ELM: A Low-Latency and Scalable Memory Encryption Scheme

Akiko Inoue, Kazuhiko Minematsu, Maya Oda, Rei Ueno, *Member, IEEE*, and Naofumi Homma, *Senior Member, IEEE*

***Abstract*— Memory encryption (ME) with authentication is becoming a key security feature of modern processors, as evident by the adoption of ME by Intel's SGX. Recently ME is actively studied from the viewpoint of system architecture. This paper studies ME from the viewpoint of symmetric-key cryptographic designs, with a primal focus on latency. A significant progress in such a direction can be observed in the SGX Integrity Tree (SIT). Using a variant of AES-GCM, SIT achieves an excellent latency. However, it has a scalability issue. By carefully examining SIT, we develop a new ME scheme dubbed ELM. We present an AES-based instantiation of ELM, and show that ELM significantly reduces latency from SIT for large memories, and achieves the provable security and equivalent hardware-protected (on-chip) area. We also present preliminary hardware implementations to substantiate our advantages.**

***Index Terms*— Memory encryption, authentication tree, latency, mode of operations, SGX.**

I. Introduction

MEMORY encryption (ME) is widely deployed in modern systems. One typical method is sector-wise encryption, such as XTS [1]. A sector-wise encryption scheme encrypts each memory sector in an independent and deterministic manner, keeping the key in an on-chip area that is physically protected. This prevents passive off-line attacks that try to extract the data from the storage devices, such as [2]. However, it is vulnerable against active online attacks and replay attacks, for the lack of authenticity. If we encrypt each sector using a nonce-based authenticated encryption (AE) and store all the nonces on-chip, it would protect against active attacks. However, this would incur a linear increase of the on-chip area. This is usually impractical because the on-chip area is much more expensive than the main (off-chip) memory.

A well-known classical solution to this problem is to use an authentication tree, also known as a Merkle Hash Tree [3]. By involving any unit memory data in the tree computation and storing the root hash value on-chip, the authenticity against active attacks is guaranteed. Instead of a cryptographic hash function, we can use a message authentication code (MAC) to build an authentication tree. Merkle tree and its (possibly MAC-based) improvements, such as PAT [4] and Bonsai Tree [5], provide an authenticity of the whole memory with a constant on-chip memory overhead, at the cost of a logarithmic computation overhead for read and write operations. Confidentiality of the memory can be achieved by an additional encryption mechanism, *e.g.*, by TEC-tree [6]. Due to the increasing threat of active attacks, tree-based ME schemes, often with a confidentiality mechanism, are gradually being deployed in real-world memory/storage systems. One prominent example is Intel's SGX [7], [8], which adopts a variant of PAT with a dedicated AES-based MAC and AE schemes similar to GMAC and GCM [9]. The widespread use of non-volatile memory also pushes the need for ME with authenticity.

Latency is a very important criterion for the aforementioned ME schemes. Merkle tree can reduce latency by utilizing parallelizability, but only for verification (read operation). When one wants to replace a piece of data with a new one (write operation), Merkle tree needs to serially update all hash values on the path from the leaf (data) to the root. PAT is the current state-of-the-art in this respect, as it is parallelizable for both read and write operations by means of a clever use of nonce-based MAC.

Tree-based MEs have been initially studied from the cryptography community, however, it recently receives significant attention from the system architecture community, such as [5], [10]–[12]. The primary focus of these studies is the data structure, such as the parameters/structures of integrity trees [5], [11] and counter/nonce representations [10], [12] that fits well into the target architecture, and cryptographic components are often considered as black boxes or instantiated by just picking a standard, *e.g.*, the use of GCM in [10]. A notable exception is the aforementioned ME inside SGX, which is also called SGX Integrity Tree (SIT). It develops dedicated AE and MAC schemes based on AES-GCM, with particular attention to latency in mind. SIT is quite efficient and enables a very low-latency read/write operation on the given tree structure that covers up to 96 Mbyte of memory on an x86 platform. Moreover, as an important subsystem of SGX, it is also quite widely deployed in practice.

In contrast to the numerous ME proposals from the system architecture community, those from cryptography community are rather scarce after [4], [6]. As mentioned above, SIT is a notable exception, however its on-chip data size is linear

Manuscript received 6 December 2021; revised 20 April 2022; accepted 9 June 2022. Date of publication 4 July 2022; date of current version 22 July 2022. This work was supported in part by the Japan Science and Technology Agency (JST) Core Research for Evolutional Science and Technology (CREST), Japan, under Grant JPMJCR19K5. The associate editor coordinating the review of this manuscript and approving it for publication was Prof. Nele Mentens. *(Corresponding author: Akiko Inoue.)*

Akiko Inoue is with NEC, Kawasaki 211-8666, Japan (e-mail: a_inoue@nec.com).

Kazuhiko Minematsu is with NEC, Kawasaki 211-8666, Japan, and also with the Graduate School of Environment and Information Sciences, Yokohama National University, Kanagawa 240-8501, Japan.

Maya Oda, Rei Ueno, and Naofumi Homma are with the Research Institute of Electrical Communication, Tohoku University, Sendai 980-8577, Japan.

Digital Object Identifier 10.1109/TIFS.2022.3188146

to the unit data size. This poses a limitation on the amount of covered memory sizes and hence is not scalable. In fact, VAULT [11] and Morphable Counter [12] are two recent MEs that aim at covering much larger memory than SIT and improving the performance, mainly by the system architecture approach.

This paper proposes a new ME scheme dubbed ELM (Encryption for Large Memory), which enables a significantly low latency for large amounts of memory. It achieves on-chip and off-chip memory overheads comparable to existing schemes, which becomes possible by closely inspecting the required operations in SIT and its baseline PAT. ELM adopts several techniques known in the symmetric-key cryptography literature. Specifically, we find that *incremental MAC* (Inc-MAC) [13], [14] can be used securely and it works quite effectively for the trees with large arity, a common feature for the modern schemes targeting large memories. By using an Inc-MAC at the internal nodes, ELM significantly reduces the write latency without harming the read latency. Our MAC is a variant of the classical XOR-MAC [15] that optimizes the latency, number of primitive calls, and security.

Another key component of ELM is a new low-latency AE scheme. It is basically a variant of OCB [16]. OCB is already quite good in terms of latency and parallelizability, because it does not need an additional authentication function unlike GCM. However, the decryption latency of OCB is not sufficiently small due to its structure. By changing the structure, our AE mode has a smaller latency than the original for decryption, while retaining the other main features such as parallelizability and provable security. In particular, when it is viewed as a mode of a tweakable block cipher (TBC) [17], the latency is *optimally small* for both encryption and decryption. ELM is obtained by combining our MAC, our AE, and a variant of tree structure of SIT which we call PAT2. Each technique itself is not ultimately novel. However, we show how to combine them in an optimal manner to reduce latency and computation, which is, to the best of our knowledge, not known in the literature. Our proposal is generic in principle, and it can be instantiated by any block cipher or TBC. Moreover, we prove that the security of ELM (and PAT2) is reduced to the pseudorandomness of the block cipher we use, which is the first *formal* security treatment of ME with authenticity.

To showcase the effectiveness of ELM, we specify an instantiation of ELM using the same components as SIT, namely AES-128 and a full 64-bit field multiplier.[1] We compare them with (a generalized variant of) SIT for various memory sizes and tree parameters under a certain practical implementation setting. Our results show that ELM has a smaller latency than SIT for most of the cases,[2] and as memory size gets larger, the difference becomes significant. We also conducted preliminary ASIC implementations, and show that the total implementation size is comparable to that of SIT. In addition, we discuss the optimization of hardware implementations for our proposal depending on the system constraints.

[1]We also use a 128-bit multiplier, but with a very small input size.

[2]This holds true even when SIT adopts a part of our idea of using Inc-MAC. See Section V-B.

II. Preliminaries

A. Notation

For a positive integer n, $\{0,1\}^n$ denotes the set of n-bit strings and let $[n] = \{1, \ldots, n\}$. $X \xleftarrow{\$} \mathcal{X}$ means that the variable X is uniformly sampled over the set $\mathcal{X}$. For binary strings A and B, $A \| B$ denotes the concatenation of A and B. The bit length of A is denoted by $|A|$, and $|A|_n := \lceil |A|/n \rceil$. Dividing a string A into blocks of n bits is denoted by $A[1] \| \cdots \| A[m] \xleftarrow{n} A$, where $m = |A|_n$ and $|A[i]| = n$, $|A[m]| \leq n$ for $1 \leq i \leq m-1$. For $t \in [|A|]$, $\mathtt{msb}_t(A)$ ($\mathtt{lsb}_t$) denotes the first (last) t bits of A. A sequence of i zeros is written as 0^i. For a function $F : \mathcal{K} \times \mathcal{X} \to \mathcal{Y}$ with the key space $\mathcal{K}$, $F(K, \cdot)$ may be written as $F_K(\cdot)$. Let $\mathrm{GF}(2^n)$ be a finite field of size 2^n with characteristic 2 and extension degree n. We focus on the case $n = 128$. Following [18], we use $(x^{128} + x^7 + x^2 + x + 1)$ for defining the field $\mathrm{GF}(2^{128})$. Here, the primitive root x is interpreted as 2 in the decimal representation. For $a \in \mathrm{GF}(2^n)$, let $2a$ denote a multiplication by x and a, which is also called doubling [18]. Similarly, let $3a$ denote $2a \oplus a$.

