Secret Can Be Public: Low-Memory AEAD Mode for High-Order Masking

Yusuke Naito[1], Yu Sasaki[2], and Takeshi Sugawara[3]

[1] Mitsubishi Electric Corporation, Kanagawa, Japan,
Naito.Yusuke@ce.MitsubishiElectric.co.jp
[2] NTT Social Informatics Laboratories, Tokyo, Japan, yu.sasaki.sk@hco.ntt.co.jp
[3] The University of Electro-Communications, Tokyo, Japan, sugawara@uec.ac.jp

Abstract. We propose a new AEAD mode of operation for an efficient countermeasure against side-channel attacks. Our mode achieves the smallest memory with high-order masking, by minimizing the states that are duplicated in masking. An s-bit key-dependent state is necessary for achieving s-bit security, and the conventional schemes always protect the entire s bits with masking. We reduce the protected state size by introducing an *unprotected* state in the key-dependent state: we protect only a half and give another half to a side-channel adversary. Ensuring independence between the unprotected and protected states is the key technical challenge since mixing these states reveals the protected state to the adversary. We propose a new mode HOMA that achieves s-bit security using a tweakable block cipher with the $s/2$-bit block size. We also propose a new primitive for instantiating HOMA with $s = 128$ by extending the SKINNY tweakable block cipher to a 64-bit plaintext block, a 128-bit key, and a $(256+3)$-bit tweak. We make hardware performance evaluation by implementing HOMA with high-order masking for $d \leq 5$. For any $d > 0$, HOMA outperforms the current state-of-the-art PFB_Plus by reducing the circuit area larger than that of the entire S-box.

Keywords: Authenticated Encryption · High-Order Masking · Side-Channel Attack · Mode of Operation · Lightweight Cryptography

1 Introduction

There is a growing demand for extending information systems to the physical world by using network-enabled embedded devices, and lightweight cryptography (LWC) is the key technology enabling secure network communication in such resource-constrained devices. Designing lightweight symmetric-key cryptography is arguably the central topic in LWC research because extremely resource-constrained devices cannot afford the cost of implementing public-key cryptography. The National Institute of Standards and Technology (NIST) is currently conducting the LWC competition to determine the next standard of authenticated encryption with associated data (AEAD) schemes [33].

Such embedded devices that need LWC can be used in a hostile environment wherein a local attacker mounts power and/or electromagnetic side-channel attacks (SCAs) [24]. Thus, LWC designers face an even more challenging task of

realizing an SCA-resistant implementation with limited resources. In fact, countermeasures against SCAs are explicitly mentioned as design requirements in NIST's competition, and ISAP [13], which was designed with a focus on robustness against SCA, has recently been chosen as a finalist in the competition [34].

Masking, which splits the target value into a number of shares, is arguably the most common countermeasure against SCA [20,32]. The security of masking is based on the $\tilde{d}$-probing model, which considers an attacker who can probe $\tilde{d}$ wires [20]. A masking scheme with the protection order d resists attacks with up to d probes. A common strategy is to design a *gadget*, typically a secure Boolean AND operation, that securely maps the input shares into the corresponding output shares and to construct a target symmetric-key algorithm using them while ensuring the compositional security.

Large performance overhead is the major drawback of masking. In particular, the number of shares significantly impacts computational complexity. The early schemes used $(td+1)$ shares with $t > 1$ for achieving the protection order d and thus called $(td+1)$-masking [20]. Later, the researchers invented a new scheme that achieves the same protection order by using $(d+1)$ shares only [39]. In this paper, we focus on the $(d+1)$-masking schemes because they have a significant performance advantage over the $(td+1)$-masking schemes.

Such a masking scheme is also effective against statistical SCA with several assumptions regarding the noise level and leakage function; the number of side-channel traces to mount an attack, which is the key difficulty indicator, increases exponentially with the protection order d [37]. A sufficient protection order heavily depends on the target, and the recent experimental evaluations suggests that $d \approx 5$ is practical. For example, Cassiers et al. verified their masking scheme up to $d = 3$ using 9 million traces which is close to the practical limit [9,10].

1.1 Low-Memory AEAD for Masking

As we reduce the circuit area for combinatorial logic gates by exploiting the area-latency trade-off with sophisticated serial architectures [26, 27], memory (register) becomes more and more dominant. The overhead of masking is also critical because it duplicates the target state for shared representation. Since reducing the memory size within a block cipher is difficult, researchers have been tackling the problem at the higher layer, and have proposed several masking-friendly AEAD modes achieving small memory sizes after masking [21,26,29].

We summarize the memory costs for achieving s-bit security in the state-of-the-art AEAD schemes in Table 1. All conventional schemes, including the conventional block cipher (BC) based and permutation (P) based schemes [12, 25], use the total memory size of $3s$ bits without SCA protection (see the column with $d = 0$). That is because we need (i) $2s$-bit information carried between blocks to achieve s-bit security against internal-state collisions, and (ii) an s-bit key indispensable for the security against exhaustive search.

In contrast, the schemes have different memory sizes after masking. As summarized in Table 1, the memory is categorized into three types:

Table 1. Memory size for masking implementations with s-bit security. The security of the existing schemes are evaluated in the conventional AE-security [30] or its related notions. HOMA is evaluated in a new security notion, which ensures the same security properties as the conventional one while leaking unprotected values to adversaries.

Scheme	Public	Key-Dependent		Key[†]	$(d+1)$ Masking				Ref.
		Protected[†]	Unprotected		$d=0$[‡]	$d=1$	$d=2$	$d=\hat{d}$	
P-based	—	$2s$	—	s	$3s$	$6s$	$9s$	$3s(\hat{d}+1)$	[12]
BC-based	—	$2s$	—	s	$3s$	$6s$	$9s$	$3s(\hat{d}+1)$	[25]
TBC-based[§]	s	s	—	s	$3s$	$5s$	$7s$	$2s(\hat{d}+1)+s$	[21, 26, 29]
HOMA	$1.5s$	$0.5s$	$0.5s$	s	$3.5s$	$5s$	$6.5s$	$1.5s(\hat{d}+1)+2s$	Ours

[†]The key and the key-dependent protected state are encoded into $(d+1)$ shares in $(d+1)$-masking.
[‡]$d=0$ corresponds to an implementation without any SCA countermeasure.
[§]This category includes PFB, Romulus, and PFB_Plus.

- *Public:* a state that can be computed only with input values to the encryption or decryption algorithm (without a key),
- *Key-dependent:* a state that requires knowledge of the key,
- *Key:* a secret key.

The public state needs no SCA protection, and the scheme with a larger public state has a smaller memory size after masking (see the column with $d > 0$). In particular, the recent beyond-the-birthday-bound schemes using Tweakable BC (TBC), namely PFB [29], Romulus [21], and PFB_Plus [26], use a public tweak for reducing the size of the key-dependent state within the internal state. These schemes achieve $2s(d+1)+s$ bits of memory with $(d+1)$-masking, which is better than the conventional BC-based or P-based schemes with $3s(d+1)$ bits.

In this paper, we pursue this direction and study a new mode of operation that minimizes the state size after $(d+1)$-masking. The key technical challenge is to reduce the key-dependent state beyond the conventional schemes. The existing masking-friendly AEADs (PFB, Romulus, and PFB_Plus) use masking to both the key-dependent state and the key. The s-bit memory for the secret key has no room for improvement. Besides, protecting the remaining key-dependent s-bit state has also been believed to be necessary for achieving s-bit security. We refer to this as "the s-bit secret barrier" hereafter. The existing masking-friendly AEADs are optimal under this belief.

1.2 Summary of Contributions

This paper makes three main contributions: (i) a new mode HOMA, (ii) an instantiation for HOMA, including a new TBC as an underlying primitive, and (iii) concrete implementations and performance benchmarking of HOMA.

(i) New Mode (Section 3) and Its Proof (Section 4.3). First, we propose a new TBC-based AEAD mode-of-operation HOMA that achieves the smallest

memory of all existing schemes for $(d+1)$ masking (see Fig. 1-(center) and -(right) for its core procedure). For further reducing memory, we consider dropping SCA protection from a part of the s-bit key-dependent states. Hence, we decompose the key-dependent state into "unprotected" and "protected" states.

- *Unprotected:* a key-dependent state without SCA protection in a raw form
- *Protected:* a key-dependent state with SCA protection in a shared form

The protected state is protected with high-order masking using $(d+1)$ shares, and has the protection order d. The unprotected state is represented without shares and an SCA adversary potentially has unlimited access. To capture this worst-case scenario, we define a security notion that all the unprotected values are revealed whereas the protected values are secret. With the leakage of the unprotected state, the secret state becomes smaller than s bits, which allows a birthday attack with $s/2$-bit complexity, as we discuss in Section 3. HOMA addresses this attack by introducing random IV without increasing memory size.

A TBC's internal state, directly updated with a key, must be protected. Hence, we design a mode such that a TBC's internal state is the only state that requires SCA protection. Moreover, the TBC's block size should be as small as possible. PFB_Plus's idea of using a small block size is beneficial to our mode. PFB_Plus divides the s-bit key-dependent state and updates a half by a TBC and another half by XORing the TBC output, as shown in Fig. 1-(left). However, simply unprotecting the latter $s/2$ bits in PFB_Plus ends up with a trivial attack. We consider $v_3 = v_1 \oplus v_2$ in Fig. 1-(left). Unprotecting the latter half of the state means that both v_3 and v_1 are revealed. This immediately reveals supposedly protected v_2 because $v_2 = v_1 \oplus v_3$. Then, a collision on the whole state can be generated only by a collision on v_3 because the difference in v_2 can be canceled by injecting the difference from A_{i+1}. Hence, security decreases to $s/2$ bits.

Addressing the issue, HOMA uses the structure in Fig. 1-(center) and -(right). Considering that each TBC call produces an $s/2$-bit random value, HOMA calls a TBC twice to sufficiently mix the s-bit internal state (and additionally calls a TBC to encrypt a plaintext block in the encryption), which enables us to prove the s-bit security of HOMA. In Fig. 1-(center) and -(right), the red lines are protected and represented with $(d+1)$ shares and the TBC and fix0 implementations are protected with $(d+1)$-masking, and the black lines remain unprotected.

With the above security notion, we prove that by fixing the TBC size to n bits, HOMA achieves $2n$-bit security. As a result, HOMA ensures s-bit security only with a protected state of size $s/2$ bits (smaller than s bits) and an s-bit key. As a drawback, HOMA needs three (resp. two) TBC calls for each data block for encryption (resp. AD processing). This yields some overhead in latency, but its impact on memory size is negligible. Another drawback is that HOMA requires a random IV of s bits, which is crucial to ensure the s-bit security when the unprotected state is $s/2$ bits, in addition to a nonce that is an additional overhead of traffic data. Note that we can comfortably assume the availability of a random generator because it is necessary for masking[4].

[4] Some masking implementations use non-cryptographic PRNGs, e.g., a simple LFSR, insufficient for the random IV. A hardware TRNG for seeding should be used instead.

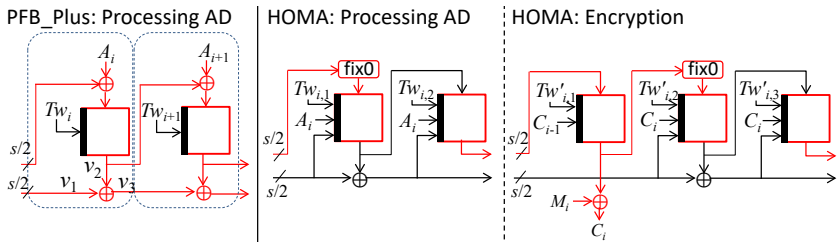

Fig. 1. PFB_Plus's structure (left) and HOMA's structure (center and right). A_j is an AD block and M_i/C_i is a plaintext/ciphertext block. The red (resp. black) lines are protected (resp. unprotected). Tw_j is a tweak. fix0 is a function fixing a LSB to 0. Each dotted circle of PFB_Plus represents a component of processing one data block.

As summarized in Table 1, HOMA uses a $1.5s$-bit public state, a $0.5s$-bit protected state, a $0.5s$-bit unprotected state, and an s-bit key. Hence, without masking implementation, the state size is $3.5s$ bits, which is worse than those of existing modes. However, with $(d+1)$ masking, HOMA achieves $1.5(d+1)+2s$, which is the smallest for $d > 0$ and asymptotically reduces memory by 25%.

(ii) Instantiation of HOMA with a New TBC (Section 5). HOMA for $s = 128$ requires a TBC that supports a 64-bit block, a 128-bit key, and a $(256 + 3)$-bit tweak, where the 3 bits are for domain separation of the mode. No existing TBC efficiently supports those configurations. Moreover, tweak- and key-schedules must be designed so that the tweak (public) is not mixed with the key (key-dependent) to avoid $(d+1)$ masking of the tweak state. We found that the tweak- and key-schedules of SKINNY [3] satisfy this requirement, thus we design a new TBC "SKINNYee" by basing its structure on SKINNY. The tweaky (a combination of a tweak and a key) size of SKINNY is 64, 128, or 192 bits, and SKINNYe [26] extended it to 256 bits, while our TBC needs $(128+256+3) = 387$ bits of key and tweak. This is challenging because the tweakey size extension done by SKINNY and SKINNYe cannot exceed 256 bits due to the limited design space. We resolve it by processing a key and a tweak as independent objects. Moreover, we absorb the 3-bit tweak by initializing a linear feedback shift register (LFSR) to a tweak-dependent value, which is more efficient than existing methods to extend the tweak size by a few bits [11,27]. Besides, we modify the LFSR clocking method of SKINNY so that the implementation is optimized for small memory.

(iii) Implementation (Section 6). We propose a hardware architecture for HOMA instantiated with SKINNYee and make a concrete performance comparison with the conventional state-of-the-art PFB_Plus. For the high-order masking, we use Cassiers et al.'s HPC2 [9, 10] for its glitch resistance, composability, and availability of an open-source implementation [8]. This is also the first HPC2 implementation of the SKINNY-based primitives and its S-box. We make an ASIC performance evaluation for the protection order $d \in \{0, \cdots, 5\}$ using a

45-nm CMOS standard cell library (see Table 3). As a result, HOMA always outperformed PFB_Plus with SCA protection, i.e., for any $d > 0$. Although the cost of the S-box circuit grows quadratically with d, in contrast to the memory size that grows only linearly, the results confirm that the memory elements still dominate the hardware cost with those practical protection orders. In particular, for any protection order $d > 0$, HOMA saved the circuit area larger than that of the entire S-box. This significant area reduction is impossible with the conventional approaches focusing on S-box, i.e., reducing S-box's multiplicative complexity [1, 16, 17] and improving each AND gadget [9, 10].

