with dn-bit tweak and n-bit key (so totally the tweakey size is $(d+1)n$ bits). By contrast, for the underlying hash function in Multiplex (e.g., Multihash without the auxiliary TBC calls for e^1, ..., e^{d-1} in Section 3, as they are not required for hashing) the rate is $d/2$, as it uses only two TBC calls in each iteration for hashing d n-bit blocks of message. The rate $d/2$ is always larger than $(d-1)/(d+1)$ for $d \geq 1$, and their ratio increases with d.

Secondly, Skinny-Hash is initially designed only as a hash function and we cannot use it as an AEAD. If we want to turn Skinny-Hash into an AEAD scheme, one option is to apply the transformation of Skinny-Hash into a duplex mode, namely first building a permutation from several TBC calls and then use it in a duplex AEAD mode. We remark that the rate of this naive method is at most $(d-1)/(d+1)$ for both associated data and message, while for Multiplex, the rate for associated data is $d/2$ and the rate for message is $d/(d+1)$.

In addition, this simple method ignores the important role of key-derivation function (KDF) and tag-generation function (TGF). In Multiplex, we plug Multihash with well-designed KDF and TGF. Regardless of what the rate is, the KDF requires three TBC calls and the TGF requires one TBC. Moreover, they are designed to be well-suited against side-channel attacks since only the first TBC and the last TBC call should be DPA secure. By contrast, if we apply the idea of Skinny-Hash to some side-channel protection friendly duplex mode such as Ascon, then both KDF and TGF require $(d+1)$ TBC calls, and these $2(d+1)$ TBC calls should be well protected against side-channel attacks.

5 Confidentiality Analysis of Multiplex

In this section, we provide the confidentiality analysis for Multiplex in the nonce misuse-resilience setting. We also discuss how the techniques of [BGP+20, GPPS20] can be applied to Multiplex to obtain CCAmL1 security at the end of the section.

Nonce-misuse resilience. The following theorem shows that Multiplex is beyond-birthday-bound secure in the nonce-misuse resilience setting. Recall that the security game

is illustrated in Figure 1. Note that here we only give the CPA security analysis as the CCA security of an AE scheme can be trivially obtained by using the standard argument with the authenticity security that is proved in the next section.

Theorem 1. *Suppose that the adversary makes at most q encryption queries, p ideal TBC queries, the total number of primitive calls among these q encryption queries being at most σ and the total number of queried users being at most u. Then we have*

$$\mathsf{Adv}^{\mathrm{priv}}_{\mathsf{Multiplex}}(\mathcal{A}) \leq \frac{(3c_1+dn+3n)p+(3c_1+dn+n)\sigma+c_1 q}{2^n} + \frac{u^2+16d(\sigma+p)^2+8q}{2^{2n+1}} + \frac{2q^2+q(\sigma+p)}{2^{3n}} + \frac{2}{2^{n/2}}$$

with $c_1 = \max\{4n, 4u/2^n\}$, assuming $q \leq 2^{n-1}$, $\sigma \leq 2^{n-3}$, and $p+\sigma \leq 2^{n-1}$.

Discussion and proof ideas. This theorem suggests that as long as the total number of primitive calls σ and the number of offline TBC queries p does not go beyond $2^n/n$, Multiplex can ensure confidentiality. The total number of users can also be as large as 2^n. The proof essentially relies on the fact that the outputs of Multiplex are indistinguishable from random strings as long as there is no two-block collision on (h_i, k_i). Since the nonce is always unique during the second encryption phase, this two-block collision happens with probability about $8d(\sigma+p)^2/2^{2n}$ by Lemma 3. For a (key, tweak) collision between the first TBC call in the KDF part and direct TBC calls or internal TBC calls, this happens with probability about $(c_1 p + c_1\sigma)/2^n$ for some constant c_1, since the multiplicities of the public key P_i can be bounded by Lemma 1. For a (key, tweak) collision between direct TBC calls and internal TBC calls, the idea is similar since the maximum number of state values with the same h_i can be bounded by Lemma 2. For a (key, tweak) collision between TGF part and direct TBC calls or internal TBC calls, this happens with probability about $q(\sigma+p)/2^{3n}$ since both the tweak and the key of the TGF are random. For a (key, tweak) collision between the first TBC call in KDF and TGF, this happens with probability about $q/2^{2n} + q^2/2^{3n}$ since the tweak of TGF is random and the keys between two different users are independently random. Regarding the key collision among many users, this happens with probability about $u^2/2^{2n}$ due to the use of a public key P_i. The formal proof contains more details and explanations.

Proof. In the security game of Figure 1, the adversary has access to three oracles, including the first encryption oracle where the nonce may be repeated, the second encryption oracle where the nonce is always fresh and unique, and the ideal TBC oracle that captures the power of offline computations. Note that the first encryption oracle and ideal TBC oracle behave exactly the same in both of the real and ideal worlds. Therewith our goal is to show that the adversary cannot tell apart the outputs of the second encryption oracle in the real world from those outputs in the ideal world, except with a negligible probability.

From the interaction with its oracles, the adversary can obtain the information that are recorded as follows.

- For each query $\textsc{Prim}(J, T, (+, x))$ with answer y, we will store an entry $(\mathsf{prim}, J, T, x, y, +)$. Similarly, for each query $\textsc{Prim}(J, T, (-, y))$ with answer x, we will store an entry $(\mathsf{prim}, J, T, x, y, -)$.
- For each query $C \,\|\, \mathsf{tag} \leftarrow \textsc{Enc}_1(i, N, A, M)$, let $A[1] \,\|\, \ldots \,\|\, A[dv] \leftarrow A$, $M[1] \,\|\, \ldots \,\|\, M[d\ell] \leftarrow M$ and $C[1] \,\|\, \ldots \,\|\, C[d\ell] \leftarrow C$. Let

$$h_0 = 0^n, k_0 = E_{K_i}(P_i \,\|\, 0^{(d-1)n}, N) \ ,$$
$$h_1 = E_{k_0}(N \,\|\, P_i \,\|\, 0^{(d-2)n}, 0^n), k_1 = E_{k_0}(N \,\|\, P_i \,\|\, 0^{(d-2)n}, \theta_1) \oplus \theta_1 \ ,$$

and for $1 \leq j \leq v$, let

$$h_{j+1} = E_{k_j}(A[dj-d+1] \,\|\, \ldots \,\|\, A[dj], h_j) \oplus h_j \ ,$$
$$k_{j+1} = E_{k_j}(A[dj-d+1] \,\|\, \ldots \,\|\, A[dj], h_j \oplus \theta_1) \oplus h_j \oplus \theta_1 \ .$$

For $1 \leq r \leq d-1$, let $e_1^r = E_{k_v}(T, h_v \oplus \theta_{r+1}) \oplus h_v \oplus \theta_{r+1}$ where $T = A[dv-d+1] \,\|\, \ldots \,\|\, A[dv]$ if AD is not empty otherwise $T = N \,\|\, P_i \,\|\, 0^{(d-2)n}$. Let $k_{v+1} = k_{v+1} \oplus 1$, and for $1 \leq j \leq \ell - 1$, let

$$h_{v+j+1} = E_{k_{v+j}}(C[dj-d+1] \,\|\, \ldots \,\|\, C[dj], h_{v+j}) \oplus h_{v+j} \ ,$$
$$k_{v+j+1} = E_{k_{v+j}}(C[dj-d+1] \,\|\, \ldots \,\|\, C[dj], h_{v+j} \oplus \theta_1) \oplus h_{v+j} \oplus \theta_1 \ ,$$
$$e_{j+1}^r = E_{k_{v+j}}(C[dj-d+1] \,\|\, \ldots \,\|\, C[dj], h_{v+j} \oplus \theta_{r+1}) \oplus h_{v+j} \oplus \theta_{r+1}$$

for $1 \leq r \leq d-1$. Let

$$h_{v+\ell+1} = E_{k_{v+\ell}}(C[d\ell-d+1] \,\|\, \ldots \,\|\, C[d\ell], h_{v+\ell}) \oplus h_{v+\ell} \ ,$$
$$k_{v+\ell+1} = E_{k_{v+\ell}}(C[d\ell-d+1] \,\|\, \ldots \,\|\, C[d\ell], h_{v+\ell} \oplus \theta_1) \oplus h_{v+\ell} \oplus \theta_1 \ .$$

We will store an entry $(\mathsf{enc}_1, i, N, A, M, C \,\|\, \mathsf{tag})$. We additionally use the entry $(\mathsf{inter}, J, T, x, y)$ to track the internal primitive calls during the computation of this query (except the first and last TBC calls). These additional entries are used for the analysis and invisible to the adversary.

- For each query $C \,\|\, \mathsf{tag} \leftarrow \textsc{Enc}_2(i, N, A, M)$, similarly to the above case, we will store an entry $(\mathsf{enc}_2, i, N, A, M, C \,\|\, \mathsf{tag})$ and use the entry $(\mathsf{inter}, J, T, x, y)$ to track the internal primitive calls.

We then define some bad events, and show that the outputs in the real world behave the same as random strings conditioned on the fact that none of these bad events occur. Let $h_{i,b}^a \,\|\, k_{i,b}^a$ be the b-th internal state for the a-th query to user i. Let $\mathsf{parent}(h_{i,b}^a \,\|\, k_{i,b}^a)$ be a sequence of values that point to $h_{i,b}^a \| k_{i,b}^a$ at the a-th query to user i, with $\mathsf{parent}(h_{i,1}^a \| k_{i,1}^a) = (h_{i,0}^a \,\|\, k_{i,0}^a, N_i^a \,\|\, P_i)$, $\mathsf{parent}(h_{i,b}^a \,\|\, k_{i,b}^a) = (h_{i,0}^a \,\|\, k_{i,0}^a, N_i^a \,\|\, P_i, A_i^a[1] \,\|\, \ldots \,\|\, A_i^a[d], \ldots, A_i^a[d(b-2)+1] \,\|\, \ldots \,\|\, A_i^a[d(b-1)])$ if $2 \leq b \leq v_i^a + 1$, and $\mathsf{parent}(h_{i,b}^a \,\|\, k_{i,b}^a) = (h_{i,0}^a \,\|\, k_{i,0}^a, N_i^a \,\|\, P_i, A_i^a[1] \,\|\, \ldots \,\|\, A_i^a[d], \ldots, A_i^a[dv_i^a - d + 1] \,\|\, \ldots \,\|\, A_i^a[dv_i^a], M_i^a[1] \,\|\, \ldots \,\|\, M_i^a[d], \ldots, M_i^a[d(b-v_i^a-2)+1] \,\|\, \ldots \,\|\, M_i^a[d(b-v_i^a-1)])$ if $v_i^a + 1 < b \leq v_i^a + \ell_i^a + 1$. We then give the definition of the first bad event bad_1. This event consists of several bad conditions, and is to ensure that the output of each query behaves like a random string when none of these conditions are triggered. The intuition of each bad condition will be explained when described. We say that the event bad_1 happens if at least one of the following conditions is violated:

(1) There are two users i and j ($i \neq j$) such that $K_i = K_j$ and $P_i = P_j$. This is to avoid key collisions among u users.

(2) There is a P_i that repeats at least c_1 times among u users. This helps to analyze other bad conditions by putting a threshold on the maximal repeated times of a public P_i.

(3) There is an ideal TBC entry $(\mathsf{prim}, J, T, x, y, *)$ or an internal primitive call $(\mathsf{inter}, J, T, x, y)$ such that $J = K_i$ and $T = P_i \,\|\, 0^{(d-1)n}$ for some user i. This is to avoid the input collision (including the secret key and tweak) of the first TBC call with that of the ideal TBC queries or internal primitive calls.