B. (Tweakable) Block Cipher

Let $\mathcal{K}$ and $\mathcal{M}$ be the set of keys and messages, respectively. Let $\mathcal{TW}$ be the set of tweaks, where a tweak is a public parameter. A tweakable block cipher (TBC) [17] is a function $\widetilde{E} : \mathcal{K} \times \mathcal{TW} \times \mathcal{M} \to \mathcal{M}$ s.t. $\widetilde{E}_K(Tw, \cdot)$ is a permutation on $\mathcal{M}$ for $\forall (K, Tw) \in \mathcal{K} \times \mathcal{TW}$. It is also denoted by $\widetilde{E}_K^{Tw}$, $\widetilde{E}^{Tw}$, or $\widetilde{E}$. If $\mathcal{TW}$ is singleton, it means a plain block cipher E_K. A TBC can be built on a block cipher [17], [18]. A block cipher E or a TBC $\widetilde{E}$ is said to be secure if it is computationally hard to distinguish from the ideal primitive with oracle access. Let $\mathrm{Perm}(n)$ denote the set of all permutations on $\{0,1\}^n$. An n-bit tweakable permutation of t-bit tweak is a function $\pi : \{0,1\}^t \times \{0,1\}^n \to \{0,1\}^n$ s.t. $\pi(Tw, \cdot) \in \mathrm{Perm}(n)$ for $\forall Tw \in \{0,1\}^t$. The set of all n-bit tweakable permutations with t-bit tweak is denoted by $\mathrm{TPerm}(t, n)$. Let $\mathsf{P} \xleftarrow{\$} \mathrm{Perm}(n)$ be a uniform random permutation (URP) and $\widetilde{\mathsf{P}} \xleftarrow{\$} \mathrm{TPerm}(t, n)$ be a tweakable URP (TURP). Let $\mathcal{A}$ be an adversary who (possibly adaptively) queries an oracle O and subsequently outputs a bit. We write $\Pr[\mathcal{A}^O \to 1]$ to denote the probability that this bit is 1. The advantages against $\widetilde{E}$ are defined as $\mathbf{Adv}^{\mathrm{tprp}}_{\widetilde{E}}(\mathcal{A}) := |\Pr[\mathcal{A}^{\widetilde{E}} \to 1] - \Pr[\mathcal{A}^{\widetilde{\mathsf{P}}} \to 1]|$, and $\mathbf{Adv}^{\mathrm{tsprp}}_{\widetilde{E}}(\mathcal{A}^{\pm}) := |\Pr[(\mathcal{A}^{\pm})^{\widetilde{E}, \widetilde{E}^{-1}} \to 1] - \Pr[(\mathcal{A}^{\pm})^{\widetilde{\mathsf{P}}, \widetilde{\mathsf{P}}^{-1}} \to 1]|$, where $\widetilde{E}^{-1}$ and $\widetilde{\mathsf{P}}^{-1}$ are decryption functions of $\widetilde{E}$ and $\widetilde{\mathsf{P}}$. When the advantage is sufficiently small, $\widetilde{E}$ is said to be secure against the underlying adversary.

C. Message Authentication Code

Message authentication code (MAC) is a symmetric-key function for message authenticity. We consider *nonce-based* MAC, where a nonce is a non-repeating value used together with a message. For the nonce space $\mathcal{N}$ and the tag space $\mathcal{T}$, a nonce-based MAC scheme MAC consists of two functions: the tagging function $\mathsf{MAC}.\mathcal{T} : \mathcal{K} \times \mathcal{N} \times \mathcal{M} \to \mathcal{T}$ and

the verification function $\mathsf{MAC}.\mathcal{V} : \mathcal{K} \times \mathcal{N} \times \mathcal{M} \times \mathcal{T} \to \{\top, \bot\}$. A tag for $(K, N, M) \in \mathcal{K} \times \mathcal{N} \times \mathcal{M}$ is derived as $T = \mathsf{MAC}.\mathcal{T}_K(N, M)$. The tuple (N, M, T) is considered to be authentic if $\mathsf{MAC}.\mathcal{V}_K(N, M, T) = \top$, and otherwise it is rejected. The security of MAC is defined as the probability that an adversary $\mathcal{A}$ creates a successful forgery by accessing $\mathsf{MAC}.\mathcal{T}_K$ and $\mathsf{MAC}.\mathcal{V}_K$. The security measure is $\mathbf{Adv}^{\mathsf{mac}}_{\mathsf{MAC}}(\mathcal{A}) := \Pr[\mathcal{A}^{\mathsf{MAC}.\mathcal{T}_K, \mathsf{MAC}.\mathcal{V}_K} \text{forges}]$, which means $\mathcal{A}$ receives $\top$ from $\mathsf{MAC}.\mathcal{V}_K$ by querying (N', M', T') while (N', M') has never been queried to $\mathsf{MAC}.\mathcal{T}_K$. Here, $\mathcal{A}$ is assumed to be nonce-respecting, that is, the nonces in the tagging queries are distinct. The nonces in the verification queries have no restriction, and $\mathcal{A}$ can repeat or reuse a nonce that was used by a tagging query.

D. Authenticated Encryption

Authenticated encryption (AE) [19] is used to ensure the privacy and authenticity of input data simultaneously. A nonce-based AE scheme AE consists of two functions: the encryption function $\mathsf{AE}.\mathcal{E} : \mathcal{K} \times \mathcal{N} \times \mathcal{M} \to \mathcal{M} \times \mathcal{T}$ and the decryption function $\mathsf{AE}.\mathcal{D} : \mathcal{K} \times \mathcal{N} \times \mathcal{M} \times \mathcal{T} \to \mathcal{M} \cup \{\bot\}$. A ciphertext and a tag for $(K, N, M) \in \mathcal{K} \times \mathcal{N} \times \mathcal{M}$ are derived as $(C, T) = \mathsf{AE}.\mathcal{E}_K(N, M)$. The tuple (N, C, T) is considered to be authentic if $\mathsf{AE}.\mathcal{D}_K(N, C, T)$ returns the message $M \neq \bot$, and otherwise it is rejected. The security of AE is evaluated by two criteria: privacy and authenticity. The privacy advantage is the probability that the adversary successfully distinguishes $\mathsf{AE}.\mathcal{E}_K$ from the random oracle $\$(*, *)$. For any query (N, M), if $(C, T) = \mathsf{AE}.\mathcal{E}_K(N, M)$, $\$(N, M)$ returns random bits of length $|C| + |T|$. Thus, $\mathbf{Adv}^{\mathsf{priv}}_{\mathsf{AE}}(\mathcal{A}) := |\Pr[\mathcal{A}^{\mathsf{AE}.\mathcal{E}_K} \to 1] - \Pr[\mathcal{A}^{\$} \to 1]|$. The authenticity advantage is the probability that the adversary creates a successful forgery by accessing $\mathsf{AE}.\mathcal{E}_K$ and $\mathsf{AE}.\mathcal{D}_K$. It is defined as $\mathbf{Adv}^{\mathsf{auth}}_{\mathsf{AE}}(\mathcal{A}) := \Pr[\mathcal{A}^{\mathsf{AE}.\mathcal{E}_K, \mathsf{AE}.\mathcal{D}_K} \text{forges}]$, which means the probability that $\mathcal{A}$ receives $M' \neq \bot$ from $\mathsf{AE}.\mathcal{D}_K$ by querying (N', C', T') while (N', M') has never been queried to $\mathsf{AE}.\mathcal{E}_K$. For both advantages, we assume the adversary is nonce-respecting in encryption queries. For authenticity, however, there is no restriction on nonce in the decryption queries, that is, $\mathcal{A}$ may repeat a nonce or reuse a nonce that was used in an encryption query.

E. Tree-Based Memory Encryption Scheme

We assume two regions in storage memory: on-chip and off-chip areas. The former is assumed to be secure in which the adversary cannot eavesdrop or tamper the stored data. The latter can be attacked by an adversary who may perform eavesdropping (getting information of plaintext from ciphertext), tampering (modify the ciphertext without being detected), and replay (replacing the ciphertext with an old legitimate one). As mentioned in the introduction, tampering can be detected by simply applying a MAC to each data unit and storing the nonce and tag off-chip. If we use a nonce-based AE instead, it also prevents eavesdropping. However, these means are not sufficient to protect from replay attacks since the adversary can perform a replay on the (nonce, ciphertext, tag) tuple. Since the on-chip area is generally much more expensive than the off-chip area, it is desirable to thwart all of these attacks with as small an on-chip area as possible. In addition, since the off-chip area is assumed to have large capacity, it is desirable to be able to perform tampering/replay detection and plaintext recovery with fewer memory accesses when only a part of the off-chip area is accessed.