1.3 Related Work

Optimization for $(td + 1)$ Masking. PFB_Plus is optimized for $(td+1)$ masking with $t > 1$, for Nikova et al.'s threshold implementation (TI) [32] in particular. $(td + 1)$-masking use the different number of shares between the linear and non-linear states: those states require $(d + 1)$ and $(td + 1)$ shares, respectively. To exploit this property, PFB_Plus increases the ratio of a linearly updated state, within the s-bit secret barrier, and achieves a smaller memory after $(td + 1)$-masking. Unfortunately, PFB_Plus's benefit disappears with a $(d + 1)$-masking, which uses the same number of shares for non-linearly and linearly updated states. TI's extension to $d \geq 2$ turned out to be non-trivial [7, 38], and researchers are studying $(d+1)$-masking as a viable option for high-order masking [39]. HOMA takes another approach of breaking the s-bit secret barrier and achieves a smaller memory with $(d+1)$ masking as shown in Table 1. Moreover, even with the 3-share TI, HOMA achieves the same memory size as PFB_Plus.

Leakage-Resilient (LR) Cryptography. LR cryptography studies symmetric-key schemes, including AEAD, with provable security against SCA [2, 5, 6, 13–15, 36]. The early LR schemes relied on the bounded leakage model that limits the amount of leakage for each measurement [15]. However, limiting the number of measurements turned out to be impractical with a stateless primitive [4]. Addressing the issue, some recent LR schemes, including TEDT [6] and Spook [5], use a leak-free primitive supposedly realized with masking [14]. These modes can be faster than HOMA because they efficiently use unprotected primitives. Meanwhile, TEDT/Spook is not optimized for memory usage; protecting its s-bit TBC with masking requires the similar memory size as the other TBC-based schemes in Table 1. The additional components, including an independent unprotected TBC/Permutation implementation, can further increase the memory size.

Other LR schemes, including ISAP [13], pursue exclusive use of leaky primitives by limiting the target to non-adaptive attackers. ISAP can go beyond Table 1 because it does not rely on masking, and the memory size is independent of the protection order d. Meanwhile, the security of these schemes relies entirely on the restricted input space to the leaky primitives, which has several limitations compared with masking. In particular, they provide no guarantee against template attacks [14] and single-trace attacks [23].

Masking-Friendly Primitives. Those primitives use the S-box with a small

multiplicative complexity to be easy to mask [1,16,17]. HOMA has a high affinity for masking-friendly primitives. Most of designs as stand-alone primtives are for block ciphers, while there are several TBCs designed along with a mode. `Clyde-128` [5], `Scream`, and `iScream` [18] are such examples. Here we design a SKINNY variant for making the performance comparison clearer.

2 Preliminaries

Notation. Let ε denote the empty string. For a positive integer i, let $\{0,1\}^i$ denote the set of all i-bit strings. Let $\{0,1\}^*$ denote the set of all bit strings. For integers $i \le j$, let $[i,j] := \{s \mid i \le s \le j\}$ be the set of integers from i to j. For a positive integer i, let $[i] := [1,i]$ and $(i) := [0,i]$. For a finite set $\mathcal{T}$, $T \xleftarrow{\$} \mathcal{T}$ denotes an element is chosen uniformly at random from $\mathcal{T}$ and is assigned to T. The concatenation of two bit strings X and Y is written as $X\|Y$ or XY when no confusion is possible. For integers $0 \le i \le j$ and $X \in \{0,1\}^j$, let $\mathsf{msb}_i(X)$ resp. $\mathsf{lsb}_i(X)$ be the most resp. least significant i bits of X, and $|X|$ be the number of bits of X, i.e., $|X| = j$. For an integer $n > 0$ and a bit string X, we denote the parsing into fixed-length n-bit strings as $(X_1, X_2, \ldots, X_\ell) \xleftarrow{n} X$, where if $X \neq \varepsilon$ then $X = X_1\|X_2\|\cdots\|X_\ell$, $|X_i| = n$ for $i \in [\ell-1]$, and $0 < |X_\ell| \le n$; if $X = \varepsilon$ then $\ell = 1$ and $X_1 = \varepsilon$.

TBC. Let n be a block size. A TBC is a set of n-bit permutations indexed by a key and a public input called tweak, that is, fixing a key and a tweak, it becomes an n-bit permutation. Let $\mathcal{K}$ be the set of keys, $\mathcal{TW}$ be the set of tweaks, and n be the input/output-block size. An encryption is denoted by $\widetilde{E} : \mathcal{K} \times \mathcal{TW} \times \{0,1\}^n \to \{0,1\}^n$, $\widetilde{E}$ having a key $K \in \mathcal{K}$ is denoted by $\widetilde{E}_K$. For an input $(K, Y, X) \in \mathcal{K} \times \mathcal{TW} \times \{0,1\}^n$, the output is denoted by $\widetilde{E}_K(Y, X)$.

In this paper, a TBC is assumed to be a secure tweakable-pseudo-random permutation (TPRP), i.e., indistinguishable from a tweakable random permutation (TRP). A tweakable permutation (TP) $\widetilde{P} : \mathcal{TW} \times \{0,1\}^n \to \{0,1\}^n$ is a set of n-bit permutations indexed by a tweak in $\mathcal{TW}$. A TP $\widetilde{P}$ having a tweak $TW \in \mathcal{TW}$ is denoted by $\widetilde{P}^{TW}$. Let $\widetilde{\mathsf{Perm}}(\mathcal{TW}, \{0,1\}^n)$ be the set of all TPs: $\mathcal{TW} \times \{0,1\}^n \to \{0,1\}^n$. A TRP is defined as $\widetilde{P} \xleftarrow{\$} \widetilde{\mathsf{Perm}}(\mathcal{TW}, \{0,1\}^n)$. In the TPRP-security game, an adversary $\mathbf{A}$ has access to either $\widetilde{E}_K$ or $\widetilde{P}$, where $K \xleftarrow{\$} \mathcal{K}$ and $\widetilde{P} \xleftarrow{\$} \widetilde{\mathsf{Perm}}(\mathcal{TW}, \{0,1\}^n)$, and after the interaction, $\mathbf{A}$ returns a decision bit $\in \{0,1\}$. The output of $\mathbf{A}$ with access to $\mathcal{O}$ is denoted by $\mathbf{A}^{\mathcal{O}} \in \{0,1\}$. Then, the TPRP-security advantage function of $\mathbf{A}$ is defined as $\mathbf{Adv}^{\mathsf{tprp}}_{\widetilde{E}_K}(\mathbf{A}) := \Pr[\mathbf{A}^{\widetilde{E}_K} = 1] - \Pr[\mathbf{A}^{\widetilde{P}} = 1]$, where the probabilities are taken over K, $\widetilde{P}$, and $\mathbf{A}$. The maximum advantage over all adversaries, running in time at most t and making at most q queries, is denoted by $\mathbf{Adv}^{\mathsf{tprp}}_{\widetilde{E}_K}(q,t) := \max_{\mathbf{A}} \left(\mathbf{Adv}^{\mathsf{tprp}}_{\widetilde{E}_K}(\mathbf{A})\right)$.

AEAD. An AEAD scheme based on a TBC $\widetilde{E}_K$, denoted by $\Pi[\widetilde{E}_K]$, is a pair of encryption and decryption algorithms $(\Pi.\mathsf{Enc}[\widetilde{E}_K], \Pi.\mathsf{Dec}[\widetilde{E}_K])$. $\mathcal{K}$, $\mathcal{IV}$, $\mathcal{M}$, $\mathcal{C}$, $\mathcal{A}$, and $\mathcal{T}$ are the sets of keys, initialization vectors, plaintexts, ciphertexts,

associated data (AD), and tags of $\Pi[\widetilde{E}_K]$, respectively. For our scheme, the set of keys of $\Pi[\widetilde{E}_K]$ is equal to that of the underlying TBC. The encryption algorithm takes an initial vector $IV \in \mathcal{IV}$, an AD $A \in \mathcal{A}$, and a plaintext $M \in \mathcal{M}$, and returns, deterministically, a pair of a ciphertext $C \in \mathcal{C}$ and a tag $T \in \mathcal{T}$. The decryption algorithm takes a tuple $(IV, A, C, T) \in \mathcal{IV} \times \mathcal{A} \times \mathcal{C} \times \mathcal{T}$ and returns, deterministically, either the distinguished invalid symbol $\mathbf{reject} \notin \mathcal{M}$ or a plaintext $M \in \mathcal{M}$. We require that for any $(IV, A, M), (IV', A', M') \in \mathcal{IV} \times \mathcal{A} \times \mathcal{M}$, $|\Pi.\mathsf{Enc}[\widetilde{E}_K](IV, A, M)| = |\Pi.\mathsf{Enc}[\widetilde{E}_K](IV, A, M')|$ is satisfied if $|M| = |M'|$. We also require that $\Pi.\mathsf{Dec}(IV, A, \Pi.\mathsf{Enc}[\widetilde{E}_K](IV, A, M)) = M$ for $IV \in \mathcal{IV}$, $A \in \mathcal{A}$, and $M \in \mathcal{M}$.

In this paper, $\mathcal{IV}$ consists of a set of nonces denoted by $\mathcal{N}$ and a set of random IVs denoted by $\mathcal{R}$ thus $\mathcal{IV} = \mathcal{N} \times \mathcal{R}$. For nonces of $\Pi.\mathsf{Enc}[\widetilde{E}_K]$, repeating the same nonce is forbidden within the same key.[5] For an input tuple $(N, R, A, M) \in \mathcal{N} \times \mathcal{R} \times \mathcal{A} \times \mathcal{M}$ of $\Pi.\mathsf{Enc}[\widetilde{E}_K]$, a random IV R is chosen independently of other elements (N, A, M) and uniformly at random from $\mathcal{R}$. Then, (N, R, A, M) is passed to $\Pi.\mathsf{Enc}[\widetilde{E}_K]$.

AE Security. We explain the AE-security notion [30], on which our security goal is based.[6]

The AE-security is the indistinguishability between the real and ideal worlds. The real-world oracles are $(\Pi.\mathsf{Enc}[\widetilde{E}_K], \Pi.\mathsf{Dec}[\widetilde{E}_K])$ wherein the key K is defined as $K \xleftarrow{\$} \mathcal{K}$. The ideal-world oracles are $(\$, \perp)$ wherein $\$$ is a random-bits oracle that returns a random bit string of length $|\Pi.\mathsf{Enc}_K[\widetilde{E}](N, R, A, M)|$ for an encryption query (N, A, M), and $\perp$ is a reject oracle that returns **reject** for any decryption query. Note that for each encryption query (N, A, M), the random IV is defined as $R \xleftarrow{\$} \mathcal{R}$. The AE-advantage function of an adversary $\mathbf{A}$ that returns a decision bit after interacting with $\Pi[\widetilde{E}_K]$ in the real world or with $(\$, \perp)$ in the ideal world is defined as $\mathbf{Adv}^{\mathsf{ae}}_{\Pi[\widetilde{E}_K]}(\mathbf{A}) = \Pr[\mathbf{A}^{\Pi.\mathsf{Enc}[\widetilde{E}_K], \Pi.\mathsf{Dec}[\widetilde{E}_K]} = 1] - \Pr[\mathbf{A}^{\$, \perp} = 1]$, where the probabilities are taken over $K, \$, \mathbf{A}$, and random IVs. $\mathbf{A}$ is nonce-respecting, that is, all nonces in queries to $\Pi.\mathsf{Enc}[\widetilde{E}_K]/\$$ are distinct. In this game, making a trivial query $(N, R, A, C, \hat{T})$ to $\Pi.\mathsf{Dec}[\widetilde{E}_K]/\perp$ is forbidden, which is defined by some previous query to $\Pi.\mathsf{Enc}[\widetilde{E}_K]/\$$.

3 Design of AEAD Mode for High-Order Masking

3.1 Intuition and Design of HOMA

High-Level Structure. To design an s-bit secure mode, the size of the key-dependent state must be at least s bits, whereas to design a masking-friendly mode, the size of the protected state must be less than s bits to be smaller than the existing designs. The minimum size of the protected state is the block size of the underlying TBC, since a state in a TBC includes information of the

[5] For $\Pi.\mathsf{Dec}[\widetilde{E}_K]$, nonces and random IVs can be repeated.

[6] The AE-security notion does not take into account SCA.

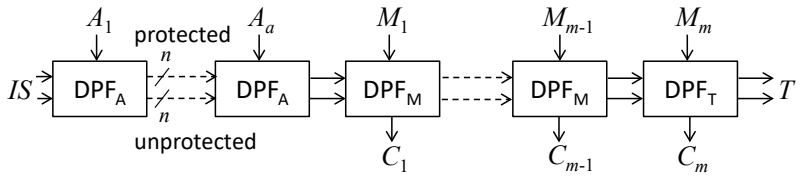

Fig. 2. The high-level structure of HOMA

key. Thus, the security of a masking-friendly mode must be beyond the block size. HOMA is designed so that, with a TBC of n-bit block, the security level is $2n$ bits, the key-dependent state size is $2n$ bits, and the unprotected state size is n bits. In other words, for the target security level s, HOMA has the s-bit key-dependent state with $s/2$-bit protected and $s/2$-bit unprotected ones.

Figure 2 shows the high-level structure of HOMA. It starts from the $2n$-bit initial state IS and updates the state by iterating a data processing function (DPF). In this iteration, we first process AD blocks $A_1, \ldots, A_a$ and then process plaintext blocks $M_1, \ldots, M_m$ while generating ciphertext blocks $C_1, \ldots, C_m$. Each DPF takes as input a public state, including a nonce and a counter, but we omit them from the figure for simplicity. In the process of the last plaintext block M_m, we define a tag T as well as the last ciphertext block C_m.

We then specify DPFs. DPFs for processing AD, plaintext blocks before the last block, and the last plaintext block (with tag generation) are similar but slightly different. We denote them by DPF$_\mathsf{A}$, DPF$_\mathsf{M}$, and DPF$_\mathsf{T}$, respectively. To design DPFs, we need to carefully define protected and unprotected states. This is because once a protected value v_p is mixed with an unprotected value v_{up} and the resulting value v is unprotected, the protected value can be leaked (e.g., if $v = v_p \oplus v_{up}$, then one can obtain v_p $(= v \oplus v_{up})$). With this important point in mind, we designed DPF$_\mathsf{A}$, DPF$_\mathsf{M}$, and DPF$_\mathsf{T}$, which are depicted in Fig. 3.