(4) There is an entry $(\mathsf{enc}_2, i, N_i^a, A_i^a, M_i^a, C_i^a \,\|\, \mathsf{tag}_i^a)$ such that $k_{i,0}^a = k_{j,0}^b$ and $N_i^a \,\|\, P_i = N_j^b \,\|\, P_j$ for some other entry $(\mathsf{enc}_*, j, N_j^b, A_j^b, M_j^b, C_j^b \,\|\, \mathsf{tag}_j^b)$ of a different user j. This is to ensure that even when (N, A, M) may repeat across two users, the initial input $(k_0, N \,\|\, P)$ remains different, therefore avoiding trivial collisions for the hash function $\mathsf{Multihash}$.

(5) There is an entry $(\mathsf{enc}_2, i, N_i^a, A_i^a, M_i^a, C_i^a \,\|\, \mathsf{tag}_i^a)$ such that $k_{i,0}^a = J$ and $N_i^a \,\|\, P_i \,\|\, 0^{(d-2)n} = T$ for some ideal TBC query $(\mathsf{prim}, J, T, x, y, *)$ or an internal primitive call $(\mathsf{inter}, J, T, x, y)$. This is to ensure that for each encryption query to $\textsc{Enc}_2$, the initial input $k_{i,0}^a$ and $N_i^a \,\|\, P_i \,\|\, 0^{(d-2)n}$ are always fresh from that of ideal TBC queries and internal primitive calls and thus provide enough randomness for the following iterations.

(6) There is an entry $(\mathsf{enc}_2, i, N_i^a, A_i^a, M_i^a, C_i^a \,\|\, \mathsf{tag}_i^a)$ such that $\mathsf{parent}(h_{i,b}^a \,\|\, k_{i,b}^a) \neq \mathsf{parent}(h_{i',b'}^{a'} \,\|\, k_{i',b'}^{a'})$ and $(h_{i,b}^a, k_{i,b}^a) \in \{(h_{i',b'}^{a'}, k_{i',b'}^{a'})\} \cup \{(h_{i',b'}^{a'} \oplus \theta_r, k_{i',b'}^{a'}) : 1 \le r \le d\} \cup \{(h_{i',b'}^{a'} \oplus \theta_{r_1} \oplus \theta_{r_2}, k_{i',b'}^{a'}) : 1 \le r_1 < r_2 \le d\}$. This is to ensure that for each encryption query, the $2n$-bit internal pairs $(h_{i,b}^a, k_{i,b}^a), (h_{i,b}^a \oplus \theta_1, k_{i,b}^a), \ldots, (h_{i,b}^a \oplus \theta_r, k_{i,b}^a)$ are always fresh from other internal pairs.

(7) There is an entry $(\mathsf{enc}_2, i, N_i^a, A_i^a, M_i^a, C_i^a \,\|\, \mathsf{tag}_i^a)$ such that one of $\{h_{i,b}^a, k_{i,b}^a, h_{i,b}^a \oplus \theta_1, \ldots, h_{i,b}^a \oplus \theta_d\}$ appears at least c_2 times for $b \ge 1$. This is to put a threshold on the maximal number of repetitions of these values that is helpful for the following analysis.

(8) There is an entry $(\mathsf{enc}_2, i, N_i^a, A_i^a, M_i^a, C_i^a \,\|\, \mathsf{tag}_i^a)$ such that $(x, J) \in \{(h_{i,b}^a, k_{i,b}^a), (h_{i,b}^a \oplus \theta_1, k_{i,b}^a), \ldots, (h_{i,b}^a \oplus \theta_d, k_{i,b}^a) : b \ge 1\}$ for some ideal TBC query $(\mathsf{prim}, J, T, x, y, *)$. This is to ensure that for each encryption query, the inputs of the internal primitive calls are always fresh from those of ideal TBC queries.

(9) There is an entry $(\mathsf{enc}_2, i, N_i^a, A_i^a, M_i^a, C_i^a \| \mathsf{tag}_i^a)$ such that $(K_i, h_{i,\ell_a+v_a+1}^a \| k_{i,\ell_a+v_a+1}^a \| 0^{(d-2)n}) = (K_j, P_j \,\|\, 0^{(d-1)n})$ for some entry $(\mathsf{enc}_*, j, N_j^a, A_j^a, M_j^a, C_j^a \,\|\, \mathsf{tag}_j^a)$. This is to avoid the input collision between the first TBC call and the last TBC call.

(10) There is an entry $(\mathsf{enc}_2, i, N_i^a, A_i^a, M_i^a, C_i^a \,\|\, \mathsf{tag}_i^a)$ such that $K_i = J$ and $h_{i,\ell_a+v_a+1}^a \,\|\, k_{i,\ell_a+v_a+1}^a \,\|\, 0^{(d-2)n} = T$ for some ideal TBC query $(\mathsf{prim}, J, T, x, y, *)$ or internal primitive call $(\mathsf{inter}, J, T, x, y)$. This is to ensure that the input of last TBC call is fresh from those of ideal TBC queries and internal primitive calls.

If bad_1 does not happen, then for each entry $(\mathsf{enc}_2, i, N_i^a, A_i^a, M_i^a, C_i^a \,\|\, \mathsf{tag}_i^a)$, tag_i^a is always a n-bit random string since the key and tweak pair $(K_i, h_{i,\ell_a+v_a+1}^a \| k_{i,\ell_a+v_a+1}^a \| 0^{(d-2)n})$ of the last TBC is fresh. On the other hand, each $C_i^a[b]$ is sampled uniformly at random from a set $\{0,1\}^n \setminus S(k_{i,b-1}^a, T)$ where $S(k_{i,b-1}^a, T)$ is the set of values that have been evaluated for the TBC under the key and tweak pair $(k_{i,b-1}^a, T)$. To capture the deviation of $C_i^a[b]$ from a uniformly random string, we will first sample a value v uniformly at random from $\{0,1\}^n$, and say the bad event bad_2 happens if $v \in S(k_{i,b-1}^a, T)$ and $v \leftarrow \{0,1\}^n \setminus S(k_{i,b-1}^a, T)$ is resampled. The value v is then assigned to $C_i^a[b]$. Hence, when neither bad_1 nor bad_2 happens, the outputs in the real world are merely random strings that are independent of the queries of the adversary. According to the fundamental lemma of game playing technique [BR06],

$$\mathsf{Adv}^{\mathrm{priv}}_{\mathsf{Multiplex}}(\mathcal{A}) \le \Pr[\,\mathsf{bad}_1 \cup \mathsf{bad}_2\,] \le \Pr[\,\mathsf{bad}_1\,] + \Pr[\,\mathsf{bad}_2 \mid \neg\mathsf{bad}_1\,] \ .$$

These two bad events will be bounded in Lemma 4 and Lemma 5 by

$$\frac{u^2 + 16d(\sigma+p)^2 + 8q}{2^{2n+1}} + \frac{(3c_1 + dn + 3n)p + (3c_1 + dn + n)\sigma + c_1 q}{2^n} + \frac{2q^2 + q(\sigma+p)}{2^{3n}} + \frac{2}{2^{n/2}},$$

and thus complete the proof. □

Lemma 4. *Assume that the adversary makes at most q encryption queries, p ideal TBC queries, with the total number of primitive calls among these q encryption queries being at most σ and the number of queried users being at most u, we have*

$$\Pr[\,\mathsf{bad}_1\,] \leq \frac{u^2 + 16d(\sigma+p)^2 + 8q}{2^{2n+1}} + \frac{3c_1(p+\sigma) + c_1 q + 2np}{2^n} + \frac{2q^2 + q(\sigma+p)}{2^{3n}} + \frac{2}{2^{n/2}}$$

Proof. The event bad_1 consists of several conditions. Let cond_i be the sub-event that the i-th condition is triggered. We analyze each condition in turn.

For cond_1, both K_i and P_i are uniformly random strings. Thus the probability that $K_i = K_j$ and $P_i = P_j$ is exactly $1/2^{2n}$. Summing this over $\binom{u}{2}$ pairs of (i,j),

$$\Pr[\,\mathsf{cond}_1\,] \leq \frac{u^2}{2^{2n+1}} \ .$$

Next, we analyze the condition cond_2. Recall that each P_i is chosen uniformly at random from the set $\{0,1\}^n$. Let $c_1 = \max\{4n, 4u/2^n\}$, and by the balls-into-bins result of Lemma 1,

$$\Pr[\,\mathsf{cond}_2\,] \leq \frac{1}{2^n} \ .$$

We then bound the condition cond_3. Conditioned on $\neg\mathsf{cond}_2$, for each ideal TBC query $(\mathsf{prim}, J, T, x, y, *)$ or internal primitive call $(\mathsf{inter}, J, T, x, y)$, there are at most c_1 users such that $P_i \parallel 0^{(d-1)n} = T$. On the other hand, the probability that $K_i = J$ for any of these c_1 users is $1/2^n$ since K_i is a uniformly random string. Summing this over at most p ideal TBC queries and σ primitive calls,

$$\Pr[\,\mathsf{cond}_3\,] \leq \frac{c_1 p + c_1 \sigma}{2^n} \ .$$

Next, we analyze the condition cond_4. For each $N_i^a \parallel P_i$, there are at most $c_1 - 1$ other users such that $N_j^b \parallel P_j = N_i^a \parallel P_i$ due to $\neg\mathsf{cond}_2$. On the other hand, conditioned on $\neg\mathsf{cond}_1$, $K_i \neq K_j$ when $P_i = P_j$. Thus the probability that $k_{i,0}^a = k_{j,0}^b$ is $1/2^n$ since they are outputs of two TBCs with different keys. Summing over at most q encryption queries,

$$\Pr[\,\mathsf{cond}_4\,] \leq \frac{(c_1 - 1)q}{2^n} \ .$$

Next, we analyze the condition cond_5. Conditioned on $\neg\mathsf{cond}_2$, for each ideal TBC query $(\mathsf{prim}, J, T, x, y, *)$ or internal primitive call $(\mathsf{inter}, J, T, x, y)$, there are at most c_1 encryption queries $(\mathsf{enc}_2, i, N_i^a, A_i^a, M_i^a, C_i^a \parallel \mathsf{tag}_i^a)$ such that $N_i^a \parallel P_i \parallel 0^{(d-2)n} = T$. The probability that $k_{i,0}^a = J$ is at most $1/(2^n - q)$ for any of these encryption queries since $k_{i,0}^a$ is distributed uniformly at random in a set of size at least $2^n - q$. Summing over at most p ideal TBC queries and σ internal primitive calls,

$$\Pr[\,\mathsf{cond}_5\,] \leq \frac{c_1(p+\sigma)}{2^n - q} \leq \frac{2c_1(p+\sigma)}{2^n}$$

by assuming $q \leq 2^{n-1}$.