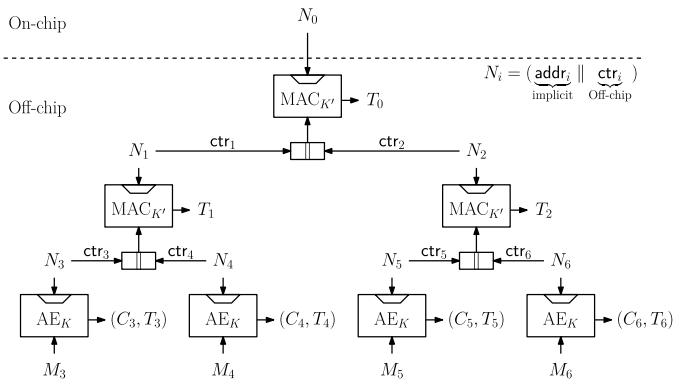


Fig. 1. An example of PAT2 with tree depth 2 and number of branches 2. A trapezoid in a box denotes the nonce input, and a box with $\|$ denotes concatenation.

As described earlier, Merkle hash tree [3] (with additional encryption layer for confidentiality) is a classical solution. It divides memory data into small chunks and associates their hashes with each leaf node of the tree. Every non-leaf node is associated with a hash of the hash values of its child nodes. By storing hash value of a root node on-chip and the others off-chip, the tampering and replay attacks can be detected. A similar scheme can be built by using MACs instead of hash functions by storing the key on-chip. Among such schemes, Parallelizable Authentication Tree (PAT) proposed by Hall and Jutla [4] (hereafter referred to as HJ05) is quite efficient for its parallelizability of both verify and update operations. It assigns a nonce to each node and stores the nonce associated with a root node in the on-chip area. PAT utilizes a MAC to compute a tag by taking the nonce assigned to its own node and nonces in its corresponding child nodes.[3]

In this paper, we use the term *ME tree* to refer to the tree-based ME scheme that also encrypts the leaf nodes. We introduce a generic construction of an ME tree, PAT2 (Fig. 1). It is mostly identical to PAT, but achieves confidentiality of memory by applying an AE scheme to the leaf nodes,[4] and it splits any nonce of PAT associated with a node into two values: an address and a local counter. The former is the memory address of the node, and the latter is a counter exclusively assigned to the node.

Fig. 1 shows an example of PAT2. Each nonce N_i assigned to node i consists of the address addr_i and the local counter ctr_i, which is initialized to 0 for all nodes. Memory data is split into four units (M_3 to M_6). After initialization, the tree keeps ctr_i, T_i for $i = 1, \ldots, 6$, and C_j for $j = 3, \ldots, 6$ at the off-chip area, and ctr_0 at the on-chip area. When verifying

[3]To be more precise, [4] proposed to use a general deterministic MAC with input being prepended by a nonce, which is a typical way to convert a deterministic MAC into a nonce-based one.

[4]In fact, An ePrint version of HJ05 [20] specifies a combination of MAC and AE schemes for confidentiality of leaf data.

a piece of data, say M_3, we check if $\mathrm{AE}.\mathcal{D}_K(N_3, C_3, T_3)$ is authentic and $\mathrm{MAC}.\mathcal{V}_{K'}(N_i, \mathsf{ctr}_{2i+1} \,\|\, \mathsf{ctr}_{2i+2}, T_i) = \top$ for $i = 0, 1$. If all hold, M_3 is considered to be authentic. When updating M_3, we first perform the above verification procedure, increment the corresponding local counters (ctr_0, ctr_1 and ctr_3), and then renew (C_3, T_3), T_1, and T_0. Note that the steps in the verification and update procedures are independent and thus parallelizable. This is a crucial advantage of PAT/PAT2 over the classical hash tree, which only allows parallel verification. In addition, since an address is given from the outer legitimate system anyway, it does not need to be explicitly stored and it is always authentic. Therefore PAT2 enables the MAC input and off-chip overhead from the original PAT to be reduced. To the best of our knowledge, this technique was first proposed by [5]. In fact, by specifying the tree structures and the MAC and AE schemes, the resulting scheme is mostly identical to SIT. Therefore, PAT2 can be viewed as an abstraction of SIT. We consider PAT2 as our baseline scheme for its simple structure and efficiency, and present our scheme based on it in Section IV.

A number of ME trees that better handle the various criteria (except for latency) have been proposed. TEC-tree [6] provides confidentiality by encrypting data stored in all nodes. MAES [21] provides security against differential power analysis attacks. VAULT [11] and Morphable Counter [12] reduce the overhead of off-chip memory and are suitable for protecting large memory (*e.g.*, GBytes); however there is a tradeoff with the average latency because their counters are compressed.

III. Components of ELM

To achieve low-latency operation, we designed dedicated MAC (PXOR-MAC) and AE (Flat-OCB).

A. PXOR-MAC: Incremental MAC

Fig. 2 shows the tagging function of PXOR-MAC using an n-bit block cipher E_K. The second key $K' \xleftarrow{\$} \{0,1\}^n$ is independent of K. The algorithm of $\mathrm{PXOR\text{-}MAC}.\mathcal{T}$ is obtained by replacing a TBC $\widetilde{E}$ of Alg. 1 with the following block cipher-based TBC.

$$\widetilde{E}^{0^n,i,j}_{K,K'}(M) = E_K(M \oplus K' \cdot i \oplus j \cdot E_K(0^n)).$$

The verification function is trivially defined and thus omitted here. We let $|N| = n$ and $|T| = \tau$. For simplicity, we exclude the case of partial blocks.

1) Properties: Since $L = E_K(0^n)$ can be computed in advance, the latency of tag computation is essentially a sum of the latencies of n-bit multiplication ($K' \cdot i$ for the block index i) and E_K. The former can be large if i has large variations; however, m is usually not too large in practice, even when the total memory size is huge. Typically, m is upper-bounded by the number of branches, *e.g.*, 2^7 according to [12]. The hardware implementation is much more efficient than a full multiplier (using Gray code; see Section V). Consequently, the latency of mask computation becomes negligible, and PXOR-MAC has the optimal latency of one E_K call for tagging and verification functions thanks to the full parallelizability of the block cipher calls.

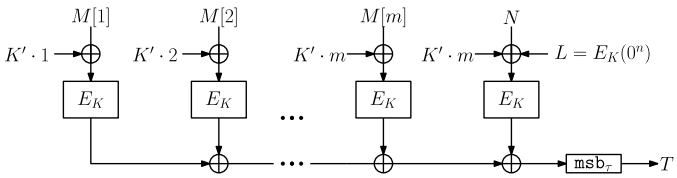


Fig. 2. PXOR-MAC.

PXOR-MAC is an Inc-MAC [13], which enables efficient tag computation when a small number of message blocks are changed. When a message block is changed together with a new nonce, the new tag is obtained by encrypting the corresponding blocks (*i.e.*, XOR of the message block and its mask) for both old and new ones, and taking an XOR of them and the old tag. We can further improve the incremental property of PXOR-MAC when used in PAT2. The MAC update function in the update procedure of PAT2 also invokes the MAC verification function, yielding some redundant E_K calls; thus, we can eliminate them. The resulting combined (verification and update) procedure, denoted by $\mathrm{PXOR\text{-}MAC}.\mathcal{VU}$, is shown in Alg. 2. It takes old nonce N_o, old plaintext M_o, old tag T_o, new nonce N_n, and new plaintext M_n. It outputs new tag T_n such that $T_\mathrm{n} = \mathrm{PXOR\text{-}MAC}.\mathcal{T}_{K,K'}(N_\mathrm{n}, M_\mathrm{n})$ holds if T_o is authentic; otherwise, it outputs $\perp$. For simplicity, Alg. 2 assumes that M_o and M_n have m blocks. For example, when one message block and nonce are changed, (which is the case of ELM), $\mathrm{PXOR\text{-}MAC}.\mathcal{VU}$ needs only $m + 3$ E_K calls except for mask derivation, while PXOR-MAC without incremental update needs $2m$. Even if we use the incremental update feature of PXOR-MAC after verification, it still needs $m + 5$ calls. This difference is not negligible as update occurs at all nodes on the path.