DPF$_\mathsf{A}$**.** Each DPF$_\mathsf{A}$ must randomize the entire $2n$-bit state to avoid a state collision so that the protected state must not be mixed with the unprotected one. We thus call a TBC twice to provide $2n$-bit randomness as Fig. 3(top,left). For each TBC call, the tweak is a concatenation of a domain separation d_i, a nonce N, a counter, the AD block A_i, and the current unprotected state value. fix0 is a function that fixes the LSB to 0.[7]

DPF$_\mathsf{M}$**.** To process each plaintext block, we first call a TBC to generate an n-bit key stream, then the same procedure as DPF$_\mathsf{A}$ is performed to update the whole state. DPF$_\mathsf{M}$ is shown in Fig. 3(top,right).

DPF$_\mathsf{T}$**.** The DPF encrypts the last plaintext block and generates a tag simultaneously. As shown in Fig. 3(bottom), we first call a TBC to generate an n-bit key stream to encrypt the plaintext block, then a TBC is iteratively applied twice to

[7] The function is introduced for the security proof that ensures that the TBC output provides a randomness to the unprotected state. It ensures that the output is chosen uniformly at random from at least 2^{n-1} elements. Note that fix0 can be removed by reserving a bit in a tweak space that takes the LSB of the TBC input.

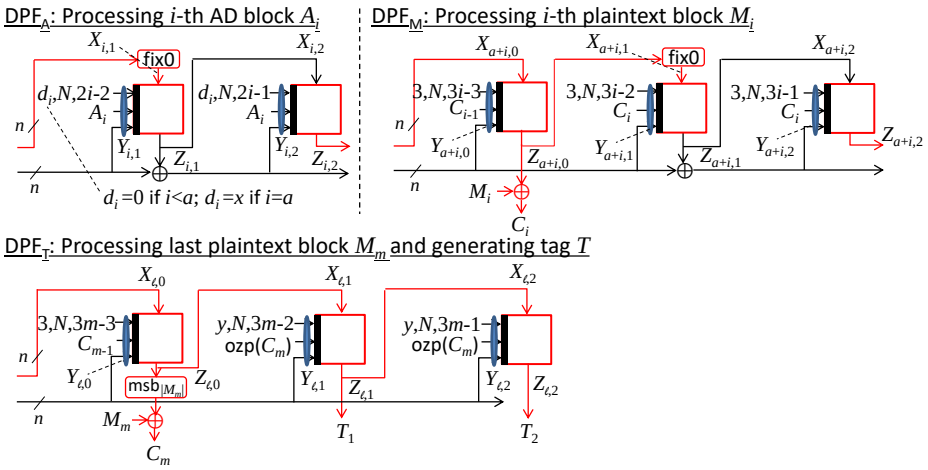

Fig. 3. DPFs of HOMA. The red lines are protected and the others are unprotected.

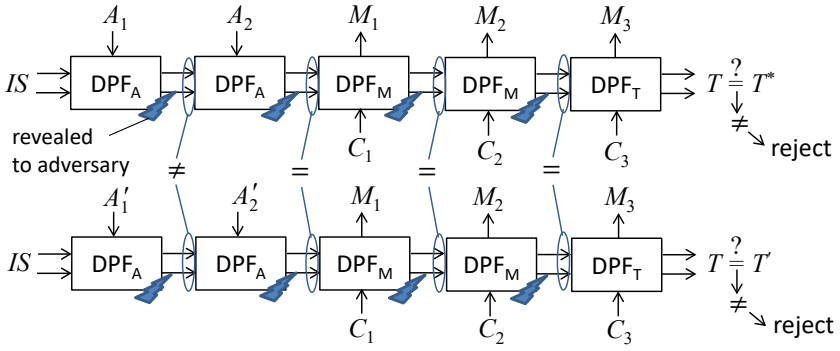

Fig. 4. A collision in decryption procedures.

generate the $2n$-bit tag. For the encryption, a tag is a conventional output, thus no protection is required, while for the decryption, the tag must be protected, since no information should be output for an invalid tag.

Random IV. For the encryption, we use a random IV of $2n-1$ bits as the initial state IS. This is because, without a random IV, the AEAD in Fig. 2 is vulnerable against a state-collision attack. The details are as follows.

Assume that IS is not random. Then an adversary can fix IS to some constant in both the encryption and decryption procedures. The SCA adversary first interacts with the decryption oracle to cause a collision of DPF, which is shown in Fig. 4. In decryption queries, the adversary can make IS values the same even in the nonce-respect setting. In this attack, distinct ADs, an identical ciphertext, and any tag are used to cause a collision of the state after processing AD (after the second AD block in Fig. 4). The key point here is that the SCA adversary can access to the unprotected state, which enables to detect the oc-

Algorithm 1 HOMA

Encryption $\mathsf{HOMA.Enc}[\widetilde{E}_K](N, R, A, M)$

1: $(H_1, H_2, C_0) \leftarrow \mathsf{HOMA.Hash}[\widetilde{E}_K](N, R, A)$
2: $(C, T) \leftarrow \mathsf{HOMA.Main}[\widetilde{E}_K](N, H_1, H_2, C_0, M)$; **return** (R, C, T)

Decryption $\mathsf{HOMA.Dec}[\widetilde{E}_K](N, R, A, C, \hat{T})$

1: $(H_1, H_2, C_0) \leftarrow \mathsf{HOMA.Hash}[\widetilde{E}_K](N, R, A)$
2: $(M, T) \leftarrow \mathsf{HOMA.Main}[\widetilde{E}_K](N, H_1, H_2, C_0, C)$
3: **if** $\hat{T} = T$ **then return** M; **else return reject**

Processing AD $\mathsf{HOMA.Hash}[\widetilde{E}_K](N, R, A)$

1: $\mathrm{St} \leftarrow \mathsf{msb}_{n-1}(R)\|0$; $\mathrm{Sb} \leftarrow \mathsf{lsb}_n(R)$; $(A_1, \ldots, A_a) \xleftarrow{n} A$
2: **for** $i = 1, \ldots, a-1$ **do** $(\mathrm{St}, \mathrm{Sb}) \leftarrow \mathsf{SUF}[\widetilde{E}_K](0, N, 2(i-1), A_i, \mathrm{St}, \mathrm{Sb})$
3: **if** $|A| \bmod n = 0$ **then** $x = 1$; **else** $x = 2$
4: $(\mathrm{St}, \mathrm{Sb}) \leftarrow \mathsf{SUF}[\widetilde{E}_K](x, N, 2(a-1), \mathsf{ozp}(A_a), \mathrm{St}, \mathrm{Sb})$; **return** $(\mathrm{St}, \mathrm{Sb}, \mathsf{ozp}(A_a))$

Main $\mathsf{HOMA.Main}[\widetilde{E}_K](N, H_1, H_2, C_0, D)$ ▷ If D is a plaintext M (resp. ciphertext C), then D' is the ciphertext C (resp. plaintext M).

1: $D' \leftarrow \varepsilon$; $\mathrm{St} \leftarrow H_1$; $\mathrm{Sb} \leftarrow H_2$; $(D_1, \ldots, D_m) \xleftarrow{n} D$
2: **for** $i = 1, \ldots, m-1$ **do**
3: $\quad \mathrm{St} \leftarrow \widetilde{E}_K((3, N, 3(i-1), C_{i-1}, \mathrm{Sb}), \mathrm{St})$; $D'_i \leftarrow \mathrm{St} \oplus D_i$
4: $\quad (\mathrm{St}, \mathrm{Sb}) \leftarrow \mathsf{SUF}[\widetilde{E}_K](3, N, 3(i-1)+1, C_i, \mathrm{St}, \mathrm{Sb})$
5: **end for**
6: $\mathrm{St} \leftarrow \widetilde{E}_K((3, N, 3(m-1), C_{m-1}, \mathrm{Sb}), \mathrm{St})$; $D'_m \leftarrow \mathsf{msb}_{|D_m|}(\mathrm{St}) \oplus D_m$
7: **if** $|D| \bmod n = 0$ **then** $y = 4$; **else** $y = 5$
8: $T_1 \leftarrow \widetilde{E}_K((y, N, 3(m-1)+1, \mathsf{ozp}(C_m), \mathrm{Sb}), \mathrm{St})$
9: $T_2 \leftarrow \widetilde{E}_K((y, N, 3(m-1)+2, \mathsf{ozp}(C_m), \mathrm{Sb}), T_1)$; **return** $(D'_1\| \cdots \|D'_m, T_1\|T_2)$

State Update $\mathsf{SUF}[\widetilde{E}_K](d, N, u, D, \mathrm{St}, \mathrm{Sb})$

1: $\mathrm{St} \leftarrow \mathsf{fix0}(\mathrm{St})$; $\mathrm{St} \leftarrow \widetilde{E}_K((d, N, u, D, \mathrm{Sb}), \mathrm{St})$ ▷ The TBC output is unprotected
2: $\mathrm{Sb} \leftarrow \mathrm{St} \oplus \mathrm{Sb}$; $\mathrm{St} \leftarrow \widetilde{E}_K((d, N, u+1, D, \mathrm{Sb}), \mathrm{St})$; **return** $(\mathrm{St}, \mathrm{Sb})$

currence of the collision of the entire state without knowing the protected state by observing if collisions on the unprotected state occur in all subsequent blocks. After finding a collision, an adversary makes an encryption query with the same (A_1, A_2), and the modified plaintext (M_1^*, M_2^*, M_3^*) under the same IS to obtain the tag T^*. Since T^* is also valid for (A'_1, A'_2) and (M_1^*, M_2^*, M_3^*), the integrity is broken by $O(2^n)$ queries (from the birthday analysis).

By introducing a random IV, the adversary cannot perform the attack unless a random IV of $2n$ bits is predicted by spending $O(2^{2n})$ complexity.

3.2 Specification of HOMA

The specification of HOMA is given in Algorithm 1. Let ν and c be nonce and counter sizes. Thus, $\mathcal{N} := \{0,1\}^\nu$. Let $\mathcal{R} := \{0,1\}^{2n-1}$, $\mathcal{A} := \{0,1\}^*$, $\mathcal{M} :=$

$\{0,1\}^*$, $\mathcal{C} := \mathcal{M}$, and $\mathcal{T} := \{0,1\}^{2n}$. Let $\mathsf{ozp} : \{0,1\}^{\leq n} \to \{0,1\}^n$ be the one-zero padding function: for $X \in \{0,1\}^{\leq n}$, $\mathsf{ozp}(X) = X$ if $|X| = n$; $\mathsf{ozp}(X) = X\|10^{n-1-|X|}$ if $|X| < n$. The set of tweaks is defined as $\mathcal{TW} := (5] \times \mathcal{N} \times \{0,1\}^c \times \{0,1\}^n \times \{0,1\}^{2n}$. HOMA.Enc (resp. HOMA.Dec) is the encryption (resp. decryption) of HOMA. HOMA.Enc takes a nonce $N \in \mathcal{N}$, a random IV $R \in \mathcal{R}$, an AD $A \in \mathcal{A}$, and a plaintext $M \in \mathcal{M}$, and returns the ciphertext $C \in \{0,1\}^{|M|}$ and the tag $T \in \mathcal{T}$, where it is required that R is chosen uniformly at random from $\mathcal{R}$ and N is a non-repeated value within the same key. HOMA.Dec takes a nonce $N \in \mathcal{N}$, an IV $R \in \mathcal{R}$, an AD $A \in \mathcal{A}$, a ciphertext $C \in \mathcal{C}$ and a tag $\hat{T} \in \mathcal{T}$, and returns the plaintext $M \in \{0,1\}^{|C|}$ if the tag is valid and **reject** if the tag is invalid. HOMA.Hash is a function that processes a nonce $N \in \mathcal{N}$, an IV $R \in \mathcal{R}$, and an AD $A \in \mathcal{A}$. HOMA.Main is a function that processes a nonce $N \in \mathcal{N}$, a plaintext/ciphertext and generates a tag. $\mathsf{SUF}[\widetilde{E}_K]$ is a function that updates the $2n$-bit state,[8] where d is a domain separation value, u is a counter value, D is a data block, St is the protected state, and Sb is the unprotected state. In HOMA, domain separation values are 0 when processing AD blocks except for the last AD block, $x \in \{1,2\}$ when processing the last AD block,[9] 3 when processing the plaintext/ciphertext blocks except for the last block, and $y \in \{4,5\}$ when processing the last plaintext/ciphertext block and generating a tag.[10] The counter value at the i-th TBC call in HOMA.Hash/HOMA.Main is $i-1$. In Algorithm 1, counter values are denoted by integers for simplicity, but the values are handled as the c-bit strings.

3.3 Protected and Unprotected Values of HOMA

We define unprotected TBC outputs in each DPF: $\mathsf{DPF_A}$: the first TBC output; $\mathsf{DPF_M}$: the second TBC output; $\mathsf{DPF_T}$: none. These outputs are the colored TBC one in SUF of Algorithm 1. Other TBC outputs are protected. In HOMA, all tweaks and a state updated with an unprotected TBC output are unprotected except for TBC computations. In Fig. 3, the colored lines are protected and other lines are unprotected.[11]

4 Security Claim and Proof of HOMA

4.1 AE Security for Masking

We define AEL-security, the security for masking, by extending the conventional AE-security [30] so that SCA adversaries for AEAD schemes with masking implementations can be considered. AEL-security is defined so that for a query to the

[8] The function SUF is the same for $\mathsf{DPF_A}$. In $\mathsf{DPF_M}$, a TBC is performed to encrypt/decrypt a plaintext/ciphertext block, then SUF is performed.

[9] If the length of the last block equals n, then $x = 1$, and otherwise $x = 2$.

[10] If the length of the last block equals n, then $y = 4$, and otherwise $y = 5$.

[11] For the encryption, T_0 and T_1 can be unprotected but plaintext blocks must be protected. The latter is necessary to ensure the privacy of plaintexts in real-world implementations but not in the security proof as an adversary chooses a plaintext.

target AEAD scheme $\Pi[\widetilde{E}_K]$, the adversary can obtain the unprotected values as well as the conventional output. Unlike the existing extension of the conventional AE-security in [2], our extension covers a larger class of leakage functions. Below, we define real-world and ideal-world oracles with leakage functions to access unprotected values.