Next, we analyze the condition cond_6. This condition implies that the adversary found a two-block collision on the hash function $\mathsf{Multihash}$ that can be reduced to the underlying compression function F by at most $\sigma + p$ TBC queries. On the other hand, conditioned on $\neg\mathsf{cond}_4$, this collision is not trivial. Hence from Lemma 3,

$$\Pr[\,\mathsf{cond}_6\,] \leq \frac{8d(\sigma+p)^2}{2^{2n}} \ .$$

We then consider the condition cond_7. Conditioned on $\neg\mathsf{cond}_6$, each pair of $h_{i,b-1}^a \| k_{i,b-1}^a$ is fresh for $b \geq 2$. In the case of $b = 1$, the pair $(k_{i,0}^a, N_i^a \parallel P_i)$ is fresh conditioned on

$\neg\mathsf{cond}_4$ and $\neg\mathsf{cond}_5$. Thus each of corresponding outputs $\{h^a_{i,b}, h^a_{i,b} \oplus \theta_1, \ldots, h^a_{i,b} \oplus \theta_d, k^a_{i,b}\}$ is chosen uniformly at random from a set of size at least $2^n - \sigma - p$. The probability that each of them hits some particular value is at most $1/(2^n - \sigma - p) \le 1/2^{n-1}$ by assuming $\sigma + p \le 2^{n-1}$. Fix $c_2 = n$ and assume $\sigma \le 2^{n-3}$. By the biased balls-into-bins result of Lemma 2,

$$\Pr[\,\mathsf{cond}_7\,] \le \frac{2}{2^{n/2}} \ .$$

Next, we analyze the condition cond_8. Conditioned on $\neg\mathsf{cond}_7$, for each ideal TBC query $(\mathsf{prim}, J, T, x, y, *)$, there are at most n values of $v \in \{h^a_{i,b}, h^a_{i,b} \oplus \theta_1, \ldots, h^a_{i,b} \oplus \theta_d : b \ge 1\}$ such that $v = x$. The probability that $k^a_{i,b} = J$ for any of these $(v, k^a_{i,b})$ is at most $1/(2^n - \sigma - p) \le 1/2^{n-1}$ by assuming $\sigma + p \le 2^{n-1}$. Summing over at most p ideal TBC queries,

$$\Pr[\,\mathsf{cond}_8\,] \le \frac{2np}{2^n} \ .$$

Next, we analyze the condtion cond_9. Conditioned on $\neg\mathsf{cond}_6$, both h^a_{i,ℓ_a+v_a+1} and k^a_{i,ℓ_a+v_a+1} are chosen uniformly at random from a set of size at least $2^n - \sigma - p$. Hence the probability that $h^a_{i,\ell_a+v_a+1} = P_j$ and $k^a_{i,\ell_a+v_a+1} = 0^n$ is at most $1/(2^n - \sigma - p)^2 \le 1/2^{2n-2}$ by assuming $\sigma + p \le 2^{n-1}$. We consider two cases:

- **Case 1:** $i = j$, and thus $K_i = K_j$. By summing over at most q encryption queries, the probability corresponding to this case is at most $4q/2^{2n}$.
- **Case 2:** $i \ne j$, and thus the conditional probability that $K_i = K_j$ is $1/2^n$. Summing over all pairs of encryption queries, we obtain a bound $2q^2/2^{3n}$.

Summing up,

$$\Pr[\,\mathsf{cond}_9\,] \le \frac{4q}{2^{2n}} + \frac{2q^2}{2^{3n}} \ .$$

Finally we consider the condition cond_{10}. Similarly to cond_9, the probability that $h^a_{i,\ell_a+v_a+1} \,\|\, k^a_{i,\ell_a+v_a+1} \,\|\, 0^{(d-2)n} = T$ is at most $1/2^{2n-2}$ by assuming $\sigma + p \le 2^{n-1}$. On the other hand, the conditional probability that $K_i = J$ is $1/2^n$ since K_i is a random string. Summing over q encryption queries, $\sigma + p$ ideal TBC queries and internal primitive calls, we get

$$\Pr[\,\mathsf{cond}_{10}\,] \le \frac{q(\sigma + p)}{2^{3n}} \ .$$

Thus totally,

$$\begin{aligned}\Pr[\,\mathsf{bad}_1\,] \quad \le \quad & \frac{u^2 + 16d(\sigma+p)^2 + 8q}{2^{2n+1}} + \frac{3c_1(p+\sigma) + c_1 q + 2np}{2^n} \\ & + \frac{2q^2 + q(\sigma+p)}{2^{3n}} + \frac{2}{2^{n/2}} \ .\end{aligned}$$

□

Lemma 5. $\Pr[\,\mathit{bad}_2 \mid \neg\mathit{bad}_1\,] \le \frac{(d+1)n\sigma + (d+1)np}{2^n}$.

Proof. Denote by $p(J,T)$ the number of ideal TBC queries that are issued by the adversary under the pair of key and tweak (J,T) and thus $\sum_{J\in\mathcal{K}, T\in\mathcal{T}} p(J,T) = p$. Recall that $S(k^a_{i,b-1}, T)$ is the set of values that have been sampled for the TBC under the pair of key and tweak $(k^a_{i,b-1}, T)$ during the computation of encryption queries. Conditioned on $\neg\mathsf{bad}_1$, the size of $S(k^a_{i,b-1}, T)$ is at most $(d+1)n + p(k^a_{i,b-1}, T)$ since each $k^a_{i,b-1}$ repeats at most n times and is used $(d+1)$ times during each iteration. Hence for each $C^a_i[b]$, the

probability that $v \in S(k^a_{i,b-1}, T)$ where $v \xleftarrow{\$} \{0,1\}^n$ is at most $((d+1)n + p(k^a_{i,b-1}, T))/2^n$. Summing over at most σ primitive calls,

$$\Pr[\,\mathsf{bad}_2 \mid \neg\mathsf{bad}_1\,] \leq \frac{(d+1)n\sigma + (d+1)np}{2^n} \ .$$

□

CCAmL1 security. We now present a heuristic analysis for the confidentiality with leakage of Multiplex. The proof follows from previous analyses for leakage-resistant modes of operation [GPPS20, BMPS21, GPPS19, BGP+20] and is standard without significant technical novelty. The security definition of CCAmL1 (CCA security with Misuse-resilience and Leakage) [GPPS19] is given in Appendix D. The main idea is that by only focusing on encryption leakages and assuming fresh nonces, each message block is encrypted with a fresh key up to the birthday bound. At a high level and following the simplified assumptions in [BBC+20], the security of Multiplex can be reduced to the Simple Power Analysis (SPA) security of a single iteration encrypting a dn-bit block of the message with a fresh key.

More formally, to prove the CCAmL1 security of Multiplex, we can resort to the hard-to-invert assumption of [YSPY10] that is also used in the proof of TEDT [BGP+20]. We briefly sketch the leakage function in encryption and challenge queries since decryption is assumed to be black-box in the CCAmL1 game. When the nonce is fresh in encryption, we can show that the initial $2n$-bit state (h_1, k_1) after KDF is random. This is because all the k_0's computed from the first protected TBC are distinct and secret random up to the birthday bound. The same holds for all the initial states (h_1, k_1) since it requires only two calls to E_{k_0} and their leakage thus remains limited. When the nonce is repeated in encryption, the adversary can easily mount a Differential Power Analysis (DPA) on the initial state by using many associated data A and messages M. Yet, the initial states for any nonce-respecting queries are independent and secret (as for challenge queries).

We then show that the following internal states remain sufficiently secret in challenge queries and thus the security is preserved. Let (h_{v+i}, k_{v+i}) be the current state and $M[di-d+1] \parallel \ldots \parallel M[di]$ the bn-bit block of message that is being processed in the computation of a challenge query. The ephemeral key k_{v+i} is used $d+1$ times in this iteration with distinct input $h_i, h_i \oplus \theta_1, \ldots, \theta_d$ for some constant d. Since k_{v+i} is not used anywhere else except with a birthday bound probability, these $d+1$ TBC calls should not leak too much information about the refreshed state $(h_{v+i+1}, k_{v+i+1}, e^1_{i+1}, \ldots, e^{d-1}_{i+1})$ which will thus be random and secret. Hence, the secrecy and randomness of an internal state propagates to the next one by (h_{v+i+1}, k_{v+i+1}). On the other hand, the computation of TGF is independent of the message processing since the last TBC call with the long-term key is protected and the final state is unique except with a negligible probability. Hence the CCAmL1 security reduces to the leakage of the one-time XOR computation of $C[di-d+1] = e^1_i \oplus M[di-d+1], \ldots, C[di-1] = e^{d-1}_i \oplus M[di-1]$, and $C[di] = h_{v+i} \oplus M[di]$, where $e^j_i = E_{k_{v+i-1}}(T, h_{v+i-1} \oplus \theta_{j+1}) \oplus h_{v+i-1} \oplus \theta_{j+1}$ for $1 \leq j \leq d-1$. Except the fact that h_{v+i} is also involved in $E_{k_{v+i}}(T, h_{v+i}) \oplus h_{v+i}, E_{k_{v+i}}(T, h_{v+i} \oplus \theta_1) \oplus h_{v+i} \oplus \theta_1, \ldots, E_{k_{v+i}}(T, h_{v+i} \oplus \theta_d) \oplus h_{v+i} \oplus \theta_d$, these XORs are the minimal number of encryption manipulations one can expect. Since h_{v+i} is involved in the computation of $E_{k_{v+i}}$ that is internal and out of the adversary's control, we can assume that little informative leaks. Thanks the hard-to-invert leakage assumption, we repeat this argument until reaching the final state as the leakages between two iterations are independent.

6 Authenticity Analysis of Multiplex

Next, we provide the authenticity analysis of Multiplex in the leakage setting.

CIML2 SECURITY. The analysis of CIML2 security is done in the unbounded leakage model with a leveled implementation [BKP+18, BPPS17]. In this model, the leakage functions are allowed to expose all internal states of unprotected (or weakly protected) building blocks to the adversary, while only the key of strongly protected components remains secret, still with their inputs and outputs allowed to leak. In the case of Multiplex, only the first and last TBCs used as KDF and TGF are strongly protected and require DPA-protection (based on masking for instance). All the other TBC calls involved in the message dependent computations require no side-channel protection. This is where a leveled design allows saving computations compared to a uniformly protected one that calls the protected TBC with the long term key at each call. The following result shows that Multiplex provides beyond-birthday-bound CIML2 security. Due to the page limitation, we only provide the proof ideas here and postpone the full proof of this theorem to Appendix E.

Theorem 2. *Suppose that the adversary makes at most totally q encryption and decryption queries, p ideal TBC queries, the total number of primitive calls among these q encryption and decryption queries being at most σ and the total number of queried users being at most u. Then we have*

$$\mathsf{Adv}^{\mathrm{CIML2}}_{\mathsf{Multiplex}}(\mathcal{A}) \leq \frac{u^2 + 8d(\sigma+p)^2 + 8q}{2^{2n+1}} + \frac{c(\sigma+p+q)+2q}{2^n} + \frac{2q^2+q(\sigma+p)}{2^{3n}}$$

by assuming $\sigma + p \leq 2^{n-1}$, $q \leq 2^{n-1}$, and $c = \max\{4n, 4u/2^n\}$.

DISCUSSION AND PROOF IDEAS. Multiplex ensures CIML2 security as long as the total number of primitive calls σ and the number of offline TBC calls p does not exceed $2^n/n$, and the number of users can be as large as 2^n.

The proof is based on the observation that as long as the final two blocks $(h_{\ell+v+1}, k_{\ell+v+1})$ are fresh, then it is hard for the adversary to predict the tag of Multiplex. For queries to the same user, since each tuple (N, A, C) is unique, this collision can be reduced to the two-block collision of the hash function Multihash that is captured by Lemma 3. On the other hand, although the tuple (N, A, C) may repeat among different users, the key pair (K_i, P_i) is unlikely to collide and thus avoids trivial collisions. For a (key, tweak) collision between the first TBC call in the KDF part and direct TBC calls or internal TBC calls, this happens with probability about $(cp + c\sigma)/2^n$ for some constant c since the multiplicities of P_i can be bounded by Lemma 1. For a (key, tweak) collision between the TGF part and direct TBC calls or internal TBC calls, this happens with probability about $q(\sigma + p)/2^{3n}$ since both the tweak and the key of TGF are random. For a (key, tweak) collision between the first TBC call in KDF and TGF, this happens with probability about $q/2^{2n} + q^2/2^{3n}$ since the tweak of TGF is random and the keys between two different users are independently random. Regarding the key collision among many users, this happens with probability about $u^2/2^{2n}$ with the help of a public key P_i. The formal proof is more detailed.

7 Concluding Remarks

We designed Multiplex, a TBC-based mode with primitive-rate $d/(d+1)$. Its underlying TBC uses dn-bit tweaks, where n is both the block length and the key length. Each message processing iteration of Multiplex carries $(d+1)$ blocks of state and, after XORing d blocks of message, absorbs the ciphertext blocks and refreshes the full state. That is, each TBC call in the iteration involves the same $(d+2)$-block input, only constants differ for separation. These calls can then be run in parallel in order to minimize the time of each iteration.

