IACR Transactions on Cryptographic Hardware and Embedded Systems
ISSN 2569-2925, Vol. 2023, No. 4, pp. 80–109. DOI:10.46586/tches.v2023.i4.80-109

MMM: Authenticated Encryption with Minimum Secret State for 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, yusk.sasaki@ntt.com
[3] The University of Electro-Communications, Tokyo, Japan, sugawara@uec.ac.jp

Abstract. We propose a new authenticated encryption (AE) mode MMM that achieves the minimum memory size with masking. Minimizing the secret state is the crucial challenge in the low-memory AE suitable for masking. Here, the minimum secret state is $s + b$ bits, composed of s bits for a secret key and b bits for a plaintext block. HOMA appeared in CRYPTO 2022 achieved this goal with $b = 64$, but choosing a smaller b was difficult because $b = s/2$ is bound to the block size of the underlying primitive, meaning that a block cipher with an unrealistically small block size (e.g., 8 bits) is necessary for further improvement. MMM addresses the issue by making b independent of the underlying primitive while achieving the minimum $(s + b)$-bit secret state. Moreover, MMM provides additional advantages over HOMA, including (i) a better rate, (ii) the security under the multi-user model, (iii) and a smaller transmission cost. We instantiate two variants, MMM-8 (with $b = 8$) and MMM-64 (with $b = 64$), using the standard tweakable block cipher SKINNY-64/192. With a $(d + 1)$-masking scheme, MMM-8 (resp. MMM-64) is smaller by $56d + 184$ (resp. 128) bits compared with HOMA. As a result of hardware performance evaluation, MMM-8 and MMM-64 achieved smaller circuit areas than HOMA with all the examined protection orders $d \in [0, 5]$. MMM-8's circuit area is only 81% of HOMA with $d = 5$, and MMM-64 achieves more than $\times 3$ speed-up with a smaller circuit area.

Keywords: Authenticated Encryption · Mode of Operation · Side-Channel Attack · Masking · Multi-User Security

1 Introduction

Resistance against side-channel attacks (SCAs) [KJJ99] is an important challenge in secure embedded systems, and masking is intensively studied as a viable countermeasure against SCAs. Masking schemes encode the intermediate values into a redundant and randomized representation called shares and implement cryptographic algorithms with shares, in the same way as multi-party computation. The security of masking is based on the probing model [ISW03], and a protection order d specifies the maximum number of probes the implementation is protected. However, large performance overhead is the most significant drawback of masking, and researchers are tackling the issue from several different directions, including more efficient masking schemes [NRR06, RBN+15, GMK16, CGLS21], randomness optimization [BBP+16], masking-friendly S-box [GGNS13, ARS+15, GJK+20], and leakage-resilient cryptography [DP08, PSV15, DEM+20, BGP+20, DM21].

Among them, some low-memory authenticated encryption with associated data schemes (AEADs) aim at minimizing the memory size after masking [NS20, IKMP20, NSS22]. They are useful in compact hardware implementations because memory elements, e.g., registers, consume considerable hardware resources. The modes exploit the asymmetry between the

Licensed under Creative Commons License CC-BY 4.0.
Received: 2023-04-15 Accepted: 2023-06-15 Published: 2023-08-31

Table 1: Comparison of Memory Size after Masking with Protection Order d with 128-bit security. MMM is evaluated with general variables with a rate of b bits and a security level of s bits, where the rate denotes the number of encrypted bits per primitive call.

Scheme	Rate	Memory Size	Security	Ref.
PFB, Romulus, PFB_Plus	128	$256(d+1)+128$	Single-User†	[NS20, IKMP20, NSS20]
HOMA	64/3	$192(d+1)+256$	Single-User†	[NSS22]
MMM	b	$(s+b)(d+1)+s$	Multi-User	This Work
MMM-8	8	$136(d+1)+128$	Multi-User	This Work
MMM-64	64	$192(d+1)+128$	Multi-User	This Work

† The proofs are given only in the su-setting, while there is no efficient attack in the mu-setting.

secret and public values; a secret value needs a large memory size to store its redundant representation. At least $d+1$ shares are needed to provide the protection order d, so secret data expands $d+1$ times after masking. On the other hand, a public value does not need such redundant representation. Hence, reducing the size of the secret value can reduce the memory size after masking.

First, tweakable block cipher (TBC)-based modes, such as PFB [NS20], Romulus [IKMP20], and PFB_Plus [NSS20], were found to be effective because TBCs effectively handle public values as tweaks, in contrast to permutation-based modes that handle the entire state with a single permutation. The state-of-the-art in this category is HOMA appeared in CRYPTO 2022 [NSS22], which further reduced the size of the secret value.

Table 1 summarizes the memory sizes of those modes. PFB, Romulus, and PFB_Plus are essentially the same in terms of memory size after masking. For achieving 128-bit security, those modes use a 128-bit internal state and a 128-bit key that should be protected with masking. Besides, they use a 128-bit public value, including a nonce, a block counter, and domain separation bits, which requires no SCA protection. As a result, PFB, Romulus, and PFB_Plus use $2s(d+1)+s$ bits of memory with s-bit security and $(d+1)$-masking, i.e., $256(d+1)+128$ bits with 128-bit security. Since the memory size depends on the security parameter s, these modes cannot reduce the memory any further, and reducing the memory size without sacrificing the the security level has become the new challenge. Then, HOMA successfully addressed the limit, achieving a new memory-area trade-off. HOMA also uses a 128-bit internal state and a 128-bit key, however, it does not protect a half of the 128-bit state, which reduces the secret state size to 192 bits. Meanwhile, the public state size increases to 256 bits as a drawback. As a result, the total state size of HOMA is $192(d+1)+256$ bits; successfully reducing the coefficient on d.

However, HOMA has several limitations to achieving this goal, as shown below.

Needs for Specialized Primitive. HOMA needs a TBC with a small block size and a large tweakey, a combination of a tweak and a key, i.e., 64-bit block and 384-bit tweakey for 128-bit security. No existing primitive satisfies this requirement, thus the designers developed a new primitive called SKINNYee [NSS22]. It was mentioned that the memory could be further improved if a TBC with an even smaller block size, e.g., 8 bits, with the same tweaky size, i.e. 384 bits, exists. However, looking at past designs, no primitive supports such configuration, and it was observed that efficiently constructing such a TBC is difficult [QDW+22].

Inflexible Speed-Area Trade-Off. Ideally, the memory size and the speed should have some trade-off so that users can flexibly choose the suitable one for a performance requirement. However, HOMA only enables the fixed relationship; the secret size of $1.5s$ bits and a rate of $s/6$ bits by using a TBC with a $0.5s$-bit block.

Inefficient Rate. PFB/Romulus requires only a single primitive call to process a 128-bit plaintext block. In contrast, HOMA needs 3 primitive calls for each 64-bit plaintext block, which results in an inefficient rate as a drawback.

Multi-User Security. The multi-user security considers a stronger adversary who can make query to multiple users with distinct keys [BT16, HTT18, LMP17]. Recently, the real-world protocols determine the AES-GCM's rekeying frequencies based on the mu-security [Res18, RTM21, TT21]. HOMA's security proof only gives a single-user bound, and a new mode ensuring high mu-security is desired.[1] Hereafter, "single-user" and "multi-user" are abbreviated to "su" and "mu," respectively.

Additional Transmission Cost. HOMA requires the use of random IVs that are updated with each query. The random IV must be transmitted as a part of a ciphertext. This increases the data transmission over the network, which can be a problem for IoT devices with power constraints and limited bandwidths.

Another notable mode-level approach for efficient SCA countermeasure is leveled implementations [PSV15, BBC+20], including leakage-resilient (LR) schemes [BGP+20, BBB+20]. They use both SCA-protected and SCA-unprotected primitives to ensure security while minimizing the use of costly protected primitives. Besides the difference in the security models, low-memory AEADs and leveled implementations aim at different performance goals; leveled implementations aim at speed, not memory size. Masking is required to realize the SCA-protected primitive [DM21] in leveled implementations, which determines the minimum memory size. Hardware cost is likely prioritized over speed with resource-constrained devices, especially when the speed is restricted by other aspects, such as a limited wireless power supply or slow wired/wireless communication. In those cases, a smaller circuit area, i.e., lower per-chip cost, is more important.

In summary, further reducing the memory size of HOMA is an important challenge. Moreover, there are other desirable properties, such as (i) compatibility with existing primitives, including standardized ones, (ii) flexibility to choose speed-area trade-off, (iii) improved speed, (iv) mu-security, and (v) reducing data transmission cost of random IV.

However, reducing the memory size of HOMA is difficult. Let b be the plaintext block size, that is, the message size to be processed in one iteration, n be the block size of the block cipher, and s be the bits of security. HOMA has $b = n$, $n = s/2$, and requires an s-bit secret key. Then, not only the s-bit key but also the n-bit internal state of the block cipher (BC) must be protected as secret. This is because, in many designs, if state values during a BC operation are leaked, an s-bit key can be derived by a simple operation such as XOR. Hence, HOMA requires at least $1.5s$ bits of the secret state. This is equal to the size of a BC with an s-bit key and an n-bit block, implying that HOMA as a whole does not use any more secret state than the one required to implement the primitive. Therefore, with HOMA's approach, the only possible direction is to make n smaller, i.e., going to primitives with smaller block sizes, but as mentioned above, this seems difficult.

Our Contributions

In this paper, we propose a new mode MMM that outperforms HOMA in the memory size and addresses the aforementioned limitations of HOMA. Then, MMM is instantiated with SKINNY [BJK+16] for two plaintext block sizes $b = 8$ and $b = 64$, called MMM-8 and MMM-64. We benchmark those instances by implementing them in hardware to show that MMM-8 outperforms HOMA in terms of the memory size after masking. MMM-64 with the same plaintext block size as HOMA can also provide a better performance.

[1] Note that the proofs of HOMA, PFB, PFB_Plus and Romulus are given only in the su-setting, while there is no efficient attack on these schemes in the mu-setting.

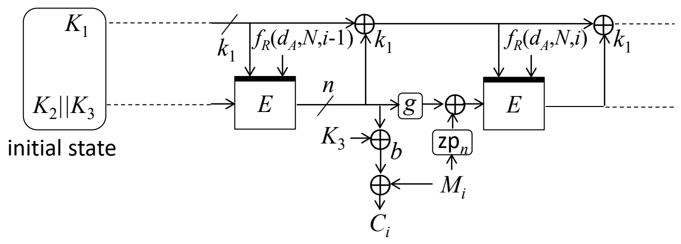

Figure 1: Encryption of MMM. zp_n appends zeros until n bits. For $l \in \{k_1, b\}$, $X \oplus_l Y$ is an XOR operation for most significant l bits.

New Approach to Achieve Minimum Secret State (Section 3 and 4). The crucial idea of MMM is to update the whole s-bit secret state by iterating an n-bit block BC, where $n \le s$, and to encrypt a b-bit plaintext block by using the b-bit part of the s-bit state, as shown in Fig. 1. A k-bit key K of MMM is partitioned into a k_1-bit key K_1, a k_2-bit key K_2, and a b-bit key K_3 such that $n = k_2 + b$, $k_1 \le n$, $k = s$ and $k = k_1 + k_2 + b$.[2] The s-bit state is initialized by K, and is partitioned into n-bit and k_1-bit states. Then, the n-bit state is updated by a BC and the remaining k_1-bit state is updated by XORing with the BC's output. In the encryption, the b-bit part of the state for encrypting a plaintext block is XORed with K_3, preventing the b-bit part from leaking to maintain the secret state size s bits. Hence, MMM achieves the optimal secret size of $s + b$ bits. The size b can be flexibly chosen from 1 to n, and MMM's secret state becomes smaller than HOMA by choosing $b < s/2$. In Fig. 1, a linear state-update function g is introduced to randomize each BC's input by the previous BC's output during the decryption, ensuring the s-bit authenticity. For each BC call, an s-bit public value consisting of a domain separation, a nonce, and a counter is input as its key element via an encoding function f_R as well as the k_1-bit state. These ensure that the BC's key elements in the encryption are all distinct for each user, ensuring that the BC's outputs are independent and the s-bit privacy is achieved. Due to $s + b$ and s bits of the secret and public states, respectively, the memory size after masking is $(s + b)(d + 1) + s$ bits, as shown in Table 1. b can be chosen by a user and can change without replacing the BC, which offers a flexible speed-area trade-off.

We prove that MMM is a secure AE in the ideal cipher (IC) model and in both su- and mu-settings. To prove the security, we use the z-bound model [HTT18] that was first proposed to prove mu-security of GCM.[3] In the z-bound model, it is assumed that the number of collisions of nonces (or pairs of nonce and counter in MMM) between different users is at most z. We prove that the bound is $O(\frac{z\sigma_{\mathcal{E}}+\sigma_{\mathcal{D}}+zp}{2^k})$ where $\sigma_{\mathcal{E}}$ (resp. $\sigma_{\mathcal{D}}$) is the number of BC calls in the encryption (resp. decryption) of MMM, p is the number of offline queries, and the tag size is k bits. In the su-setting, $z = 1$ is ensured from its definition, hence the k-bit security is ensured without introducing randomness. In the mu-setting, randomness is necessary to reduce z, but this random number does not need to be updated during the lifetime and does not need to be sent to the communication partner. By ensuring that the randomized nonce/counter size is long enough, z is bounded to be small and the k-bit security is ensured.[4]

Regarding the underlying BC, if $k_1 > 0$, both the k_1-bit secret value and s-bit public value are input to the BC's key element. If the public value is updated by the secret value, the updated value needs to be treated as secret, resulting in an increase of the memory

[2]Note that $k_1 \ge 0$, and if $k_1 = 0$, the k_1-bit state is removed.

[3]It was originally called the d-bound model [HTT18]. This paper uses z instead of d to avoid ambiguity with the protection order d of masking.

[4]One specific implementation of the z-bound model is a randomized nonce, in which each user determines one random number at the beginning, and each time a nonce is generated, that random number is XORed to the nonce. In MMM, a counter is also randomized in addition to a nonce to further reduce z.

Table 2: Comparison of primitive requirements

Mode	Underlying primitive type	Unnecessity of independent key schedule update for optimized implementation	Availability in international standard
HOMA	standard-model secure TBC	–	–
PFB/Romulus	standard-model secure TBC	–	✓
MMM with $k_1 \geq 1$	ideal cipher	–	✓
MMM with $k_1 = 0$	ideal cipher	✓	✓

size. To avoid it, we have a condition for the BC that the k_1-bit secret part and s-bit public part must be independently updated in the key schedule function. Note that many BCs satisfy this condition. In particular, in lightweight ciphers, the key schedule has a lot of independence because the amount of data mixing is reduced to a minimum for small implementations. Specifically, key schedules of Midori [BBI+15] and GIFT [BPP+17] discussed in Sect. 6 have such properties. If $k_1 = 0$, the condition is not required.

MMM does not use random IVs, which reduces the transmission cost compared with HOMA. The reason why HOMA needs random IVs comes from its design nature, which leaves the half of its secret state unprotected. Without randomness, this reveals the information that a collision occurred during decryption. To avoid this problem, HOMA introduced random IVs that are updated with each query, so that the collided pair in the past cannot be used in future decryption attempts. Unlike HOMA, the entire s-bit state of MMM is protected, so the occurrence of a collision during decryption is not detected even without randomness.

HOMA uses unprotected states without SCA protection, expressed as a leakage oracle in the security notion. In contrast, MMM/PFB/Romulus' notion doesn't include a leakage oracle simply because they have no unprotected state. MMM's security is proved with the IC model, unlike HOMA/PFB/Romulus with the standard model, as summarized in Table 2. The IC model requires stronger assumptions on BC, which enables more flexible use of BC, enabling MMM's better performances. The omission of the condition on the key schedule function of the underlying primitive is only applied to MMM with $k_1 = 0$. HOMA cannot be instantiated with existing TBCs, while PFB/Romulus and MMM can be instantiated with SKINNY. Those requirements on the underlying primitive by each mode is compared in Table 2.