First, the real-world oracles are defined. Let $\mathsf{EncUPV}[\widetilde{E}_K](N, R, A, M)$ resp. $\mathsf{DecUPV}[\widetilde{E}_K](N, R, A, C, \hat{T})$ be a leakage function for the encryption resp. the decryption, which returns unprotected values in the process of $\Pi.\mathsf{Enc}[\widetilde{E}_K](N, R, A, M)$ resp. $\Pi.\mathsf{Dec}[\widetilde{E}_K](N, R, A, C, \hat{T})$.

- Enc. oracle $\mathsf{EncL_R}[\widetilde{E}_K]$: For a query $(N, A, M) \in \mathcal{N} \times \mathcal{A} \times \mathcal{M}$, $R \xleftarrow{\$} \mathcal{R}$, and returns the outputs of $\Pi.\mathsf{Enc}[\widetilde{E}_K](N, R, A, M)$ and of $\mathsf{EncUPV}[\widetilde{E}_K](N, R, A, M)$.
- Dec. oracle $\mathsf{DecL_R}[\widetilde{E}_K]$: For a query $(N, R, A, C, \hat{T}) \in \mathcal{N} \times \mathcal{R} \times \mathcal{A} \times \mathcal{C} \times \mathcal{T}$, returns the outputs of $\Pi.\mathsf{Dec}[\widetilde{E}_K]$ $(N, R, A, C, \hat{T})$ and of $\mathsf{DecUPV}[\widetilde{E}_K](N, R, A, C, \hat{T})$.

Next, the ideal-world oracles are defined. The leakage of unprotected values is supported by introducing a simulator $\mathcal{S} = (\mathcal{S}_{\mathsf{encL}}, \mathcal{S}_{\mathsf{decL}})$ that simulates $(\mathsf{EncUPV}[\widetilde{E}_K], \mathsf{DecUPV}[\widetilde{E}_K])$.

- Enc. oracle $\mathsf{EncL_I}$: For a query $(N, A, M) \in \mathcal{N} \times \mathcal{A} \times \mathcal{M}$, $\mathsf{EncL_I}$ returns the outputs of $\$(N, A, M)$ and of $\mathcal{S}_{\mathsf{encL}}(N, R, A, C, T)$ where $(R, C, T) = \$(N, A, M)$.
- Dec. oracle $\mathsf{DecL_I}$: For a query $(N, R, A, C, \hat{T}) \in \mathcal{N} \times \mathcal{R} \times \mathcal{A} \times \mathcal{C} \times \mathcal{T}$, returns the outputs of $\perp$ and of $\mathcal{S}_{\mathsf{decL}}(N, R, A, C, \hat{T})$.

The simulator's task is to simulate unprotected values of the real world by using only public values.[12] If such simulator exists, i.e., the real and ideal worlds are indistinguishable, then one can ensure that the unprotected values provide nothing to differentiate the AEAD scheme from an ideal AEAD $(\$, \perp)$. Note that the simulator must be a polynomial-time algorithm, since the simulator represents a procedure of some polynomial-time adversary in the ideal world.

The AEL-security advantage function of an adversary $\mathbf{A}$, that returns a decision bit, after making all queries, is defined as

$$\mathbf{Adv}^{\mathsf{ael}}_{\Pi[\widetilde{E}_K],\mathcal{S}}(\mathbf{A}) = \Pr[\mathbf{A}^{\mathsf{EncL_R}[\widetilde{E}_K],\mathsf{DecL_R}[\widetilde{E}_K]} = 1] - \Pr[\mathbf{A}^{\mathsf{EncL_I},\mathsf{DecL_I}} = 1],$$

where the probabilities are taken over $K, R, \$, \mathcal{S}, \mathbf{A}$. Hereafter, we refer a query to $\mathsf{EncL_R}[\widetilde{E}_K]/\mathsf{EncL_I}$ (resp. $\mathsf{DecL_R}[\widetilde{E}_K]/\mathsf{DecL_I}$) an encryption (resp. decryption) query. This game forbids $\mathbf{A}$ making a trivial query: some encryption query-responses are forwarded to the decryption oracle.

A scheme $\Pi[\widetilde{E}_K]$ is AEL-secure if there exists a simulator such that the advantage function is bounded by a negligible probability. The goal of HOMA is to obtain a bound of $2n$-bit security (negligible up to $O(2^{2n})$ query complexity).

[12] To ensure the privacy, a plaintext M must be kept private to an adversary. Thus, the plaintext must not be included in a tuple of simulator's inputs.

Comparisons with Existing Notions. Barwell et al. [2] extended the conventional AE-security notion, where two oracles $\ell(\Pi.\mathsf{Enc}[\widetilde{E}_K]), \ell(\Pi.\mathsf{Dec}[\widetilde{E}_K])$ are introduced in addition to the standard oracles $\Pi.\mathsf{Enc}[\widetilde{E}_K]$, $\Pi.\mathsf{Dec}[\widetilde{E}_K]$, \$, and $\perp$. $\ell(\Pi.\mathsf{Enc}[\widetilde{E}_K])$ (resp. $\ell(\Pi.\mathsf{Dec}[\widetilde{E}_K])$) returns leak values of $\Pi.\mathsf{Enc}[\widetilde{E}_K]$ (resp. $\Pi.\mathsf{Dec}[\widetilde{E}_K]$) as well as the output of $\Pi.\mathsf{Enc}[\widetilde{E}_K]$ (resp. $\Pi.\mathsf{Dec}[\widetilde{E}_K]$). The real-world oracles are $(\Pi.\mathsf{Enc}[\widetilde{E}_K], \Pi.\mathsf{Dec}[\widetilde{E}_K], \ell(\Pi.\mathsf{Enc}[\widetilde{E}_K]), \ell(\Pi.\mathsf{Dec}[\widetilde{E}_K]))$ and the ideal-world ones are $(\$, \perp, \ell(\Pi.\mathsf{Enc}[\widetilde{E}_K]), \ell(\Pi.\mathsf{Dec}[\widetilde{E}_K]))$. Hence, this notion does not permit adversaries to obtain leak values of the first or second oracle. AEL-security is defined so that there is no such restriction.

Berti et al. [6] defined two notions for privacy and integrity. The notion for integrity, called CIML2, is the integrity part of the AE-security one with encryption and decryption leakages. The notion for privacy, called muCIML2, is different from the privacy part of the AE-security one. The adversary's goal of muCIML2 is to guess a bit b of a challenge ciphertext C_b while having access to leakage functions as well as the encryption and decryption oracles, where two plaintext M_1 and M_2 are chosen by an adversary, b is a random bit, and C_b is the encrypted value of M_b. Since \$ and $\perp$ leak no information of plaintexts, any scheme indistinguisbale from $(\$, \perp)$ is secure in the sense of the goal of muCIML2. Hence, the AEL-security notion covers the security goals of CIML2 and of muCIML2. Berti et al. designed an AEAD mode secure regarding CIML2 and muCIML2 in the multi-user setting and the misuse setting. On the other hand, our security proof of HOMA don't consider these settings. Note that the AEL-security notion can be extended to the one covering these settings by adding multiple users and permitting adversaries to make misuse queries.

4.2 AEL-Security of HOMA

The following theorem shows that $\mathsf{HOMA}[\widetilde{E}_K]$ is AEL-secure up to $O(2^{2n})$ decryption query complexity.

Theorem 1 (Security of HOMA). *There exists a simulator $\mathcal{S}$ such that for any adversary $\mathbf{A}$ running in time t, $\mathbf{Adv}^{\mathsf{ael}}_{\mathsf{HOMA}[\widetilde{E}_K],\mathcal{S}}(\mathbf{A}) \leq \mathbf{Adv}^{\mathsf{tprp}}_{\widetilde{E}}(\sigma, t+O(\sigma)) + \frac{19\sigma_{\mathcal{D}}}{2^{2n}}$, and $\mathcal{S}$ runs in time $t + O(\sigma)$ and requires an $O(\sigma)$-bit memory, where $\sigma_{\mathcal{D}}$ (resp. σ) is the number of TBC calls in all* HOMA.Dec *(resp.* HOMA*) procedures.*

Intuition of the Security of HOMA. Assume that the TBC is a TRP. Then, there are the following differences between the real and ideal worlds.

1. Enc.: (real) ciphertexts and tags are defined by a TRP; (ideal) those are defined by \$.
2. Dec.: (real) a plaintext might be returned; (ideal) all responses are **reject**.
3. Unprotected values: (real) the values are defined by HOMA; (ideal) the values are defined by a simulator.

For the difference (1), in the real world, since each tweak includes a nonce and a counter, each output of $\widetilde{P}$ in the encrypt is random. Thus, the difference yields no attack.

For the difference (2), we consider two-types of decryption query in the real world: In a decryption query, (2)-1: the nonce is not in the previous encryption queries; (2)-2: the nonce is in some previous encryption query. In the type (2)-1, the tag is chosen independently from all tags in encryption queries, and thus the probability of forging the tag is $O(1/2^{2n})$. In the case (2)-2, forging the tag implies that an internal state collision occurs between the encryption and decryption queries (the nonces are the same).[13] As mentioned in Section 3, a collision in previous decryption queries with the same nonce cannot be used without detecting the random IV in the encryption query. The probability of detecting the random IV is $O(1/2^{2n})$. Then, to obtain the internal state collision, some $2n$-bit internal state, which is freshly defined in the decryption query, must collide with some internal state in the encryption query. The collision probability is at most $O(\ell/2^{2n})$ for the data length ℓ. Summing the bound $O(\ell/2^{2n})$ for each decryption query, the probability of forging a tag in some decryption query, i.e. the distinguishing probability from the difference is at most $O(\sigma_{\mathcal{D}}/2^{2n})$.

For the difference (3), we define a simulator so that unprotected values include no information differentiating the real and ideal worlds. The detail is given in Section 4.3.

Hence, we obtain the AEL-Security bound $O(\sigma_{\mathcal{D}}/2^{2n})$.

4.3 Proof of Theorem 1

First, the TBC $\widetilde{E}_K$ is replaced with a TRP $\widetilde{P}$. Then, for any adversary $\mathbf{A}$, there exists an adversary $\mathbf{A}'$ such that $\mathbf{Adv}^{\mathsf{ael}}_{\mathsf{HOMA}[\widetilde{E}_K],\mathcal{S}}(\mathbf{A}) \leq \mathbf{Adv}^{\mathsf{tprp}}_{\widetilde{E}}(\sigma, t + O(\sigma)) + \mathbf{Adv}^{\mathsf{ael}}_{\mathsf{HOMA}[\widetilde{P}],\mathcal{S}}(\mathbf{A}')$. Hereafter, we bound $\mathbf{Adv}^{\mathsf{ael}}_{\mathsf{HOMA}[\widetilde{P}],\mathcal{S}}(\mathbf{A}')$, the AEL-security advantage of HOMA using $\widetilde{P}$.

Simulator $\mathcal{S}$. Our simulator is defined below. Both of $\mathcal{S}_{\mathsf{encL}}$ and $\mathcal{S}_{\mathsf{decL}}$ run the decryption procedure $\mathsf{HOMA.Dec}$ and return unprotected values defined in this procedure. The underlying TBC is instantiated with a TRP $\widetilde{P}' \in \widetilde{\mathsf{Perm}}(\mathcal{TW}, \{0,1\}^n)$, which the simulators realize by lazy sampling.[14]

[13] A TRP offers independent permutations if the tweaks are distinct. In HOMA, a nonce is a tweak element, thus HOMA procedures with distinct nonces are independently performed (even if the R values are the same). Thus, encryption queries whose nonces are different from the nonce of the decryption query do not affect the internal state collision probability.

[14] A TRP $\widetilde{P}$ keeps a table $\mathcal{L}$ that is initially empty. For an input $(X, Y) \in \{0,1\}^n \times \mathcal{TW}$ to $\widetilde{P}$, the output Z is defined as follows: if $\mathcal{L}(X, Y) = \varepsilon$ **then** $Z \xleftarrow{\$} \{0,1\}^n \backslash \mathcal{L}(*, Y)$ and $\mathcal{L}(X, Y) \leftarrow Z$, where $\mathcal{L}(*, Y)$ is the set of all outputs whose tweaks are Y, and otherwise $Z \leftarrow \mathcal{L}(X, Y)$.

- $\mathcal{S}_{\mathsf{encL}}(N, R, A, C, T)$: runs $\mathsf{HOMA.Dec}[\widetilde{P}'](N, R, A, C, T)$; returns the unprotected values defined in $\mathsf{HOMA.Dec}[\widetilde{P}'](N, R, A, C, T)$.
- $\mathcal{S}_{\mathsf{decL}}(N, R, A, C, \hat{T})$: runs $\mathsf{HOMA.Dec}[\widetilde{P}'](N, R, A, C, \hat{T})$; returns the unprotected values defined in $\mathsf{HOMA.Dec}[\widetilde{P}'](N, R, A, C, \hat{T})$.

$\mathcal{S}$ runs in time $t + O(\sigma)$ and requires an $O(\sigma)$-bit memory. Note again that the TRP $\widetilde{P}'$ is realized by the simulators as well as the decryption procedure $\mathsf{HOMA.Dec}[\widetilde{P}']$, which is given in Algorithm 1 where $\widetilde{E}_K$ is replaced with $\widetilde{P}'$.

Notations. Let $q_{\mathcal{E}}$ (resp. $q_{\mathcal{D}}$) be the number of encryption (resp. decryption) queries, and $q := q_{\mathcal{E}} + q_{\mathcal{D}}$. Let $\sigma_{\mathcal{D},A}$ (resp. $\sigma_{\mathcal{D},C}$) be the total number of TRP calls in $\mathsf{HOMA.Hash}$ (resp. $\mathsf{HOMA.Main}$) by decryption queries, thus $\sigma_{\mathcal{D}} = \sigma_{\mathcal{D},A} + \sigma_{\mathcal{D},C}$. For convenience, we express the α-th encryption (resp. β-th decryption) query as the α-th (resp. $(\beta+q_{\mathcal{E}})$-th) query. For $\alpha, \beta \in [q]$ such that the β-th query is made after the α-th query, the relation is denoted by $\alpha \lhd \beta$. Let $\ell := a + m$ denote the total length of data blocks by a query. For the j-th TRP call at the i-th DPF call in HOMA, the input block, the output block, and the tweak in $\mathsf{HOMA.Hash}$ (resp. $\mathsf{HOMA.Main}$) are denoted by $X_{i,j}$, $Z_{i,j}$, and $Y_{i,j}$, (resp., $X_{i,j-1}$, $Z_{i,j-1}$, and $Y_{i,j-1}$). See also Fig. 3. Let $XY_{i,j} := X_{i,j} \| Y_{i,j}$. Note that in the ideal world, these values are defined by $\mathcal{S}$. For $\alpha \in [q]$, a value V defined at the α-th query is denoted by $V^{(\alpha)}$. The lengths a, m and ℓ of the α-th query are denoted by a_α, m_α and ℓ_α. For $\alpha \in [q]$, let $\mathcal{C}_i^{(\alpha)} := (XY_{a_\alpha,1}^{(\alpha)}, C_1^{(\alpha)}, \ldots, C_i^{(\alpha)})$ be an array of an input to the second last TRP call in $\mathsf{HOMA.Hash}$ and the ciphertext blocks up to the i-th block defined at the α-th query, $\mathcal{C}_0^{(\alpha)} := (XY_{a_\alpha,1}^{(\alpha)})$.