Algorithm 1 $\mathrm{PXOR\text{-}MAC\text{-}T}.\mathcal{T}_{\widetilde{E}}(N, M)$

$M[1] \,\|\, \cdots \,\|\, M[m] \xleftarrow{n} M$, $T \leftarrow 0^\tau$
for $1 \le i \le m$ **do**
  $T \leftarrow T \oplus \mathtt{msb}_\tau(\widetilde{E}^{0^n,i,0}_{K,K'}(M[i]))$
$T \leftarrow T \oplus \mathtt{msb}_\tau(\widetilde{E}^{0^n,m,1}_{K,K'}(N))$
return T

Algorithm 2 $\mathrm{PXOR\text{-}MAC}.\mathcal{VU}_{E_K}(N_\mathrm{o}, M_\mathrm{o}, T_\mathrm{o}, N_\mathrm{n}, M_\mathrm{n})$

$L \leftarrow E_K(0^n)$, $T' \leftarrow 0^\tau$, $T_\mathrm{n} \leftarrow T_\mathrm{o}$
$M_\mathrm{o}[1] \,\|\, \cdots \,\|\, M_\mathrm{o}[m] \xleftarrow{n} M_\mathrm{o}$, $M_\mathrm{n}[1] \,\|\, \cdots \,\|\, M_\mathrm{n}[m] \xleftarrow{n} M_\mathrm{n}$
for $1 \le i \le m$ **do**
  $S \leftarrow \mathtt{msb}_\tau(E_K(M_\mathrm{o}[i] \oplus K' \cdot i))$, $T' \leftarrow T' \oplus S$
  if $M_\mathrm{o}[i] \ne M_\mathrm{n}[i]$ **then**
    $T_\mathrm{n} \leftarrow T_\mathrm{n} \oplus S \oplus \mathtt{msb}_\tau(E_K(M_\mathrm{n}[i] \oplus K' \cdot i))$
$S \leftarrow \mathtt{msb}_\tau(E_K(N_\mathrm{o} \oplus K' \cdot m \oplus L))$, $T' \leftarrow T' \oplus S$
$T_\mathrm{n} \leftarrow T_\mathrm{n} \oplus S \oplus \mathtt{msb}_\tau(E_K(N_\mathrm{n} \oplus K' \cdot m \oplus L))$
if $T' \ne T_\mathrm{o}$ **then**
  return $\perp$
return T_n

2) Security: We assume the underlying block cipher of PXOR-MAC is an n-bit URP P, which is denoted by $\mathrm{PXOR\text{-}MAC}_\mathsf{P}$.

Theorem 1: Let $\sigma_{\mathtt{mac}}$ be the number of accesses to P in the MAC game s.t. $\sigma_{\mathtt{mac}} \leq 2^{n-1}$, and q_v be the number of queries to the verification oracle. We obtain

$$\mathbf{Adv}^{\mathtt{mac}}_{\mathsf{PXOR\text{-}MAC}_{\mathsf{P}}}(\mathcal{A}) \leq \frac{2q_v}{2^\tau} + \frac{4.5\sigma^2_{\mathtt{mac}}}{2^n}.$$

Note that this advantage is an information-theoretic bound. The computational counterpart is derived from our bound. Since this is fairly standard [22], we omit it here. This theorem means that PXOR-MAC has the security bound of $\min\{O(2^\tau), O(2^{n/2})\}$, which is almost the same as that of MAC used in SIT. PXOR-MAC only fulfills "basic" Inc-MAC security [13], which requires that the update function of Inc-MAC always takes benign inputs (in our case this corresponds to require $T_{\mathrm{o}} = \mathsf{PXOR\text{-}MAC}.\mathcal{T}_{K,K'}(N_{\mathrm{o}}, M_{\mathrm{o}})$ holds whenever $\mathcal{VU}$ outputs $T_{\mathrm{n}} \neq \perp$). We stress that, while a stronger, "tamper-proof" security notion was also defined by [13], basic security is enough for our case since the adversary cannot intervene in the update procedure.

3) Proof of Theorem 1: Let $\widetilde{\mathsf{P}}$ be an n-bit TURP having the same tweak space as $\widetilde{E}$, and $\widetilde{\mathsf{E}}$ be a TBC involving P and an independent key K', defined as $\widetilde{\mathsf{E}}^{0^n,i,j}(M) = \mathsf{P}(M \oplus K' \cdot i \oplus j \cdot \mathsf{P}(0^n))$. Then we obtain

$$\mathbf{Adv}^{\mathtt{mac}}_{\mathsf{PXOR\text{-}MAC}_{\mathsf{P}}}(\mathcal{A}) \leq \mathbf{Adv}^{\mathtt{mac}}_{\mathsf{PXOR\text{-}MAC\text{-}T}_{\widetilde{\mathsf{P}}}}(\mathcal{A}) + \mathbf{Adv}^{\mathrm{tprp}}_{\widetilde{\mathsf{E}}}(\mathcal{B}),$$

where $\mathcal{B}$ is the adversary against $\widetilde{\mathsf{E}}$. Regarding the first term, we can prove $\mathbf{Adv}^{\mathtt{mac}}_{\mathsf{PXOR\text{-}MAC\text{-}T}_{\widetilde{\mathsf{P}}}}(\mathcal{A}) \leq 2q_v/2^\tau$ in the same manner as PMAC [18], and thus we omit. We then evaluate the second term. We define the offset function F as $F_{K'}((i,j), \mathsf{P}(0^n)) = K' \cdot i \oplus j \cdot \mathsf{P}(0^n)$, where $i \in \{1,2,\ldots\}$, $j \in \{0,1\}$. Then, $\widetilde{\mathsf{E}}^{0^n,i,j}(M) = \mathsf{P}(M \oplus F_{K'}((i,j), \mathsf{P}(0^n)))$ holds for any (i, j, M). We introduce the definition and the lemma for offset functions. They are simplified ones of Definition 4.1 and Theorem 4.1 in [23].

Definition 1: Let $V \xleftarrow{\$} \{0,1\}^n$. An offset function F is said to be $(\varepsilon, \gamma, \rho)$-uniform if F satisfies the following conditions.

$$\max_{l \neq l', \delta \in \{0,1\}^n} \Pr[F(l,V) \oplus F(l',V) = \delta] \leq \varepsilon,$$
$$\max_{l, \delta \in \{0,1\}^n} \Pr[F(l,V) = \delta] \leq \gamma,$$
$$\max_{l, \delta \in \{0,1\}^n} \Pr[F(l,V) \oplus V = \delta] \leq \rho.$$

Lemma 1: Suppose that $\widetilde{\mathsf{E}}$ uses an $(\varepsilon, \gamma, \rho)$-uniform offset function F. We obtain the following evaluation.

$$\mathbf{Adv}^{\mathrm{tprp}}_{\widetilde{\mathsf{E}}}(\mathcal{B}) \leq q^2 \left(2\varepsilon + \gamma + \rho + \frac{1}{2^{n+1}}\right),$$

where q is the number of encryption queries s.t. $q \leq 2^{n-1}$. Since K' and $\mathsf{P}(0^n)$ are uniformly random and independent, $(\varepsilon, \gamma, \rho) = (1/2^n, 1/2^n, 1/2^n)$ trivially holds. Thus, we obtain $\mathbf{Adv}^{\mathrm{tprp}}_{\widetilde{\mathsf{E}}}(\mathcal{B}) \leq 4.5\sigma^2_{\mathtt{mac}}/2^n$. This concludes the proof.

B. Flat-OCB: Low-Latency AE

To encrypt the leaf data of PAT2, an AE scheme is needed. We take OCB as the baseline AE for its efficiency: it needs m plus a few block cipher (E_K) calls to process m blocks (while a generic composition of Encryption and MAC needs at least $2m$ calls). Because these m calls are parallelizable, OCB has quite a low latency. However, when looking into the dependency among "a few block cipher calls" mentioned above, we find a problem in the decryption. In more detail, the final E_K call in the OCB decryption takes the sum of the plaintext blocks after the plaintext blocks are obtained by ECB-like decryption (see *e.g.*, [16, Fig. 5]). This results in one call that cannot be computed in parallel, and adds a significant latency compared to the encryption. This is not desirable for applications requiring low latency. We present a solution to this problem. Concretely, we first propose an improved version of the TBC-based interpretation of OCB (ΘCB3 in [16], hereafter referred to as ΘCB), which we call Flat-ΘCB and show its block cipher-based instantiation, Flat-OCB. It has a smaller decryption latency than OCB, while keeping the same encryption latency.

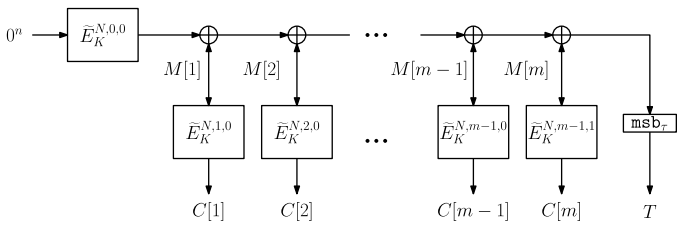


Fig. 3. Flat-ΘCB.