Instantiation and Hardware Performance Evaluation (Section 5). The BC's key size in MMM is $k_1 + s$ bits, which is only 192 bits for $s = 128$ and $n = 64$. This enables us to instantiate MMM with the ISO-standard cipher SKINNY-64/192 [ISO22]. We benchmark two instances with different b, namely MMM-8 and MMM-64, and compare them with HOMA. SKINNY's tweakey schedule function allows users to choose the size of the public value, tweak, and the secret value, key, and to update each independently. This property fully satisfies the condition on the underlying cipher.

We evaluate the hardware performance of MMM-8 and MMM-64 by implementing them using the high-order masking scheme HPC2, which provides composable security under the glitch-robust probing model [CS20]. We use the fullVerif verification tool [Cas21] to verify the SCA resistance of our implementations. As a result, MMM-8 and MMM-64 outperform HOMA with the protection orders $d \in [0, 5]$ (see Table 3 in Section 5). With $d = 5$, for example, MMM-8 achieves 14,039 GE, which is only 81% of HOMA with 17,261 GE. This advantage comes from the smaller memory; MMM-8 is smaller by $56d + 184$ bits than HOMA (see Table 1). MMM-64 achieves 16,526 GE at $d = 5$, which is still smaller than HOMA, while achieving more than $\times 3$ speed-up because of the better rate.

```
$\underline{\mathcal{E}_R(\nu, N, A, M)}$
if $\nu > u$ then return $\varepsilon$
return $\Pi^E_{K_\nu}.\mathsf{Enc}(N, A, M)$

$\underline{\mathcal{D}_R(\nu, N, A, C, T')}$
if $\nu > u$ then return **reject**
return $\Pi^E_{K_\nu}.\mathsf{Dec}(N, A, C, T')$
```

```
$\underline{\mathcal{E}_I(\nu, N, A, M)}$
if $\nu > u$ then return $\varepsilon$
return $\$_\nu(N, A, M)$

$\underline{\mathcal{D}_I(\nu, N, A, C, T')}$
return **reject**
```

Figure 2: Encryption and decryption oracles in the real (left) and ideal (right) worlds.

2 Preliminaries

Notations. For integers $0 \le i \le j$, let $[i, j] := \{i, i+1, \ldots, j\}$, $[j] := [1, j]$, and $[j]_0 := [0, j]$. If $i > j$ then $[i, j] := \emptyset$. Let ε be an empty string, $\emptyset$ an empty set, and $\{0,1\}^*$ be the set of all bit strings. For an integer $n \ge 0$, let $\{0,1\}^n$ be the set of all n-bit strings, $\{0,1\}^0 := \{\varepsilon\}$, and $\{0,1\}^{\le n} := \cup_{i \in (n]}\{0,1\}^i$. Let 0^i be the bit string of i-bit zeros. For $X \in \{0,1\}^j$, let $|X| := j$. The concatenation of two bit strings X and Y is written as $X\|Y$ or XY when no confusion is possible. For a non-empty set $\mathcal{T}$, $T \xleftarrow{\$} \mathcal{T}$ means that an element is chosen uniformly at random from $\mathcal{T}$ and assigned to T.

A block cipher (BC) is a set of permutations indexed by a key. Throughout this paper, block and key bit-lengths of BC are denoted by n and t, respectively. An encryption of BC is denoted by $E : \{0,1\}^t \times \{0,1\}^n \to \{0,1\}^n$, and its decryption is denoted by $E^{-1} : \{0,1\}^t \times \{0,1\}^n \to \{0,1\}^n$. Let $\mathbf{BC}$ be the set of all encryptions of n-bit block and t-bit key BC.

Authenticated Encryption. An authenticated encryption (AE) scheme based on a BC E, denoted by Π^E_K, is a pair of encryption and decryption algorithms $(\Pi^E_K.\mathsf{Enc}, \Pi^E_K.\mathsf{Dec})$. $\mathcal{K}$, $\mathcal{N}$, $\mathcal{M}$, $\mathcal{C}$, $\mathcal{A}$, and $\mathcal{T}$ are the sets of keys, nonces, plaintexts, ciphertexts, associated data (AD), and tags, respectively. The encryption algorithm takes a tuple $(N, A, M) \in \mathcal{N} \times \mathcal{A} \times \mathcal{M}$, and returns, deterministically, a pair $(C, T) \in \mathcal{C} \times \mathcal{T}$. The decryption algorithm takes a tuple $(N, A, C, T') \in \mathcal{N} \times \mathcal{A} \times \mathcal{C} \times \mathcal{T}$ and returns, deterministically, either the distinguished invalid symbol **reject** $\notin \mathcal{M}$ or a plaintext $M \in \mathcal{M}$. We require that for any $(N, A, M), (N', A', M') \in \mathcal{N} \times \mathcal{A} \times \mathcal{M}$, $|\Pi^E_K.\mathsf{Enc}(N, A, M)| = |\Pi^E_K.\mathsf{Enc}(N, A, M')|$ is satisfied if $|M| = |M'|$. We also require that $\Pi^E_K.\mathsf{Dec}(N, A, \Pi^E_K.\mathsf{Enc}(N, A, M)) = M$ for any $N \in \mathcal{N}$, $A \in \mathcal{A}$, and $M \in \mathcal{M}$.

Multi-User AE Security in the Ideal Cipher Model. We evaluate multi-user-AE (mu-AE) security of our AE scheme in the ideal cipher (IC) model, where an adversary has access to multiple users and the underlying IC. mu-AE-security is the indistinguishability between the real and ideal worlds.

Let u be the number of users. Let $\mathbf{A}$ be an adversary in the mu-AE-security game. $\mathbf{A}$ has access to either real-world oracles $(\mathcal{E}_R, \mathcal{D}_R, E, E^{-1})$ or ideal-world ones $(\mathcal{E}_I, \mathcal{D}_I, E, E^{-1})$. An IC's encryption is defined as $E \xleftarrow{\$} \mathbf{BC}$. The pairs of encryption and decryption oracles $(\mathcal{E}_R, \mathcal{D}_R)$ and $(\mathcal{E}_I, \mathcal{D}_I)$ are defined in Fig. 2. $\mathbf{A}$ has access to the ν-th user's encryption (resp. decryption) oracle by making queries to $\mathcal{E}_W$ (resp. $\mathcal{D}_W$) with a user number $\nu \in [u]$ where $W \in \{R, I\}$. User's keys K_i are defined as $K_i \xleftarrow{\$} \mathcal{K}$ where $i \in [u]$. $\$_\nu(N, A, M)$ is a random-bit oracle of the ν-th user and has the same interface as $\Pi^E_{K_\nu}.\mathsf{Enc}$. For a query (ν, N, A, M), the response is defined as $(C, T) \xleftarrow{\$} \{0,1\}^{|\Pi^E_{K_\nu}.\mathsf{Enc}(N,A,M)|}$. At the end of this game, $\mathbf{A}$ return a decision bit in $\{0,1\}$. Let $\mathbf{A}^{\mathcal{E}_W, \mathcal{D}_W, E, E^{-1}} \in \{0,1\}$ be an output of $\mathbf{A}$ with access to oracles $(\mathcal{E}_W, \mathcal{D}_W, E, E^{-1})$ where $W \in \{R, I\}$, Then, the mu-AE-security

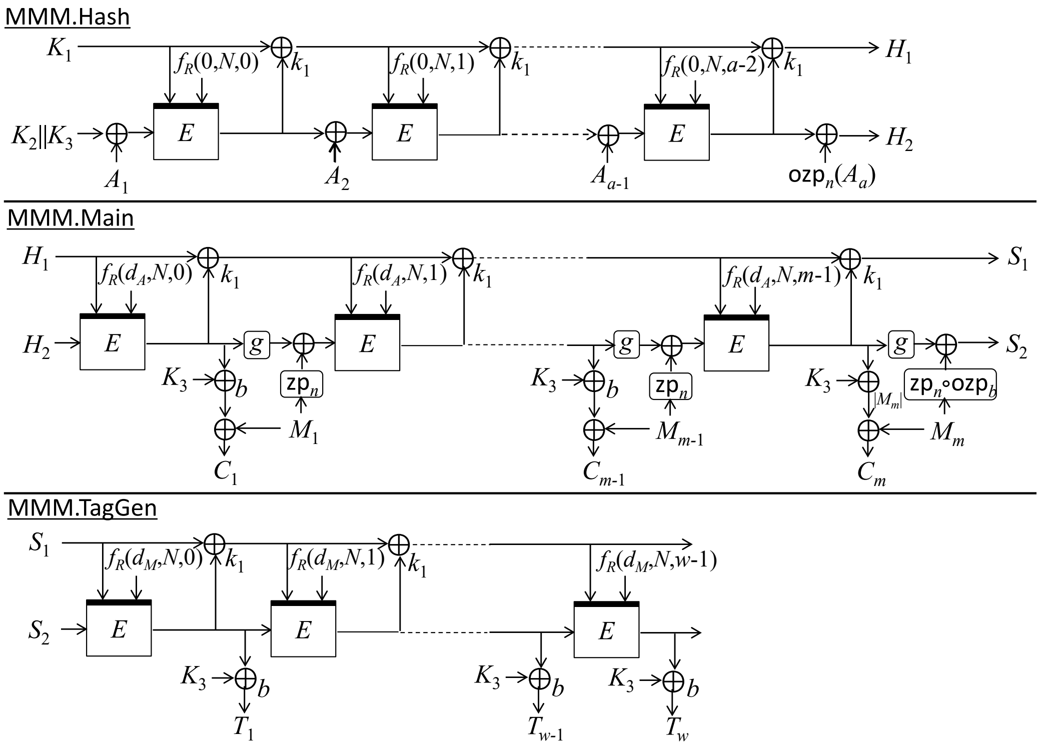

Figure 3: Structure of encryption of MMM.

advantage function of $\mathbf{A}$ is defined as

$$\mathbf{Adv}_{\Pi}^{\mathsf{mu\text{-}ae}}(\mathbf{A}) := \Pr\left[\mathbf{A}^{\mathcal{E}_R,\mathcal{D}_R,E,E^{-1}} = 1\right] - \Pr\left[\mathbf{A}^{\mathcal{E}_I,\mathcal{D}_I,E,E^{-1}} = 1\right] .$$

$\mathbf{A}$ is nonce-respecting, that is, all nonces in queries to the encryption oracle with the same-user number are distinct. In this game, making a trivial query (ν, N, A, C, T') to the decryption oracle is forbidden, which was received by some previous query to the encryption oracle. For $W \in \{R, I\}$, we refer the particular queries to as follows:

- offline queries: queries to E or E^{-1},
- forward queries: queries to E,
- inverse queries: queries to E^{-1},
- online queries: queries to $\mathcal{E}_W$ or $\mathcal{D}_W$,
- encryption queries: queries to $\mathcal{E}_W$, and
- decryption queries: queries to $\mathcal{D}_W$.

3 AE with Minimum Secret State: Design and Security

In this section, we first give a specification of our mode MMM and explain how to design MMM. We then show the security bound of MMM. Finally, we give an overview of the security proof.

Algorithm 1 $\mathsf{MMM.Enc}_K^E$

```
Input: $N, A, M$
Output: $C, T$
1: $(H_1, H_2, d_A) \leftarrow \mathsf{MMM.Hash}_K^E(N, A)$        ▷ Algorithm 3
2: $(C, T) \leftarrow \mathsf{MMM.Main}_K^E(\mathsf{enc}, N, d_A, H_1, H_2, M)$        ▷ Algorithm 4
3: return $(C, T)$
```

Algorithm 2 $\mathsf{MMM.Dec}_K^E$

```
Input: $N, A, C, T'$
Output: $RV$
1: $(H_1, H_2, d_A) \leftarrow \mathsf{MMM.Hash}_K^E(N, A)$        ▷ Algorithm 3
2: $(M, T) \leftarrow \mathsf{MMM.Main}_K^E(\mathsf{dec}, N, d_A, H_1, H_2, C)$        ▷ Algorithm 4
3: if $T' = T$ then $RV \leftarrow M$; else $RV \leftarrow$ reject
4: return $RV$
```

Algorithm 3 $\mathsf{MMM.Hash}_K^E$

```
Input: $N, A,$
Output: $H_1, H_2, d_A$
1: $(A_1, \ldots, A_a) \xleftarrow{n} A$; $S_1 \leftarrow K_1$; $S_2 \leftarrow (K_2 \| K_3) \oplus \mathsf{ozp}_n(A_1)$
2: if $A \neq \varepsilon$ and $|A_a| \bmod n = 0$ then $d_A \leftarrow 1$ else $d_A \leftarrow 2$ end if
3: for $i = 2, \ldots, a$ do
4:     $S_2 \leftarrow E(S_1 \| f_R(0, N, i-2), S_2)$; $S_1 \leftarrow S_1 \oplus_{k_1} S_2$; $S_2 \leftarrow S_2 \oplus \mathsf{ozp}_n(A_i)$
5: end for
6: $H_1 \leftarrow S_1$; $H_2 \leftarrow S_2$; return $(H_1, H_2, d_A)$
```

Algorithm 4 $\mathsf{MMM.Main}_K^E$

```
Input: $\mathsf{flag}, N, d_A, H_1, H_2, D$
Output: $D', T$
        ▷ If $\mathsf{flag} = \mathsf{enc}$, then $D$ is a plaintext $M$ and $D'$ is the ciphertext $C$.
        ▷ If $\mathsf{flag} = \mathsf{dec}$, then $D$ is a ciphertext $C$ and $D'$ is the plaintext $M$.
1: $S_1 \leftarrow H_1$; $S_2 \leftarrow H_2$; $(D_1, \ldots, D_m) \xleftarrow{b} D$
2: if $D \neq \varepsilon$ and $|D_m| \bmod b = 0$ then $d_M \leftarrow 3$ else $d_M \leftarrow 4$ end if
3: for $i = 1, \ldots, m$ do
4:     $S_2 \leftarrow E_K(S_1 \| f_R(d_A, N, i-1), S_2)$; $S_1 \leftarrow S_1 \oplus_{k_1} S_2$; $D'_i \leftarrow D_i \oplus_{|D_i|} K_3 \oplus_{|D_i|} S_2$
5:     if $\mathsf{flag} = \mathsf{enc}$ then $S_2 \leftarrow g(S_2) \oplus \mathsf{zp}_n \circ \mathsf{ozp}_b(D_i)$ end if
6:     if $\mathsf{flag} = \mathsf{dec}$ then $S_2 \leftarrow g(S_2) \oplus \mathsf{zp}_n \circ \mathsf{ozp}_b(D'_i)$ end if
7: end for
8: $D' \leftarrow D'_1 \| \cdots \| D'_m$; $T \leftarrow \mathsf{MMM.TagGen}_K^E(N, d_M, S_1, S_2)$        ▷ Algorithm 5
9: return $(D', T)$
```

Algorithm 5 $\mathsf{MMM.TagGen}_K^E$

```
Input: $N, d_M, S_1, S_2$
Output: $T$
1: for $i = 1, \ldots, w$ do
2:     $S_2 \leftarrow E_K(S_1 \| f_R(d_M, N, i-1), S_2)$; $S_1 \leftarrow S_1 \oplus_{k_1} S_2$; $T_i \leftarrow S_2 \oplus_b K_3$
3: end for
4: $T \leftarrow \mathsf{msb}_\tau(T_1 \| \cdots \| T_w)$; return $T$
```

3.1 Specification of MMM

The specification of MMM.Enc which is an encryption of MMM is given in Figure 3 and Algorithm 1. The specification of MMM.Dec which is a decryption of MMM is given in Figure 2. Algorithms 3, 4 and 5 are subroutines of MMM.Enc and MMM.Dec. Algorithm 3 shows MMM.Hash that processes AD. Algorithm 4 shows MMM.Main that encrypts (resp. decrypts) a plaintext (resp. a ciphertext). Algorithm 5 shows MMM.TagGen that generates a tag.

Let b be the size of plaintext and tag blocks, w the block size of tags, and $\tau := wb$ the tag size. Let k be the key size of MMM such that $n \le k \le 2n$. Let $K \in \{0,1\}^k$ be a key of MMM. In MMM, the key is partitioned into three keys: first k_1-bit key K_1, second k_2-bit key K_2, and third b-bit key K_3 such that $k = k_1 + k_2 + b$, $n = k_2 + b$, and $k_1, k_2 \ge 0$. Let v be the size of a counter space. Then, the sets of MMM are defined as $\mathcal{K} := \{0,1\}^k$, $\mathcal{A} := \{0,1\}^{\le nv}$, $\mathcal{M} = \mathcal{C} := \{0,1\}^{\le b(v-1)}$, and $\mathcal{T} := \{0,1\}^\tau$.