Transcript. In the following proof, for each encryption query, if $|C| \bmod n \neq 0$, i.e., $|C_m| < n$, then a $(n-|C_m|)$-bit string C_L is appended to the ciphertext C and the modified ciphertext $\tilde{C} = C \| C_L$ is returned instead of C. In the real world, $C_L := \mathsf{lsb}_{n-|C_m|}(Z_{\ell,0})$ (thus, $Z_{\ell,0} = (M_m \| 0^{n-|C_m|}) \oplus (C_m \| C_L)$), and in the ideal world $C_L \xleftarrow{\$} \{0,1\}^{n-|C_m|}$. For $i \in [m-1]$, let $\tilde{C}_i := C_i$ and $\tilde{C}_m := C_m \| C_L$, thus $\tilde{C} = \tilde{C}_1 \| \cdots \| \tilde{C}_m$. Let $\tilde{M}_i := M_i$ and $\tilde{M}_m := M_m \| 0^{n-|M_m|}$.

The following proof, in addition to the standard outputs, permits $\mathbf{A}'$ to obtain the following protected values after making all queries but before returning a decision bit.

- $\mathcal{Z}_2 := \{Z_{i,2}^{(\alpha)} \mid \alpha \in [q], i \in [\ell_\alpha - 1]\}$.
- $\mathcal{Z}_{0,1} := \{Z_{a_\beta+i,0}^{(\beta)} \mid \beta \in [q_{\mathcal{E}}+1, q], i \in [m_\beta - 1]$ s.t. $\forall \alpha \in [q_{\mathcal{E}}]$ s.t. $\alpha \lhd \beta : N^{(\alpha)} \neq N^{(\beta)}\}$.
- $\mathcal{Z}_{0,2} := \{Z_{a_\beta+i,0}^{(\beta)} \mid \beta \in [q_{\mathcal{E}}+1, q], i \in [m_\beta - 1]$ s.t. $\exists \alpha \in [q_{\mathcal{E}}]$ s.t. $\alpha \lhd \beta \wedge N^{(\alpha)} = N^{(\beta)} \wedge \mathcal{C}_{i-1}^{(\alpha)} \neq \mathcal{C}_{i-1}^{(\beta)}\}$.
- $\mathcal{Z}_t := \{T_1^{(\beta)}, T_2^{(\beta)} \mid \beta \in [q_{\mathcal{E}}+1, q]\}$.

Note that the TBC outputs $Z_{a_\alpha+i,0}^{(\alpha)}, Z_{\ell_\alpha,1}^{(\alpha)}, Z_{\ell_\alpha,2}^{(\alpha)}$ for $\alpha \in [q_{\mathcal{E}}], i \in [m_\alpha]$ (defined by encryption queries) remain secret (in the ideal world). Then, a transcript τ that $\mathbf{A}'$ obtains in the game consists of

- $\big((N^{(\alpha)}, A^{(\alpha)}, M^{(\alpha)}), (R^{(\alpha)}, \tilde{C}^{(\alpha)}, T^{(\alpha)})\big)$ for $\alpha \in [q_{\mathcal{E}}]$,
- $\big((N^{(\beta)}, R^{(\beta)}, A^{(\beta)}, C^{(\beta)}, \hat{T}^{(\beta)}), RV^{(\beta)}\big)$ for $\beta \in [q_{\mathcal{E}}+1, q]$, where $RV^{(\beta)}$ is an output of the β-th query: plaintext $M^{(\beta)}$ or **reject**,
- $Z^{(\alpha)}_{i,1}$ for $\alpha \in [q]$ and $i \in [\ell_\alpha - 1]$,
- $\mathcal{Z}_2, \mathcal{Z}_{0,1}, \mathcal{Z}_{0,2}$, and $\mathcal{Z}_t$.

Bound of the Advantage. Let τ be a transcript that $\mathbf{A}'$ obtains by queries in the game. Let T_R be a transcript in the real world obtained by sampling $\widetilde{P}$ and R. Let T_I be a transcript in the ideal world obtained by sampling \$, $\widetilde{P}'$, and R. We call a transcript τ *valid* if $\Pr[\mathsf{T}_I = \tau] > 0$. Let $\mathcal{T}$ be all valid transcripts such that $\forall \tau \in \mathcal{T} : \Pr[\mathsf{T}_R = \tau] \leq \Pr[\mathsf{T}_I = \tau]$. Then, we have $\mathbf{Adv}^{\mathsf{ael}}_{\mathsf{HOMA}[\widetilde{P}],\mathcal{S}}(\mathbf{A}') = \mathsf{SD}(\mathsf{T}_R, \mathsf{T}_I) = \sum_{\tau \in \mathcal{T}} (\Pr[\mathsf{T}_I = \tau] - \Pr[\mathsf{T}_R = \tau])$. We bound the statistical distance $\mathsf{SD}(\mathsf{T}_R, \mathsf{T}_I)$ using the following collision event: coll_m:

- coll_m: $\exists \alpha \in [q_{\mathcal{E}}], \beta \in [q_{\mathcal{E}}+1, q], i \in [m_\alpha]$ s.t.
$$XY^{(\alpha)}_{a_\alpha+i-1,1} \neq XY^{(\beta)}_{a_\beta+i-1,1} \wedge XY^{(\alpha)}_{a_\alpha+i,1} = XY^{(\beta)}_{a_\beta+i,1}.$$

Let coll^r_m (resp. coll^i_m) be the real (resp. ideal) world event. Using the event, we have $\mathsf{SD}(\mathsf{T}_R, \mathsf{T}_I) \leq \Pr[\mathsf{coll}^r_m] + \Pr[\mathsf{coll}^i_m] + \mathsf{SD}(\mathsf{T}^*_R, \mathsf{T}^*_I)$, where T^*_R (resp. T^*_I) is the transcript T_R (resp. T_I) conditioned on $\neg\mathsf{coll}^r_m$ (resp. $\neg\mathsf{coll}^i_m$). The bounds of $\Pr[\mathsf{coll}^r_m]$, $\Pr[\mathsf{coll}^i_m]$, and $\mathsf{SD}(\mathsf{T}^*_R, \mathsf{T}^*_I)$ are given in the following analyses, ensuring $\mathbf{Adv}^{\mathsf{ael}}_{\mathsf{HOMA}[\widetilde{P}],\mathcal{S}}(\mathbf{A}') \leq \frac{8\sigma_{\mathcal{D},C}}{2^{2n}} + \frac{11\sigma_{\mathcal{D}}}{2^{2n}} \leq \frac{19\sigma_{\mathcal{D}}}{2^{2n}}$.

Bounds of $\Pr[\mathsf{coll}^r_m]$, $\Pr[\mathsf{coll}^i_m]$. The following analysis holds for both worlds. Fix $\alpha \in [q_{\mathcal{E}}], \beta \in [q_{\mathcal{D}}+1, q]$, and $i \in [m_\alpha]$ such that $XY^{(\alpha)}_{a_\alpha+i-1,1} \neq XY^{(\beta)}_{a_\beta+i-1,1}$ and $N^{(\alpha)} = N^{(\beta)}$. For $\gamma \in \{\alpha, \beta\}$, $X^{(\gamma)}_{a_\gamma+i,1} = \mathsf{fix0}(Z^{(\gamma)}_{a_\gamma+i,0})$ is satisfied, and $Z^{(\alpha)}_{a_\alpha+i,0}$ and $Z^{(\beta)}_{a_\beta+i,0}$ are sampled separately and uniformly at random from at least $2^n - 1$ elements. We thus have $\Pr[X^{(\alpha)}_{a_\alpha+i,1} = X^{(\beta)}_{a_\beta+i,1}] \leq 2/2^n$. $Z^{(\alpha)}_{a_\alpha+i-1,1}$ and $Z^{(\beta)}_{a_\beta+i-1,1}$, which are used to define $Y^{(\alpha)}_{a_\alpha+i,1}$ and $Y^{(\beta)}_{a_\beta+i,1}$, respectively, are sampled separately and uniformly at random from at least 2^{n-1} elements due to $\mathsf{fix0}$. We thus have $\Pr[Y^{(\alpha)}_{a_\alpha+i,1} = Y^{(\beta)}_{a_\beta+i,1}] \leq 2/2^n$. Summing the bound $4/2^{2n}$ for each β, i, we have $\Pr[\mathsf{coll}^r_m] \leq 4\sigma_{\mathcal{D},C}/2^{2n}$ and $\Pr[\mathsf{coll}^i_m] \leq 4\sigma_{\mathcal{D},C}/2^{2n}$.

Bound of $\mathsf{SD}(\mathsf{T}^*_R, \mathsf{T}^*_I)$. We bound $\mathsf{SD}(\mathsf{T}^*_R, \mathsf{T}^*_I)$ by using the coefficent H technique [35]. Here, $\mathcal{T}$ is partitioned into two transcripts: good transcripts $\mathcal{T}_{\mathsf{good}}$ and bad transcripts $\mathcal{T}_{\mathsf{bad}}$.

Lemma 1 (Coefficent H technique [35]). *If* $\forall \tau \in \mathcal{T}_{\mathsf{good}} : \frac{\Pr[\mathsf{T}^*_R = \tau]}{\Pr[\mathsf{T}^*_I = \tau]} \geq 1 - \mu$ *s.t.* $0 \leq \mu \leq 1$, *then* $\mathsf{SD}(\mathsf{T}^*_R, \mathsf{T}^*_I) \leq \Pr[\mathsf{T}^*_I \in \mathcal{T}_{\mathsf{bad}}] + \mu$.

In the following proof, good and bad transcripts are defined. Then $\Pr[\mathsf{T}^*_I \in \mathcal{T}_{\mathsf{bad}}]$ is upper-bounded, and $\frac{\Pr[\mathsf{T}^*_R = \tau]}{\Pr[\mathsf{T}^*_I = \tau]}$ is lower-bounded. Finally, an upper-bound of $\mathsf{SD}(\mathsf{T}^*_R, \mathsf{T}^*_I)$ is obtained, putting the bounds into the above lemma.

Good and Bad Transcripts. We define bad events below.

- forge: $\exists \alpha \in [q_{\mathcal{D}}+1, q]$ s.t. $T^{(\alpha)} = \hat{T}^{(\alpha)}$.
- coll_{iv}: $\exists \alpha \in [q_{\mathcal{E}}], \beta \in [q_{\mathcal{E}}+1, q]$ s.t. $\beta \lhd \alpha \wedge (N^{(\alpha)}, R^{(\alpha)}) = (N^{(\beta)}, R^{(\beta)})$.
- coll_{h}: $\exists \alpha \in [q_{\mathcal{E}}], \beta \in [q_{\mathcal{E}}+1, q]$ s.t.
$$(R^{(\alpha)}, A^{(\alpha)}) \neq (R^{(\beta)}, A^{(\beta)}) \wedge XY^{(\alpha)}_{a_\alpha,1} = XY^{(\beta)}_{a_\beta,1}.$$
- coll_{c}: $\exists \alpha \in [q_{\mathcal{E}}], \beta \in [q_{\mathcal{E}}+1, q], i \in [m_\alpha]$ s.t.
$$\mathcal{C}^{(\alpha)}_{i-1} \neq \mathcal{C}^{(\beta)}_{i-1} \wedge (\mathsf{fix0}(\tilde{M}^{(\alpha)}_i) \oplus \tilde{C}^{(\alpha)}_i), Y^{(\alpha)}_{a_\alpha+i,1}) = (X^{(\beta)}_{a_\beta+i,1}, Y^{(\beta)}_{a_\beta+i,1}).$$

Note that if $i > m_\beta$, then $X^{(\beta)}_{a_\beta+i,1} := \varepsilon$ and $Y^{(\beta)}_{a_\beta+i,1} := \varepsilon$.

We define bad transcripts $\mathcal{T}_{\mathsf{bad}}$ that satisfy one of the bad events. Good transcripts are defined as $\mathcal{T}_{\mathsf{good}} := \mathcal{T} \backslash \mathcal{T}_{\mathsf{bad}}$.

Lower-Bound of $\mathbf{Pr}[\mathsf{T}^*_{\boldsymbol{R}} = \tau] / \mathbf{Pr}[\mathsf{T}^*_{\boldsymbol{I}} = \tau]$. We give an overview of this evaluation. The detail is given in the full version of this paper [28].

There are the following differences between the real and ideal worlds.

1. Dec.: (real) a plaintext might be returned; (ideal) all responses are **reject**.
2. Enc.: (real) ciphertexts and tags are defined by a TRP; (ideal) those are defined by \$.
3. Protected and unprotected values: (real) the values are defined by HOMA; (ideal) the values are defined by the simulator.

We thus show that as long as no bad event occurs, the differences yield no distinguishing attack.

For the difference (1), by $\neg\mathsf{forge}$, the difference yields no attack.

For the difference (2), in the real world, since each tweak includes a nonce and a counter, each output of $\widetilde{P}$, which is used to encrypt a plaintext, is random. Thus, the difference yields no attack.

For the difference (3), in the real world, protected values and unprotected values are defined by a TRP as well as ciphertext blocks, whereas in the ideal world, these values are defined by a TRP but independently of ciphertext blocks that are defined by \$. The detail of the difference is shown below, where $\alpha \in [q_{\mathcal{E}}], \beta \in [q_{\mathcal{E}}+1, q]$ and $i \in [m_\alpha]$ such that $N^{(\alpha)} = N^{(\beta)}$ and $(R^{(\alpha)}, A^{(\alpha)}, C^{(\alpha)}_1, \ldots, C^{(\alpha)}_{i-1}) \neq (R^{(\beta)}, A^{(\beta)}, C^{(\beta)}_1, \ldots, C^{(\beta)}_{i-1})$.