Algorithm 3 Flat-ΘCB.$\mathcal{E}_{\widetilde{E}}(N, M)$

$M[1] \| \cdots \| M[m] \xleftarrow{n} M$, $T \leftarrow \mathtt{msb}_\tau(\widetilde{E}_K^{N,0,0}(0^n))$
for $1 \leq i \leq m-1$ **do**
  $C[i] \leftarrow \widetilde{E}_K^{N,i,0}(M[i])$, $T \leftarrow T \oplus \mathtt{msb}_\tau(M[i])$
$C[m] \leftarrow \widetilde{E}_K^{N,m-1,1}(M[m])$, $T \leftarrow T \oplus \mathtt{msb}_\tau(M[m])$
$C \leftarrow C[1] \| \cdots \| C[m]$
return C, T

Algorithm 4 $\mathsf{MASK}_{K_1,K_2,K_3,K_4}(N)$

$N_1 \leftarrow \mathtt{msb}_{n/2}(N)$, $N_2 \leftarrow \mathtt{lsb}_{n/2}(N)$
return $\Delta \leftarrow (N_1 \cdot K_1 \| N_2 \cdot K_2) \oplus (N_2 \cdot K_3 \| N_1 \cdot K_4)$

Alg. 3 and Fig. 3 show the encryption of Flat-ΘCB. Decryption is naturally obtained, thus omitted. It is based on n-bit TBC, $\widetilde{E}$, using n-bit nonce. We exclude the case of partial blocks for simplicity. Flat-ΘCB is similar to ΘCB, however, the crucial difference is how the tag T is generated. While ΘCB encrypts the checksum $M[1] \oplus M[2] \oplus \cdots \oplus M[m]$ using $\widetilde{E}$ to produce T, ours first encrypts N and then takes a sum with the checksum. To build a block cipher-based AE, we instantiate $\widetilde{E}$ with an n-bit block cipher E_K as

$$\widetilde{E}_K^{N,i,j}(M) = E_K(M \oplus \Delta \oplus 2^i 3^j L) \oplus \Delta \oplus 2^i 3^j L, \quad (1)$$

where $i \in \{0,1,2,\ldots\}$, $j \in \{0,1\}$, $L = E_K(0^n)$ and Δ is derived from N. The value Δ needs to be pairwise independent for a pair of distinct inputs, and this may be realized by various means. To keep the compatibility with SIT, we generate Δ using $n/2$-bit multiplications as Alg. 4 for some even n

TABLE I

COMPARISON OF AE MODES. m IS THE NUMBER OF INPUT BLOCKS, AND "MUL" IN THE LATENCY COLUMNS DENOTES $n/2$-bit FULL GF MULTIPLICATION. THE LAST COLUMN DENOTES THE TOTAL ON-CHIP SIZE. FOR SIMPLICITY, THE ENCRYPTION AND DECRYPTION LATENCY OF (T)BC ARE ASSUMED TO BE IDENTICAL, AND (T)BC HAS n-bit BLOCK SIZE AND n-bit KEY

Scheme	Enc latency	Dec latency	Circuit size to achieve the best latency	The total number of primitive calls	Total size of key and preprocessed data (bits)
ΘCB [16]	1 TBC	2 TBCs	$m+1$ TBCs	$m+1$ TBCs	n
Flat-ΘCB (This work)	1 TBC	1 TBC	$m+1$ TBC	$m+1$ TBC	n
OCB [16]	2 BCs	3 BC	$m+1$ BC	$m+2$ BC	n
SIT-AE [7]	1 BC + 1 Mul.	max{1 BC, 1 Mul.}	$m+1$ BC + $2m$ Mul	$m+1$ BC + $2m$ Mul	$2n+mn$
Flat-OCB (This work)	1 BC + 1 Mul	1 BC + 1 Mul	$m+1$ BC + 4 Mul	$m+1$ BC + 4 Mul	$4n$

and random and independent $n/2$-bit keys (K_1, K_2, K_3, K_4). We define TBC as the block cipher-based TBC defined in (1) with MASK (Alg. 4). We obtain Flat-OCB by instantiating $\widetilde{E}$ of Flat-ΘCB with TBC.

1) Properties: Table I compares Flat-ΘCB and Flat-OCB with other AEs. SIT-AE is a GCM-like AE inside SIT [8]. Notably, as a mode of TBC, the latency of Flat-ΘCB is the lowest achievable – one TBC call – for both encryption and decryption, which is achieved for the first time. In contrast, the decryption latency of ΘCB is two for the aforementioned reason. For other properties, Flat-ΘCB has the same figures as those of ΘCB. Similarly, Flat-OCB has the same encryption latency as OCB, and has the smaller decryption latency than OCB. SIT-AE has even smaller decryption latency, however, it requires $2m$ multiplier circuits to achieve this, whereas Flat-OCB requires only 4 multipliers.[5] Moreover, the key size of SIT-AE is linear to m, which will have a non-negligible impact. In contrast, Flat-OCB requires $3n$-bit key, plus the n-bit preprocessed data $L = E_K(0^n)$ used inside TBC.

2) Security: We assume that the underlying block cipher is an n-bit URP P, and only present an information-theoretic bound based on P.

Theorem 2: The advantages of Flat-ΘCB and Flat-OCB are obtained as follows:

$$\mathbf{Adv}^{\mathrm{priv}}_{\mathsf{Flat\text{-}\Theta CB}_{\widetilde{\mathsf{P}}}}(\mathcal{A}) = 0, \quad \mathbf{Adv}^{\mathrm{auth}}_{\mathsf{Flat\text{-}\Theta CB}_{\widetilde{\mathsf{P}}}}(\mathcal{A}^{\pm}) \le \frac{2q_d}{2^{\tau}},$$

$$\mathbf{Adv}^{\mathrm{priv}}_{\mathsf{Flat\text{-}OCB}_{\mathsf{P}}}(\mathcal{A}) \le \frac{4.5\sigma^2_{\mathtt{priv}}}{2^n},$$

$$\mathbf{Adv}^{\mathrm{auth}}_{\mathsf{Flat\text{-}OCB}_{\mathsf{P}}}(\mathcal{A}^{\pm}) \le \frac{2q_d}{2^{\tau}} + \frac{4.5\sigma^2_{\mathtt{auth}}}{2^n},$$

where $\sigma_{\mathtt{priv}}$, $\sigma_{\mathtt{auth}}$, and q_d are the parameters for $\mathcal{A}$ and $\mathcal{A}^{\pm}$. The parameter $\sigma_{\mathtt{priv}}$ (resp. $\sigma_{\mathtt{auth}}$) is the number of accesses to P in the privacy (resp. authenticity) game s.t. $\sigma_{\mathtt{priv}}, \sigma_{\mathtt{auth}} \le 2^{n-1}$. The parameter q_d is the number of queries to the decryption oracle.

Flat-ΘCB has the same advantages as those of ΘCB ($\mathbf{Adv}^{\mathrm{priv}}_{\Theta\mathsf{CB}_{\widetilde{\mathsf{P}}}}(\mathcal{A}) = 0$, $\mathbf{Adv}^{\mathrm{auth}}_{\Theta\mathsf{CB}_{\widetilde{\mathsf{P}}}}(\mathcal{A}^{\pm}) \le (2^{n-\tau}q_d)/(2^n - 1)$), hence there is no security penalty, up to the constant. The security of Flat-OCB is comparable to those of OCB and SIT-AE. Assuming $n = 128$ and $\tau = 64$, Flat-OCB has 64-bit data security that is comparable to OCB and SIT-AE.

[5]The proper number of multipliers for hardware implementation depends on the system constraint/architecture. See also Section V.

3) Proof of Theorem 2: Let $\widetilde{\mathsf{P}}$ be an n-bit TURP having the same tweak space as TBC. Let $\widetilde{\mathsf{E}}$ be the TBC mode obtained by replacing E_K in TBC with P. Then we obtain the following inequations.

$$\mathbf{Adv}^{\mathrm{priv}}_{\mathsf{Flat\text{-}OCB}_{\mathsf{P}}}(\mathcal{A}) \le \mathbf{Adv}^{\mathrm{priv}}_{\mathsf{Flat\text{-}\Theta CB}_{\widetilde{\mathsf{P}}}}(\mathcal{A}) + \mathbf{Adv}^{\mathrm{tprp}}_{\widetilde{\mathsf{E}}}(\mathcal{B}),$$

$$\mathbf{Adv}^{\mathrm{auth}}_{\mathsf{Flat\text{-}OCB}_{\mathsf{P}}}(\mathcal{A}^{\pm}) \le \mathbf{Adv}^{\mathrm{auth}}_{\mathsf{Flat\text{-}\Theta CB}_{\widetilde{\mathsf{P}}}}(\mathcal{A}^{\pm}) + \mathbf{Adv}^{\mathrm{tsprp}}_{\widetilde{\mathsf{E}}}(\mathcal{B}^{\pm}),$$

where $\mathcal{B}$ and $\mathcal{B}^{\pm}$ are adversaries against $\widetilde{\mathsf{E}}$. To derive the advantages of $\mathcal{B}$ and $\mathcal{B}^{\pm}$, we use the tsprp version of the methodology in [23], which is almost the same as the tprp one (*i.e.*, Def. 1 and Lem. 1), thus we omit the details. We define the offset function F of $\widetilde{\mathsf{E}}$ as $F((N, i, j), \mathsf{P}(0^n)) = \mathsf{MASK}(N) \oplus 2^i \cdot 3^j \mathsf{P}(0^n)$. Since keys of MASK and $\mathsf{P}(0^n)$ are uniformly random and independent, and MASK is pairwise independent, $(\varepsilon, \gamma, \rho) = (1/2^n, 1/2^n, 1/2^n)$ trivially holds. Thus, $\mathbf{Adv}^{\mathrm{tprp}}_{\widetilde{\mathsf{E}}}(\mathcal{B}) \le 4.5\sigma^2_{\mathtt{priv}}/2^n$, and $\mathbf{Adv}^{\mathrm{tsprp}}_{\widetilde{\mathsf{E}}}(\mathcal{B}^{\pm}) \le 4.5\sigma^2_{\mathtt{auth}}/2^n$ hold.