For an integer $i \ge 0$ and $X \in \{0,1\}^j$, let $\mathsf{msb}_i(X)$ (resp. $\mathsf{lsb}_i(X)$) be the most (resp. least) significant i-bit part of X. For an integer $l \ge 0$ and bit strings X, Y such that $|X| \ge l$ and $|Y| \ge l$, let $X \oplus_l Y := \mathsf{msb}_l(X) \oplus \mathsf{msb}_l(Y)$. Note that if $|X| = |Y| = l$, then l is omitted. Let ozp_i and zp_i be respectively one-zero and zero padding functions where for $X \in \{0,1\}^{\le i}$ $\mathsf{ozp}_i(X) = X$ and $\mathsf{zp}_i(X) = X$ if $|X| = i$; $\mathsf{ozp}_i(X) = X \| 10^{i-|X|-1}$ and $\mathsf{zp}_i(X) = X \| 0^{i-|X|}$ if $|X| < i$. Let "$\circ$" be a function composition where $\mathsf{zp}_i \circ \mathsf{ozp}_j(X) = \mathsf{zp}_i(\mathsf{ozp}_j(X))$. For an integer $l \ge 0$ and a bit string X, $(X_1, \ldots, X_\ell) \xleftarrow{l} X$ means the parsing into fixed-length l-bit strings, where if $X \ne \varepsilon$ then $X = X_1 \| \cdots \| X_\ell$, $|X_i| = l$ for $i \in [\ell - 1]$, and $0 < |X_\ell| \le l$; if $X = \varepsilon$ then $\ell = 1$ and $X_1 = \varepsilon$.

Let $g : \{0,1\}^n \to \{0,1\}^n$ be a state-update function such that for the equation $X \oplus g(X) = Y$ with two n-bit variables X and Y, if Y is fixed, then the equation offers a unique solution for X. Let $f_R : [4]_0 \times \mathcal{N} \times [v-1]_0 \to \{0,1\}^{t-k_1}$ be an injective function parameterized by a value R that takes a tuple of domain separation, nonce, and counter, where R is uniquely fixed for each user. Hence, f_R requires the condition $2^{t-k_1} \ge 5 \times |\mathcal{N}| \times v$. An example of the function is that for a domain separation ds, a nonce N, and a counter ctr such that $|(N\|ctr)| = |R|$, $f_R(ds, N, ctr) := ds\|((N\|ctr) \oplus R)$, where each value is seen as a bit-string. In the single-user setting, R can be fixed to a constant such as all 0 bits. In the multi-user setting, R is chosen uniformly at random from $\{0,1\}^{|R|}$. The definitions of domain separation are given in step 2 of MMM.Hash and step 2 of MMM.Main. In step 2 of MMM.Hash, a domain separation value d_A is defined as $d_A = 1$ if $A \ne \varepsilon$ and the last AD block satisfies $|A_a| \bmod n = 0$; $d_A = 2$ else. In step 2, a domain separation value d_M is defined as $d_A = 3$ if $M \ne \varepsilon$ and the last plaintext (resp. ciphertext) block satisfies $|M_m| \bmod n = 0$ (resp. $|C_m| \bmod n = 0$); $d_A = 4$ else.

MMM keeps a secret state of k bits and a secret value of b bits. The k-bit state is divided into two parts: k_1-bit part and n-bit one, where $n = k_2 + b$. The k_1-bit (resp. n-bit) part is initialized by the first key K_1 (resp. the second and third keys $K_2\|K_3$). Then the state is updated by iterating a BC and taking a data block. Each BC call takes a state value and a tuple of domain separation, nonce, and counter via the function f_R.

In MMM.Enc (resp. MMM.Dec), (1) AD A is processed by the hash function MMM.Hash, where A is partitioned into n-bit blocks and each data block is injected into the n-bit secret state, (2) a plaintext M (resp. a ciphertext C) is processed by the main function MMM.Main, where M/C is partitioned into b-bit blocks, each data block is injected into the n-bit state and encrypted (resp. decrypted), and (3) a tag is defined by the tag generation function MMM.TagGen. In MMM.Dec, the generated tag T is compared with the inputted tag T'. If $T = T'$ then it returns the plaintext and otherwise **reject**.

In Appendix A, we give a figure of MMM with the conditions $k_1 = n$, $b = n$, and $\tau = 2n$, which MMM-64 satisfies.

3.2 Design of MMM

This section describes our design approach for MMM. Let s be the bit-security goal.

MMM is defined to have the minimum memory size keeping secret values. The minimum size is $s + b$ bits, where the s-bit and b-bit secret memories are respectively required to ensure s-bit security and to encrypt a b-bit plaintext block. Our mode is designed to iterate a BC sequentially to achieve the minimum memory size. In particular, it uses no memory other than that required for BC computations.

In MMM, the s-bit secret state is initialized by the secret key and then updated by iterating the BC. The BC's block size n must be less than or equal to MMM's key size k $(= s)$ as the state of the BC's round function is secret. If $k_1 (= k - n) > 0$, then each BC call takes the remaining k_1-bit secret as a part of the key element. For preventing an attack using a collision of the secret state, the k_1-bit secret state is updated by using the BC's output. Hence, for each BC call, MMM updates the k-bit state.

In the hash function MMM.Hash that processes AD blocks, no state is outputted, and the whole of BC's output block can be used for processing AD blocks A_i. Thus the length of AD blocks is n bits. In the main procedure MMM.Main that processes plaintext/ciphertext blocks, a b-bit value of BC's output is used to encrypt/decrypt a plaintext/ciphertext block M_i/C_i. To keep the whole s-bit state secret, the b-bit value is hidden by XORing with a b-bit part of the secret key K_3. Similarly, in the tag-generation procedure MMM.TagGen, each tag block T_i is defined by XORing the b-bit part of BC's output with K_3. Hence, K_3 is kept in these procedures, and the total secret memory size of MMM is $s + b$ bits.

In MMM.Main, each BC's output is updated by using the state-update function g. If the function is not used, i.e., g is an identity map, then for each BC call, the b-bit part of BC's input block is $C_i \oplus K_3$ and is unaffected by the previous BC's output. As ciphertext blocks are under the adversary's control, one cannot rely on the b-bit part for ensuring the authenticity of MMM, reducing the security level to $s - b$ bits. On the other hand, the function g ensures that for each BC call, the BC's input block is affected by the previous (randomized) BC's output, ensuring the s-bit security regarding authenticity.

Finally, in MMM, each BC call takes a tuple of domain separation, nonce, and counter. The tuple ensures that BC's key elements defined in the encryption of MMM are all distinct. Thus, one can ensure that ciphertext and tag blocks defined in the encryption are all independently drawn and the s-bit privacy is ensured.

3.3 mu-AE Security of MMM in the z-Bound Model

We first define the z-bound model [HTT18].

Definition 1 (z-bound model)**.** Let $L^{(\nu)}$ be the set of the least significant $(t-k_1)$-bit parts of key element of BC defined in encryption queries to the ν-th user. For $V \in \{0,1\}^{t-k_1}$, if $V \in L^{(\nu)}$ then $L^{(\nu)}[V] := 1$; otherwise $L^{(\nu)}[V] := 0$. We say that an adversary is z-repeating if $\forall V \in \{0,1\}^{t-k_1} : \sum_{\alpha \in [\nu]} L^{(\nu)}[V] \leq z$. This means that the same $(t-k_1)$-bit key element of BC is repeated at most z times in encryption queries to different users. We stress that the bound only holds for encryption queries and there is no such restriction on decryption queries. The single-user setting corresponds to $z = 1$. ■

We then show the mu-AE-security of MMM in the z-bound and ideal cipher models.

Theorem 1 (mu-AE-Security of MMM)**.** *For any nonce-respecting and z-repeating adversary* $\mathbf{A}$*, we have*

$$\mathbf{Adv}^{\mathsf{mu\text{-}ae}}_{\mathsf{MMM}}(\mathbf{A}) \leq \frac{4z\sigma_{\mathcal{E}}}{2^k} + \frac{4(z+k)\sigma_{\mathcal{D}}}{2^\tau} + \frac{10w^2zp}{2^k} \ ,$$

where $\sigma_{\mathcal{E}}$ is the number of IC calls in encryption queries, $\sigma_{\mathcal{D}}$ is the number of IC calls in decryption queries, p is the number of offline queries, and w is be the block size of tags. ■

The bound shows that when $\tau = k$, k and z are negligible in the bound,[5] the bound shows that MMM is $\mathsf{mu\text{-}AE}$-secure up to 2^k query complexity, meaning the security is ensured as long as $\sigma_{\mathcal{E}} \ll 2^k$, $\sigma_{\mathcal{D}} \ll 2^k$, and $p \ll 2^k$.

3.4 Overview of Proof of Theorem 1

In this proof, we consider the following bad events.

- bad_1: a collision occurs in IC's inputs defined by encryption queries.
- bad_2: a collision occurs between IC's inputs in encryption and decryption queries.
- bad_3: a collision occurs between offline query-response tuples and IC's input-output tuples defined by encryption queries.
- bad_4: a tag is forged at some decryption query.

We first assume that these bad events do not occur. Regarding the privacy of $\mathsf{MMM.Enc}$, i.e., the indistinguishability between the encryption oracles in the real and ideal worlds, bad_1 ensures that IC's outputs in encryption queries are independently defined. bad_3 ensures that the outputs are defined independently of offline query-response tuples. Hence, b-bit output blocks of $\mathsf{MMM.Enc}$ can be seen as random values. Consequently, the encryption oracles in the real and ideal worlds are indistinguishable. Regarding the authenticity of $\mathsf{MMM.Dec}$, i.e., the indistinguishability between the decryption oracles in the real and ideal worlds, bad_4 ensures that all responses from $\mathsf{MMM.Dec}$ are **reject**, and thus the decryption oracles are indistinguishable. Hence, the real-world and ideal-world oracles are indistinguishable. Note that the bad_2 is not used in the above evaluation but is used for evaluating $\Pr[\mathsf{bad}_4]$. Therefore, the $\mathsf{mu\text{-}AE}$-security advantage of MMM is bounded by the sum of the probabilities for the bad events.

Regarding bad_1, bad_2, and bad_3, these events consider collisions with IC's inputs defined in online queries. Each IC's input is defined by the previous state in MMM, which is randomized by the previous IC's outputs and the user's key. Hence, the k-bit part of each IC's input is a random value. Using the randomness and the parameter z, we obtain the upper-bounds $\Pr[\mathsf{bad}_1] \le O(z\sigma_{\mathcal{E}}/2^k)$, $\Pr[\mathsf{bad}_2] \le O(z\sigma_{\mathcal{E}}/2^k)$, and $\mathsf{bad}_3 \le O(zp/2^k)$.

Regarding bad_4, assuming bad_2 does not occur, an adversary cannot use outputs from the encryption oracles. Then, for each decryption query, we have only to consider the two cases: (1) a tag is defined independently of offline query-response tuples; (2) a tag depends on some offline query-response tuples. In case (1), each tag can be seen as an (almost) τ-bit random value, and the probability of forging some tag is $O(q_{\mathcal{D}}/2^\tau)$ where $q_{\mathcal{D}}$ is the number of decryption queries. In case (2), fixing a decryption query, an adversary recovered the tag before the decryption query, and the probability of forging the tag is bounded by that of recovering the tag by offline queries. The probability can be upper-bounded by using a multi-collision technique for tag candidates obtained from offline query-response tuples. Using the technique, the probability of forging a tag in some decryption query is bounded by $O(k\sigma_{\mathcal{D}}/2^\tau)$.

Combining these bounds, we obtain the upper-bound of the advantage function $O(z\sigma_{\mathcal{E}}/2^k + k\sigma_{\mathcal{D}}/2^\tau + zp/2^k)$.

4 Proof of Theorem 1

Without loss of generality, we assume that an adversary is deterministic, makes no repeated online query to the same user, and makes no repeated offline query.

[5]If the whole part of $N\|ctr$ is randomized such as $f_R(ds, N, ctr) := ds\|((N\|ctr) \oplus R)$ and $|N\|ctr|$ is about k bits, we can have $z = O(k/\log_2 k)$ by using the multi-collision analysis such as [JLM+19].

4.1 Deriving the Upper-Bound using Coefficient H Technique

Our proof uses the coefficient H technique [Pat08]. In this technique, we can analyze bad events in the ideal world, and thus make proofs simple. Let T_R be a transcript in the real world obtained by random samples of user's keys and an IC. Let T_I be a transcript in the ideal world obtained by random samples of random-bit oracles, an IC, and additional values defined later. For a transcript τ, an adversary's view (or information obtained) in the security game, we call τ a valid transcript if $\Pr[\mathsf{T}_I = \tau] > 0$. Let $\mathbf{T}$ be the set of all valid transcripts such that $\forall \tau \in \mathbf{T} : \Pr[\mathsf{T}_R = \tau] \leq \Pr[\mathsf{T}_I = \tau]$. Then, the advantage function $\mathbf{Adv}^{\mathsf{mu\text{-}ae}}_{\mathsf{MMM}}(\mathbf{A})$ is upper-bounded by the statistical distance $\mathsf{SD}(\mathsf{T}_R, \mathsf{T}_I)$ as follows.

$$\mathbf{Adv}^{\mathsf{mu\text{-}ae}}_{\mathsf{MMM}}(\mathbf{A}) \leq \mathsf{SD}(\mathsf{T}_R, \mathsf{T}_I) := \sum_{\tau \in \mathbf{T}} (\Pr[\mathsf{T}_I = \tau] - \Pr[\mathsf{T}_R = \tau]) \ .$$

Using the following lemma, we can derive the upper-bound of $\mathbf{Adv}^{\mathsf{mu\text{-}ae}}_{\mathsf{MMM}}(\mathbf{A})$.

Lemma 1. *Let $\mathbf{T}_{\mathsf{good}}$ and $\mathbf{T}_{\mathsf{bad}}$ be respectively the sets of good transcripts and of bad ones into which $\mathbf{T}$ is partitioned. For good transcripts $\mathbf{T}_{\mathsf{good}}$ and bad transcripts $\mathbf{T}_{\mathsf{bad}}$, if $\forall \tau \in \mathbf{T}_{\mathsf{good}} : \frac{\Pr[\mathsf{T}_R = \tau]}{\Pr[\mathsf{T}_I = \tau]} \geq 1 - \varepsilon$ s.t. $0 \leq \varepsilon \leq 1$, then $\mathsf{SD}(\mathsf{T}_R, \mathsf{T}_I) \leq \Pr[\mathsf{T}_I \in \mathbf{T}_{\mathsf{bad}}] + \varepsilon$.* ■

In the following proof, we (1) partition $\mathbf{T}$ into $\mathbf{T}_{\mathsf{good}}$ and $\mathbf{T}_{\mathsf{bad}}$ (Sect. 4.7); (2) upper-bound $\Pr[\mathsf{T}_I \in \mathbf{T}_{\mathsf{bad}}]$ (Eq. (1) in Sect. 4.8); and (3) lower-bound $\frac{\Pr[\mathsf{T}_R = \tau]}{\Pr[\mathsf{T}_I = \tau]}$ for any $\tau \in \mathbf{T}_{\mathsf{good}}$ (Eq. (8) in Sect. 4.9). We finally obtain the $\mathsf{mu\text{-}AE}$-security bound in Theorem 1.

4.2 Lazy Sampling for IC.

In this proof, an IC's encryption E and decryption E^{-1} are realized by lazy sampling. Let E be a table that is initially empty and keeps query-response tuples of E or E^{-1}. Let $\mathsf{E}_{\mathsf{c}}[Z] := \{Y \mid (Z, X, Y) \in \mathsf{E}\}$ and $\mathsf{E}_{\mathsf{p}}[Z] := \{X \mid (Z, X, Y) \in \mathsf{E}\}$ be tables that respectively keep ciphertext (resp. plaintext) elements in tuples with the key element Z defined in E. For a forward query (Z, X) to E (resp. inverse query (Z, Y) to E^{-1}), the response Y (resp. X) is defined as $Y \xleftarrow{\$} \{0,1\}^n \backslash \mathsf{E}_{\mathsf{c}}[Z]$ (resp. $Y \xleftarrow{\$} \{0,1\}^n \backslash \mathsf{E}_{\mathsf{p}}[Z]$), and the query-response tuple (Z, X, Y) is added to E: $\mathsf{E} \xleftarrow{\cup} \{(Z, X, Y)\}$.