- **Real**: If $(\mathsf{fix0}(\tilde{M}^{(\alpha)}_i \oplus \tilde{C}^{(\alpha)}_i), Y^{(\alpha)}_{i,1}) = (X^{(\beta)}_{i,1}, Y^{(\beta)}_{i,1})$ then $Z^{(\alpha)}_{i,1} = Z^{(\beta)}_{i,1}$, since $X^{(\alpha)}_{i,1} = \mathsf{fix0}(Z^{(\alpha)}_{i,0}) \wedge Z^{(\alpha)}_{i,0} = \tilde{M}^{(\alpha)}_i \oplus \tilde{C}^{(\alpha)}_i$.
- **Ideal**: It occurs that $(\mathsf{fix0}(\tilde{M}^{(\alpha)}_i \oplus \tilde{C}^{(\alpha)}_i), Y^{(\alpha)}_{i,1}) = (X^{(\beta)}_{i,1}, Y^{(\beta)}_{i,1}) \wedge Z^{(\alpha)}_{i,1} \neq Z^{(\beta)}_{i,1}$, since $X^{(\alpha)}_{i,1} = \mathsf{fix0}(Z^{(\alpha)}_{i,0})$ but $\tilde{C}^{(\alpha)}_i$ is defined independently of $Z^{(\alpha)}_{i,0}$.

In both worlds, by $\neg\mathsf{coll}_h \wedge \neg\mathsf{coll}_{iv}$, $\mathcal{C}^{(\alpha)}_{i-1} \neq \mathcal{C}^{(\beta)}_{i-1}$ is satisfied. Then, in the real world, by $\neg\mathsf{coll}_m$, $(\mathsf{fix0}(\tilde{M}^{(\alpha)}_i \oplus \tilde{C}^{(\alpha)}_i), Y^{(\alpha)}_{i,1}) \neq (X^{(\beta)}_{i,1}, Y^{(\beta)}_{i,1})$ is satisfied, thus the real-word event does not occur. By $\neg\mathsf{coll}_c$, the ideal-world event does not occurs. Hence, no attack using the difference (3) exists.

Hence, the real and ideal worlds are indistinguishable, that is, $\forall \tau \in \mathcal{T}_{\mathsf{good}}$: $\Pr[\mathsf{T}^*_R = \tau] / \Pr[\mathsf{T}^*_I = \tau] \geq 1$.

Upper-Bound of $\Pr[\mathsf{T}_I \in \mathcal{T}_{\mathsf{bad}}]$. $\Pr[\mathsf{T}_I \in \mathcal{T}_{\mathsf{bad}}]$ is bounded by $\Pr[\mathsf{forge}] + \Pr[\mathsf{coll}_{iv}] + \Pr[\mathsf{coll}_h] + \Pr[\mathsf{coll}_c] \leq \frac{q_{\mathcal{D}}}{2^{2n}} + \frac{2q_{\mathcal{D}}}{2^{2n}} + \frac{2\sigma_{\mathcal{D},A}}{2^{2n}} + \frac{8\sigma_{\mathcal{D},C}}{2^{2n}} \leq \frac{11\sigma_{\mathcal{D}}}{2^{2n}}$, where for each event ev of the four events, $\Pr[\mathsf{ev}]$ is the probability that ev occurs as long as other events have not occurred. The bounds are given in the following analyses.
$\underline{\Pr[\mathsf{forge}]}$. For each $\alpha \in [q_{\mathcal{E}}+1, q]$, each of $T_1^{(\alpha)}$ and $T_2^{(\alpha)}$ is chosen uniformly at random from $\{0,1\}^n$, thus $\Pr[\mathsf{forge}] \leq q_{\mathcal{D}}/2^{2n}$.
$\underline{\Pr[\mathsf{coll}_{iv}]}$. For each $\alpha \in [q_{\mathcal{E}}], \beta \in [q_{\mathcal{E}}+1, q]$ such that $\beta \lhd \alpha$ and $N^{(\alpha)} = N^{(\alpha)}$, $R^{(\alpha)}$ is chosen uniformly at random from $\{0,1\}^{2n-1}$, thus $\Pr[\mathsf{coll}_{iv}] \leq 2q_{\mathcal{D}}/2^{2n}$.
$\underline{\Pr[\mathsf{coll}_h]}$. We first fix $\alpha \in [q_{\mathcal{E}}], \beta \in [q_{\mathcal{D}}+1, q]$ such that $N^{(\alpha)} = N^{(\beta)} \wedge (R^{(\alpha)}, A^{(\alpha)}) \neq (R^{(\beta)}, A^{(\beta)})$, and consider an event $\mathsf{coll}_h[\alpha, \beta]$: coll_h occurs due to the α-th and β-th queries. By $(R^{(\alpha)}, A^{(\alpha)}) \neq (R^{(\beta)}, A^{(\beta)})$, $\mathsf{coll}_h[\alpha, \beta]$ implies that an internal-state collision occurs in $\mathsf{HOMA.Hash}[\widetilde{P}]$: $\exists i \in [a_\beta]$ s.t. $XY_{i-1,1}^{(\alpha)} \neq XY_{i-1,1}^{(\beta)} \wedge XY_{i,1}^{(\alpha)} = XY_{i,1}^{(\beta)}$. If $XY_{i-1,1}^{(\alpha)} \neq XY_{i-1,1}^{(\beta)}$, then the outputs $Z_{i-1,1}^{(\alpha)}$ and $Z_{i-1,1}^{(\beta)}$ are sampled separately, and the next outputs $Z_{i-1,2}^{(\alpha)}$ and $Z_{i-1,2}^{(\beta)}$ are sampled separately. We thus have $\Pr[XY_{i,1}^{(\alpha)} = XY_{i,1}^{(\beta)}] \leq (2/2^n) \cdot (1/2^n) = 2/2^{2n}$.

Using the bound $2/2^{2n}$, we have $\Pr[\mathsf{coll}_h] \leq \sum_{\beta=1}^{q_{\mathcal{D}}} 2a_\beta/2^{2n} \leq 2\sigma_{\mathcal{D},A}/2^{2n}$.
$\underline{\Pr[\mathsf{coll}_c]}$. Fix $\alpha \in [q_{\mathcal{E}}], \beta \in [q_{\mathcal{D}}+1, q], i \in [m_\alpha]$ s.t. $N^{(\alpha)} = N^{(\beta)} \wedge \mathcal{C}_{i-1}^{(\alpha)} \neq \mathcal{C}_{i-1}^{(\beta)}$. For the condition $Y_{a_\alpha+i,1}^{(\alpha)} = Y_{a_\beta+i,1}^{(\beta)}$, by $\neg\mathsf{coll}_m$, $XY_{a_\alpha+i-1,1}^{(\alpha)} \neq XY_{a_\beta+i-1,1}^{(\beta)}$ is satisfied, thus the outputs $Z_{a_\alpha+i-1,1}^{(\alpha)}$ and $Z_{a_\beta+i-1,1}^{(\beta)}$ are separately sampled from at least 2^{n-1} elements due to $\mathsf{fix0}$. Thus, we have $\Pr[Y_{a_\alpha+i,1}^{(\alpha)} = Y_{a_\beta+i,1}^{(\beta)}] \leq 2/2^n$. For the condition $\mathsf{fix0}(\tilde{M}_i^{(\alpha)} \oplus \tilde{C}_i^{(\alpha)}) = X_{a_\beta+i,1}^{(\beta)}$, since $\tilde{C}_i^{(\alpha)}$ is chosen from $\{0,1\}^n$, $Z_{a_\beta+i,0}^{(\beta)}$ is chosen from at least $2^n - 1$ elements, and $X_{a_\beta+i,1}^{(\beta)} = \mathsf{fix0}(Z_{a_\beta+i,0}^{(\beta)})$ is satisfied, we have $\Pr[\mathsf{fix0}(\tilde{M}_i^{(\alpha)} \oplus \tilde{C}_i^{(\alpha)}) = X_{a_\beta+i,1}^{(\beta)}] \leq 2/(2^n - 1) \leq 4/2^n$.

Summing the bound $(2/2^n) \cdot (4/2^n)$ for each β, i, we have $\Pr[\mathsf{coll}_c] \leq 8\sigma_{\mathcal{D},C}/2^{2n}$.

5 A TBC Optimized for HOMA

HOMA requires a TBC that accepts a $0.5s$-bit plaintext, an s-bit key, and a $2s+3$-bit tweak, where $s = 128$ for 128-bit security. We design a new TBC, SKINNYee, which is optimized to be used in HOMA by basing the scheme on SKINNY64 [3]. We conjecture that SKINNYee is a TPRP and satisfies the requirement of HOMA.

5.1 SKINNY64 and SKINNYe with TK4

SKINNY64 is a TBC that supports a block size of 64 bits. SKINNY64 adopts the tweakey framework [22], which enables the designers to avoid making a distinction between a tweak and a key, and those two are treated as a single object "tweakey." The design is called TKn when the tweakey size is n times as big as the block size. SKINNY64 supports the tweakey size of 64 bits (TK1), 128 bits (TK2), and 192 bits (TK3). Later, Naito et al. [26] proposed SKINNYe

(version 2) to extend the tweakey size of SKINNY64 to 256 bits (TK4). Here we describe the specifications of SKINNYe, which is a base of our work.

SKINNYe operates on the data structure (state) of 16 sequences of 4-bit data (nibble) $d_0, \ldots, d_{15}$ that are formatted into a 4×4 two-dimensional array; The first row is $d_0 \ldots, d_3$, the second row is $d_4 \ldots, d_7$, and so on. A 64-bit plaintext is divided into 16 nibbles, and those form a data state. A 256-bit tweakey forms 4 tweakey states. Then, the following round transformation is iterated 44 times.

`SubCells(SC).` A 4-bit S-box is applied to each nibble.

`AddConstants(AC).` A 7-bit constant specified for each round is XORed to particular 7 bits of the state.

`AddRoundTweakey(ART).` A 32-bit value called sub-tweakey is generated from the 256-bit tweakey state, and those are XORed to the top two rows of the data state. Then 3 tweakey states are updated as explained later.

`ShiftRows(SR).` The position of each nibble in row $i, i \in \{0, 1, 2, 3\}$ is cyclically shifted to right by i positions.

`MixColumns(MC).` Let (x, y, z, w) be 4 nibbles in a column. The value is updated to $(x \oplus z \oplus w, x, y \oplus z, x \oplus z)$. This transformation is applied to each column.

Regarding `AC`, a 6-bit affine LFSR denoted by $(rc_5, rc_4, rc_3, rc_2, rc_1, rc_0)$ is used to generate round constants. In each round, this LFSR is updated by $(rc_5 \| rc_4 \| \cdots \| rc_0) \rightarrow (rc_4 \| rc_3 \| rc_2 \| rc_1 \| rc_0 \| rc_5 \oplus rc_4 \oplus 1)$. Then, 3 nibble values $rc_3 \| rc_2 \| rc_1 \| rc_0$, $0 \| 0 \| rc_5 \| rc_4$, and `0x2` are XORed to the first, the second, and the third rows of the left-most column of the state, respectively.

Regarding ART, first, the 32-bit sub-tweakey value is computed by extracting the top 2 rows from each of 4 tweakey states and XORing them. Second, nibble positions are permuted by the permutation P_T: $(0, \ldots, 15) \rightarrow (9, 15, 8, 13, 10, 14, 12, 11, 0, 1, 2, 3, 4, 5, 6, 7)$. All tweakey states are updated with the same P_T. Third, all nibbles in the second, the third, and the fourth tweakey states are updated by applying the following $LFSR_2$, $LFSR_3$, and $LFSR_4$, respectively.

$$
\begin{aligned}
LFSR_2 &: (x_3 \| x_2 \| x_1 \| x_0) \rightarrow (x_2 \| x_1 \| x_0 \| x_3 \oplus x_2), \\
LFSR_3 &: (x_3 \| x_2 \| x_1 \| x_0) \rightarrow (x_0 \oplus x_3 \| x_3 \| x_2 \| x_1), \\
LFSR_4 &: (x_3 \| x_2 \| x_1 \| x_0) \rightarrow (x_1 \| x_0 \| x_3 \oplus x_2 \| x_2 \oplus x_1).
\end{aligned}
$$

5.2 Elastic-Tweak Framework for Small Tweaks

Elastic-tweak is a design to convert BCs or TBCs to accept a few (more) bits of tweak [11]. The input tweak is first expanded to a relatively large size for security reasons and then XORed to the data state in every few rounds. The framework was later improved to be more lightweight by realizing the expanded tweak state with LFSR [27], but it still preserves the principle of expanding the tweak, which is disadvantageous for small implementations.

5.3 Design Approach of SKINNYee

We first give an overview of our approach to design our TBC. Recall that HOMA requires a 64-bit block TBC that supports a 128-bit key and a 259-bit tweak. By adopting the same approach as SKINNYe, such TBCs are realized if the tweakey size of SKINNY64 can be extended to 448 bits (TK7). However, we found that this approach is not reasonable for two reasons.

- The idea behind the tweakey of SKINNY is to not make any distinction between a key and a tweak. For example, a 192-bit tweakey can be an x-bit key and a $(192-x)$-bit tweak for some x, $1 \leq x \leq 192$. This functionality is not necessary for HOMA because the key size and the tweak size are fixed.
- We actually investigated the possibility of designing TK7 by searching for $LFSR_5$, $LFSR_6$, and $LFSR_7$ for the extra tweakey states. Because the search space is limited, all 4-bit LFSRs can be tested exhaustively. Our experiments showed that no LFSR exists to ensure security for TK7. TK7 can still be achieved by replacing LFSRs with more complex computations, but this requires to compromise implementation efficiency.

Our aim is not a general-purpose TBC. From the above considerations, we determined to treat a key and a tweak as independent objects instead of a tweakey.

Among 259 bits of the tweak, 3 bits are for the domain separation. The elastic-tweak gives us a hint that those can be processed efficiently by introducing different computations from the other tweak value. However, we found that the elastic-tweak is not suitable for HOMA because an additional computation to process a small tweak increases the memory size. Instead, we enlarge the size of an LFSR to compute the round constant by a few bits and initialize the LFSR to be different values depending on the 3-bit tweak.

Lastly, we design SKINNYee by reusing as many components of SKINNY as possible for two reasons. First, the benchmark becomes fair when we later compare the benchmark of our scheme with other SKINNY-based schemes. Second, SKINNY has received a lot of third-party security analysis, and the fact that SKINNY still stands against any cryptanalytic attempts enhances the reliability of the design. To take over those cryptanalytic attempts, the amount of modification from SKINNY should be minimized. In the end, we decided not to modify `SC`, `SR`, and `MC` from the original. So, modifications from SKINNY are made on `AC`, `ART`, and a new operation to process a 128-bit key.