Besides its interest for flexible leakage-resistant modes of operation and as a good candidate for comparing TBCs and permutations, outlined in the introduction, we believe these results also open interesting research avenues. In particular, it would be worth trying to further optimize the performances of Multiplex by relying on shorter-block TBC. For instance, there is a 64-bit version of Skinny [BJK+16] that requires less rounds than the 128-bit version, still with 256-bit tweak. Of course the amount of TBC calls will double to process the same amount of message bits: since we would have to consider $n/2$-bit blocks of message, the primitive rate would approach $d/(d+2)$ if we carefully deal with h_i, since 2 calls would be necessary to produce a fresh ephemeral key. Nevertheless, the time of this double amount of TBC executions could reduce to the time of a single execution when parallelism can be exploited. This shows that a primitive with a smaller block length might decrease the overall encryption time. It would then also be interesting to evaluate the cost of the iteration. In Appendix B, we give an intuitive idea of how this *short-block* Multiplex would work for the iterations in the message-processing part. Equipping it with a suitable key-derivation function and a tag-generation function may achieve a comparable security as Multiplex. We leave it as an interesting open problem. Besides studying such an efficiency improvement, studying the black-box security of Multiplex in the standard model would also be a nice path towards comparing the security of modes with optimal primitive-rate which do not resort on idealized assumption. In this paper, we had to rely on the ideal TBC setting since, in the unbounded leakage model, proving CIML2 first required showing beyond-birthday collision-resistance in the message-processing part where the adversary is given all the internal values. However, we carefully designed Multiplex so that the key-input of each TBC call can be proven being independent through the successive iterations by relying on the pseudorandomness of the TBC.

Acknowledgments

Thomas Peters and François-Xavier Standaert are respectively associate researcher and senior research associate of the Belgian Fund for Scientific Research (F.R.S.-FNRS). Research carried out while Yaobin Shen was a post-doc at the UCL Crypto Group. This work and its presentation have been funded in parts by the ERC consolidator grant 724725 (SWORD), the ERC Advanced Grant 101096871 (BRIDGE) and Ant Research, Ant Group. Views and opinions expressed are those of the authors only and do not necessarily reflect those of the EU or the European Research Council. Neither the European Union nor the granting authority can be held responsible for them.

References

[ADL17] Tomer Ashur, Orr Dunkelman, and Atul Luykx. Boosting authenticated encryption robustness with minimal modifications. In *CRYPTO (3)*, volume 10403 of *LNCS*, pages 3–33. Springer, 2017.

[AFL+14] Farzaneh Abed, Christian Forler, Eik List, Stefan Lucks, and Jakob Wenzel. Counter-bdm: A provably secure family of multi-block-length compression functions. In *Progress in Cryptology - AFRICACRYPT 2014 - 7th International Conference on Cryptology in Africa, Marrakesh, Morocco, May 28-30, 2014. Proceedings*, pages 440–458, 2014.

[BBB+20] Davide Bellizia, Francesco Berti, Olivier Bronchain, Gaëtan Cassiers, Sébastien Duval, Chun Guo, Gregor Leander, Gaëtan Leurent, Itamar Levi, Charles Momin, Olivier Pereira, Thomas Peters, François-Xavier Standaert, Balazs Udvarhelyi, and Friedrich Wiemer. Spook: Sponge-based leakage-resistant

authenticated encryption with a masked tweakable block cipher. *IACR Trans. Symmetric Cryptol.*, 2020(S1):295–349, 2020.

[BBC+20] Davide Bellizia, Olivier Bronchain, Gaëtan Cassiers, Vincent Grosso, Chun Guo, Charles Momin, Olivier Pereira, Thomas Peters, and François-Xavier Standaert. Mode-level vs. implementation-level physical security in symmetric cryptography - A practical guide through the leakage-resistance jungle. In *CRYPTO (1)*, volume 12170 of *LNCS*, pages 369–400. Springer, 2020.

[BDPA11] Guido Bertoni, Joan Daemen, Michaël Peeters, and Gilles Van Assche. Duplexing the sponge: Single-pass authenticated encryption and other applications. In *Selected Areas in Cryptography*, volume 7118 of *LNCS*, pages 320–337. Springer, 2011.

[BGP+20] Francesco Berti, Chun Guo, Olivier Pereira, Thomas Peters, and François-Xavier Standaert. TEDT, a leakage-resist AEAD mode for high physical security applications. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2020(1):256–320, 2020.

[BGPS21] Francesco Berti, Chun Guo, Thomas Peters, and François-Xavier Standaert. Efficient leakage-resilient macs without idealized assumptions. In Mehdi Tibouchi and Huaxiong 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 *LNCS*, pages 95–123. Springer, 2021.

[BHT18] Priyanka Bose, Viet Tung Hoang, and Stefano Tessaro. Revisiting AES-GCM-SIV: multi-user security, faster key derivation, and better bounds. In *Advances in Cryptology - EUROCRYPT 2018 - 37th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Tel Aviv, Israel, April 29 - May 3, 2018 Proceedings, Part I*, pages 468–499, 2018.

[BJK+16] Christof Beierle, Jérémy Jean, Stefan Kölbl, Gregor Leander, Amir Moradi, Thomas Peyrin, Yu Sasaki, Pascal Sasdrich, and Siang Meng Sim. The SKINNY family of block ciphers and its low-latency variant MANTIS. In *CRYPTO (2)*, volume 9815 of *LNCS*, pages 123–153. Springer, 2016.

[BJK+20] Christof Beierle, Jérémy Jean, Stefan Kölbl, Gregor Leander, Amir Moradi, Thomas Peyrin, Yu Sasaki, Pascal Sasdrich, and Siang Meng Sim. SKINNY-AEAD and skinny-hash. *IACR Trans. Symmetric Cryptol.*, 2020(S1):88–131, 2020.

[BKP+18] Francesco Berti, François Koeune, Olivier Pereira, Thomas Peters, and François-Xavier Standaert. Ciphertext integrity with misuse and leakage: Definition and efficient constructions with symmetric primitives. In *Proceedings of the 2018 on Asia Conference on Computer and Communications Security, AsiaCCS 2018, Incheon, Republic of Korea, June 04-08, 2018*, pages 37–50, 2018.

[BMPS21] Olivier Bronchain, Charles Momin, Thomas Peters, and François-Xavier Standaert. Improved leakage-resistant authenticated encryption based on hardware AES coprocessors. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2021(3):641–676, 2021.

[BPPS17] Francesco Berti, Olivier Pereira, Thomas Peters, and François-Xavier Standaert. On leakage-resilient authenticated encryption with decryption leakages. *IACR Trans. Symmetric Cryptol.*, 2017(3):271–293, 2017.

[BR06] Mihir Bellare and Phillip Rogaway. The security of triple encryption and a framework for code-based game-playing proofs. In *Advances in Cryptology - EUROCRYPT 2006, 25th Annual International Conference on the Theory and Applications of Cryptographic Techniques, St. Petersburg, Russia, May 28 - June 1, 2006, Proceedings*, pages 409–426, 2006.

[BT16] Mihir Bellare and Björn Tackmann. The multi-user security of authenticated encryption: AES-GCM in TLS 1.3. In *CRYPTO (1)*, volume 9814 of *LNCS*, pages 247–276. Springer, 2016.

[CJPS22] Benoît Cogliati, Jérémy Jean, Thomas Peyrin, and Yannick Seurin. A long tweak goes a long way: High multi-user security authenticated encryption from tweakable block ciphers. Cryptology ePrint Archive, Paper 2022/846, 2022. https://eprint.iacr.org/2022/846.

[DEM+20] Christoph Dobraunig, Maria Eichlseder, Stefan Mangard, Florian Mendel, Bart Mennink, Robert Primas, and Thomas Unterluggauer. ISAP v2.0. *IACR Trans. Symmetric Cryptol.*, 2020(S1):390–416, 2020.

[DEMS21] Christoph Dobraunig, Maria Eichlseder, Florian Mendel, and Martin Schläffer. Ascon v1.2: Lightweight authenticated encryption and hashing. *J. Cryptol.*, 34(3):33, 2021.

[DJS19] Jean Paul Degabriele, Christian Janson, and Patrick Struck. Sponges resist leakage: The case of authenticated encryption. In *ASIACRYPT (2)*, volume 11922 of *LNCS*, pages 209–240. Springer, 2019.

[DM19] Christoph Dobraunig and Bart Mennink. Leakage resilience of the duplex construction. In *ASIACRYPT (3)*, volume 11923 of *LNCS*, pages 225–255. Springer, 2019.

[DM21] Christoph Dobraunig and Bart Mennink. Leakage resilient value comparison with application to message authentication. In *EUROCRYPT (2)*, volume 12697 of *LNCS*, pages 377–407. Springer, 2021.

[DMA17] Joan Daemen, Bart Mennink, and Gilles Van Assche. Full-state keyed duplex with built-in multi-user support. In *ASIACRYPT (2)*, volume 10625 of *LNCS*, pages 606–637. Springer, 2017.

[DP08] Stefan Dziembowski and Krzysztof Pietrzak. Leakage-resilient cryptography. In *FOCS*, pages 293–302. IEEE Computer Society, 2008.

[DP10] Yevgeniy Dodis and Krzysztof Pietrzak. Leakage-resilient pseudorandom functions and side-channel attacks on feistel networks. In *CRYPTO*, volume 6223 of *LNCS*, pages 21–40. Springer, 2010.

[FPS12] Sebastian Faust, Krzysztof Pietrzak, and Joachim Schipper. Practical leakage-resilient symmetric cryptography. In *CHES*, volume 7428 of *LNCS*, pages 213–232. Springer, 2012.

[GPPS19] Chun Guo, Olivier Pereira, Thomas Peters, and François-Xavier Standaert. Authenticated encryption with nonce misuse and physical leakage: Definitions, separation results and first construction - (extended abstract). In *LATINCRYPT*, volume 11774 of *LNCS*, pages 150–172. Springer, 2019.

[GPPS20] Chun Guo, Olivier Pereira, Thomas Peters, and François-Xavier Standaert. Towards low-energy leakage-resistant authenticated encryption from the duplex sponge construction. *IACR Trans. Symmetric Cryptol.*, 2020(1):6–42, 2020.

[Hir06] Shoichi Hirose. Some plausible constructions of double-block-length hash functions. In *Fast Software Encryption, 13th International Workshop, FSE 2006, Graz, Austria, March 15-17, 2006, Revised Selected Papers*, pages 210–225, 2006.

[HT17] Viet Tung Hoang and Stefano Tessaro. The multi-user security of double encryption. In *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 II*, pages 381–411, 2017.

[JLM+19] Philipp Jovanovic, Atul Luykx, Bart Mennink, Yu Sasaki, and Kan Yasuda. Beyond conventional security in sponge-based authenticated encryption modes. *J. Cryptol.*, 32(3):895–940, 2019.

[JNP14] Jérémy Jean, Ivica Nikolic, and Thomas Peyrin. Tweaks and keys for block ciphers: The TWEAKEY framework. In *ASIACRYPT (2)*, volume 8874 of *LNCS*, pages 274–288. Springer, 2014.

[Lis21] Eik List. TEDT2 - highly secure leakage-resilient tbc-based authenticated encryption. In *LATINCRYPT*, volume 12912 of *LNCS*, pages 275–295. Springer, 2021.

[LRW11] Moses D. Liskov, Ronald L. Rivest, and David A. Wagner. Tweakable block ciphers. *J. Cryptol.*, 24(3):588–613, 2011.

[MCS22] Charles Momin, Gaëtan Cassiers, and François-Xavier Standaert. Unprotected and masked hardware implementations of spook v2. *IACR Cryptol. ePrint Arch.*, page 254, 2022.

[MR04] Silvio Micali and Leonid Reyzin. Physically observable cryptography (extended abstract). In *TCC*, volume 2951 of *LNCS*, pages 278–296. Springer, 2004.

[MRV15] Bart Mennink, Reza Reyhanitabar, and Damian Vizár. Security of full-state keyed sponge and duplex: Applications to authenticated encryption. In *ASIACRYPT (2)*, volume 9453 of *LNCS*, pages 465–489. Springer, 2015.