4.3 Definition

Let u be the maximum number of users. Let q, $q_{\mathcal{E}}$, and $q_{\mathcal{D}}$ be the total number of online, forward, and inverse queries, respectively. The 1st through $q_{\mathcal{E}}$-th (resp. $(q_{\mathcal{E}}+1)$-th through q-th) online queries are assigned to encryption (resp. decryption). Hence, α-th encryption (resp. decryption) query is said to be the α-th (resp. $(q_{\mathcal{E}} + \alpha)$-th) online query. Note that this assignment defines just online-query numbers, and does not restrict the order of adversary's queries.

For $\alpha \in [q]$, values defined at the α-th online query are denoted by using the superscript of (α), and the lengths a and m at the α-th online query are denoted by a_α and m_α. Let $\ell_\alpha := a_\alpha + m_\alpha + w - 1$ be the number of IC calls in the α-th online query. Let ν_α be the user number of the α-th online query. Note that $K^{(\alpha)} = K_{\nu_\alpha}$. Let $(W_i^{(\alpha)}, U_i^{(\alpha)}, V_i^{(\alpha)})$ be an input-output tuple of an i-th IC call defined in an α-th online query.

For $\beta \in [p]$, the β-th offline query-response tuple is denoted by $(Z^{(\beta)}, X^{(\beta)}, Y^{(\beta)})$, where $Y^{(\beta)} = E(Z^{(\beta)}, X^{(\beta)})$ for a forward query and $X^{(\beta)} = E^{-1}(Z^{(\beta)}, Y^{(\beta)})$ for an inverse query. Let $\mathcal{Q}_{\mathsf{off}}$ be the set of offline query-response tuples.

Hereafter, We call a period from the start of the game to the end of $\mathbf{A}$'s queries "query phase", and a phase after finishing $\mathbf{A}$'s queries "decision phase". In the decision phase, dummy internal values are defined in the ideal world (defined below).

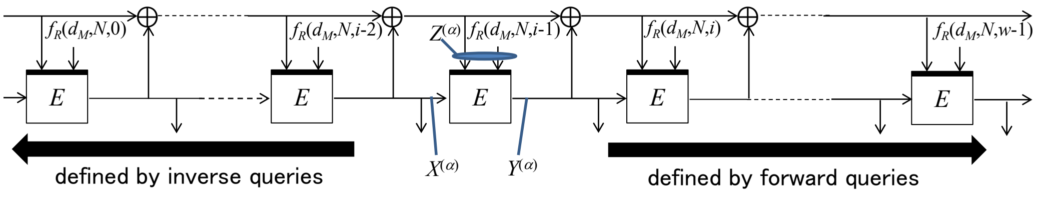

Figure 4: Tag-search queries. $(Z^{(\alpha)}, X^{(\alpha)}, Y^{(\alpha)})$ is the input-output tuple at the i-th IC call. The input-output tuples from the $(i-1)$-th down to 1st (resp. from $(i+1)$-th to w-th) IC calls are defined by inverse (resp. forward) queries. Note that if $i = 1$ (resp. $i = w$) then the input-output tuples are defined by only forward (resp. inverse) queries.

4.4 Additional Queries

In this proof, we allow $\mathbf{A}$ to obtain the following offline query-response tuples in addition to the standard offline query-response ones.

- **Full-block queries.** To ensure the randomnesses of the outputs of E or E^{-1}, we use the technique given in [AFK+11]. In the query stage, for a key element Z of E, after making s offline queries with Z, we permit $\mathbf{A}$ to obtain the remaining input-output tuples of E with Z, i.e., $\mathbf{A}$ obtains all input-output tuples with Z. The additional queries, which we call full-block queries, ensure that the outputs of E or E^{-1} are chosen uniformly at random from roughly $2^n - s$ elements in $\{0,1\}^n$. Specifically, fixing Y^*, for a full-block query (Z, X), the probability that the output Y is equal to Y^* is $(2^n - s - 1)!/(2^n - s)! = 1/(2^n - s)$. Without loss of generality, the full-block queries are forward ones. We choose the parameter s such that $(s+1)/2^n = 1/(w+1)$ $(\Leftrightarrow s = 2^n/(w+1) - 1)$.
- **Tag-search queries.** In the query stage, we permit $\mathbf{A}$ to obtain offline query-response tuples with the structure of $\mathsf{MMM.TagGen}$. After an offline query-response tuple $(Z^{(\alpha)}, X^{(\alpha)}, Y^{(\alpha)})$ is defined, if for some $d_M \in [3,4]$ and $i \in [w]$, $\mathsf{lsb}_{t-k_1}(Z^{(\alpha)}) = f_R(d_M, N, i-1)$, then it is regarded as the i-th input-output block of $\mathsf{MMM.TagGen}$ and $\mathbf{A}$ obtains the remaining $w-1$ input-output blocks defined by forward or inverse queries. See Fig. 4. The tag-search queries are useful for evaluating the probability of forging a tag. By the tag-search queries, for each decryption query, we have only to consider two cases that all input-output tuples that defines the tag are in $\mathcal{Q}_{\mathsf{off}}$ or not.

Let p_{fb} be the number of offline queries including the full-block ones, and p_{add} the total number of offline query-response pairs including the full-block and tag-search queries. Then, we have $p_{\mathsf{fb}} \leq (w+1)p$ and $p_{\mathsf{add}} \leq w(w+1)p$.

4.5 Revealing User's Keys and Internal Values

In the decision phase, the proof permits $\mathbf{A}$ to obtain all user's keys and all BC's outputs defined by online queries. In the ideal world, these values are defined by Algorithm 6. In the algorithm, firstly, dummy user's keys $K_1, \ldots, K_u$ are defined. Secondly, for each $\alpha \in [q_{\mathcal{E}}]$, dummy internal input-output tuples for the α-th online (or encryption) query are defined so that the tuples are consistent with the ciphertext and tag. Dummy input-output tuples corresponding with $A^{(\alpha)}$ are defined by performing $\mathsf{MMM.Hash}^E_{K^{(\alpha)}}(N^{(\alpha)}, A^{(\alpha)})$. Then input-output tuples corresponding with $C^{(\alpha)}$ are defined in Steps 9-14. In these steps, the most significant b-bit part of $(a_\alpha + i - 1)$-output block $V^{(\alpha)}_{a_\alpha+i-1}$ is defined by using $K^{(\alpha)}_3$, $M^{(\alpha)}_i$, and $C^{(\alpha)}_i$ according to the structure of $\mathsf{MMM.Main}$ and the remaining $n-b$-bit

Algorithm 6 Dummy Internal Values in the Ideal World

Input:

- Online query-response tuples $(N^{(\alpha)}, A^{(\alpha)}, M^{(\alpha)}, C^{(\alpha)}, T^{(\alpha)})$ where $\alpha \in [q_{\mathcal{E}}]$
- Online query-response tuples $(N^{(\alpha)}, A^{(\alpha)}, C^{(\alpha)}, T'^{(\alpha)}, RV^{(\alpha)})$ where $\alpha \in [q_{\mathcal{E}}+1, q]$ and $RV^{(\alpha)} \in \{\textbf{reject}, M^{(\alpha)}\}$

Output:

- Dummy user's keys K_i where $i \in [u]$
- Dummy input-output tuples of IC in online queries $(W_i^{(\alpha)}, U_i^{(\alpha)}, V_i^{(\alpha)})$ where $\alpha \in [q], i \in [\ell_\alpha]$

```
1: // Dummy keys
2: for $i \in [u]$ do $K_i \xleftarrow{\$} \{0,1\}^k$
3: // Dummy internal values of MMM.Enc
4: for $\alpha \in [q_{\mathcal{E}}]$ do
5:     $(S_1, S_2) \leftarrow \mathsf{MMM.Hash}^E_{K^{(\alpha)}}(N^{(\alpha)}, A^{(\alpha)})$
6:     for $i \in [a_\alpha - 1]$ do
7:         $(W_i^{(\alpha)}, U_i^{(\alpha)}, V_i^{(\alpha)}) \leftarrow$ the input-output tuple at the $i$-th IC call of
                   $\mathsf{MMM.Hash}^E_{K^{(\alpha)}}(N^{(\alpha)}, A^{(\alpha)})$
8:     end for
9:     for $i \in [m_\alpha]$ do
10:        $U^{(\alpha)}_{a_\alpha+i-1} \leftarrow S_2$; $W^{(\alpha)}_{a_\alpha+i-1} \leftarrow S_1 \| f_R(d_A^{(\alpha)}, N^{(\alpha)}, i-1)$
11:        $Tmp \xleftarrow{\$} \{0,1\}^{n-|M_i^{(\alpha)}|}$; $S_2 \leftarrow V^{(\alpha)}_{a_\alpha+i-1} \leftarrow \left(K_3^{(\alpha)} \oplus_{|M_i^{(\alpha)}|} M_i^{(\alpha)} \oplus C_i^{(\alpha)}\right) \| Tmp$
12:        $\mathsf{E} \xleftarrow{\cup} \{(W^{(\alpha)}_{a_\alpha+i-1}, U^{(\alpha)}_{a_\alpha+i-1}, V^{(\alpha)}_{a_\alpha+i-1})\}$
13:        $S_1 \leftarrow S_1 \oplus_{k_1} S_2$; $S_2 \leftarrow g(S_2) \oplus \mathsf{zp}_n \circ \mathsf{ozp}_b(M_i^{(\alpha)})$
14:    end for
15:    for $i \in [w]$ do
16:        $U^{(\alpha)}_{a_\alpha+m_\alpha+i-1} \leftarrow S_2$; $W^{(\alpha)}_{a_\alpha+m_\alpha+i-1} \leftarrow S_1 \| f_R(d_M^{(\alpha)}, N^{(\alpha)}, i-1)$
17:        $Tmp \xleftarrow{\$} \{0,1\}^{n-b}$; $S_2 \leftarrow V^{(\alpha)}_{a_\alpha+m_\alpha+i-1} \leftarrow \left(K_3^{(\alpha)} \oplus_{|T_i^{(\alpha)}|} T_i^{(\alpha)}\right) \| Tmp$
18:        $\mathsf{E} \xleftarrow{\cup} \{(W^{(\alpha)}_{a_\alpha+m_\alpha+i-1}, U^{(\alpha)}_{a_\alpha+m_\alpha+i-1}, V^{(\alpha)}_{a_\alpha+m_\alpha+i-1})\}$; $S_1 \leftarrow S_1 \oplus_{k_1} S_2$
19:    end for
20: end for
21: // Dummy internal values of MMM.Dec
22: for $\alpha \in [q_{\mathcal{E}}+1, q]$ do
23:     for $i \in [\ell_\alpha]$ do
24:         $(W_i^{(\alpha)}, U_i^{(\alpha)}, V_i^{(\alpha)}) \leftarrow$ the input-output tuple at the $i$-th IC call
                   of $\mathsf{MMM.Dec}^E_{K^{(\alpha)}}(N^{(\alpha)}, A^{(\alpha)}, C^{(\alpha)}, T'^{(\alpha)})$
25:     end for
26: end for
27: return $(K_1, \ldots, K_u, (W_1^{(1)}, U_1^{(1)}, V_1^{(1)}), \ldots, (W_{\ell_q}^{(q)}, U_{\ell_q}^{(q)}, V_{\ell_q}^{(q)}))$
```

value is randomly chosen, and the obtained tuple is added to the IC's table E in Step 12. Thirdly, input-output tuples corresponding with $T^{(\alpha)}$ are defined in Steps 15-19 according to the structure of $\mathsf{MMM.TagGen}$. Finally, dummy input-output values for decryption queries are defined by performing $\mathsf{MMM.Dec}^{E}_{K^{(\alpha)}}(N^{(\alpha)}, A^{(\alpha)}, C^{(\alpha)}, T'^{(\alpha)})$ in Steps 22-26.

4.6 Adversary's View

The adversary's view is summarized in a transcript τ, which consists of

- user's keys K_i where $i \in [u]$,
- online query-response tuples $(N^{(\alpha)}, A^{(\alpha)}, M^{(\alpha)}, C^{(\alpha)}, T^{(\alpha)})$ where $\alpha \in [q_{\mathcal{E}}]$,
- online query-response tuples $(N^{(\alpha)}, A^{(\alpha)}, C^{(\alpha)}, T'^{(\alpha)}, RV^{(\alpha)})$ where $\alpha \in [q_{\mathcal{E}} + 1, q]$ and $RV^{(\alpha)} \in \{\mathbf{reject}, M^{(\alpha)}\}$ is the response to the α-th online query,
- (dummy) input-output tuples of IC in online queries $(W_i^{(\alpha)}, U_i^{(\alpha)}, V_i^{(\alpha)})$ where $\alpha \in [q], i \in [\ell_\alpha]$, and
- offline query-response tuples $(Z^{(\alpha)}, X^{(\alpha)}, Y^{(\alpha)})$ where $\alpha \in [p_{\mathsf{add}}]$.

4.7 Good and Bad Transcripts

We define the following four bad events. bad_1 is that a collision in IC's inputs defined by encryption queries occurs. bad_2 is that a collision between IC's inputs in encryption and decryption queries occurs. bad_3 is that a collision between primitive and encryption queries occurs.[6] bad_4 is that a tag is forged at some decryption query. The formal definitions of the bad events are given below.

- bad_1: $\exists \alpha, \beta \in [q_{\mathcal{E}}], i \in [\ell_\alpha], j \in [\ell_\beta]$ s.t. $(\alpha, i) \neq (\beta, j) \wedge \Big((W_i^{(\alpha)}, U_i^{(\alpha)}) = (W_j^{(\beta)}, U_j^{(\beta)}) \vee (W_i^{(\alpha)}, V_i^{(\alpha)}) = (W_j^{(\beta)}, V_j^{(\beta)}) \Big)$.
- bad_2: $\exists \alpha \in [q_{\mathcal{E}}], \beta \in [q_{\mathcal{E}} + 1, q], i \in [\ell_\alpha], j \in [\ell_\beta]$ s.t. $(\alpha, i) \neq (\beta, j) \wedge (W_i^{(\alpha)}, U_i^{(\alpha)}) \neq (W_j^{(\beta)}, U_j^{(\beta)}) \wedge (W_{i+1}^{(\alpha)}, U_{i+1}^{(\alpha)}) = (W_{j+1}^{(\beta)}, U_{j+1}^{(\beta)})$.
- bad_3: $\exists \alpha \in [q_{\mathcal{E}}], \beta \in [p], i \in [\ell_\alpha]$ s.t. $(W_i^{(\alpha)}, U_i^{(\alpha)}) = (Z^{(\beta)}, X^{(\beta)}) \vee (W_i^{(\alpha)}, V_i^{(\alpha)}) = (Z^{(\beta)}, Y^{(\beta)})$.
- bad_4: $\exists \alpha \in [q_{\mathcal{D}} + 1, q]$ s.t. $T^{(\alpha)} = T'^{(\alpha)}$.

Let $\mathsf{bad} := \mathsf{bad}_1 \vee \mathsf{bad}_2 \vee \mathsf{bad}_3 \vee \mathsf{bad}_4$. We define bad transcripts $\mathcal{T}_{\mathsf{bad}}$ that satisfy one of the above events and good ones by $\mathcal{T}_{\mathsf{bad}} := \mathcal{T} \backslash \mathcal{T}_{\mathsf{bad}}$.