We then evaluate the security bounds of Flat-ΘCB. For the privacy, $\mathbf{Adv}^{\mathrm{priv}}_{\mathsf{Flat\text{-}\Theta CB}_{\widetilde{\mathsf{P}}}}(\mathcal{A}) = 0$ holds since every TURP call in the game takes different tweaks. For the authenticity, We start with the case $q_d = 1$. Let $\{(N_1, M_1, C_1, T_1), \ldots, (N_{q_e}, M_{q_e}, C_{q_e}, T_{q_e})\}$ be the transcript in encryption queries, and (N', C', T') be the decryption query. If $\forall i \in [q_e], N' \ne N_i$, the probability $\mathcal{A}^{\pm}$ forges is at most $1/2^{\tau}$. We then evaluate the case $\exists \alpha \in [q_e], N' = N_{\alpha}$. If $|C_{\alpha}|_n \ne |C'|_n := m'$, the TURP which decrypts $C'[m']$ takes a different tweak from all tweaks invoked in the encryption queries. Thus, the probability $\mathcal{A}^{\pm}$ forges is at most $1/2^{\tau}$. If $|C_{\alpha}|_n = |C'|_n = m'$, all tweaks invoked in the decryption query are the same as those in the α-th encryption query. However, $C'[i] \ne C_{\alpha}[i]$ holds for $\exists i \in [m']$, and thus, the probability $\mathcal{A}^{\pm}$ successfully guesses $\mathtt{msb}_{\tau}(M^*[i])$, where $M^*[i]$ is the decrypted value of $C'[i]$, is at most $2/2^{\tau}$. From the above, we obtain the advantage for the case $q_d = 1$: $\mathbf{Adv}^{\mathrm{auth}}_{\mathsf{Flat\text{-}\Theta CB}_{\widetilde{\mathsf{P}}}}(\mathcal{A}^{\pm}) \le 2/2^{\tau}$. We apply the standard conversion from single to multiple decryption queries [24] and obtain the bound $q_d\,(2/2^{\tau})$ for $q_d \ge 1$. This concludes the proof of Flat-ΘCB.

IV. ELM

ELM is based on PAT2, where the internal MAC and AE are instantiated by PXOR-MAC and Flat-OCB.

A. Notations for the Tree

Let $b \ge 2$ be the number of branches, and let d be the tree depth, where the root has depth 0 and a leaf node has

depth d. ELM uses a balanced tree, so we have b^d leaves. The entire memory (plaintext) is divided into ℓ-bit chunks and i-th leaf node is denoted by $\mathsf{leaf}(i)$ for $i \in [b^d]$. The whole input memory is $M = M[1] \| \cdots \| M[b^d]$ s.t. $|M[i]| = \ell$ for $i \in [b^d]$ and $M[i]$ is associated with $\mathsf{leaf}(i)$. The ciphertext chunk corresponding to $M[i]$ is denoted by $C[i]$, which is stored in $\mathsf{leaf}(i)$. For node u, the memory address, the counter, and the tag are denoted by $\mathsf{add}(u)$, $\mathsf{ctr}(u)$, and $\mathsf{tag}(u)$, respectively, where $|\mathsf{add}(u)| = \alpha$, $|\mathsf{ctr}(u)| = \beta$, and $|\mathsf{tag}(u)| = \tau$. The entire data to construct the tree stored in the on-chip and off-chip areas is denoted by σ, which includes $C[i]$ for $i \in [b^d]$, $\mathsf{ctr}(u)$, and $\mathsf{tag}(u)$ for all nodes u. Note that we exclude the key and the preprocessed data from σ. As we adopt PAT2, we exclude the node addresses from σ and assume they are given by the system when needed. For the root node u_{r}, we store $\mathsf{ctr}(u_{\mathsf{r}})$ on-chip. The leftover data of σ is stored off-chip. We may also use σ to refer to the tree construction itself. We also write a node u, leaf node, plaintext chunk, and ciphertext chunk of σ as u^σ, $\mathsf{leaf}(i)^\sigma$, $M^\sigma[i]$, and $C^\sigma[i]$, respectively. If no confusion is possible, we omit their superscript σ. For any non-leaf node u^σ and $i \in [b]$, $\mathsf{ch}_i(u^\sigma)$ denotes its i-th child node.

Algorithm 5 IT: Initialization of a Tree Construction σ

Input $M = M[1] \| \cdots \| M[b^d]$ s.t. $|M[i]| = \ell$ for $i \in [b^d]$
Output σ
1: $\sigma \leftarrow 0^{\left(\frac{b^d-1}{b-1}\right)\times(\beta+\tau)+b^d\times(\ell+\beta+\tau))}$
2: **for** all nodes u **do**
3: $\quad \mathsf{ctr}(u^\sigma) \leftarrow 0^{\beta-1}1$
4: **for** $1 \le i \le b^d$ **do**
5: $\quad (C[i], \mathsf{tag}(\mathsf{leaf}(i)^\sigma))$
$\quad\quad \leftarrow \mathsf{Flat\text{-}OCB}.\mathcal{E}(\mathsf{add}(\mathsf{leaf}(i)^\sigma) \| \mathsf{ctr}(\mathsf{leaf}(i)^\sigma), M[i])$
6: $M' \leftarrow \mathsf{ctr}(\mathsf{ch}_1(u_{\mathsf{r}}^\sigma)) \| \cdots \| \mathsf{ctr}(\mathsf{ch}_b(u_{\mathsf{r}}^\sigma))$
7: $\mathsf{tag}(u_{\mathsf{r}}^\sigma) \leftarrow \mathsf{PXOR\text{-}MAC}.\mathcal{T}(\mathsf{add}(u_{\mathsf{r}}^\sigma) \| \mathsf{ctr}(u_{\mathsf{r}}^\sigma), M')$
8: **for** all intermediate nodes u **do**
9: $\quad \mathsf{tag}(u^\sigma) \leftarrow \mathsf{PXOR\text{-}MAC}.\mathcal{T}(\mathsf{add}(u^\sigma) \| \mathsf{ctr}(u^\sigma), M')$
10: **return** σ

Algorithm 6 VT: Checking the Validity of $\mathsf{leaf}(idx)$.

Input idx, σ
Output $\top$ or $\bot$
1: $(u_0^\sigma, \cdots, u_d^\sigma) \leftarrow$ path of nodes from root to specified leaf
(*i.e.*, u_0^σ is the root node u_{r}^σ, and u_d^σ is equal to $\mathsf{leaf}(idx)$.)
2: **for** $0 \le i \le d-1$ **do**
3: $\quad$ **if** $\mathsf{PXOR\text{-}MAC}.\mathcal{V}(\mathsf{add}(u_i^\sigma) \| \mathsf{ctr}(u_i^\sigma),$
$\quad\quad \mathsf{ctr}(\mathsf{ch}_1(u_i^\sigma)) \| \cdots \| \mathsf{ctr}(\mathsf{ch}_b(u_i^\sigma)), \mathsf{tag}(u_i^\sigma)) = \bot$ **then**
4: $\quad\quad$ **return** $\bot$
5: **if** $\mathsf{Flat\text{-}OCB}.\mathcal{D}(\mathsf{add}(u_d^\sigma) \| \mathsf{ctr}(u_d^\sigma), C[idx], \mathsf{tag}(u_d^\sigma)) = \bot$ **then**
6: $\quad$ **return** $\bot$
7: **return** $\top$

B. Specifications of ELM

ELM consists of three algorithms: IT, VT, and UT defined in Algs. 5, 6, and 7. ELM's key consists of the AE and MAC keys, which are chosen uniformly at random and independent. IT initializes the tree. It takes an input plaintext M, and outputs a tree σ. Here, σ consists of the local counters being initialized to zero, the tags for the non-leaf nodes, and the (ciphertext, tag) pairs for the leaf nodes. VT checks the validity of a specified leaf node. It is associated with a read operation. VT takes an index of a leaf node $idx \in [b^d]$ and a tree σ as input. It outputs $\top$ if all the verifications of PXOR-MAC and the decryption of Flat-OCB are successful, and otherwise $\bot$. UT renews plaintext chunk assigned to the specified leaf node, which is associated with a write operation. UT takes an index of leaf node idx, an update value (new plaintext) B s.t. $|B| = \ell$, and a tree σ as input. It returns a renewed tree $\tilde{\sigma}$ if the verification is successful, otherwise $\bot$. Note that it is essential for UT to check the validity of the data associated in node path in order to prevent a replay attack. If the verification in UT is bypassed, the adversary can roll back the value of $\mathsf{ctr}(\cdot)$ and mount a replay attack. For the verification and update of intermediate nodes in UT, we use $\mathsf{PXOR\text{-}MAC}.\mathcal{VU}$ to eliminate redundant computations. Similarly, Flat-OCB in lines 7–7 of Alg. 7 can eliminate some redundant field multiplications in deriving Δ by caching.