[NSS22] Yusuke Naito, Yu Sasaki, and Takeshi Sugawara. Secret can be public: Low-memory aead mode for high-order masking. Cryptology ePrint Archive, Paper 2022/812, 2022. https://eprint.iacr.org/2022/812.

[Pey20] Thomas Peyrin. Tweakable block cipher-based cryptography, 2020. FSE, Invited talk.

[Pie09] Krzysztof Pietrzak. A leakage-resilient mode of operation. In *EUROCRYPT*, volume 5479 of *LNCS*, pages 462–482. Springer, 2009.

[PS16] Thomas Peyrin and Yannick Seurin. Counter-in-tweak: Authenticated encryption modes for tweakable block ciphers. In *CRYPTO (1)*, volume 9814 of *LNCS*, pages 33–63. Springer, 2016.

[PSV15] Olivier Pereira, François-Xavier Standaert, and Srinivas Vivek. Leakage-resilient authentication and encryption from symmetric cryptographic primitives. In *CCS*, pages 96–108. ACM, 2015.

[Rog02] Phillip Rogaway. Authenticated-encryption with associated-data. In *Proceedings of the 9th ACM Conference on Computer and Communications Security, CCS 2002, Washington, DC, USA, November 18-22, 2002*, pages 98–107, 2002.

[SPS+22] Yaobin Shen, Thomas Peters, François-Xavier Standaert, Gaëtan Cassiers, and Corentin Verhamme. Triplex: an efficient and one-pass leakage-resistant mode of operation. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2022(4):135–162, 2022.

[VCS22] Corentin Verhamme, Gaëtan Cassiers, and François-Xavier Standaert. Analyzing the leakage resistance of the NIST's lightweight crypto competition's finalists. In *CARDIS*, volume xxxx of *LNCS*, pages yyy–zzz. Springer, 2022.

[YS13] Yu Yu and François-Xavier Standaert. Practical leakage-resilient pseudorandom objects with minimum public randomness. In *CT-RSA*, volume 7779 of *LNCS*, pages 223–238. Springer, 2013.

[YSPY10] Yu Yu, François-Xavier Standaert, Olivier Pereira, and Moti Yung. Practical leakage-resilient pseudorandom generators. In *CCS*, pages 141–151. ACM, 2010.

A Code Description of Multiplex

Procedure $\mathcal{E}(K \parallel P, N, A, M)$
Input: key $K \in \{0,1\}^n$,
public key $P \in \{0,1\}^n$
nonce $N \in \{0,1\}^n$
associated data $A \in \{0,1\}^*$
plaintext $M \in \{0,1\}^*$
Output: ciphertext $C \in \{0,1\}^{|M|}$
tag $\mathsf{tag} \in \{0,1\}^n$

Initialize
$A[1] \parallel \ldots \parallel A[dv] \leftarrow \mathrm{pad}(A)$
$M[1] \parallel \ldots \parallel M[d\ell] \leftarrow \mathrm{pad}(M)$
$k_0 \leftarrow E_K(P \parallel 0^{(d-1)n}, N)$
$h_0 \leftarrow 0^n$; $T \leftarrow N \parallel P \parallel 0^{(d-2)n}$
$h_1 \leftarrow \mathsf{DM}(h_0, k_0, T, 0^n)$
$k_1 \leftarrow \mathsf{DM}(h_0, k_0, T, \theta_1)$

Processing Associated Data
for $i \leftarrow 1$ **to** v **do**
$T \leftarrow A[di-d+1] \parallel \ldots \parallel A[di]$
$h_{i+1} \leftarrow \mathsf{DM}(h_i, k_i, T, 0^n)$
$k_{i+1} \leftarrow \mathsf{DM}(h_i, k_i, T, \theta_1)$

Processing Plaintext
for $j \leftarrow 1$ **to** $d-1$ **do**
$e_1^j \leftarrow \mathsf{DM}(h_v, k_v, T, \theta_{j+1})$
$k_{v+1} \leftarrow k_{v+1} \oplus 1$
for $i \leftarrow 1$ **to** $\ell - 1$ **do**
for $j \leftarrow 1$ **to** $d-1$
$a \leftarrow di - d + j$
$C[a] \leftarrow e_i^j \oplus M[a]$
$C[di] \leftarrow h_{v+i} \oplus M[di]$
$T \leftarrow C[di-d+1] \parallel \ldots \parallel C[di]$
$h_{v+i+1} \leftarrow \mathsf{DM}(h_{v+i}, k_{v+i}, T, 0^n)$
$k_{v+i+1} \leftarrow \mathsf{DM}(h_{v+i}, k_{v+i}, T, \theta_1)$
for $j \leftarrow 1$ **to** $d-1$
$e_{i+1}^j \leftarrow \mathsf{DM}(h_{v+i}, k_{v+i}, T, \theta_{j+1})$
for $j \leftarrow 1$ **to** $d-1$
$a \leftarrow d\ell - d + j$
$C[a] \leftarrow e_\ell^j \oplus M[a]$
$C[d\ell] \leftarrow h_{v+\ell} \oplus M[d\ell]$
$T \leftarrow C[d\ell-d+1] \parallel \ldots \parallel C[d\ell]$
$h_{\ell+v+1} \leftarrow \mathsf{DM}(h_{\ell+v}, k_{\ell+v}, T, 0^n)$
$k_{\ell+v+1} \leftarrow \mathsf{DM}(h_{\ell+v}, k_{\ell+v}, T, \theta_1)$
$C \leftarrow \lceil C[1] \parallel \ldots \parallel C[d\ell] \rceil^{|M|}$

Finalize
$\mathsf{tag} \leftarrow E_K(h_{\ell+v+1} \parallel k_{\ell+v+1} \parallel 0^{(d-2)n}, 0^n)$
return $C \parallel \mathsf{tag}$

Procedure $\mathcal{D}(K \parallel P, N, A, C \parallel \mathsf{tag})$
Input: key $K \in \{0,1\}^n$,
public key $P \in \{0,1\}^n$
nonce $N \in \{0,1\}^n$
associated data $A \in \{0,1\}^*$
ciphertext $C \in \{0,1\}^*$
tag $\mathsf{tag} \in \{0,1\}^n$
Output: plaintext $M \in \{0,1\}^{|C|}$ or $\perp$

Initialize
$A[1] \parallel \ldots \parallel A[dv] \leftarrow \mathrm{pad}(A)$
$C[1] \parallel \ldots \parallel C[d\ell] \leftarrow \mathrm{pad}(C)$
$k_0 \leftarrow E_K(P \parallel 0^{(d-1)n}, N)$
$h_0 \leftarrow 0^n$; $T \leftarrow N \parallel P \parallel 0^{(d-2)n}$
$h_1 \leftarrow \mathsf{DM}(h_0, k_0, T, 0^n)$
$k_1 \leftarrow \mathsf{DM}(h_0, k_0, T, \theta_1)$

Processing Associated Data
for $i \leftarrow 1$ **to** v **do**
$T \leftarrow A[di-d+1] \parallel \ldots \parallel A[di]$
$h_{i+1} \leftarrow \mathsf{DM}(h_i, k_i, T, 0^n)$
$k_{i+1} \leftarrow \mathsf{DM}(h_i, k_i, T, \theta_1)$

Processing Ciphertext
for $j \leftarrow 1$ **to** $d-1$ **do**
$e_1^j \leftarrow \mathsf{DM}(h_v, k_v, T, \theta_{j+1})$
$k_{v+1} \leftarrow k_{v+1} \oplus 1$
for $i \leftarrow 1$ **to** $\ell - 1$ **do**
for $j \leftarrow 1$ **to** $d-1$
$a \leftarrow di - d + j$
$M[a] \leftarrow e_i^j \oplus C[a]$
$M[di] \leftarrow h_{v+i} \oplus C[di]$
$T \leftarrow C[di-d+1] \parallel \ldots \parallel C[di]$
$h_{v+i+1} \leftarrow \mathsf{DM}(h_{v+i}, k_{v+i}, T, 0^n)$
$k_{v+i+1} \leftarrow \mathsf{DM}(h_{v+i}, k_{v+i}, T, \theta_1)$
for $j \leftarrow 1$ **to** $d-1$
$e_{i+1}^j \leftarrow \mathsf{DM}(h_{v+i}, k_{v+i}, T, \theta_{j+1})$
for $j \leftarrow 1$ **to** $d-1$
$a \leftarrow d\ell - d + j$
$M[a] \leftarrow e_\ell^j \oplus C[a]$
$M[d\ell] \leftarrow h_{v+\ell} \oplus C[d\ell]$
$T \leftarrow C[d\ell-d+1] \parallel \ldots \parallel C[d\ell]$
$h_{\ell+v+1} \leftarrow \mathsf{DM}(h_{\ell+v}, k_{\ell+v}, T, 0^n)$
$k_{\ell+v+1} \leftarrow \mathsf{DM}(h_{\ell+v}, k_{\ell+v}, T, \theta_1)$
$M \leftarrow \lceil M[1] \parallel \ldots \parallel M[d\ell] \rceil^{|C|}$

Finalize
$x \leftarrow E_K^{-1}(h_{\ell+v+1} \parallel k_{\ell+v+1} \parallel 0^{(d-2)n}, \mathsf{tag})$
if $x = 0^n$ **then return** M **else** $\perp$

Inner Function $\mathsf{DM}(h, k, m, \theta)$
$e \leftarrow E_k(m, h \oplus \theta) \oplus h \oplus \theta$
return e

Figure 6: Authenticated encryption and decryption procedures of Multiplex construction based on a TBC with n-bit key and dn-bit tweak. Here $\theta_1, \ldots, \theta_d \in \{0,1\}^n$ are distinct non-zero constants, e.g., $1, \ldots, d$.

B A Shorter-Block Variant of Multiplex

In this section, we provide an intuitive idea how to build Multiplex from a short-block tweakable block cipher (TBC) with n-bit security. For a short-block TBC, the key size is usually larger than the block size in order to to keep the security. Here we focus on the case where the key size is twice as large as the block size. We first describe a variant of Multihash denoted by $\overline{\text{Multihash}}$ from a short-block TBC, and then show how to encrypt and authenticate messages by using this hash function.

HASH FUNCTION. The hash function $\overline{\text{Multihash}}$ is based on a compression function $\overline{F}$ that is defined below. The main idea is to keep a $2n$-bit state as in F for Multihash. Now, this state is simply seen as 4 blocks of $n/2$ bits. However, while in F the state can be carried by the n-bit key-input and the n-bit plaintext-input in each TBC call, here we can only carry 3 blocks of $n/2$ bits by following this strategy as the TBC now only supports $(n/2)$-bit plaintext-input. To hope keeping a good collision-resistance bound, we still have to make all the TBC calls in $\overline{F}$ directly depending on the 4th block as well and we thus put this last $n/2$ bits in the tweak. Assuming the tweak size is $d \cdot n/2$, for some integer $d \geq 2$, we still have room to absorb $d-1$ blocks. This leads to the following definition with rate $(d-1)/(d+1)$, where we also generate the auxiliary blocks that will help designing a Multiplex variant with shorter blocks.