4.8 Analysis for Bad Transcripts

Without loss of generality, assume that an adversary aborts if one of the bad events occurs. Hence, for $i \in [4]$, $\Pr[\mathsf{bad}_i]$ is the probability that bad_i occurs as long as the other events have not occurred. Then, $\Pr[\mathsf{T}_I \in \mathbf{T}_{\mathsf{bad}}]$ is upper-bounded as follows.

$$\Pr[\mathsf{T}_I \in \mathbf{T}_{\mathsf{bad}}] \leq \Pr[\mathsf{bad}_1] + \Pr[\mathsf{bad}_2] + \Pr[\mathsf{bad}_3] + \Pr[\mathsf{bad}_4] \ .$$

[6]Note that a collision between decryption and primitive queries is evaluated in an evaluation of bad_4.

These probabilities are upper-bounded by Eqs. (2), (3), (4), and (5), as we will evaluate in the following subsections. Using these bounds, we have

$$\begin{aligned}\Pr[\mathsf{T}_I \in \mathbf{T}_{\mathsf{bad}}] &\leq \frac{4z\sigma_{\mathcal{E}}}{2^k} + \frac{4z\sigma_{\mathcal{D}}}{2^k} + \frac{2w(w+1)zp}{2^k} + \frac{6wp}{2^k} + \frac{2k\sigma_{\mathcal{D}}}{2^\tau} \\ &\leq \frac{4z\sigma_{\mathcal{E}}}{2^k} + \frac{4(z+k)\sigma_{\mathcal{D}}}{2^\tau} + \frac{10w^2zp}{2^k} \quad . \end{aligned} \tag{1}$$

4.8.1 Bounding $\Pr[\mathsf{bad}_1]$

For any $\alpha, \beta \in [q_{\mathcal{E}}]$, $i \in [\ell_\alpha]$, and $j \in [\ell_\beta]$ such that $\nu_\alpha = \nu_\beta$ and $i \neq j$, $W_i^{(\alpha)} \neq W_j^{(\beta)}$ is satisfied due to tuples of domain separation, nonce, and counter. We thus consider $\alpha, \beta \in [q_{\mathcal{E}}]$ such that $\alpha < \beta$ and $\nu_\alpha \neq \nu_\beta$.

Fix $\alpha, \beta \in [q_{\mathcal{E}}]$, $i \in [\ell_\alpha]$, and $j \in [\ell_\beta]$ such that $\alpha < \beta$, $\nu_\alpha \neq \nu_\beta$, and $\mathsf{lsb}_{t-k_1}(W_i^{(\alpha)}) = \mathsf{lsb}_{t-k_1}(W_j^{(\beta)})$. As $\mathsf{msb}_{k_1}(W_i^{(\alpha)})$ and $\mathsf{msb}_{k_1}(W_j^{(\beta)})$ are respectively defined by using $K_1^{(\alpha)}$ and $K_1^{(\beta)}$ which are independent, we have $\Pr[\mathsf{msb}_{k_1}(W_i^{(\alpha)}) = \mathsf{msb}_{k_1}(W_j^{(\beta)})] \leq 1/2^{k_1}$. For some data block $D \in \{0,1\}^n$ $U_j^{(\beta)} = g(V_{j-1}^{(\beta)}) \oplus D$ is satisfied, and by the randomness of $V_{i-1}^{(\beta)}$ that is chosen from at least $2^n - s$ elements in $\{0,1\}^n$, we have $\Pr[U_i^{(\alpha)} = U_j^{(\beta)}] \leq 1/(2^n - s) \leq 2/2^n$. Similarly, we have $\Pr[V_i^{(\alpha)} = V_j^{(\beta)}] \leq 1/(2^n - s) \leq 2/2^n$. In the z-bound model, the number of pairs $(\beta, j) \in [q_{\mathcal{E}}] \times [\ell_\beta]$ such that $\nu_\alpha \neq \nu_\beta$ and $\mathsf{lsb}_{t-k_1}(W_i^{(\alpha)}) = \mathsf{lsb}_{t-k_1}(W_j^{(\beta)})$ is at most z. We thus have

$$\Pr[\mathsf{bad}_1] \leq \sum_{\alpha \in [q_{\mathcal{E}}]} \sum_{i \in [\ell_\alpha]} \frac{4z}{2^{n+k_1}} = \frac{4z\sigma_{\mathcal{E}}}{2^k} \quad . \tag{2}$$

4.8.2 Bounding $\Pr[\mathsf{bad}_2]$

Fix $\alpha \in [q_{\mathcal{E}}], \beta \in [q_{\mathcal{E}} + 1, q], i \in [\ell_\alpha], j \in [\ell_\beta]$ s.t. $(\alpha, i) \neq (\beta, j) \wedge (W_i^{(\alpha)}, U_i^{(\alpha)}) \neq (W_j^{(\beta)}, U_j^{(\beta)})$. Then we evaluate the collision probability $\Pr[(W_{i+1}^{(\alpha)}, U_{i+1}^{(\alpha)}) = (W_{j+1}^{(\beta)}, U_{j+1}^{(\beta)})]$. $(W_{i+1}^{(\alpha)}, U_{i+1}^{(\alpha)})$ and $(W_{j+1}^{(\beta)}, U_{j+1}^{(\beta)})$ are defined by using the previous output blocks. In the β-th online query, the previous output blocks of $(W_{j+1}^{(\beta)}, U_{j+1}^{(\beta)})$ are chosen uniformly at random from at least $2^n - s - 1$ elements in $\{0,1\}^n$. Using the randomness, we have

$$\begin{aligned}\Pr[U_{i+1}^{(\alpha)} = U_{j+1}^{(\beta)}] &\leq \frac{1}{2^n - s - 1} \leq \frac{1}{2^n(1 - (s+1)/2^n} \leq \frac{1}{2^n(1 - 1/(w+1))} \leq \frac{2}{2^n} \quad , \\ \Pr[W_{i+1}^{(\alpha)} = W_{j+1}^{(\beta)}] &\leq \frac{2^{n-k_1}}{2^n - s - 1} \leq \frac{2 \cdot 2^{n-k_1}}{2^n} = \frac{2}{2^{k_1}} \quad . \end{aligned}$$

Hence, we have

$$\Pr[(W_{i+1}^{(\alpha)}, U_{i+1}^{(\alpha)}) = (W_{j+1}^{(\beta)}, U_{j+1}^{(\beta)})] \leq \frac{2}{2^n} \cdot \frac{2}{2^{k_1}} = \frac{4}{2^k} \quad .$$

Since $\mathbf{A}$ is a nonce-respecting and z-repeating adversary, for each $\beta \in [q_{\mathcal{E}} + 1, q]$, the number of such $\alpha \in [q_{\mathcal{E}}]$ is at most z. Using the bound, we have

$$\Pr[\mathsf{bad}_2] \leq \sum_{\alpha \in [q_{\mathcal{E}}+1, q]} \sum_{i \in [\ell_\alpha]} \frac{4z}{2^k} \leq \frac{4z\sigma_{\mathcal{D}}}{2^k} \quad . \tag{3}$$

4.8.3 Bounding $\Pr[\mathsf{bad}_3]$

Fix $\alpha \in [q_{\mathcal{E}}]$, $i \in [\ell_\alpha]$ and $\beta \in [p_{\mathsf{add}}]$ such that $\mathsf{lsb}_{t-k_1}(W_i^{(\alpha)}) = \mathsf{lsb}_{t-k_1}(Z^{(\beta)})$, and evaluate the collision probability $\Pr[(W_i^{(\alpha)}, U_i^{(\alpha)}) = (Z^{(\beta)}, X^{(\beta)}) \vee (W_i^{(\alpha)}, V_i^{(\alpha)}) = (Z^{(\beta)}, Y^{(\beta)})]$. Then we have $\Pr[\mathsf{msb}_{k_1}(W_i^{(\alpha)}) = \mathsf{msb}_{k_1}(Z^{(\beta)})] \leq 1/2^{k_1}$ as $\mathsf{msb}_{k_1}(W_i^{(\alpha)})$ is defined by using $K_1^{(\alpha)}$; $\Pr[U_i^{(\alpha)} = X^{(\beta)}] \leq 1/2^n$ by using the randomness of $V_{i-1}^{(\alpha)}$ (chosen from $\{0,1\}^n$) and the relation $U_i^{(\alpha)} = g(V_{i-1}^{(\alpha)}) \oplus D$ for the i-th data block D; and $\Pr[V_i^{(\alpha)} = Y^{(\beta)}] \leq 1/2^n$ by using the randomness of $V_i^{(\alpha)}$ (chosen from $\{0,1\}^n$). In the z-bound model, for each $\beta \in [p_{\mathsf{add}}]$, the number of pairs (α, i) such that $\mathsf{lsb}_{t-k_1}(W_i^{(\alpha)}) = \mathsf{lsb}_{t-k_1}(Z^{(\beta)})$ is at most z. Hence we have

$$\Pr[\mathsf{bad}_3] \leq \sum_{\beta \in [p_{\mathsf{add}}]} \frac{2z}{2^{n+k_1}} = \frac{2zp_{\mathsf{add}}}{2^k} \leq \frac{2w(w+1)zp}{2^k} \ . \qquad (4)$$

4.8.4 Bounding $\Pr[\mathsf{bad}_4]$

We first give the following definition. The definition considers sequences of offline-query response tuples relevant to $\mathsf{MMM.TagGen}$. The definition is useful for evaluating the probability of forging a tag. By tag-search queries, for each decryption query, we have only to consider the two cases that a sequence of input-output tuples that defines the tag are in $\mathcal{Q}_{\mathsf{off}}$ or not. The definition is used to evaluate the former case.

Definition 2 (Tag-candidate sequence)**.** $\{(Z_1, X_1, Y_1), \dots, (Z_w, X_w, Y_w)\} \in (\mathcal{Q}_{\mathsf{off}})^w$ is a tag-candidate sequence if for some $K^* \in \{0,1\}^k$, $S_1^* \in \{0,1\}^{k_1}$, $S_2^* \in \{0,1\}^n$, and $N^* \in \mathcal{N}$, these tuples are internal input-output tuples of E in $\mathsf{MMM.TagGen}_{K^*}^E(N^*, S_1^*, S_2^*)$, i.e.,

- $\exists N^* \in \mathcal{N}, d_M \in [3,4]$ s.t. $\forall i \in [w]$:
  $\mathsf{msb}_{k_1}(Z_i) = \mathsf{msb}_{k_1}(Z_1) \oplus_{k_1} \left(\bigoplus_{j \in [i-1]} Y_j\right) \wedge \mathsf{lsb}_{t-k_1}(Z_i) = f_R(d_M, N^*, i-1)$, and
- $\forall i \in [w-1] : X_{i+1} = Y_i$.

Let $\mathcal{S}_{\mathsf{TagCan}} \subseteq (\mathcal{Q}_{\mathsf{off}})^w$ be the set of all tag-candidate sequences. Let $\phi : \mathcal{S}_{\mathsf{TagCan}} \to (\{0,1\}^b)^w$ be a function that for an input $\mathcal{T} = \{(Z_1, X_1, Y_1), \dots, (Z_w, X_w, Y_w)\} \in \mathcal{S}_{\mathsf{TagCan}}$ returns $(\mathsf{msb}_b(Y_1), \dots, \mathsf{msb}_b(Y_w))$. Let $[\phi(\mathcal{T})]_i := \mathsf{msb}_b(Y_i)$ be the i-th element of $\phi(\mathcal{T})$. ■

Note that by tag-search queries, we have $|\mathcal{S}_{\mathsf{TagCan}}| \leq p_{\mathsf{fb}} \leq (w+1)p$. In this evaluation, we use the following μ-multi-collision event for the function ϕ such that $\mu := 2^{k-(\tau-b)}k$.

- $\mathsf{mcoll} : \exists T_1^*, \dots, T_w^* \in \{0,1\}^b, \mathcal{T}_1, \dots, \mathcal{T}_\mu \in \mathcal{S}_{\mathsf{TagCan}}$ s.t.
  $\forall i \in [\mu] : [\phi(\mathcal{T}_i)]_1 \oplus T_1^* = \cdots = [\phi(\mathcal{T}_i)]_w \oplus T_w^*$.

Note that for a decryption query, if all input-output tuples that define the tag are in $\mathcal{Q}_{\mathsf{off}}$ (Event $\mathsf{E2}$ in the following evaluation), then there exists $\mathcal{T} \in \mathcal{S}_{\mathsf{TagCan}}$ such that $[\phi(\mathcal{T})]_1 \oplus T_1'^{(\alpha)} = \cdots = [\phi(\mathcal{T})]_w \oplus T_w'^{(\alpha)} = K_3^{(\alpha)}$. Assuming mcoll does not occur, the probability that $T^{(\alpha)} = T'^{(\alpha)}$ is satisfied with the event $\mathsf{E2}$ can be upper-bounded by the probability that the internal state connects with one of the μ-multi-collision elements.

Using the multi-collision event, we have

$$\Pr[\mathsf{bad}_4] \leq \Pr[\mathsf{mcoll}] + \Pr[\mathsf{bad}_4 | \neg\mathsf{mcoll}] \ .$$

These probabilities are evaluated in the following and the bounds are given in Eqs. (6) and (7). The bounds offer the following bound.

$$\Pr[\mathsf{bad}_4] \leq \left(\frac{6w^2}{k} \cdot \frac{p}{2^k}\right)^k + \frac{2k\sigma_{\mathcal{D}}}{2^\tau} \ . \qquad (5)$$

Bounding $\Pr[\mathsf{mcoll}]$. We first show the following lemma.

Lemma 2. $\forall T_1^*, \ldots, T_w^* \in \{0,1\}^b, \mathcal{T} \in \mathcal{S}_{\mathsf{TagCan}} : \Pr[[\phi(\mathcal{T})]_1 \oplus T_1^* = \cdots = [\phi(\mathcal{T})]_w \oplus T_w^*] \leq w/2^{\tau-b}$. ■

Proof. Fix $T_1^*, \ldots, T_w^* \in \{0,1\}^b, \mathcal{T} \in \mathcal{S}_{\mathsf{TagCan}}$, and evaluate the probability $\Pr[[\phi(\mathcal{T})]_1 \oplus T_1^* = \cdots = [\phi(\mathcal{T})]_w \oplus T_w^*]$. Let $\mathcal{T} := \{(Z_1, X_1, Y_1), \ldots, (Z_w, X_w, Y_w)\}$. Assume that a j-th input-output tuple (Z_j, X_j, Y_j) is defined before the others in $\mathcal{T}$ are defined. If the query for the j-th input-output block is a forward (resp. inverse) one, then the randomness of X_j $(= Y_{j-1})$ (resp. Y_j) cannot be used but the other outputs in $\mathcal{T}$ can be used, and we thus have $\Pr[\mathsf{msb}_b(Y_i) \oplus T_i^* = \mathsf{msb}_b(Y_{j-1}) \oplus T_{j-1}^*] \leq 2^{n-b}/(2^n - s)$ for each $i \in [w]\backslash\{j-1\}$ (resp. $\Pr[\mathsf{msb}_b(Y_i) \oplus T_i^* = \mathsf{msb}_b(Y_j) \oplus T_j^*] \leq 2^{n-b}/(2^n - s)$ for each $i \in [w]\backslash\{j\}$). Using the bounds, we have

$$\begin{aligned}&\Pr[[\phi(\mathcal{T})]_1 \oplus T_1^* = \cdots = [\phi(\mathcal{T})]_w \oplus T_w^*] \\ &\leq \left(\frac{1}{2^b(1-s/2^n)}\right)^{w-1} \leq \frac{1}{(1-\frac{1}{w+1})^{w-1}} \cdot \frac{1}{2^{(w-1)b}} \leq \frac{1}{1-\frac{w-1}{w+1}} \cdot \frac{1}{2^{\tau-b}} \leq \frac{w}{2^{\tau-b}} \ .\end{aligned}$$

□

Using the above lemma, we have

$$\begin{aligned}\Pr[\mathsf{mcoll}] &\leq \sum_{T_1^*,\ldots,T_w^* \in \{0,1\}^b, \mathcal{T}_1,\ldots,\mathcal{T}_\mu \in \mathcal{S}_{\mathsf{TagCan}}} \Pr[\forall i \in [\mu] : [\phi(\mathcal{T}_i)]_1 \oplus T_1^* = \cdots = [\phi(\mathcal{T}_i)]_w \oplus T_w^*] \\ &\leq 2^\tau \binom{p_{\mathsf{fb}}}{\mu} \left(\frac{w}{2^{\tau-b}}\right)^\mu \leq 2^\tau \cdot \left(\frac{ewp_{\mathsf{fb}}}{\mu 2^{\tau-b}}\right)^\mu \leq \left(\frac{6w^2}{k} \cdot \frac{p}{2^k}\right)^k \leq \frac{6wp}{2^k} \ , \qquad (6)\end{aligned}$$

using Stirling's approximation ($x! \geq (x/e)^x$ for any x).