5.4 Specifications of SKINNYee

SKINNYee accepts a 128-bit key, a 256-bit tweak, and a 3-bit tweak for the domain separation. The design is based on SKINNYe (TK4). The round transformation of SKINNYee is given in Fig. 5. Modifications we made are listed below.

- The 256-bit tweak is assigned to the 256-bit tweakey of SKINNYe.
- A new operation `AddRoundKey` is added between `SB` and `SR`. The 128-bit key is divided into four 32-bit data K_0, K_1, K_2, K_3. In round i, a 32-bit subkey is $K_{i \bmod 4}$. The subkey is XORed to the bottom two rows of the data state.

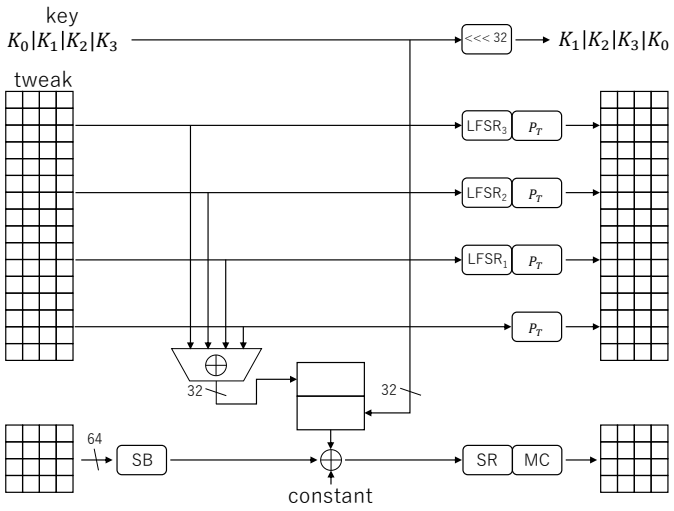

Fig. 5. Round Transformation of SKINNYee.

- `AC` is drastically modified. We define a 10-bit LFSR $rc_9, \ldots, rc_0$, which clocks $(rc_9\| \cdots \|rc_0)$ to $(rc_8\|rc_7\|rc_6\|rc_5\|rc_4\|rc_3\|rc_2\|rc_1\|rc_0\|rc_9 \oplus rc_3 \oplus rc_2 \oplus rc_0)$. At the beginning, $rc_9\|rc_8\|rc_7$ is initialized to the 3-bit tweak for the domain separation, and the other 7 bits are initialized to $rc_8 = \ldots = rc_1 = 0$ and $rc_0 = 1$. In each round, for $i = 0, 1, \ldots, 15$, we first XOR the 4-bit value $(rc_3\|rc_2\|rc_1\|rc_0)$ to the i-th nibble of the data state and then clock the LFSR.
- The number of rounds increases to 56.

5.5 Design Rationale

Rationale for the `AddRoundKey` is as follows. First, the tweak value is not mixed with the secret value derived by the key, which enables us not need to protect tweak states, otherwise the mixed state needs to be duplicated into several shares. Second, if both the subtweak and the subkey are XORed in the top two rows, some unknown interaction between the tweak and the key may occur. Specifically, when all nibbles in the first tweak state (never updated with LFSR) and all nibbles of the key have the same value, the XOR of the subtweak and the subkey can be a constant value. To avoid such cases, we decided to XOR subkeys to the bottom two rows. Note that the TPRP security required by the mode is a security notion for a single key, thus we exclude the use case that the adversary injects some difference in the key. Hence, we do not have to worry about related-key attacks. Moreover, the tweak value is computed by the HOMA mode, and the adversary cannot control it to be suitable for the attack. For the key schedule, we chose to use 4 parts of 32 bits of the 128-bit key in turn. This avoids using extra memory for the key schedule, thus it is very suitable for our

goal. Also note that the key schedule forms a cycle in every 4 rounds, and the key state is back to the original after the whole encryption process (56 rounds). This saves us the cost to implement the key schedule inverse.

We drastically modified AC. The first modification is the small-tweak dependent initialization of the LFSR. A single-bit difference in the initial value of the LFSR significantly changes the generated constant sequences, which is sufficient to separate the TBC invocations for different small-tweak values. Besides, we XOR the 4-bit constant to all nibbles by repeating exactly the same procedure 16 times in each round. This modification increases the total computational cost, thus may speed-down the round-based implementation, which was the original goal of SKINNY. Meanwhile, our goal is a small memory, thus iterating the same procedure 16 times is more suitable. The size of the LFSR was determined from the number of clocks for the whole encryption procedure. Our constant generation requires 16 clocks per round, thus it requires $16 \times 56 = 896$ clocks. We chose the LFSR size to be 10 bits to avoid having the same LFSR state. The feedback function of the 10-bit LFSR was chosen so that the cycle period is 1,023.

The number of rounds increased from that of SKINNY64 with TK3 (40 rounds) and SKINNYe with TK4 (44 rounds). This is because, in SKINNYee, each key nibble is XORed to the data state only in every 4 rounds, while in the previous designs, each key nibble is XORed in every 2 rounds. This does not immediately imply that the number of rounds of SKINNYee must be doubled. Many cryptanalyses, e.g. differential cryptanalysis, are divided into a 'distinguisher' and a 'key-recovery part.' The distinguisher is usually irrelevant to the key schedule, and the less-frequent use of each key nibble only affects the key-recovery part. We expect that the number of key-recovery rounds should be doubled in the worst-case scenario for SKINNY64 and SKINNYe. The maximum number of key-recovery rounds in literature was 11 [40],[15] thus we increased the number of rounds of SKINNYee by 12 from SKINNYe.

5.6 Security Analysis against Various Cryptanalyses

The security goal of SKINNYee is the TPRP security, which is a notion for a single-key. Hence, we focus on the evaluation in the single-key setting. When an adversary can inject any difference in the plaintext and the tweak, the number of active S-boxes for SKINNYee (in the single-key) is the same as one for SKINNYe in the related-tweakey (TK4) setting. The minimum number of active S-boxes can be evaluated by using mixed integer linear programming (MILP). The results are shown in Table 2, which show that 29 rounds ensure at least 64 S-boxes [26], and the maximum differential characteristic probability is upper-bounded by $2^{-2\times 64} = 2^{-128}$. Hence 56 rounds of SKINNYee is sufficiently secure.

Another popular approach is linear cryptanalysis. It has some advantage with respect to working in the known-plaintext setting, which allows an attacker to ignore the effects of random IV implemented in HOMA. The evaluation with

[15] The longest attack in literature with respect to the number of distinguisher rounds plus key-recovery rounds reaches $22 + 8 = 30$ rounds with TK3 [19].

Table 2. Tight bounds of the number of active Sboxes of SKINNYee.

	1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**
Diff	0	0	0	0	0	0	0	0	1	2	3	6	9	12	16
Lin	1	2	5	8	13	19	25	32	38	43	48	52	55	58	64
	16	**17**	**18**	**19**	**20**	**21**	**22**	**23**	**24**	**25**	**26**	**27**	**28**	**29**	**30**
Diff	19	21	24	30	35	39	41	43	46	50	54	58	62	66	72
Lin	70	76	80	85	90	96	102	107	110	115	121	127	130	135	141

MILP ensures at least 64 linearly active S-boxes only after 15 rounds [26]. Hence we conclude that HOMA is secure against linear cryptanalysis.

There are several cryptanalytic approaches that focus on features defined over 4 plaintext-ciphertext pairs. Boomerang-type attacks and differential-linear attacks are such examples. Roughly speaking, boomerang-type attacks combine 2 independent relatively short differential characteristics instead of a single long differentials characteristic, meanwhile the probability of each active S-box is squared. Table 2 shows that two 15-round characteristic with 16 active S-boxes may be able to be combined to construct 30-round distinguisher with probability $(2^{(-2)\times 16})^2 \times (2^{(-2)\times 16})^2 = 2^{-128}$. Dependency between two characteristics may increase or decrease the number of rounds a bit, but we conclude that 56 rounds of SKINNYee is sufficiently secure. In differential-linear attacks two differential characteristics and one linear characteristic is combined. For example, two 15-round differential characteristic with 16 active S-boxes may be able to be combined with a 8-round linear characteristic with 32 S-boxes. Again, dependency between two characteristics may increase or decrease the number of rounds a bit, but we conclude that 56 rounds of SKINNYee is sufficiently secure.

Meet-in-the-middle attacks divide the computation structure to two independently computed sub-parts. The designers of SKINNY [3] evaluated the maximum number of attacked rounds based on the number of rounds required for the full diffusion, which showed that the meet-in-the-middle attack would not reach 23 rounds. The use of large tweak in SKINNYee may extend the number of rounds for the full diffusion by 3, which may increase the number of rounds of independently computed parts and two techniques (partial-matching and initial structure) by 3. Hence, the number of attacked rounds is at most $23 + 5 \times 3 = 38$ even with an optimistic evaluation for the attacker.

Some attacks, such as invariant subspace and non-linear invariant, work regardless of the number of rounds (often with a weak key restriction), but no such attacks have been reported for SKINNY or its variants.

6 Implementation

6.1 Targets and Design Policy

We evaluate the hardware performance of HOMA instantiated with SKINNYee. Hereafter, we refer to the SKINNYee's 256-bit tweak as $\mathsf{TK}_1||\mathsf{TK}_2||\mathsf{TK}_3||\mathsf{TK}_4$ wherein each TK_i is a 64-bit chunk scheduled independently. We use them for the following purposes:

- TK_1: Upper 64 bits of the nonce,
- TK_2: Upper 36 bits: a lower part of the nonce, lower 28 bits: a counter,
- TK_3: Unprotected data,
- TK_4: Either an associated data block A_i or a ciphertext block C_i (see Fig. 1).

For a fair comparison, we also implement the current state-of-the-art PFB_Plus instantiated with SKINNYe [26] (see Table 1) with the same design policy. The circuit components needed for SKINNYe and SKINNYee are mostly common, which help us to evaluate the difference from the modes of operation. We respect PFB_Plus's original tweak configuration: $\mathsf{TK}_1||\mathsf{TK}_2$ stores the secret key, while $\mathsf{TK}_3||\mathsf{TK}_4$ stores the nonce and counter concatenated.[16]

We follow the design policy of the conventional PFB_Plus implementation [26], which works as a coprocessor that provides a set of commands for block-wise processing. We can realize all AEAD operations by combining those commands. The implementation keeps the key, nonce, and a counter during their lifetime to avoid the hidden cost of an external storage.

6.2 Masked S-box Implementation

We choose Cassiers et al.'s HPC2 [9,10] as a target masking scheme for its glitch resistance, composability, and the availability of an open-source implementation [8]. In particular, composability ensures the security of a circuit composed of the gadgets, which greatly simplifies the security analysis of the entire implementation [9]. Although HPC2 is a great option, we stress that HOMA's low-memory advantage (see Table 1) is independent of a particular masking scheme. An efficient masking scheme in the future will make the HOMA's advantage even higher because an efficient masking makes memory elements even more dominant in hardware cost.

Figure 6-(left) shows our 3-stage pipelined implementation of the SKINNY 4-bit S-box using the HPC2 AND gadgets. The gadget has built-in registers, and its two input ports have different latency. We arrange the gadgets in the pipeline in a way that minimizes the number of pipeline stages on the basis of Cassiers et al.'s S-box representation optimized for HPC2 [10]. The circuit uses four HPC2 AND gadgets, and each pipeline stage calculates (a part of) the S-box independently. Each AND gadget uses $(7d^2 + 11d + 4)/2$ bits of internal

[16] For both implementations, we use 28 bits as a counter and the remaining bits as a nonce, by following the conventional PFB_Plus implementation [26].

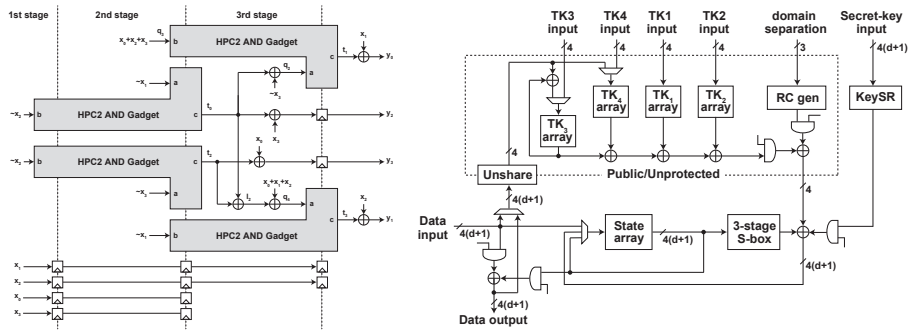

Fig. 6. (Left) three-stage pipelined implementation of the SKINNY 4-bit S-box. The shaded boxes are the HPC2 AND gadgets. We follow the original expression for the symbol names [10]. (Right) hardware architecture of HOMA.

registers. We also need $10d$ bits of the pipeline registers, as shown in the bottom of Fig. 6-(left), for carrying the inputs to later stages. As a result, the S-box circuit uses $(14d^2 + 18d + 8)$ bits of registers in total. Each HPC2 AND gadget uses $d(d+1)/2$ bits of a random number, and the S-box circuit consumes $2d(d+1)$ random bits/cycle at maximum. The total number of random bits for running a TBC is $2d(d+1) \times 16 \times N_{round}$ wherein N_{round} is the round number.

6.3 Hardware Design

Architecture. Figure 6-(right) shows the proposed nibble-serial hardware architecture, which uses the 2-dimensional arrays of registers as a basic building block, by following the conventional PFB_Plus and SKINNY implementations [3, 26].

The state array is a 64-bit register arranged in a 4×4 matrix, which efficiently realizes the nibble-wise data scan, as well as the `MixColumns` and `ShiftRows` operations. We use a scan flip-flop, a special register with a built-in 2-way selector, for efficiently implementing the array. Each round function takes 24 cycles, and the entire SKINNYee operation finishes in 1344 (=24 $\times$ 56) cycles.[17] The TK_1–TK_4 arrays are the similar 4×4 matrices that efficiently realize the nibble-wise data scan and the tweakey schedule [26]. We implement the newly-introduced 128-bit key $K_0||K_1||K_2||K_3$ using a simple (4×32)-bit shift register shown as KeySR in Fig. 6-(right).