Algorithm 7 UT: Updating the Message of $\mathsf{leaf}(idx)$ to B.

Input idx, B, σ
Output $\tilde{\sigma}$ or $\bot$
1: $\tilde{\sigma} \leftarrow \sigma$, $(u_0, \cdots, u_d) \leftarrow$ path of nodes from root to specified leaf
2: **for** $0 \le i \le d$ **do**
3: $\quad \mathsf{ctr}(u_i^{\tilde{\sigma}}) \leftarrow \mathsf{ctr}(u_i^{\tilde{\sigma}}) + 1$
4: **for** $0 \le i \le d-1$ **do**
5: $\quad N_{\mathrm{o}} \leftarrow \mathsf{add}(u_i^\sigma) \| \mathsf{ctr}(u_i^\sigma)$, $N_{\mathrm{n}} \leftarrow \mathsf{add}(u_i^\sigma) \| \mathsf{ctr}(u_i^{\tilde{\sigma}})$
6: $\quad M_{\mathrm{o}} \leftarrow \mathsf{ctr}(\mathsf{ch}_1(u_i^\sigma)) \| \cdots \| \mathsf{ctr}(\mathsf{ch}_b(u_i^\sigma))$
7: $\quad M_{\mathrm{n}} \leftarrow \mathsf{ctr}(\mathsf{ch}_1(u_i^{\tilde{\sigma}})) \| \cdots \| \mathsf{ctr}(\mathsf{ch}_b(u_i^{\tilde{\sigma}}))$
8: $\quad \mathsf{tag}(u_i^{\tilde{\sigma}}) \leftarrow \mathsf{PXOR\text{-}MAC}.\mathcal{VU}(N_{\mathrm{o}}, M_{\mathrm{o}}, \mathsf{tag}(u_i^\sigma), N_{\mathrm{n}}, M_{\mathrm{n}})$
9: $\quad$ **if** $\mathsf{tag}(u_i^{\tilde{\sigma}}) = \bot$ **then**
10: $\quad\quad$ **return** $\bot$
11: **if** $\mathsf{Flat\text{-}OCB}.\mathcal{D}(\mathsf{add}(u_d^\sigma) \| \mathsf{ctr}(u_d^\sigma), C^\sigma[idx], \mathsf{tag}(u_d^\sigma)) = \bot$ **then**
12: $\quad$ **return** $\bot$
13: $(C^{\tilde{\sigma}}[idx], \mathsf{tag}(u_d^{\tilde{\sigma}})) \leftarrow \mathsf{Flat\text{-}OCB}.\mathcal{E}(\mathsf{add}(u_d^\sigma) \| \mathsf{ctr}(u_d^{\tilde{\sigma}}), B)$
14: **return** $\tilde{\sigma}$

C. Efficiency of ELM

ELM is designed to achieve low latency by utilizing the incremental property of MAC and full parallelizability of the cryptographic components and the tree structure. In particular, the incremental property greatly reduces the latency of UT. An Inc-MAC with basic security works (See Sec. III-A.1). Suppose $\alpha = \beta = n/2$ and some even b. One VT call needs $(1+2/b)d$ E_K calls for intermediate and root nodes. One UT call needs $(3+2/b)d$ E_K calls for intermediate and root nodes, while UT with a non-incremental MAC needs at least twice as many E_K calls as VT does. In addition, ELM is scalable for its constant on-chip size (the key and the preprocessed data, total $7n$ bits). However, (a generalized version of) SIT needs an on-chip area linear to b and β.

D. Security of ELM

To the best of our knowledge, the provable security of PAT2 have not been formally shown in the literature. Among

many ME proposals, the concrete provable security of the proposed scheme (as in the same manner to the seminal Bellare *et al*. [22]) is rarely shown, or even the formal, game-based security notion is often missing. It is true that HJ05 defines the security notion for PAT and proves the security, however, PAT has slightly different tree construction from PAT2, and more importantly, it lacks a confidentiality notion as the original PAT does not encrypt data. We remedy this situation by defining two appropriate security notions for ME trees: *privacy* and *unforgeability*. The former is for the confidentiality of plaintext and is defined analogously to the standard "privacy" notion for AE. The latter is mostly identical to the notion introduced by HJ05. We show the concrete security bounds of PAT2 under these notions. Although our analysis is not surprising, we think a formal security framework is important and will facilitate further studies on ME schemes. In particular, the security model is identical to what SIT considers.

1) Security Notion of ME Tree: Suppose that Tree is an ME tree scheme defined as a tuple of three functions: the initialization function IT, the verification function VT, and the update function UT. We assume that UT contains VT and performs VT first. Recall that $\mathsf{IT}(M) = \sigma$, $\mathsf{VT}(idx, \sigma) = \top$ or $\bot$, and $\mathsf{UT}(idx, B, \sigma) = \tilde{\sigma}$ or $\bot$. Also recall that σ includes data stored in the on-chip area.[6]

For the privacy of Tree, we define IT-\$ and UT-\$. They return their ciphertexts and tags to be stored in the leaf nodes as random strings whose lengths are the same as those of IT and UT, respectively. Regarding other variables, for example, the data associated with the intermediate nodes, they return the same outputs as IT and UT. The privacy security of Tree is defined as the probability that an adversary $\mathcal{A}$ successfully distinguishes (IT, UT) from (IT-\$, UT-\$). It is written as

$$\mathbf{Adv}^{\mathtt{pt}}_{\mathsf{Tree}}(\mathcal{A}) := |\Pr[\mathcal{A}^{\mathsf{IT},\mathsf{UT}} \to 1] - \Pr[\mathcal{A}^{\mathsf{IT\text{-}\$},\mathsf{UT\text{-}\$}} \to 1]|,$$

where $\mathcal{A}$ plays the following game.

1) $\mathcal{A}$ queries M to the tree initialization oracle (IT or IT-\$) and obtains $\tilde{\sigma}_0$.
2) $\mathcal{A}$ makes q adaptive queries to the update oracle (UT or UT-\$). Let $\{(idx_1, B_1, \sigma_1, \tilde{\sigma}_1), \ldots, (idx_q, B_q, \sigma_q, \tilde{\sigma}_q)\}$ be the transcript obtained by the update queries. Here, we assume that $\sigma_i = \tilde{\sigma}_{i-1}$ for $i \in [q]$ so that $\mathcal{A}$ can always obtain an updated tree, not $\bot$.
3) $\mathcal{A}$ guesses which oracle pair she has queried ((IT, UT) or (IT-\$, UT-\$)) and accordingly outputs a bit.

For the unforgeability of Tree, our definition follows [4]. It is defined as the advantage of an adversary $\mathcal{A}'$ querying IT and UT successfully distinguishes VT from $\bot_{\mathsf{T}}(\cdot,\cdot)$ which always returns $\bot$ for any inputs. The unforgeability advantage of $\mathcal{A}'$ is defined as

$$\mathbf{Adv}^{\mathtt{uft}}_{\mathsf{Tree}}(\mathcal{A}') := |\Pr[\mathcal{A}'^{\mathsf{IT},\mathsf{UT},\mathsf{VT}} \to 1] - \Pr[\mathcal{A}'^{\mathsf{IT},\mathsf{UT},\bot_{\mathsf{T}}} \to 1]|,$$

where $\mathcal{A}'$ plays the following game.

[6]Recall that σ does not include the secret key and the preprocessed data. In this paper, we do not assume the confidentiality of $\mathsf{Sec}(\sigma)$, thus the adversary can look into it. It is a weaker assumption than that assuming both the confidentiality and tamper freeness.