Bounding $\Pr[\mathsf{bad_4}|\neg\mathsf{mcoll}]$. First fix $\alpha \in [q_\mathcal{E} + 1, q]$ and evaluate $\Pr[T^{(\alpha)} = T'^{(\alpha)}]$. We consider the following two events.

- $\mathsf{E1}$: All input-output tuples that define the tag $T^{(\alpha)}$ are new, that is, $\forall i \in [w] : (W^{(\alpha)}_{\ell_\alpha - i+1}, U^{(\alpha)}_{\ell_\alpha - i+1}, V^{(\alpha)}_{\ell_\alpha - i+1}) \notin \mathcal{Q}_{\mathsf{off}}$.
- $\mathsf{E2}$: All input-output tuples that define the tag $T^{(\alpha)}$ have been defined by offline queries, that is, $\forall i \in [w] : (W^{(\alpha)}_{\ell_\alpha - i+1}, U^{(\alpha)}_{\ell_\alpha - i+1}, V^{(\alpha)}_{\ell_\alpha - i+1}) \in \mathcal{Q}_{\mathsf{off}}$.

By tag-search queries, $\mathsf{E1} = \neg\mathsf{E2}$ is satisfied. Using these events, we have

$$\begin{aligned}\Pr[T^{(\alpha)} = T'^{(\alpha)}|\neg\mathsf{mcoll}] &= \Pr[T^{(\alpha)} = T'^{(\alpha)} \wedge \mathsf{E1}|\neg\mathsf{mcoll}] + \Pr[T^{(\alpha)} = T'^{(\alpha)} \wedge \mathsf{E2}|\neg\mathsf{mcoll}] \\ &\leq \Pr[T^{(\alpha)} = T'^{(\alpha)}|\mathsf{E1} \wedge \neg\mathsf{mcoll}] + \Pr[T^{(\alpha)} = T'^{(\alpha)} \wedge \mathsf{E2}|\neg\mathsf{mcoll}] \ .\end{aligned}$$

These probabilities are evaluated below.

We first evaluate $\Pr[T^{(\alpha)} = T'^{(\alpha)}|\mathsf{E1} \wedge \neg\mathsf{mcoll}]$. In this case, as $\mathsf{bad_2}$ does not occur, each input $(W^{(\alpha)}_{\ell_\alpha - i+1}, U^{(\alpha)}_{\ell_\alpha - i+1})$ is new, and thus we have $\Pr[\mathsf{msb}_b(V^{(\alpha)}_{\ell_\alpha - i+1}) = T^{(\alpha)}_{w-i+1}] \leq \frac{2^{n-b}}{2^n - s} \leq \frac{1}{1-1/(w+1)} \cdot \frac{1}{2^b}$. Using the bound, we have

$$\Pr[T^{(\alpha)} = T'^{(\alpha)}|\mathsf{E1} \wedge \neg\mathsf{mcoll}] \leq \frac{1}{(1-1/(w+1))^w} \cdot \frac{1}{2^{wb}} \leq \frac{1}{1-w/(w+1)} \cdot \frac{1}{2^\tau} = \frac{w+1}{2^\tau} \ .$$

We next evaluate $\Pr[T^{(\alpha)} = T'^{(\alpha)} \wedge \mathsf{E2}|\neg\mathsf{mcoll}]$. For $\gamma \in [\ell_\alpha - w]$, let $\mathsf{E2}[\gamma]$ be an event that the γ-th input for the α-th online query is new but the following input-output tuples (from the $(\gamma+1)$-th to ℓ_α-th blocks) are in $\mathcal{Q}_{\mathsf{off}}$. Fix $\gamma \in [\ell_\alpha - w]$ and evaluate $\Pr[T^{(\alpha)} = T'^{(\alpha)} \wedge \mathsf{E2}[\gamma]]$. By $\neg\mathsf{mcoll}$, there are at most $(\mu - 1)$ tag-generation sequences $\mathcal{T}_1, \ldots, \mathcal{T}_{\mu-1} \in \mathcal{S}_{\mathsf{TagCan}}$ such that $\forall i \in [\mu - 1] : [\phi(\mathcal{T}_i)]_1 \oplus T_1'^{(\alpha)} = \cdots = [\phi(\mathcal{T}_i)]_w \oplus T_w'^{(\alpha)}$. Then $T^{(\alpha)} = T'^{(\alpha)} \wedge \mathsf{E2}[\gamma]$ implies that $\exists j \in [\mu - 1]$ s.t.

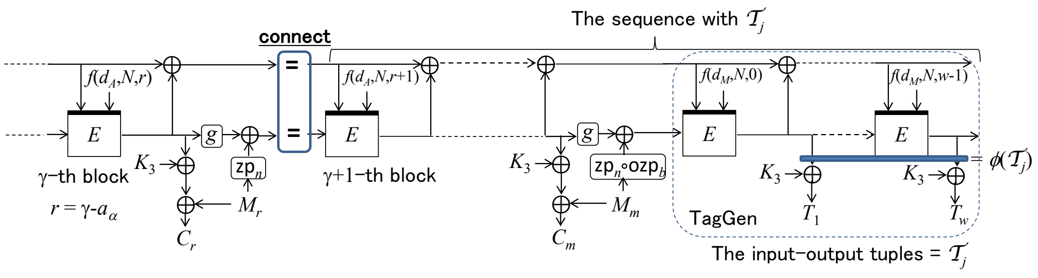

Figure 5: Event $\mathsf{E2}_2[\gamma]$. The event considers the case that the internal state after the γ-th IC call connects with the sequence with $\mathcal{T}_j$, where the γ-th input-output tuple is not in $\mathcal{Q}_{\mathsf{off}}$. Since $\mathcal{T}_j$, the ciphertext blocks, and the tag blocks are fixed such that $\forall i \in [w] : T_i = T_i'^{(\alpha)}$, the sequence with $\mathcal{T}_j$ is uniquely determined.

- $\mathsf{E2}_1[\gamma]$: $[\phi(\mathcal{T}_j)]_1 \oplus T_1'^{(\alpha)} = \cdots = [\phi(\mathcal{T}_j)]_w \oplus T_w'^{(\alpha)} = K_3^{(\alpha)}$ and
- $\mathsf{E2}_2[\gamma]$: the state after the γ-th block connects with the sequence whose the last w input-output tuples are equal to $\mathcal{T}_j$. See Fig. 5.

For $\mathsf{E2}_1[\gamma]$, by the randomness of K_3, we have $\Pr[\mathsf{E2}_1[\gamma]] \le \frac{1}{2^b}$. For $\mathsf{E2}_2[\gamma]$, the least significant n-bit part of the k-bit state after the γ-th IC call is defined by using the γ-th IC's output, and the remaining k_1-bit part is defined by using the user's key $K_1^{(\alpha)}$. Since each IC's output is chosen uniformly at random from at least $2^n - s \le 2^{n-1}$, we have $\Pr[\mathsf{E2}_2[\gamma]] \le \frac{1}{2^{n-1}} \cdot \frac{1}{2^{k_1}} = \frac{2}{2^k}$. Summing the bound $(\mu - 1) \cdot \Pr[\mathsf{E2}_1[\gamma]] \cdot \Pr[\mathsf{E2}_2[\gamma]] \le \frac{2 \cdot 2^{k-(\tau-b)} k}{2^{k+b}} = \frac{2k}{2^\tau}$ for each $\gamma \in [\ell_\alpha - w]$, we have

$$\Pr[T^{(\alpha)} = T'^{(\alpha)} \wedge \mathsf{E2} | \neg \mathsf{mcoll}] \le (\ell_\alpha - w) \cdot \frac{2k}{2^\tau} \ .$$

Finally, summing the above bounds for each α, we obtain the following bound.

$$\Pr[\mathsf{bad}_4 | \neg \mathsf{mcoll}] \le \sum_{\alpha \in [q_{\mathcal{D}}]} \left(\frac{w+1}{2^\tau} + \frac{2k(\ell_\alpha - w)}{2^\tau} \right) \le \sum_{\alpha \in [q_{\mathcal{D}}]} \frac{2k\ell_\alpha}{2^\tau} = \frac{2k\sigma_{\mathcal{D}}}{2^\tau} \ . \tag{7}$$

4.9 Analysis for Good Transcripts

We give an intuition of this analysis. The detail is given in Appendix B. This analysis considers only good transcripts which do not satisfy the bad events. For $\tau \in \mathbf{T}_{\mathsf{good}}$, we evaluate $\frac{\Pr[\mathsf{T}_R = \tau]}{\Pr[\mathsf{T}_I = \tau]}$.

Regarding offline queries, in both worlds, the responses are defined by an IC. Hence, there is no difference between the real and ideal worlds.

Regarding encryption queries, by bad_3 there is no collision between IC's inputs of τ for encryption and offline queries. By bad_1 there is no input collision in IC's inputs of τ defined by encryption queries. The IC's outputs of T_R are defined by an IC and those of T_I are chosen uniformly at random from $\{0,1\}^n$. Hence, the probability that T_R satisfies the IC's output in τ is grater than or equal to the one for T_R.

Regarding decryption queries, by bad_4, the IC's outputs in τ are such that the responses are all **reject**. Hence, we have only consider the IC's outputs for decryption queries. In both worlds, for each decryption query, the outputs are defined by an IC, thus there is no difference between the real and ideal worlds.

Hence, we have

$$\frac{\Pr[\mathsf{T}_R = \tau]}{\Pr[\mathsf{T}_I = \tau]} \ge 1 \ . \tag{8}$$

5 Hardware Implementation

5.1 Instantiation of MMM for Implementation

We instantiate MMM-8 and MMM-64 for 128-bit security to make hardware performance comparison with HOMA. We use the ISO-standard TBC SKINNY [BJK+16, ISO22] as a primitive. In particular, we chose SKINNY-64/192 with a 64-bit block and 192-bit tweakey because HOMA uses another 64-bit SKINNY variant SKINNYee [NSS22]. We refer to SKINNY-64/192's independently scheduled 64-bit tweakey blocks as TK_1, TK_2, and TK_3. We assign the 64-bit secret K_1 to the simplest TK_1. Meanwhile, the public value $f_R(d, N, i) \in \{0,1\}^{128}$ is stored in TK_2 and TK_3 concatenated.

The state-update function $g\colon \{0,1\}^{128} \rightarrow \{0,1\}^{128}$ is defined using the SKINNY's 4-bit LFSR function $f_{\mathrm{LFSR}}\colon x_3||x_2||x_1||x_0 \mapsto x_2||x_1||x_0||x_3 \oplus x_2$ with x_3, x_2, x_1, $x_0 \in \{0,1\}$. g is its parallel application, i.e., $g\colon X_{15}|| \cdots ||X_0 \mapsto f_{\mathrm{LFSR}}(X_{15})|| \cdots ||f_{\mathrm{LFSR}}(X_0)$ with $X_{15}, \cdots, X_0 \in \{0,1\}^4$. g defined as above satisfies the condition in Section 3.2, i.e., X satisfying $X \oplus g(X) = Y$ is unique for a fixed Y.

HOMA is the main target for performance comparison because it is the conventional smallest [NSS22]. We implement MMM and HOMA with the same design policy, including fullVerif, and use the same hardware components as much as possible. We implement HOMA by following the hardware architecture shown in the original paper [NSS22].

5.2 HPC2 Masking Scheme and Composition-based Verification

We use HPC2 as a masking scheme [CGLS21, CL20], following the previous HOMA implementation [NSS22]. HPC2 provides security with glitches, i.e., transient signal propagation that can break the assumptions behind some masking schemes [MMSS19]. HPC2 also offers composability with Probe Isolating Non-Interference (PINI) [CS20], which ensures the security of a circuit composed of the HPC2 gadgets.

Although composability greatly simplifies the security analysis, additional high-level assumptions should be satisfied for security. They include (i) fresh and independent randomness, (ii) no additional computations than specified by the algorithm, and (iii) no shuffling of the order of the shares. Ensuring the security with these assumptions, Cassiers et al. proposed the composition-based verification tool fullVerif [Cas21]. The tool receives a design in hardware description language (HDL) using special annotations and a gadget library. FullVerif uses the source codes to construct composition graphs, through logic synthesis and simulation, and checks the resulting graphs for security conditions.

We verify our MMM and HOMA implementations using fullVerif. There are two additional tasks for developing the HDL codes compliant with fullVerif. First, the design should be constructed from the verified hardware components. For this purpose, we use the fullVerif's component library containing basic gadgets, including AND, XOR, and MUX. This requirement prohibits low-level optimizations, such as manually instantiating a particular standard cell, e.g., clock gating and scan flip-flops in the previous HOMA implementation [NSS22]. Second, we should annotate the SCA-protected modules and their input and output ports with special directives. The annotation used for fullVerif represents the information such as the security property, the number of shares, and data types [Cas21].

We develop the HDL codes in the design flow in Fig. 6, wherein verification with fullVerif is conducted after the conventional functional verification. We implement the hardware modules in HDL using the verified component library and with annotations. Then, we make functional verification using a logic simulator using testbench, followed by fullVerif verification. Running fullVerif with the source codes and testbench checks the consistency between the annotations and reports pass or fail. If fullVerif reports a failure, we fix the HDL description based on the error messages. We repeat this development

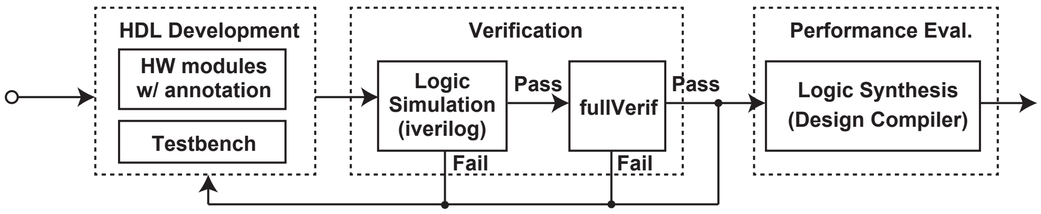

Figure 6: Design and evaluation flow with fullVerif

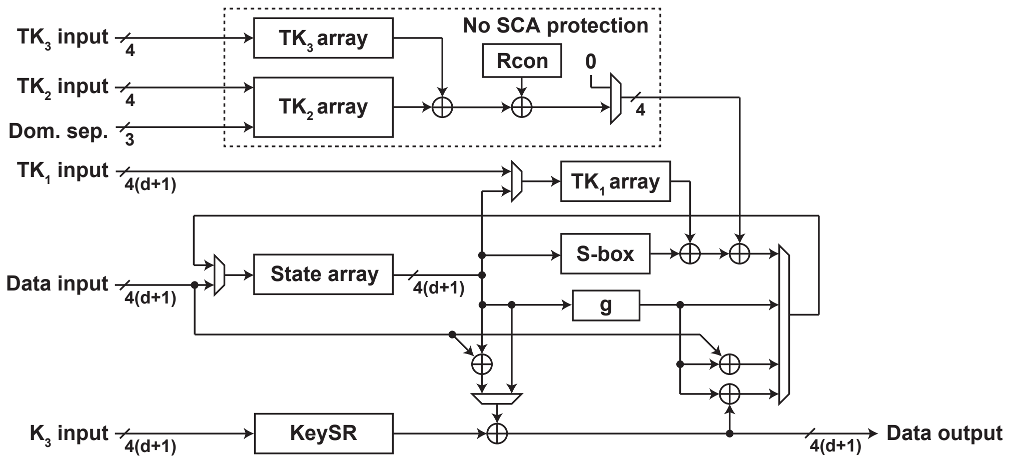

Figure 7: Datapath architecture of diagram of our MMM-8 and MMM-64 implementations

process from low-level modules to the top module while keeping the fullVerif compliance of the finished modules. The final MMM and HOMA implementations are compliant with fullVerif and used for performance evaluation.