HOMA needs to update TK_3 and TK_4 using the TBC output namely Y_{TBC}, such as $\mathsf{TK}_3 \leftarrow \mathsf{TK}_3 \oplus Y_{TBC}$ and $\mathsf{TK}_4 \leftarrow M_i \oplus Y_{TBC}$, in addition to SKINNYee encryption. Our architecture implements those operations in a nibble-oriented manner. The TK_2 array also integrates a 28-bit adder for updating the counter in place, meanwhile the state array integrates the fix0 operation.

[17] 19 cycles for S-box calculation with pipeline latency, 4 cycles for `MixColumns`, and 1 cycle for `ShiftRows`

Table 3. Hardware performances in gate equivalent (GE) for $d \in \{0, \cdots, 5\}$

Component	HOMA						PFB_Plus					
	$d=0$	$d=1$	$d=2$	$d=3$	$d=4$	$d=5$	$d=0$	$d=1$	$d=2$	$d=3$	$d=4$	$d=5$
Total	4,981	6,283	8,226	10,392	12,782	15,487	4,569	6,884	9,667	12,675	15,941	19,724
S-box	161	501	1,087	1,897	2,931	4,189	161	501	1,087	1,897	2,931	4,189
State array	542	1,046	1,573	2,097	2,621	3,240	540	1,049	1,571	2,094	2,619	3,238
TK_1 array	636	549	549	549	549	549	637	1,231	1,845	2,459	3,083	3,818
TK_2 array	844	749	744	748	744	748	674	1,296	1,938	2,578	3,239	3,989
TK_3 array	675	585	586	585	585	586	746	656	657	656	656	656
TK_4 array	675	577	576	577	577	576	865	782	782	780	780	781
KeySR	735	1,468	2,201	2,935	3,668	4,402	—	—	—	—	—	—
Shift reg.	—	—	—	—	—	—	377	754	1,131	1,508	1,885	2,262

Implementation of Shares. The state array and KeySR are simply duplicated for masking, which ensures the component-wise independence. The components in the unprotected region (see Fig. 6-(right)) have no SCA protection. The Unshare module interfaces the protected and unprotected regions by converting the data in shared representation into its bare form. Besides the S-box circuit, this Unshare module is the only place wherein shares can interact. To avoid an exploitable leakage by unsharing the unwanted intermediate data, the Unshare module has a dedicated input register, which strictly controls the incoming data from flowing into the XOR gates that make actual unsharing.

PFB_Plus Implementation. Our PFB_Plus design follows the conventional one [26] and is adjusted for the pipelined S-box circuit in Fig. 6. As a result, the state array and the S-box circuit are mostly the same between our HOMA and PFB_Plus implementations. Meanwhile, there are important differences in the tweakey arrays. In particular, the TK_1 and TK_2 arrays for PFB_Plus store the secret key, which stays in the protected region and is duplicated for masking. PFB_Plus needs an additional state outside the TBC, and we implemented it using a simple shift register similar to KeySR.

6.4 Performance Evaluation and Comparison

We describe the HOMA and PFB_Plus implementations at the register-transfer level except for the direct instantiation of the scan flip-flops [26]. We evaluate the performances by synthesizing the circuits using Synopsys Design Compiler with the NanGate 45-nm standard cell library [31]. To make component-wise comparison, we preserve the hierarchy of the components shown in Fig. 6-(right). Tables 3 show the post-synthesis performances of HOMA and PFB_Plus. We examine the protection orders $d \in \{0, \cdots, 5\}$ by considering the experimental security evaluation in the original paper [9, 10].

The results are consistent with the memory advantage in Table 1, and HOMA outperforms PFB_Plus in all the cases with SCA protection, i.e., $d > 0$. In those cases, HOMA's area reduction is larger than that of the entire S-box. For example, at $d = 5$, HOMA saves 4,237 GE wherein the S-box circuit uses 4,189 GE. In

other words, HOMA achieves the area reduction that is impossible with the conventional approaches focusing on S-box, i.e., reducing S-box's multiplicative complexity [1, 16, 17] and improving each AND gadget [9, 10].

The results confirm that the memory elements still dominate the overall circuit area with the practical protection orders, and HOMA saves a considerable amount of hardware resources. As discussed in Section 6.2, the cost of the AND gadgets and the entire S-box circuit grows quadratically with the protection order d, which will eventually overwhelm the memory elements that grow only linearly. Although we can confirm the S-box circuit's quadratic growth in Tables 3, the memory elements still dominate the total cost with $d \in \{0, \cdots, 5\}$. Besides, the simple key schedule of SKINNYee greatly contributes to the small area: the shift-register based KeySR achieves lower per-bit cost than that of the TK_1 and TK_2 arrays that PFB_Plus uses for storing the key.

HOMA essentially trades the area with latency; HOMA (resp. PFB_Plus) calls the TBC twice (resp. once) for each 64-bit message block. Also, the number of clock cycles for each TBC is extended by roughly 56/44 because SKINNYee has 56 rounds compared with 44 rounds of SKINNYe. However, we believe the area has priority in embedded-system applications, and that would be why serialized architectures having only a single S-box circuit is popular in previous literature.

7 Conclusions

We proposed an AEAD scheme that has the smaller memory usage with $(d+1)$ high-order masking. Achieving this goal, we proposed the strategy that a key-dependent state is separated into public and secret states. We then proposed the new mode HOMA that the half of the state is public, and the new TBC needed for its instantiation. We proved that for $(d+1)$ high-order masking, our scheme outperforms the previous state-of-the-art with respect to circuit area.

Designing an AEAD scheme with a smaller memory usage with $(d+1)$ high-order masking is an interesting future research. One promising approach is to extend the ratio of unprotected state in our design strategy. While SKINNYee was designed based on SKINNY for the purpose of clarifying performance comparisons, designing a new TBC with a new structure for the extended mode that requires a higher number of TK states is another interesting challenge.

References

1. Albrecht, M.R., Rechberger, C., Schneider, T., Tiessen, T., Zohner, M.: Ciphers for MPC and FHE. In: EUROCRYPT 2015. vol. 9056, pp. 430–454 (2015)
2. Barwell, G., Martin, D.P., Oswald, E., Stam, M.: Authenticated Encryption in the Face of Protocol and Side Channel Leakage. In: ASIACRYPT 2017. pp. 693–723 (2017)
3. Beierle, C., Jean, J., Kölbl, S., Leander, G., Moradi, A., Peyrin, T., Sasaki, Y., Sasdrich, P., Sim, S.M.: The SKINNY Family of Block Ciphers and Its Low-Latency Variant MANTIS. In: CRYPTO 2016. pp. 123–153 (2016)

4. Belaïd, S., Grosso, V., Standaert, F.: Masking and leakage-resilient primitives: One, the other(s) or both? Cryptogr. Commun. **7**(1), 163–184 (2015)
5. Bellizia, D., Berti, F., Bronchain, O., Cassiers, G., Duval, S., Guo, C., Leander, G., Leurent, G., Levi, I., Momin, C., Pereira, O., Peters, T., Standaert, F., Udvarhelyi, B., Wiemer, F.: Spook: Sponge-based leakage-resistant authenticated encryption with a masked tweakable block cipher. IACR Trans. Symmetric Cryptol. **2020**(S1), 295–349 (2020)
6. Berti, F., Guo, C., Pereira, O., Peters, T., Standaert, F.: TEDT, a Leakage-Resist AEAD Mode for High Physical Security Applications. IACR Trans. Cryptogr. Hardw. Embed. Syst. **2020**(1), 256–320 (2020)
7. Bilgin, B., Gierlichs, B., Nikova, S., Nikov, V., Rijmen, V.: Higher-Order Threshold Implementations. In: ASIACRYPT 2014. pp. 326–343 (2014)
8. Cassiers, G.: fullVerif. `https://github.com/cassiersg/fullverif` (2021)
9. Cassiers, G., Gregoire, B., Levi, I., Standaert, F.X.: Hardware Private Circuits: From Trivial Composition to Full Verification. IEEE Transactions on Computers pp. 1–1 (2020)
10. Cassiers, G., Levi, I.: AND depth 2, 4 ANDs, 4-bit (Optimized) S-boxes. IACR Cryptol. ePrint Arch. **2020**, 185 (2020), `https://eprint.iacr.org/2020/185`
11. Chakraborti, A., Datta, N., Jha, A., Mancillas-López, C., Nandi, M., Sasaki, Y.: Elastic-Tweak: A Framework for Short Tweak Tweakable Block Cipher. IACR Cryptol. ePrint Arch. **2019**, 440 (2019), `https://eprint.iacr.org/2019/440`
12. Chakraborti, A., Datta, N., Nandi, M., Yasuda, K.: Beetle Family of Lightweight and Secure Authenticated Encryption Ciphers. IACR Trans. Cryptogr. Hardw. Embed. Syst. **2018**(2), 218–241 (2018)
13. Dobraunig, C., Eichlseder, M., Mangard, S., Mendel, F., Mennink, B., Primas, R., Unterluggauer, T.: ISAP v2.0. IACR Trans. Symmetric Cryptol. **2020**(S1), 390–416 (2020)
14. Dobraunig, C., Mennink, B.: Leakage resilient value comparison with application to message authentication. In: EUROCRYPT 2021. pp. 377–407 (2021)
15. Dziembowski, S., Pietrzak, K.: Leakage-Resilient Cryptography. In: IEEE Symposium on Foundations of Computer Science, FOCS 2008. pp. 293–302 (2008)
16. Gérard, B., Grosso, V., Naya-Plasencia, M., Standaert, F.: Block Ciphers That Are Easier to Mask: How Far Can We Go? In: CHES 2013. pp. 383–399 (2013)
17. Goudarzi, D., Jean, J., Kölbl, S., Peyrin, T., Rivain, M., Sasaki, Y., Sim, S.M.: Pyjamask: Block Cipher and Authenticated Encryption with Highly Efficient Masked Implementation. IACR Trans. Symmetric Cryptol. **2020**, 31–59 (2020)
18. Grosso, V., Leurent, G., Standaert, F.X., Varici, K., Durvaux, F., Gaspar, L., Kerckhof, S.: `SCREAM` & `iSCREAM` side-channel resistant authenticated encryption with masking. Submitted to CAESAR (2014)
19. Hadipour, H., Bagheri, N., Song, L.: Improved Rectangle Attacks on SKINNY and CRAFT. IACR Cryptol. ePrint Arch. p. 1317 (2020)
20. Ishai, Y., Sahai, A., Wagner, D.A.: Private Circuits: Securing Hardware against Probing Attacks. In: CRYPTO 2003. pp. 463–481 (2003)
21. Iwata, T., Khairallah, M., Minematsu, K., Peyrin, T.: Duel of the Titans: The Romulus and Remus Families of Lightweight AEAD Algorithms. IACR Trans. Symmetric Cryptol. **2020**(1), 43–120 (2020)
22. Jean, J., Nikolić, I., Peyrin, T.: Tweaks and Keys for Block Ciphers: The TWEAKEY Framework. In: ASIACRYPT 2014. vol. 8874, pp. 274–288 (2014)
23. Kannwischer, M.J., Pessl, P., Primas, R.: Single-Trace Attacks on Keccak. IACR Trans. Cryptogr. Hardw. Embed. Syst. **2020**(3), 243–268 (2020)

24. Kocher, P.C., Jaffe, J., Jun, B.: Differential Power Analysis. In: CRYPTO '99. pp. 388–397 (1999)
25. Naito, Y., Matsui, M., Sugawara, T., Suzuki, D.: SAEB: A Lightweight Blockcipher-Based AEAD Mode of Operation. IACR Trans. Cryptogr. Hardw. Embed. Syst. **2018**(2), 192–217 (2018)
26. Naito, Y., Sasaki, Y., Sugawara, T.: Lightweight Authenticated Encryption Mode Suitable for Threshold Implementation. In: EUROCRYPT 2020. pp. 705–735 (2020)
27. Naito, Y., Sasaki, Y., Sugawara, T.: LM-DAE: Low-Memory Deterministic Authenticated Encryption for 128-bit Security. IACR Trans. Symmetric Cryptol. **2020**(4), 1–38 (2020)
28. Naito, Y., Sasaki, Y., Sugawara, T.: Secret Can Be Public: Low-Memory AEAD Mode for High-Order Masking. IACR Cryptol. ePrint Arch. **2022**, 812 (2022), `https://eprint.iacr.org/2022/812`
29. Naito, Y., Sugawara, T.: Lightweight Authenticated Encryption Mode of Operation for Tweakable Block Ciphers. IACR Trans. Cryptogr. Hardw. Embed. Syst. **2020**(1), 66–94 (2020)
30. Namprempre, C., Rogaway, P., Shrimpton, T.: Reconsidering Generic Composition. In: EUROCRYPT 2014 (2014)
31. NanGate: NanGate FreePDK45 Open Cell Library. `https://si2.org/open-cell-library/` (2021), accessed: 2021-05-06
32. Nikova, S., Rechberger, C., Rijmen, V.: Threshold Implementations Against Side-Channel Attacks and Glitches. In: Information and Communications Security, 8th International Conference, ICICS 2006. pp. 529–545 (2006)
33. NIST: National Institute of Standards and Technology: Submission Requirements and Evaluation Criteria for the Lightweight Cryptography Standardization Process. `https://csrc.nist.gov/Projects/lightweight-cryptography` (2018)
34. NIST: National Institute of Standards and Technology: Lightweight Cryptography Standardization: Finalists Announced. `https://csrc.nist.gov/News/2021/lightweight-crypto-finalists-announced` (2021)
35. Patarin, J.: The "Coefficients H" Technique. In: SAC 2008. pp. 328–345 (2008)
36. Pereira, O., Standaert, F., Vivek, S.: Leakage-Resilient Authentication and Encryption from Symmetric Cryptographic Primitives. In: CCS 2015. pp. 96–108 (2015)
37. Prouff, E., Rivain, M.: Masking against Side-Channel Attacks: A Formal Security Proof. In: EUROCRYPT 2013. pp. 142–159 (2013)
38. Reparaz, O.: A Note on the Security of Higher-Order Threshold Implementations. IACR Cryptol. ePrint Arch. **2015**, 1 (2015), `http://eprint.iacr.org/2015/001`
39. Reparaz, O., Bilgin, B., Nikova, S., Gierlichs, B., Verbauwhede, I.: Consolidating Masking Schemes. In: CRYPTO 2015. vol. 9215, pp. 764–783 (2015)
40. Tolba, M., Abdelkhalek, A., Youssef, A.M.: Impossible Differential Cryptanalysis of Reduced-Round SKINNY. In: AFRICACRYPT 2017. pp. 117–134 (2017)