Definition 3. Let $E : \{0,1\}^n \times \{0,1\}^{dn/2} \times \{0,1\}^{n/2} \to \{0,1\}^{n/2}$ be a TBC. Then $\overline{F} : \{0,1\}^{4n/2} \times \{0,1\}^{(d-1)n/2} \to \{0,1\}^{(d+1)n/2}$ is an extended compression function such that $(h_i^1, h_i^2, k_i^1, k_i^2, e_i^1, \ldots, e_i^{d-3}) = \overline{F}(h_{i-1}^1, h_{i-1}^2, k_{i-1}^1, k_{i-1}^2, t_i)$ where $h_{i-1}^1, h_{i-1}^2, k_{i-1}^1, k_{i-1}^2, \in \{0,1\}^{n/2}$ is the state, $t_i \in \{0,1\}^{(d-1)n/2}$ is being absorbed, $h_i^1, h_i^2, k_i^1, k_i^2$ is the main output (the refreshed state), and $e_i^1, \ldots, e_i^{d-3} \in \{0,1\}^{n/2}$ the auxiliary blocks that are computed as follows:

$$\begin{cases} h_i^1 = E_{k_{i-1}^1 \,\|\, k_{i-1}^2}(h_{i-1}^2 \,\|\, t_i, h_{i-1}^1) \oplus h_{i-1}^1 \\ k_i^1 = E_{k_{i-1}^1 \,\|\, k_{i-1}^2}(h_{i-1}^2 \,\|\, t_i, h_{i-1}^1 \oplus \theta_1) \oplus h_{i-1}^1 \oplus \theta_1 \\ k_i^2 = E_{k_{i-1}^1 \,\|\, k_{i-1}^2}(h_{i-1}^2 \,\|\, t_i, h_{i-1}^1 \oplus \theta_2) \oplus h_{i-1}^1 \oplus \theta_2 \\ h_i^2 = E_{k_{i-1}^1 \,\|\, k_{i-1}^2}(h_{i-1}^2 \,\|\, t_i, h_{i-1}^1 \oplus \theta_3) \oplus h_{i-1}^1 \oplus \theta_3 \\ e_i^1 = E_{k_{i-1}^1 \,\|\, k_{i-1}^2}(h_{i-1}^2 \,\|\, t_i, h_{i-1}^1 \oplus \theta_4) \oplus h_{i-1}^1 \oplus \theta_4 \\ \qquad\qquad \vdots \\ e_i^{d-3} = E_{k_{i-1}^1 \,\|\, k_{i-1}^2}(h_{i-1}^2 \,\|\, t_i, h_{i-1}^1 \oplus \theta_d) \oplus h_{i-1}^1 \oplus \theta_d \end{cases}$$

Here $\theta_1, \ldots, \theta_d \in \{0,1\}^{n/2}$ are distinct non-zero constants, e.g., $1, \ldots, d$.

See the left of Figure 7 for the illustration of compression function $\overline{F}$ when $d = 4$. Following the same paradigm in Section 3, we can obtain the hash function $\overline{\text{Multihash}}$ that is based on the compression function $\overline{F}$. Note that we consider the case when the key size of a TBC is twice large as the block size. In the hash function $\overline{\text{Multihash}}$, it requires two TBC calls to produce the n-bit key.

MULTIPLEX FROM A SHORT-BLOCK TBC. Following a similar paradigm of Multiplex in Section 4, we can turn the hash function $\overline{\text{Multihash}}$ with a key-derivation function and a tag-generation function into a full-fledged AE scheme. For that purpose, the KDF and the TGF variants would have to make (at least) two protected calls to the short-block TBC. This Multiplex variant would allow encrypting $d-1$ blocks of $n/2$-bit message in parallel per iteration with only $d+1$ calls to the underlying short-block TBC. Given a TBC with 256 bits of tweak, we have $d = 4$, and the mode would allow encrypting $3 \cdot 64 = 192$ bits of message per iteration with primitive-rate 3/5 (while the only previous AE with short

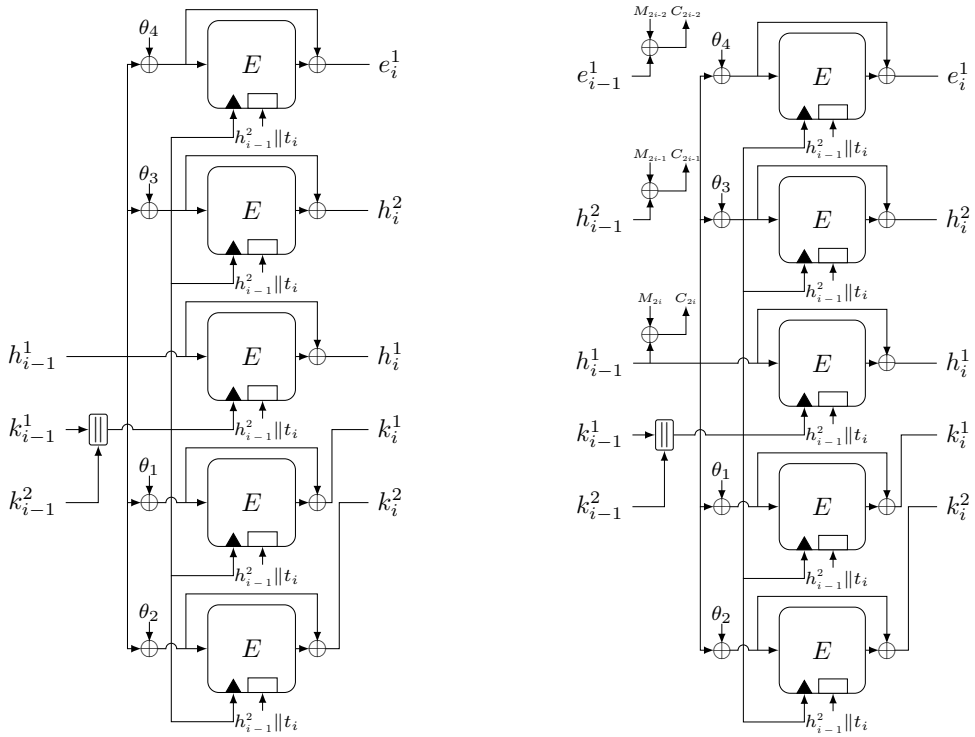

Figure 7: The compression function $\overline{F}$ that is based on a tweakable block cipher E with $n/2$-bit block, n-bit key and $2n$-bit tweak (left). An intuitive idea to turn the compression function $\overline{F}$ into encryption and authentication(right) where $t_i = C_{2i-2} \parallel C_{2i-1} \parallel C_{2i}$.

blocks aiming at leakage resilience [NSS22] only enjoys a rate 1/3, but for different security goals). Here we only provide one iteration of our short-block Multiplex scheme that is illustrated in the right of Figure 7.

C Multi-Block Hash Function Using a Tweakable Block Cipher

In this section, we introduce the multi-block-length (MBL) hash function called MBLhash based on a tweakable block cipher with n-bit key and dn-bit tweak that is collision-resistant up to $O(2^{(d+1)n/2})$ queries in the ideal TBC model.

Note that this MBL compression function can be regarded as a generalization of Hirose's double-block length compression function [Hir06] to output arbitrary blocks without increasing the size of key. A similar MBL hash function called Counter-bDM is proposed by Abed et al. [AFL+14]. It is based on a block cipher and the key size increases linearly with the output blocks, while for the hash function MBLhash, the size of key is fixed and only the size of tweak increases linearly with the output blocks.

Compression function. The MBL hash function MBLhash is built on top of a MBL compression function F^* that is described as follows.

Definition 4. Let $E : \{0,1\}^n \times \{0,1\}^{dn} \times \{0,1\}^n \to \{0,1\}^n$ be a tweakable block cipher. Then $F^* : \{0,1\}^{(d+1)n} \times \{0,1\}^n \to \{0,1\}^{(d+1)n}$ is a multi-block compression function such that $(h_i, k_i, e_i^1, \ldots, e_i^{d-1}) = F^*(h_{i-1}, k_{i-1}, e_{i-1}^1, \ldots, e_{i-1}^{d-1}, t_i)$ where $h_{i-1}, k_{i-1}, e_{i-1}^1, \ldots, e_{i-1}^{d-1}$, $t_i \in \{0,1\}^n$, and $h_i, k_i, e_i^1, \ldots, e_i^{d-1} \in \{0,1\}^n$ are computed

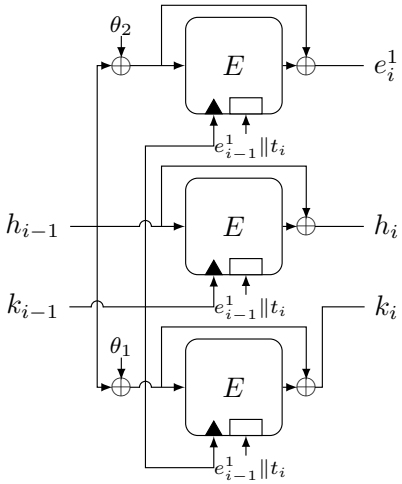

Figure 8: The MBL compression function F^* based on a tweakable block cipher E with n-bit key and $2n$-bit tweak.

as follows:

$$\begin{cases} h_i = E_{k_{i-1}}(e_{i-1}^1 \parallel \ldots \parallel e_{i-1}^{d-1} \parallel t_i, h_{i-1}) \oplus h_{i-1} \\ k_i = E_{k_{i-1}}(e_{i-1}^1 \parallel \ldots \parallel e_{i-1}^{d-1} \parallel t_i, h_{i-1} \oplus \theta_1) \oplus h_{i-1} \oplus \theta_1 \\ e_i^1 = E_{k_{i-1}}(e_{i-1}^1 \parallel \ldots \parallel e_{i-1}^{d-1} \parallel t_i, h_{i-1} \oplus \theta_2) \oplus h_{i-1} \oplus \theta_2 \\ \qquad\qquad \vdots \\ e_i^{d-1} = E_{k_{i-1}}(e_{i-1}^1 \parallel \ldots \parallel e_{i-1}^{d-1} \parallel t_i, h_{i-1} \oplus \theta_d) \oplus h_{i-1} \oplus \theta_d \end{cases}$$

Here $\theta_1, \ldots, \theta_d \in \{0,1\}^n$ are distinct non-zero constants, e.g., $1, \ldots, d$.

A pictorial illustration of the compression function F^* when $d = 2$ is given in Figure 8.

Next, we give the definition of MBL hash function $\mathsf{MBLhash} : \{0,1\}^* \to \{0,1\}^{(d+1)n}$ composed of the compress function F. Let $\mathrm{pad} : \{0,1\}^* \to (\{0,1\}^n)^+$ be an injective padding function, e.g., one-zero padding $\mathrm{pad}(t) = t \parallel 10^a$ where $a = n - 1 - (|t| \mod n)$. The specification of MBL hash function $\mathsf{MBLhash}$ is given in Figure 9.

We say a collision occurs if $\mathsf{MBLhash}(t) = \mathsf{MBLhash}(t')$ for any two different messages $t, t' \in \{0,1\}^*$. This collision requires a multi-block collision on the outputs, namely $(h_\ell, k_\ell, e_\ell^1, \ldots, e_\ell^{d-1}) = (h'_{\ell'}, k'_{\ell'}, e'^1_{\ell'}, \ldots, e'^{d-1}_{\ell'})$. For an adversary $\mathcal{A}$ with oracle access to E and E^{-1}, let $\mathsf{Adv}_H^{\mathsf{mcoll}}(\mathcal{A})$ be the probability that $\mathcal{A}$ finds a collision of the hash function $\mathsf{MBLhash}$. The following lemma shows that $\mathsf{MBLhash}$ is collision-resistance up to $O(2^{(d+1)n/2})$ queries. The proof can be adapted directly from [AFL+14, Theorem 3] since the key and tweak play the same role in the ideal TBC model.

Lemma 6. *For any adversary $\mathcal{A}$ making at most q queries to E and E^{-1}, we have*

$$\mathsf{Adv}^{\mathsf{mcoll}}_{\mathsf{MBLhash}}(\mathcal{A}) \leq \frac{(d+1)^2 \cdot 2^{d+3} \cdot q^2}{2^{(d+1)n}} + \frac{(d+1)^3 \cdot 2^{d+6} \cdot q^2}{2^{(d+2)n}}.$$

D CCAmL1 Security

In this section, we give the definition of CCAmL1 [GPPS19] in the multi-user setting. In CCAmL1 game (illustrated in Figure 10), the adversary has the access to encryption oracles with leakage and decryption oracle without leakage. The goal of adversary is to attack the confidentiality of messages encrypted with fresh nonces (nonce-misuse resilience).