5.3 Datapath Architecture

Fig. 7 shows the datapath architecture of our MMM circuit. The overall architecture follows the previous HOMA implementation, which realizes a compact circuit area using a set of systolic arrays that store data and modify them in place. The state array stores SKINNY's 64-bit state and executes ShiftRows and MixColumns in place using the nibble-sliding technique [JMPS17]. The TK_1, TK_2, and TK_3 arrays are the other systolic arrays that integrate the tweakey schedule. KeySR is a cyclic shift register for storing K_3, and MMM-8 and MMM-64 differ only in the size of this KeySR.

We use the same S-box circuit used in the previous HOMA implementation, which is based on Cassiers et al.'s optimized S-box representation [CL20], composed of four HPC2 AND gadgets and three pipeline stages. Since each AND gadget consumes $d(d+1)/2$ bits of randomness for each operation, the S-box circuit consumes $2d(d+1)$-bit random number for each cycle at maximum.

TBC dominates the execution time, and single SKINNY encryption finishes in 1,050 cycles, corresponding to 131.25 cycles/byte. Here, a single SKINNY round takes 26 cycles (with 19, 3, and 4 cycles for S-box, ShiftRows, and MixColumns, respectively), and SKINNY-64/192 is composed of 40 rounds ($26 \times 40 = 1040$ cycles).

As shown in Fig. 7, the TK_2 and TK_3 arrays storing the public value are unprotected from SCA. The S-box circuit is the only place wherein the component in a share can interact with each other, and it is implemented as described above [NSS22]. The other secret components (the state array, TK_1 array, g, and KeySR) are linear (affine), and each component in a share is processed independently. The last-level elements of the SCA-protected modules are always the fullVerif's verified gadgets, and the modules are parameterized with the protection order d.

5.4 Performance Comparison

Table 3 summarizes the post-synthesis performances of MMM-8, MMM-64, and HOMA for the protection orders $d \in [0, 5]$, shown in NAND-equivalent gate count (GE). For further comparison, the table also shows the performance of our Romulus-N1 [IKMP20] implementation (see Appendix C for details) and the HOMA and PFB_Plus implementations in the previous work [NSS22]. The table also shows the circuit sizes of the components preserved during the synthesis. All the targets are evaluated using Synopsys Design Compiler and the NanGate 45-nm standard cell library [Nan21].

As a result, MMM-8 and MMM-64 are smaller than HOMA for all $d \in [0, 5]$. As summarized in Table 1, MMM-8, MMM-64 and HOMA use $192(d+1)+256$, $136(d+1)+128$, and $192(d+1)+128$ bits of memory, respectively. In other words, MMM-8 (resp. MMM-64) is smaller by $56d + 184$ (resp. 128) bits compared with HOMA, which results in the smaller circuit area. The circuit area of each component in Table 3 further verifies that the difference comes from the memory elements for storing the secret-key elements. With $d = 5$, for example, MMM-8 is smaller than HOMA by $3,222$ $(= 17,261 - 14,039)$ GE, which mostly comes from those in TK_1 and KeySR: MMM-8 and HOMA use $(3,275 + 390)$ and $(549 + 5,720)$ GE for these components, respectively.

The Romulus-N1 implementation uses large circuit area by the two reasons. First, Romulus-N1 needs $120(d + 1)$-bit (resp. $64(d + 1)$-bit) more memory compared with MMM-8 (resp. MMM-64), as summarized in Table 1. Second, Romulus-N1 use the 8-bit SKINNY S-box comprising 8 AND gadgets, which is more costly than the 4-bit SKINNY S-box with only 4 AND gadgets. Although our Romulus-N1 has room for reducing a register by 64 bits (256–448 GE, see Appendix C), the advantages of MMM-8 and MMM-64 are more significant.

Table 3 also shows the performance of the conventional HOMA and PFB_Plus implementations [NSS22]. Our HOMA implementation is larger than the previous one for $d > 0$, which is caused by not using low-level optimizations for fullVerif in all of the designs in comparison. In particular, without clock gating, an additional MUX is necessary for realizing a register with enable, which increases the per-bit cost of a memory. Despite this disadvantage, our MMM-8 implementation outperforms both the previous HOMA and PFB_Plus implementations for all $d \in [0, 5]$.

We finally compare speed with Table 4 summarizing the speed in cycle per byte (cpb) for processing a message block in AE encryption. MMM-64 achieves 131.25 cpb, achieving more than $\times 3$ speed-up from HOMA (549.75 cpb) because of the smaller number of TBC calls per message block (see Table 1). MMM-64 is comparable with PFB_Plus in speed by the same reason. Romulus-N1 achieved a better speed (83.13 cpb) because of its byte-serial architecture that processes a byte (cf. a nibble) for each cycle, roughly doubling the speed, which comes at the cost of a larger area as described previously.

6 Discussion on Other Primitive Choices

MMM allows flexibility in choosing the size of K_1 and $K_2 \| K_3$ according to user needs and available primitive designs, and allows settings other than SKINNY64/192 and $|K_1| = 64$ implemented in the previous section. The plaintext block size b can be up to the BC's block

Table 3: NAND-equivalent circuit area (GE) of MMM-8, MMM-64, HOMA, Romulus-N1, and PFB_Plus for the protection order $d \in [0, 5]$.

Order d	Target	Total	S-box	State	TK_1	TK_2	TK_3	TK_4	KeySR
0	MMM-8	3,343	161	594	549	732	583	—	—
0	MMM-64	3,767	161	549	549	729	582	—	539
0	HOMA	4,878	161	595	549	704	583	588	955
0	Romulus-N1	5,915	598	1,082	1,261	1,184	1,186	—	—
1	MMM-8	5,023	500	1,165	1,094	732	583	—	130
1	MMM-64	5,847	499	1,165	1,094	729	582	—	960
1	HOMA	6,893	500	1,157	549	704	583	588	1,907
1	Romulus-N1	9,156	1,532	2,147	1,162	1,184	1,186	—	—
2	MMM-8	6,924	1,086	1,713	1,639	730	582	—	195
2	MMM-64	8,167	1,086	1,713	1,639	729	582	—	1,439
2	HOMA	9,138	1,086	1,715	549	701	578	590	2,860
2	Romulus-N1	12,899	2,967	3,202	1,188	1,094	3,479	—	—
3	MMM-8	9,073	1,896	2,282	2,185	729	582	—	260
3	MMM-64	10,732	1,896	2,282	2,185	729	582	—	1,918
3	HOMA	11,622	1,896	2,279	549	703	579	588	3,814
3	Romulus-N1	17,139	4,869	4,268	1,137	1,089	4,629	—	—
4	MMM-8	11,457	2,934	2,852	2,733	729	582	—	324
4	MMM-64	13,530	2,934	2,852	2,733	729	582	—	2,397
4	HOMA	14,341	2,929	2,851	549	702	579	590	4,769
4	Romulus-N1	21,858	7,173	5,329	1,171	1,090	5,775	—	—
5	MMM-8	14,039	4,193	3,404	3,275	729	582	—	390
5	MMM-64	16,526	4,193	3,404	3,275	729	582	—	2,876
5	HOMA	17,261	4,187	3,405	549	702	579	588	5,720
5	Romulus-N1	26,991	9,978	6,388	1,135	1,083	6,916	—	—
0	HOMA[†]	4,981	161	542	636	844	675	675	735
0	PFB_Plus[†]	4,569	161	540	637	674	746	865	
1	HOMA[†]	6,283	501	1,046	549	749	585	577	1,468
1	PFB_Plus[†]	6,884	501	1,049	1,231	1,296	656	782	—
2	HOMA[†]	8,226	1,087	1,573	549	744	586	576	2,201
2	PFB_Plus[†]	9,667	1,087	1,571	1,845	1,938	657	782	—
3	HOMA[†]	10,392	1,897	2,097	549	748	585	577	2,935
3	PFB_Plus[†]	12,675	1,897	2,094	2,459	2,578	656	780	—
4	HOMA[†]	12,782	2,931	2,621	549	744	585	577	3,668
4	PFB_Plus[†]	15,941	2,931	2,619	3,083	3,239	656	780	—
5	HOMA[†]	15,487	4,189	3,240	549	748	586	576	4,402
5	PFB_Plus[†]	19,724	4,189	3,238	3,818	3,989	656	781	—

[†]HOMA and PFB_Plus performance obtained from [NSS22, Table 3].

size n and may have different properties depending on its choice. This section describes the selection of primitives when $s = 128$ and $n = 128$.

When b is chosen to be 128 for the fastest speed in MMM, the size of K_3 is 128 bits, and K_1 is 0 bits. First, since $|K_1| = 0$, the input to the underlying BC is 128 bits in total: the counter, the nonce, and the domain separation. Hence, the key size that the underlying BC must support is 128 bits. This is a popular parameter especially for $n = 128$. Hence,

Table 4: Execution speed in cycle/byte for processing one message block in AE encryption.

Target	Primitive	Latency (cycles)	Message block length (bytes)	Enc. speed (cycle/byte)
MMM-8	SKINNY-64/192	1,050	1	1,050.00
MMM-64	SKINNY-64/192	1,050	8	131.25
HOMA	SKINNYee	1,466	8/3	549.75
Romulus-N1	SKINNY-128/384+	1,330	16	83.13
HOMA[†]	SKINNYee	1,344	8/3	504.00
PFB_Plus[†]	SKINNYe	1,056	8	132.00

[†]HOMA and PFB_Plus performance in the previous work [NSS22].

the choice of primitives is diverse, and basing the scheme on a standardized BC is easy. Second, due to $|K_1| = 0$, the state corresponding to K_1, H_1, and S_1 disappears, and there will be no secret value in the key element of each BC call. Recall that, for $b = 64$, the key element has both secret and public values, and these values are updated simultaneously in the key schedule function in BC. At this time, it was necessary to have a BC with a special key schedule function in which the secret part and the public part are updated independently. If all the bits of the key element are public, then there is no longer a separation of data that must not be mixed, and any operation on the BC's key schedule function can be used as an instance of MMM. For example, the AES key schedule function can be used without problems.

One concern is that many 128-bit block BCs use 8-bit S-boxes instead of 4-bit S-boxes. Since the masking implementation cost depends on the S-box size, BCs using 4-bit S-boxes should be used as an instance of MMM. For example, GIFT-128 [BPP+17] and Midori128 [BBI+15] are examples of BCs with a 128-bit block using a 4-bit S-box.

Another point that needs to be addressed carefully is the gap between the assumption about primitives by mode and the premise of security by primitive designs. MMM assumes that the BC behaves as an ideal cipher. On the other hand, many lightweight BC designs may only consider security in the single-key setting. For example, regarding GIFT-128 and Midori128, the designers do not claim any related-key security, though related-key attacks have not been found yet.

Contrarily, designing new decent BCs with the above features, suitable as an instance of MMM, is an interesting open problem.

7 Conclusion

In this paper, we proposed a new AE mode MMM that achieves the minimum memory size with masking. Our idea is to update the whole s-bit secret state by iterating an n-bit block BC, and to encrypt a b-bit plaintext block by using the b-bit part of the s-bit state. Compared to HOMA, MMM achieves several advantages, which includes a better rate, mu-security, and a smaller transmission cost. Then two SKINNY-based instances, MMM-8 and MMM-64, were proposed and implemented With a $(d + 1)$-masking scheme. MMM-8 and MMM-64 achieved smaller circuit areas than HOMA with all $d \in [0, 5]$.

A Example of MMM

The structure of MMM with $k_1 = n$, $b = n$, and $\tau = 2n$ is given in Fig. 8. The conditions offer the following conditions $k_2 = 0$, $b = n$, and $k = 2n$. Note that MMM-64 satisfy the

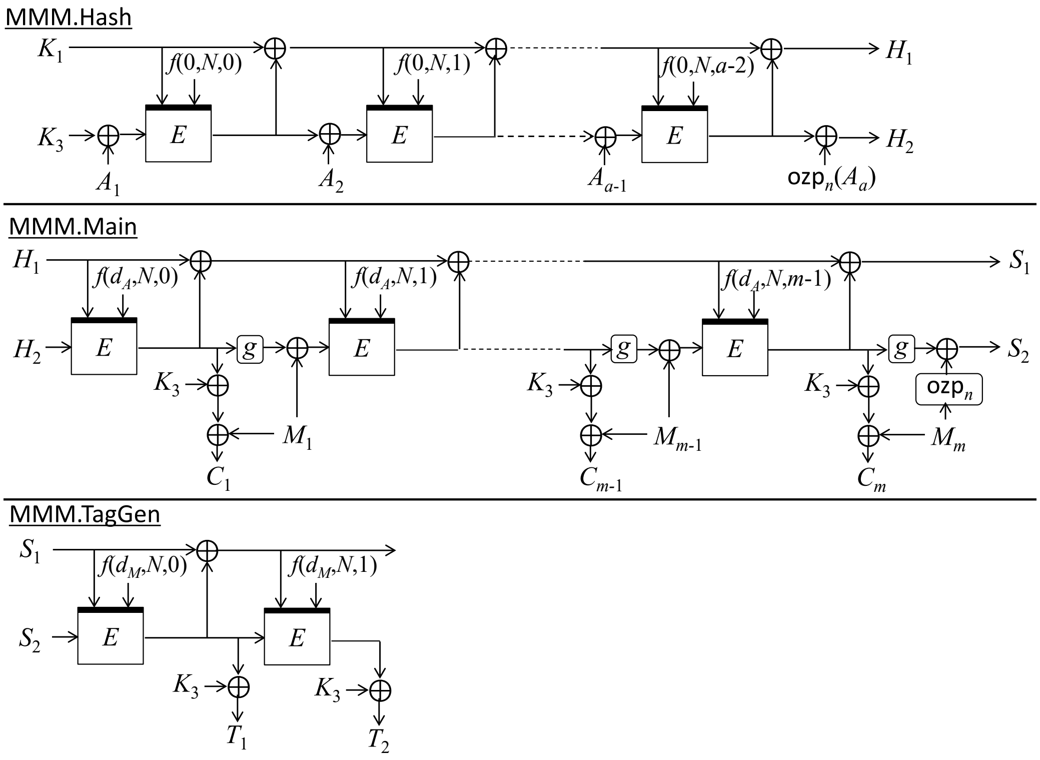

Figure 8: Structure of MMM with $k_1 = n$, $b = n$, and $\tau = 2n$.

conditions.

B Analysis for Good Transcripts

Let $\tau \in \mathbf{T}_{\mathsf{good}}$. Tags and ciphertexts in τ are defined from (dummy) input-output tuples in online queries. By $\neg\mathsf{bad}_4$, response of decryption queries in τ are all **reject**. Hence, we have only to consider (dummy) user's keys, (dummy) input-output tuples of an IC in online queries, and offline query-response tuples. Let τ_K be the subset of τ whose elements are user's keys. Let τ_{online} be the set of offline query-response tuples, τ_{enc} the set of input-output tuples defined by encryption query-response tuples that are not in τ_{online}, and τ_{dec} the set of input-output tuples defined by decryption query-response tuples that are not in $\tau_{\mathsf{enc}} \cup \tau_{\mathsf{online}}$.

Firstly, we evaluate $\Pr[\mathsf{T}_R \vdash \tau_{\mathsf{offline}}]$ and $\Pr[\mathsf{T}_I \vdash \tau_{\mathsf{offline}}]$, the probabilities for offline queries. In both worlds, the responses to offline queries are defined by an IC. We thus have $\Pr[\mathsf{T}_R \vdash \tau_{\mathsf{online}}] = \Pr[\mathsf{T}_I \vdash \tau_{\mathsf{online}}]$. Hereafter, we assume that $\mathsf{T}_R \vdash \tau_{\mathsf{online}}$ and $\mathsf{T}_I \vdash \tau_{\mathsf{online}}$ are satisfied.

Secondly, we evaluate $\Pr[\mathsf{T}_R \vdash \tau_K]$ and $\Pr[\mathsf{T}_I \vdash \tau_K]$, the probabilities for user's keys. In both worlds, all user's keys are chosen uniformly at random from $\{0,1\}^k$. We thus have $\Pr[\mathsf{T}_R \vdash \tau_K] = \Pr[\mathsf{T}_I \vdash \tau_K]$. Hereafter, we assume that $\mathsf{T}_R \vdash \tau_K$ and $\mathsf{T}_I \vdash \tau_K$ are satisfied.

Thirdly, we evaluate $\Pr[\mathsf{T}_R \vdash \tau_{\mathsf{enc}}]$ and $\Pr[\mathsf{T}_I \vdash \tau_{\mathsf{enc}}]$, the probabilities for encryption queries. Note that by bad_1, there is no collision among IC's inputs in the encryption queries, and by bad_3, there is no collision between τ_{online} and τ_{enc}. Hence, we all inputs in τ_{enc} are new. In the real world, all internal values in encryption queries are defined by an IC and in the ideal world, by Algorithm 6. In this algorithm, all internal values for the hash function are defined by an IC and the remaining ones are chosen uniformly at random from $\{0,1\}^n$, as all ciphetext blocks and tag ones are chosen uniformly at random from $\{0,1\}^b$. Hence, the output spaces in the ideal world are larger than those in the real one, and we have $\Pr[\mathsf{T}_R \vdash \tau_{\mathsf{enc}}] \geq \Pr[\mathsf{T}_I \vdash \tau_{\mathsf{enc}}]$. Hereafter, we assume that $\mathsf{T}_R \vdash \tau_{\mathsf{enc}}$ and $\mathsf{T}_I \vdash \tau_{\mathsf{enc}}$ are satisfied.

Fourthly, we evaluate $\Pr[\mathsf{T}_R \vdash \tau_{\mathsf{dec}}]$ and $\Pr[\mathsf{T}_I \vdash \tau_{\mathsf{dec}}]$, the probabilities for decryption queries. In both worlds, the internal values in decryption queries are defined by an IC, that is, chosen from the same spaces. Note that the ideal-world values are defined by Algorithm 6. Hence, we have $\Pr[\mathsf{T}_R \vdash \tau_{\mathsf{dec}}] = \Pr[\mathsf{T}_I \vdash \tau_{\mathsf{dec}}]$

Finally, using the above evaluations, we have

$$\frac{\Pr[\mathsf{T}_R = \tau]}{\Pr[\mathsf{T}_I = \tau]} = \frac{\Pr[\mathsf{T}_R \vdash \tau_{\mathsf{offline}}] \cdot \Pr[\mathsf{T}_R \vdash \tau_K] \cdot \Pr[\mathsf{T}_R \vdash \tau_{\mathsf{enc}}] \cdot \Pr[\mathsf{T}_R \vdash \tau_{\mathsf{dec}}]}{\Pr[\mathsf{T}_I \vdash \tau_{\mathsf{offline}}] \cdot \Pr[\mathsf{T}_I \vdash \tau_K] \cdot \Pr[\mathsf{T}_I \vdash \tau_{\mathsf{enc}}] \cdot \Pr[\mathsf{T}_I \vdash \tau_{\mathsf{dec}}]} \geq 1 \ .$$

C Romulus-N1 Hardware Design

This appendix describes our Romulus-N1 hardware design protected with the HPC2 high-order masking, discussed in Section 5.4. Romulus-N1 is a TBC-based AE [IKMP20] that appeared at the NIST LWC competition. Romulus-N1's underlying TBC is SKINNY-128-384+, a 40-round version of SKINNY-128-384. The primitive is a variant of 128-bit SKINNY with the 8-bit S-box, which contrasts with MMM, HOMA, and PFB_Plus that used 64-bit SKINNY with the 4-bit S-box. A byte is a natural processing unit for Romulus-N1, and we design its implementation with the byte-serial (cf. nibble-serial) architecture using the state and tweakey arrays [BJK+16].

The state array is a 128-bit systolic array integrating MixColumns and ShiftRows. The TK_1–TK_3 arrays are similar 128-bit systolic arrays that integrate forward and inverse tweakey schedules. TK_3 (cf. TK_1) stores the secret key and needs SCA protection, and TK_1 integrates a 56-bit LFSR circuit for a counter. These arrays are optimized with the byte-sliding technique to minimize the hardware cost [JMPS17]. A single SKINNY-128-384+ round takes 30 cycles, and a single SKINNY-128-384+ takes 1,330 cycles involving 40 rounds of operations and the other overhead for recovering the original values overwritten by the on-the-fly tweakey schedule. Although Romulus-N1 uses only the upper half of TK_1, we implement the TK_1 array with a 128-bit register for simplicity and consistency with the other tweakey array implementations; The unused 64 bits can be reduced by designing a sophisticated circuit that keeps track of the unused bits in the tweakey schedule. The room for further optimization is estimated as 256–448 GE with a typical register cost (4–7 GE/bit). In addition to the state and tweakey arrays, we also implement the ρ and ρ^{-1} functions for feeding plaintext/ciphertext blocks [IKMP20]. This module needs SCA protection, but it can be easily realized exploiting the linearity of ρ and ρ^{-1}.

We finally discuss the implementation of the SKINNY 8-bit S-box. Since the HPC2 AND gadget needs a different number of cycles (1 and 2 cycles) for its inputs, sophisticated scheduling is necessary to minimize the pipeline stages. In the previous work, Cassiers et al. approached the problem by using a SAT solver to find an efficient Boolean circuit for 4-bit S-boxes [CL20], which is used in our MMM and HOMA implementations. Unfortunately, the same approach is infeasible with 8-bit S-boxes for a large computational complexity [Sto16]. Therefore, we schedule the HPC2 AND gadgets without changing the S-box's original Boolean representation comprising 8 NOR gates [BJK+16]. Realizing NOR using AND is straightforward because $\overline{x_0 \vee x_1} = \overline{x_0} \wedge \overline{x_1}$. Then, we assign each AND gadget by analyzing the data dependency graph. Fig. 9 shows the final design comprising a 6-stage pipeline with one additional input stage for controlling incoming data. Each S-box calculation consumes $4d(d+1)$ random bits with $d+1$ shares. The S-box circuit uses $28d^2 + 92d + 16$

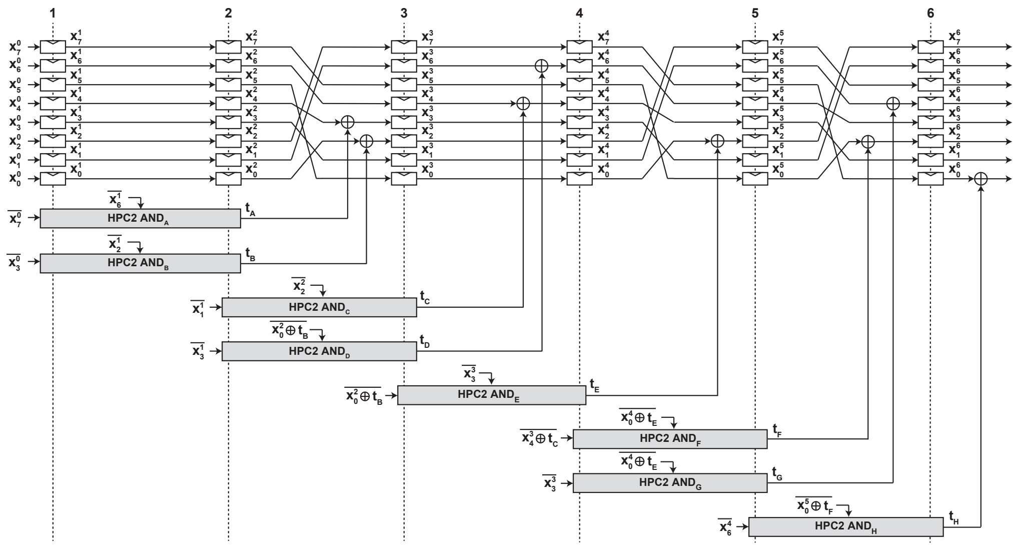

Figure 9: Pipelined circuit for 8-bit SKINNY S-box using high-order masking HPC2.

bits of registers: $(8 \times 6)d$ bits for pipeline registers and $(7d^2 + 11d + 4)/2$ bits for each AND gadget.

References

[AFK+11] Frederik Armknecht, Ewan Fleischmann, Matthias Krause, Jooyoung Lee, Martijn Stam, and John P. Steinberger. The preimage security of double-block-length compression functions. In *ASIACRYPT 2011*, volume 7073 of *LNCS*, pages 233–251. Springer, 2011.

[ARS+15] Martin R. Albrecht, Christian Rechberger, Thomas Schneider, Tyge Tiessen, and Michael Zohner. Ciphers for MPC and FHE. In *EUROCRYPT 2015*, volume 9056, pages 430–454, 2015.

[BBB+20] Davide Bellizia, Francesco Berti, Olivier Bronchain, Gaëtan Cassiers, Sébastien Duval, Chun Guo, Gregor Leander, Gaëtan Leurent, Itamar Levi, Charles Momin, Olivier Pereira, Thomas Peters, François-Xavier Standaert, Balazs Udvarhelyi, and Friedrich Wiemer. Spook: Sponge-based leakage-resistant authenticated encryption with a masked tweakable block cipher. *IACR Trans. Symmetric Cryptol.*, 2020(S1):295–349, 2020.

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

[BBI+15] Subhadeep Banik, Andrey Bogdanov, Takanori Isobe, Kyoji Shibutani, Harunaga Hiwatari, Toru Akishita, and Francesco Regazzoni. Midori: A block cipher for low energy. In *ASIACRYPT 2015*, volume 9453 of *LNCS*, pages 411–436. Springer, 2015.

[BBP$^+$16] Sonia Belaïd, Fabrice Benhamouda, Alain Passelègue, Emmanuel Prouff, Adrian Thillard, and Damien Vergnaud. Randomness complexity of private circuits for multiplication. In *EUROCRYPT 2016*, pages 616–648, 2016.

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

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

[BPP$^+$17] Subhadeep Banik, Sumit Kumar Pandey, Thomas Peyrin, Yu Sasaki, Siang Meng Sim, and Yosuke Todo. GIFT: A small Present - towards reaching the limit of lightweight encryption. In *CHES 2017*, volume 10529 of *LNCS*, pages 321–345. Springer, 2017.

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

[Cas21] Gaëtan Cassiers. fullVerif. https://github.com/cassiersg/fullverif, 2021.

[CGLS21] Gaëtan Cassiers, Benjamin Grégoire, Itamar Levi, and François-Xavier Standaert. Hardware private circuits: From trivial composition to full verification. *IEEE Trans. Computers*, 70(10):1677–1690, 2021.

[CL20] Gaëtan Cassiers and Itamar Levi. AND depth 2, 4 ANDs, 4-bit (Optimized) S-boxes. *IACR Cryptol. ePrint Arch.*, 2020:185, 2020.

[CS20] Gaëtan Cassiers and François-Xavier Standaert. Trivially and efficiently composing masked gadgets with probe isolating non-interference. *IEEE Trans. Inf. Forensics Secur.*, 15:2542–2555, 2020.

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

[DM21] Christoph Dobraunig and Bart Mennink. Leakage resilient value comparison with application to message authentication. In *EUROCRYPT 2021*, pages 377–407, 2021.

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

[GGNS13] Benoît Gérard, Vincent Grosso, María Naya-Plasencia, and François-Xavier Standaert. Block ciphers that are easier to mask: How far can we go? In *CHES 2013*, pages 383–399, 2013.

[GJK$^+$20] Dahmun Goudarzi, Jérémy Jean, Stefan Kölbl, Thomas Peyrin, Matthieu Rivain, Yu Sasaki, and Siang Meng Sim. Pyjamask: Block cipher and authenticated encryption with highly efficient masked implementation. *IACR Trans. Symmetric Cryptol.*, 2020:31–59, 2020.

[GMK16] Hannes Groß, Stefan Mangard, and Thomas Korak. Domain-oriented masking: Compact masked hardware implementations with arbitrary protection order. *IACR Cryptol. ePrint Arch.*, 2016:486, 2016.

[HTT18] Viet Tung Hoang, Stefano Tessaro, and Aishwarya Thiruvengadam. The multi-user security of gcm, revisited: Tight bounds for nonce randomization. In *CCS 2018*, pages 1429–1440. ACM, 2018.

[IKMP20] Tetsu Iwata, Mustafa Khairallah, Kazuhiko Minematsu, and Thomas Peyrin. Duel of the titans: The Romulus and Remus families of lightweight AEAD algorithms. *IACR Trans. Symmetric Cryptol.*, 2020(1):43–120, 2020.

[ISO22] ISO. ISO/IEC 18033-7:2010 Information security — Encryption algorithms — Part 7: Tweakable block ciphers. 2022.

[ISW03] Yuval Ishai, Amit Sahai, and David A. Wagner. Private circuits: Securing hardware against probing attacks. In *CRYPTO 2003*, pages 463–481, 2003.

[JLM$^+$19] Philipp Jovanovic, Atul Luykx, Bart Mennink, Yu Sasaki, and Kan Yasuda. Beyond Conventional Security in Sponge-Based Authenticated Encryption Modes. *J. Cryptol.*, 32(3):895–940, 2019.

[JMPS17] Jérémy Jean, Amir Moradi, Thomas Peyrin, and Pascal Sasdrich. Bit-Sliding: A generic technique for bit-serial implementations of SPN-based primitives - applications to AES, PRESENT and SKINNY. In *CHES 2017*, volume 10529, pages 687–707, 2017.

[KJJ99] Paul C. Kocher, Joshua Jaffe, and Benjamin Jun. Differential Power Analysis. In *CRYPTO '99*, pages 388–397, 1999.

[LMP17] Atul Luykx, Bart Mennink, and Kenneth G. Paterson. Analyzing multi-key security degradation. In *ASIACRYPT 2017*, volume 10625, pages 575–605. Springer, 2017.

[MMSS19] Thorben Moos, Amir Moradi, Tobias Schneider, and François-Xavier Standaert. Glitch-resistant masking revisited or why proofs in the robust probing model are needed. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2019(2):256–292, 2019.

[Nan21] NanGate. NanGate FreePDK45 Open Cell Library. `https://si2.org/open-cell-library/`, 2021.

[NRR06] Svetla Nikova, Christian Rechberger, and Vincent Rijmen. Threshold implementations against side-channel attacks and glitches. In *Information and Communications Security, 8th International Conference, ICICS 2006*, pages 529–545, 2006.

[NS20] Yusuke Naito and Takeshi Sugawara. Lightweight authenticated encryption mode of operation for tweakable block ciphers. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2020(1):66–94, 2020.

[NSS20] Yusuke Naito, Yu Sasaki, and Takeshi Sugawara. Lightweight authenticated encryption mode suitable for threshold implementation. In *EUROCRYPT 2020*, pages 705–735, 2020.

[NSS22] Yusuke Naito, Yu Sasaki, and Takeshi Sugawara. Secret can be public: Low-memory AEAD mode for high-order masking. In *CRYPTO 2022*, volume 13509 of *LNCS*, pages 315–345, 2022.

[Pat08] Jacques Patarin. The "Coefficients H" technique. In *SAC 2008*, pages 328–345, 2008.

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

[QDW$^+$22] Lingyue Qin, Xiaoyang Dong, Anyu Wang, Jialiang Hua, and Xiaoyun Wang. Mind the TWEAKEY schedule: Cryptanalysis on SKINNYe-64-256. In *ASIACRYPT 2022*, volume 13791 of *LNCS*, pages 287–317. Springer, 2022.

[RBN$^+$15] Oscar Reparaz, Begül Bilgin, Svetla Nikova, Benedikt Gierlichs, and Ingrid Verbauwhede. Consolidating masking schemes. In *CRYPTO 2015*, volume 9215, pages 764–783, 2015.

[Res18] Eric Rescorla. RFC 8446: The transport layer security (TLS) protocol version 1.3. https://doi.org/10.17487/RFC8446, 2018.

[RTM21] Eric Rescorla, Hannes Tschofenig, and Nagendra Modadugu. The datagram transport layer security (DTLS) protocol version 1.3 – draft-ietf-tls-dtls13-43. https://tools.ietf.org/html/draft-ietf-tls-dtls13-43, 2021.

[Sto16] Ko Stoffelen. Optimizing S-box implementations for several criteria using SAT solvers. In *FSE 2016*, volume 9783, pages 140–160, 2016.

[TT21] Martin Thomson and Sean Turner. Using TLS to secure QUIC. *RFC*, 9001:1–52, 2021.