Hash function $\mathsf{MBLhash}(t)$
$t_1 \parallel \ldots \parallel t_\ell \leftarrow \mathrm{pad}(t)$; $(h_0, k_0, e_0^1, \ldots, e_0^{d-1}) \leftarrow (0^n, 0^n, 0^n, \ldots, 0^n)$
for $i \leftarrow 1$ **to** ℓ **do**
  $(h_i, k_i, e_i^1, \ldots, e_i^{d-1}) \leftarrow F(h_{i-1}, k_{i-1}, e_{i-1}^1, \ldots, e_{i-1}^{d-1}, t_i)$
return $(h_\ell, k_\ell, e_\ell^1, \ldots, e_\ell^{d-1})$

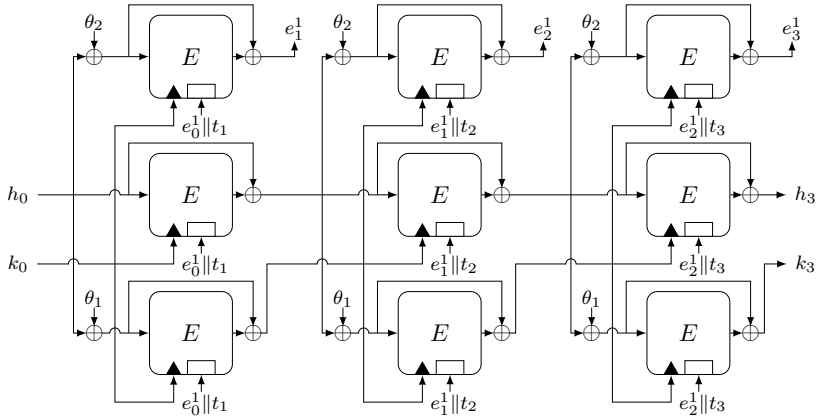

Figure 9: The MBL hash function $\mathsf{MBLhash}$ that is built from the compression function F^* with n-bit key and $2n$-bit tweak.

Game $\mathsf{G}_\Pi^{\mathrm{CCAmL1}}(\mathcal{A})$
$K_1, K_2, \ldots, \xleftarrow{\$} \mathcal{K}$; $b \xleftarrow{\$} \{0,1\}$
$b' \xleftarrow{\$} \mathcal{A}^{\textsc{Prim},\textsc{Lenc}_1,\textsc{Lenc}_2,\textsc{Dec}}$
return $(b' = b)$

Procedure $\textsc{Prim}(J, T, X)$
if $X = (+, x)$ **then return** $E_J(T, x)$
if $X = (-, y)$ **then return** $E_J^{-1}(T, y)$

Procedure $\textsc{Dec}(i, N, A, C \parallel \mathsf{tag})$
$M \leftarrow \mathcal{D}(K_i, N, A, C \parallel \mathsf{tag})$
return M

Procedure $\textsc{Lenc}_1(i, N, A, M)$
$C \parallel \mathsf{tag} \leftarrow \mathcal{E}(K_i, N, A, M)$
$L_e \leftarrow \mathcal{L}_E(K_i, N, A, M)$
return $(C \parallel \mathsf{tag}, L_e)$

Procedure $\textsc{Lenc}_2(i, N, A, M^0, M^1)$
if $|M^0| \neq |M^1|$ **then return** $\perp$
$C^b \parallel \mathsf{tag}^b \leftarrow \mathcal{E}(K_i, N, A, M^b)$
$L_e^b \leftarrow \mathcal{L}_E(K_i, N, A, M^b)$
return $(C^b \parallel \mathsf{tag}^b, L_e^b)$

Figure 10: Game $\mathsf{G}_\Pi^{\mathrm{CCAmL1}}$ defines the multi-user CCAmL1 security of an AE scheme Π.

The advantage of the adversary is captured by the left-or-right framework. That is, the probability that the adversary can tell apart the encryption of two different messages of equal length. This framework can avoid the conceptual difficulty to define the leakage of ideal objects in the real-or-random game. For queries to the same user, the adversary may repeat the nonce in the first encryption oracle $\textsc{Lenc}_1$ but the nonce in the second one $\textsc{Lenc}_2$ (challenge queries) is unique and fresh. For queries to different users, the adversary can repeat the nonce in both oracles. The adversary also has the access to decryption oracle $\textsc{Dec}$, but she cannot forward queries from $\textsc{Lenc}_2$ to $\textsc{Dec}$ since otherwise resulting in trivial win. Formally,

$$\mathsf{Adv}_\Pi^{\mathrm{CCAmL1}}(\mathcal{A}) = 2\Pr\left[\,\mathsf{G}_\Pi^{\mathrm{CCAmL1}}(\mathcal{A})\,\right] - 1$$

is the advantage of the adversary against the CCAmL1 security of an AE Π.

E Proof of Theorem 2

In the CIML2 security game of Figure 2, the adversary is granted access to three oracles, including the encryption oracle, the decryption oracle, and the offline ideal TBC oracle. The adversary forges successfully if any of her queries passes the decryption oracle. Our goal is to show that the successful probability is negligible. From the interaction with her oracles, the adversary can obtain the information that are recorded as follows.

- For each query $\textsc{Prim}(J, T, (+, x))$ with answer y, we will store an entry $(\mathsf{prim}, J, T, x, y, +)$. Similarly, for each query $\textsc{Prim}(J, T, (-, y))$ with answer x, we will store an entry $(\mathsf{prim}, J, T, x, y, -)$.
- For each query $(C\|\mathsf{tag}, L_e) \leftarrow \textsc{Enc}(i, N, A, M)$, let $A[1]\| \dots \|A[dv] \leftarrow A$, $M[1]\| \dots \| M[d\ell] \leftarrow M$ and $C[1] \| \dots \| C[d\ell] \leftarrow C$. Let

$$h_0 = 0^n, k_0 = E_{K_i}(P_i \| 0^{(d-1)n}, N) \ ,$$
$$h_1 = E_{k_0}(N \| P_i \| 0^{(d-2)n}, 0^n), k_1 = E_{k_0}(N \| P_i \| 0^{(d-2)n}, \theta_1) \oplus \theta_1 \ ,$$

  and for $1 \le j \le v$, let

$$h_{j+1} = E_{k_j}(A[dj - d + 1] \| \dots \| A[dj], h_j) \oplus h_j \ ,$$
$$k_{j+1} = E_{k_j}(A[dj - d + 1] \| \dots \| A[dj], h_j \oplus \theta_1) \oplus h_j \oplus \theta_1 \ .$$

  For $1 \le r \le d-1$, let $e_1^r = E_{k_v}(T, h_v \oplus \theta_{r+1}) \oplus h_v \oplus \theta_{r+1}$ where $T = A[dv - d + 1]\| \dots \|A[dv]$ if AD is not empty otherwise $T = N\|P_i\|0^{(d-2)n}$. Let $k_{v+1} = k_{v+1} \oplus 1$, and for $1 \le j \le \ell - 1$, let

$$h_{v+j+1} = E_{k_{v+j}}(C[dj - d + 1] \| \dots \| C[dj], h_{v+j}) \oplus h_{v+j} \ ,$$
$$k_{v+j+1} = E_{k_{v+j}}(C[dj - d + 1] \| \dots \| C[dj], h_{v+j} \oplus \theta_1) \oplus h_{v+j} \oplus \theta_1 \ ,$$
$$e_{j+1}^r = E_{k_{v+j}}(C[dj - d + 1] \| \dots \| C[dj], h_{v+j} \oplus \theta_{r+1}) \oplus h_{v+j} \oplus \theta_{r+1}$$

  for $1 \le r \le d-1$. Let

$$h_{v+\ell+1} = E_{k_{v+\ell}}(C[d\ell - d + 1] \| \dots \| C[d\ell], h_{v+\ell}) \oplus h_{v+\ell} \ ,$$
$$k_{v+\ell+1} = E_{k_{v+\ell}}(C[d\ell - d + 1] \| \dots \| C[d\ell], h_{v+\ell} \oplus \theta_1) \oplus h_{v+\ell} \oplus \theta_1 \ .$$

  Denote by $\boldsymbol{h} = (h_0, \dots, h_{\ell+v+1})$ and $\boldsymbol{k} = (k_0, \dots, k_{\ell+v+1})$. In the unbounded leakage model with leveled implementation, except the key K_i of the first and final TBC call, all the values of $\boldsymbol{h}$ and $\boldsymbol{k}$ are exposed to the adversary. Hence we will store an entry $(\mathsf{enc}, i, N, A, M, T \| \mathsf{tag}, \boldsymbol{h}, \boldsymbol{k})$. Note that the adversary can learn the internal primitive calls from this entry that are recorded by $(\mathsf{leak}, J, T, x, y)$ as follows:

  - $(\mathsf{leak}, k_0, N \| P_i \| 0^{(d-2)n}, 0^n, h_1)$ and $(\mathsf{leak}, k_0, N \| P_i \| 0^{(d-2)n}, \theta_1, k_1 \oplus \theta_1)$ during the initialization;
  - for $1 \le j \le v$, $(\mathsf{leak}, k_j, A[dj-d+1]\| \dots \|A[dj], h_j, h_j \oplus h_{j+1})$, $(\mathsf{leak}, k_j, A[dj-d+1]\| \dots \|A[dj], h_j \oplus \theta_1, h_j \oplus k_{j+1} \oplus \theta_1)$, and $(\mathsf{leak}, k_v, A[dj-d+1]\| \dots \|A[dj], h_v \oplus \theta_{r+1}, h_v \oplus e_1^r \oplus \theta_{r+1})$ for $1 \le r \le d-1$ during the associated dara processing;
  - for $1 \le j \le \ell - 1$, $(\mathsf{leak}, k_{v+j}, C[dj - d + 1] \| \dots \| C[dj], h_{v+j}, h_{v+j} \oplus h_{v+j+1})$, $(\mathsf{leak}, k_{v+j}, C[dj-d+1]\| \dots \|C[dj], h_{v+j} \oplus \theta_1, h_{v+j} \oplus k_{v+j+1} \oplus \theta_1)$, $(\mathsf{leak}, k_{v+j}, C[dj-d+1] \| \dots \| C[dj], h_{v+j} \oplus \theta_{r+1}, h_{v+j} \oplus e_{j+1}^r \oplus \theta_{r+1})$ for $1 \le r \le d-1$, and finally $(\mathsf{leak}, k_{v+\ell}, C[dj - d + 1] \| \dots \| C[dj], h_{v+\ell}, h_{v+\ell+1})$ and $(\mathsf{leak}, k_{v+\ell}, C[dj - d + 1] \| \dots \| C[dj], h_{v+\ell} \oplus \theta_1, h_{v+\ell} \oplus k_{v+\ell+1} \oplus \theta_1)$ during the message processing.

- For each query $(M, L_d) \leftarrow \textsc{Dec}(i, N, A, C \,\|\, \mathsf{tag})$ (here M can be either a message or a false symbol $\perp$ by abusing of notation), similarly to the encryption query, we will store an entry $(\mathsf{dec}, i, N, A, M, C \,\|\, \mathsf{tag}, \boldsymbol{h}, \boldsymbol{k}, x)$ where x is the checking value that is computed as $x = E_{K_i}^{-1}(h_{\ell+v+1} \,\|\, k_{\ell+v+1} \,\|\, 0^{(d-2)n}, \mathsf{tag})$. We also use $(\mathsf{leak}, J, T, x, y)$ to record the internal primitive calls that the adversary can learn from this entry.

Note that in the CIML2 game, the adversary can make a decryption query even when this query has appeared in previous encryption queries. This is to capture the scenario that the adversary may obtain some more leakage information by repeating the query. However, in the unbounded leakage model with leveled implementation, all the internal values are exposed to the adversary by previous encryption queries. Hence without loss of generality, we can simply ignore such trivial decryption queries.

We now proceed to show that the probability that the adversary forges successfully is negligible. To this end, we first define a bad event. This event consists of several bad conditions that will be explained when described. The purpose of this event is to ensure that the adversary is unlikely to forge a tag when none of these bad conditions are triggered as shown below. The event bad is said to happen if at least one of the following conditions is violated (note that conditions 1-4 are similar to those in the proof of Theorem 1 and thus we omit the corresponding explanations here):

(1) There are two users i and j $(i \neq j)$ such that $K_i = K_j$ and $P_i = P_j$.

(2) There is a P_i that repeats at least c_1 times among u users.

(3) There is an ideal TBC entry $(\mathsf{prim}, J, T, x, y, *)$ or an internal primitive call $(\mathsf{leak}, J, T, x, y)$ such that $J = K_i$ and $T = P_i \,\|\, 0^{(d-1)n}$ for some user i.

(4) There is an entry $(\mathsf{dec}, i, N_i^a, A_i^a, M_i^a, C_i^a \,\|\, \mathsf{tag}_i^a)$ such that $k_{i,0}^a = k_{j,0}^b$ and $N_i^a \,\|\, P_i = N_j^b \,\|\, P_j$ for some other entry $(*, j, N_j^b, A_j^b, M_j^b, C_j^b \,\|\, \mathsf{tag}_j^b)$ of a different user j.

(5) There is an entry $(\mathsf{dec}, i, N_i^a, A_i^a, M_i^a, C_i^a \,\|\, \mathsf{tag}_i^a, \boldsymbol{h}_i^a, \boldsymbol{k}_i^a, x_i^a)$ such that $h_{i,\ell_a+v_a+1}^a \,\|\, k_{i,\ell_a+v_a+1}^a = h_{i,\ell_b+v_b+1}^b \| k_{i,\ell_b+v_b+1}^b$ for some previous encryption query $(\mathsf{enc}, i, N_i^b, A_i^b, M_i^b, C_i^b \,\|\, \mathsf{tag}_i^b, \boldsymbol{h}_i^b, \boldsymbol{k}_i^b)$ with $(N_i^a, A_i^a, C_i^a) \neq (N_i^b, A_i^b, C_i^b)$. This condition and the following condition (6) are to ensure that for each decryption query, the tweak $h_{i,\ell_a+v_a+1}^a \,\|\, k_{i,\ell_a+v_a+1}^a$ is always different from those of other encryption and decryption queries of the same user.

(6) There is an entry $(\mathsf{dec}, i, N_i^a, A_i^a, M_i^a, C_i^a \,\|\, \mathsf{tag}_i^a, \boldsymbol{h}_i^a, \boldsymbol{k}_i^a, x_i^a)$ such that $h_{i,\ell_a+v_a+1}^a \,\|\, k_{i,\ell_a+v_a+1}^a = h_{i,\ell_b+v_b+1}^b \| k_{i,\ell_b+v_b+1}^b$ for some previous decryption query $(\mathsf{dec}, i, N_i^b, A_i^b, M_i^b, C_i^b \,\|\, \mathsf{tag}_i^b, \boldsymbol{h}_i^b, \boldsymbol{k}_i^b, x_i^b)$ with $(N_i^a, A_i^a, C_i^a) \neq (N_i^b, A_i^b, C_i^b)$.

(7) There is an entry $(\mathsf{dec}, i, N_i^a, A_i^a, M_i^a, C_i^a \,\|\, \mathsf{tag}_i^a, \boldsymbol{h}_i^a, \boldsymbol{k}_i^a, x_i^a)$ such that $h_{i,\ell_a+v_a+1}^a \,\|\, k_{i,\ell_a+v_a+1}^a = h_{j,\ell_b+v_b+1}^b \,\|\, k_{j,\ell_b+v_b+1}^b$ for some previous encryption query $(\mathsf{enc}, j, N_j^b, A_j^b, M_j^b, C_j^b \,\|\, \mathsf{tag}_j^b, \boldsymbol{h}_j^b, \boldsymbol{k}_j^b)$ with $i \neq j$ and $P_i \neq P_j$. This condition and the following condition (8) are to ensure that for each decryption query, the tweak $h_{i,\ell_a+v_a+1}^a \,\|\, k_{i,\ell_a+v_a+1}^a$ is always different from those of other encryption or decryption queries of a different user. Note that we only need to consider the case when $P_i \neq P_j$ since otherwise $K_i \neq K_j$ conditioned on that condition (1) does not happen.

(8) There is an entry $(\mathsf{dec}, i, N_i^a, A_i^a, M_i^a, C_i^a \,\|\, \mathsf{tag}_i^a, \boldsymbol{h}_i^a, \boldsymbol{k}_i^a, x_i^a)$ such that $h_{i,\ell_a+v_a+1}^a \,\|\, k_{i,\ell_a+v_a+1}^a = h_{j,\ell_b+v_b+1}^b \| k_{j,\ell_b+v_b+1}^b$ for some previous decryption query $(\mathsf{dec}, j, N_j^b, A_j^b, M_j^b, C_j^b \,\|\, \mathsf{tag}_j^b, \boldsymbol{h}_j^b, \boldsymbol{k}_j^b, x_j^b)$ with $i \neq j$ and $P_i \neq P_j$.

(9) There is an entry $(\mathsf{dec}, i, N_i^a, A_i^a, M_i^a, C_i^a \,\|\, \mathsf{tag}_i^a, \boldsymbol{h}_i^a, \boldsymbol{k}_i^a, x_i^a)$ such that $(K_i, h_{i,\ell_a+v_a+1}^a \,\|\, k_{i,\ell_a+v_a+1}^a \,\|\, 0^{(d-2)n}) = (K_j, P_j \,\|\, 0^{(d-1)n})$ for some other entry. This is to avoid the

input collision (including key and tweak) between the first TBC and the last TBC call.

(10) There is an entry $(\mathsf{dec}, i, N_i^a, A_i^a, M_i^a, C_i^a \parallel \mathsf{tag}_i^a, \boldsymbol{h}_i^a, \boldsymbol{k}_i^a, x_i^a)$ such that $K_i = J$ and $h_{i,\ell_a+v_a+1}^a \parallel k_{i,\ell_a+v_a+1}^a \parallel 0^{(d-2)n} = T$ for some ideal TBC query $(\mathsf{prim}, J, T, x, y, *)$ or some leaked primitive call $(\mathsf{leak}, J, T, x, y)$. This is to ensure that the input of the last TBC call is fresh from those of ideal TBC queries and leaked primitive calls.

Observe that

$$\Pr[\mathcal{A} \text{ forges}] \leq \Pr[\mathcal{A} \text{ forges} \mid \neg\mathsf{bad}] + \Pr[\mathsf{bad}] \ . \tag{1}$$

A bound on the probability that event bad happens is given in Lemma 7.

We then analyze the conditional probability that $\mathcal{A}$ forges successfully given that bad does not happen. Such a forgery requires $x_i^a = 0^n$ for some decryption query $(\mathsf{dec}, i, N_i^a, A_i^a, M_i^a, C_i^a \parallel \mathsf{tag}_i^a, \boldsymbol{h}_i^a, \boldsymbol{k}_i^a, x_i^a)$. We consider the following cases.

- $(N_i^a, A_i^a, C_i^a) = (N_i^b, A_i^b, C_i^b)$ for some previous encryption query $(\mathsf{enc}, j, N_j^b, A_j^b, M_j^b, C_j^b \parallel \mathsf{tag}_j^b, \boldsymbol{h}_j^b, \boldsymbol{k}_j^b)$, then $\mathsf{tag}_i^a \neq \mathsf{tag}_i^b$ and x_i^a cannot be 0^n.
- $(N_i^a, A_i^a, C_i^a) = (N_i^a, A_i^a, C_i^a)$ for some previous decryption query $(\mathsf{dec}, i, N_i^b, A_i^b, M_i^b, C_i^b \parallel \mathsf{tag}_i^b, \boldsymbol{h}_i^b, \boldsymbol{k}_i^b, x_i^b)$, then $\mathsf{tag}_i^a \neq \mathsf{tag}_i^b$, and the probability that $x_i^a = 0^n$ is at most $1/(2^n - q)$ since conditioned on $\neg\mathsf{bad}$, the value x_i^a is always chosen uniformly at random from a set of size at least $2^n - q$.
- If neither of the above cases happens, then conditioned on $\neg\mathsf{bad}$, the key and tweak pair $(K_i, h_{i,\ell_a+v_a+1}^a \parallel k_{i,\ell_a+v_a+1}^a \parallel 0^{(d-2)n})$ is always fresh. Thus x_i^a is a uniformly random string and the probability that $x_i^a = 0^n$ is $1/2^n$.

Summing over at most q decryption queries, we get

$$\Pr[\mathcal{A} \text{ forges} \mid \neg\mathsf{bad}] \leq \frac{q}{2^n - q} \leq \frac{2q}{2^n}$$

by assuming $q \leq 2^{n-1}$. The proof is completed via Equation 1 and the bound of Lemma 7.

Lemma 7. *Assume that the adversary makes at most totally q encryption and decryption queries, p ideal TBC queries, with the total number of primitive calls among these q encryption and decryption queries being at most σ and the number of queried users being at most u, we have*

$$\Pr[\mathsf{bad}] \leq \frac{u^2 + 8d(\sigma+p)^2 + 8q}{2^{2n+1}} + \frac{c(\sigma+p+q)}{2^n} + \frac{2q^2 + q(\sigma+p)}{2^{3n}}$$

where $c = \max\{4n, 4u/2^n\}$ and assuming $\sigma + p \leq 2^{n-1}$.

Proof. The event bad consists of several conditions. Denote by cond_i the sub-event that the i-th condition is triggered. Note that the analyzes of conditions from cond_1 to cond_4 are similar to those in the proof of Lemma 4, and can be bounded by

$$\Pr\left[\bigvee_{i=1}^{4} \mathsf{cond}_i\right] \leq \frac{u^2}{2^{2n+1}} + \frac{1}{2^n} + \frac{cp + c\sigma}{2^n} + \frac{(c-1)q}{2^n}$$

where $c = \max\{4n, 4u/2^n\}$.

Next, we analyze the conditions from cond_5 to cond_8. If any of these four conditions is violated, then it implies that the adversary found a two-block collision of the hash function

Multihash by at most $\sigma + p$ TBC queries. On the other hand, conditioned on $\neg\mathsf{cond}_4$, this collision is not trivial. By Lemma 3, we get

$$\Pr\left[\bigvee_{j=5}^{8}\mathsf{cond}_j\right] \le \frac{8d(\sigma+p)^2}{2^{2n}} \ .$$

We then consider the condition cond_9. Since there is no collision on the hash function Multihash as bounded in the above case, both h^a_{i,ℓ_a+v_a+1} and k^a_{i,ℓ_a+v_a+1} are chosen uniformly at random from a set of size at least $2^n - \sigma - p$. Thus the probability that $h^a_{i,\ell_a+v_a+1} = P_j$ and $k^a_{i,\ell_a+v_a+1} = 0^n$ is at most $1/(2^n-\sigma-p)^2 \le 1/2^{2n-2}$ by assuming $\sigma + p \le 2^{n-1}$. Similarly to the analysis in condition (9) of Lemma 4, we have

$$\Pr[\,\mathsf{cond}_9\,] \le \frac{4q}{2^{2n}} + \frac{2q^2}{2^{3n}} \ .$$

Finally we analyze the condition cond_{10}. Following the similar argument in condition (10) of Lemma 4, we get

$$\Pr[\,\mathsf{cond}_{10}\,] \le \frac{q(\sigma+p)}{2^{3n}} \ .$$

Wrapping up,

$$\Pr[\,\mathsf{bad}\,] \le \frac{u^2 + 8d(\sigma+p)^2 + 8q}{2^{2n+1}} + \frac{c(\sigma+p+q)}{2^n} + \frac{2q^2+q(\sigma+p)}{2^{3n}} \ .$$

□