1) $\mathcal{A}'$ queries M to IT and obtains $\tilde{\sigma}_0$.
2) $\mathcal{A}'$ makes q' adaptive queries to UT. Let $\{(idx_1, B_1, \sigma_1, \tilde{\sigma}_1), \ldots, (idx_{q'}, B_{q'}, \sigma_{q'}, \tilde{\sigma}_{q'})\}$ be the transcript obtained by update queries. As well as the privacy game, we assume that $\sigma_i = \tilde{\sigma}_{i-1}$ for $i \in [q]$.
3) $\mathcal{A}'$ queries (idx', σ') to the verification oracle (VT or $\bot_{\mathsf{T}}$) and obtains $\top$ or $\bot$. Let $(u_0, \ldots, u_d)$ be the path of nodes from the root node to $\mathsf{leaf}(idx')$. To exclude a trivial win, we assume that there exists $i \in \{0, \ldots, d\}$ such that $u_i^{\sigma'}$ stores different data from that stored in $u_i^{\tilde{\sigma}_{q'}}$. Moreover, $\mathsf{Sec}(\sigma') = \mathsf{Sec}(\tilde{\sigma}_{q'})$ also must hold since the data in the on-chip area cannot be tampered.
4) $\mathcal{A}'$ guesses which oracle pair she has queried ((IT, UT, VT) or (IT, UT, $\bot_{\mathsf{T}}$)) and accordingly outputs a bit.

We stress that $\mathcal{A}'$ can perform a verification query s.t. $u_i^{\sigma'}$ stores the same data as that stored in $u_i^{\tilde{\sigma}_j}$ for $0 \le i \le d$ and $0 \le j \le q' - 1$, unless the data stored in $u_i^{\tilde{\sigma}_j}$ is the same as that stored in $u_i^{\tilde{\sigma}_{q'}}$ for all $i \in \{0, \ldots, d\}$ as described in the third operation of the above game. This condition is essential for the unforgeability notion to capture an adversary who performs a replay attack. We remark that the security notion $\mathbf{Adv}^{\mathtt{uft}}$ also captures the adversary who tampers the data before update, i.e., a forgery attack against UT, since UT contains VT.

2) Security Bounds of PAT2 and ELM: The algorithms of PAT2 are obtained by replacing PXOR-MAC and Flat-OCB in Algs. 5, 6, and 7 to general MAC and AE denoted by MAC and AE. Note that PXOR-MAC.$\mathcal{VU}$ in Alg. 7 is interpreted as a combined function that outputs $T_{\mathrm{n}} = \mathsf{MAC}.\mathcal{T}(N_{\mathrm{n}}, M_{\mathrm{n}})$ if $\mathsf{MAC}.\mathcal{V}(N_{\mathrm{o}}, M_{\mathrm{o}}, T_{\mathrm{o}}) = \top$, otherwise outputs $\bot$.

Theorem 3: We obtain the following bounds:

$$\mathbf{Adv}^{\mathtt{pt}}_{\mathsf{PAT2}}(\mathcal{A}) \le \mathbf{Adv}^{\mathtt{priv}}_{\mathsf{AE}}(\mathcal{A}_{\mathtt{ae}}), \tag{2}$$

$$\mathbf{Adv}^{\mathtt{uft}}_{\mathsf{PAT2}}(\mathcal{A}') \le \mathbf{Adv}^{\mathtt{auth}}_{\mathsf{AE}}(\mathcal{A}^{\pm}_{\mathtt{ae}}) + \mathbf{Adv}^{\mathtt{mac}}_{\mathsf{MAC}}(\mathcal{A}_{\mathtt{mac}}). \tag{3}$$

The parameters of $\mathcal{A}_{\mathtt{ae}}$, $\mathcal{A}^{\pm}_{\mathtt{ae}}$, and $\mathcal{A}_{\mathtt{mac}}$ can be determined by those of $\mathcal{A}$ and $\mathcal{A}'$. Theorem 3 means that the security of PAT2 can be reduced to those of MAC and AE. This is not surprising, but we cannot find such a formal treatment (in particular for the combination of MAC and AE to guarantee privacy and unforgeability) in the literature. The security bounds of ELM can be derived by combining the advantages of PAT2 (2), (3) and those of Flat-OCB and PXOR-MAC. When assuming the underlying block cipher of ELM is P, we obtain

$$\mathbf{Adv}^{\mathtt{pt}}_{\mathsf{ELM}}(\mathcal{A}) \le \frac{4.5\sigma^2_{\mathtt{priv}}}{2^n},$$

$$\mathbf{Adv}^{\mathtt{uft}}_{\mathsf{ELM}}(\mathcal{A}') \le \left(\frac{2q_d}{2^\tau} + \frac{4.5\sigma^2_{\mathtt{auth}}}{2^n}\right) + \left(\frac{2q_v}{2^\tau} + \frac{4.5\sigma^2_{\mathtt{mac}}}{2^n}\right),$$

where $\sigma_{\mathtt{priv}}$, q_d, $\sigma_{\mathtt{auth}}$, q_v, and $\sigma_{\mathtt{mac}}$ are parameters of $\mathcal{A}_{\mathtt{ae}}$, $\mathcal{A}^{\pm}_{\mathtt{ae}}$. This means that the security of ELM can be reduced to that of the underlying block cipher. Also, the security bounds of ELM are standard birthday type ($O(2^{n/2})$) as with GCM and OCB.

E. Proof of Theorem 3

Let K_M and K_A be the keys of MAC and AE. Suppose that they are uniformly random and independent.

1) Privacy: We assume that $\mathcal{A}$ is given K_M, denoted by $\mathcal{A}(K_M)$. Since $\mathcal{A}(K_M)$ can compute the data associated with the root node and the intermediate nodes, we can assume that $\mathcal{A}(K_M)$ obtains only the data associated with leaf nodes from the tree initialization oracle and the update oracle. Let $\mathcal{A}_{\mathtt{ae}}$ be the privacy adversary against AE. If $\mathcal{A}(K_M)$ queries IT (resp. IT-\$), $\mathcal{A}_{\mathtt{ae}}$ can simulate it by querying $\mathsf{AE}.\mathcal{E}$ (resp. \$) in the same manner as Alg. 5. Note that a query to IT/IT-\$ invokes nonce-respecting encryption queries of $\mathcal{A}_{\mathtt{ae}}$ since $\mathsf{add}(\mathsf{leaf}(i)) \,\|\, \mathsf{ctr}(\mathsf{leaf}(i)) \neq \mathsf{add}(\mathsf{leaf}(j)) \,\|\, \mathsf{ctr}(\mathsf{leaf}(j))$ necessarily holds for $1 \leq i \neq j \leq b^d$. Regarding the update queries, $\mathcal{A}(K_M)$ invokes $\mathsf{AE}.\mathcal{D}$. However, $\mathsf{AE}.\mathcal{D}$ in the update queries always outputs $\top$ since $\sigma_i = \tilde{\sigma}_{i-1}$ for $i \in [q]$. Thus, $\mathcal{A}_{\mathtt{ae}}$ can always output $\top$ regardless of inputs to simulate $\mathsf{AE}.\mathcal{D}$ in update queries. The adversary $\mathcal{A}_{\mathtt{ae}}$ can simulate the leftover pure update function in the same manner as the simulation of the initialization oracles. Also, the update query of $\mathcal{A}(K_M)$ invokes nonce-respecting encryption queries of $\mathcal{A}_{\mathtt{ae}}$ due to the node-unique property of $\mathsf{add}(\cdot)$ and the one-time property of $\mathsf{ctr}(\cdot)$. Namely, the sequence of queries in the privacy game of $\mathcal{A}(K_M)$ can be simulated by $\mathcal{A}_{\mathtt{ae}}$. Thus, $\mathbf{Adv}^{\mathtt{pt}}_{\mathsf{PAT2}}(\mathcal{A})$ can be upper-bounded by

$$\begin{aligned}&|\Pr[\mathcal{A}(K_M)^{\mathsf{IT},\mathsf{UT}} \to 1] - \Pr[\mathcal{A}(K_M)^{\mathsf{IT\text{-}\$},\mathsf{UT\text{-}\$}} \to 1]| \\ &\quad = \mathbf{Adv}^{\mathtt{priv}}_{\mathsf{AE}}(\mathcal{A}_{\mathtt{ae}}),\end{aligned}$$

where $\mathcal{A}_{\mathtt{ae}}$ queries $b^d + q$ times to encryption oracle because an initialization query of $\mathcal{A}(K_M)$ invokes b^d encryption queries of $\mathcal{A}_{\mathtt{ae}}$ and update queries of $\mathcal{A}(K_M)$ invoke q encryption queries of $\mathcal{A}_{\mathtt{ae}}$.

2) Unforgeability: Let $\mathcal{A}'_{\mathtt{ma}}$ denote an adversary who queries $\mathsf{MAC}.\mathcal{T}$, $\mathsf{AE}.\mathcal{E}$, $\mathsf{MAC}.\mathcal{V}$, and $\mathsf{AE}.\mathcal{D}$. We show $\mathcal{A}'_{\mathtt{ma}}$ can simulate IT, UT, and VT following the rules of the security notions of MAC and AE. As in the case of the privacy advantage, $\mathcal{A}'_{\mathtt{ma}}$ can simulate IT and UT by querying $\mathsf{MAC}.\mathcal{T}$ and $\mathsf{AE}.\mathcal{E}$. Note that all queries are nonce-respecting. For the simulation of IT, $\mathcal{A}'_{\mathtt{ma}}$ queries $(b^d - 1)/(b - 1)$ times to $\mathsf{MAC}.\mathcal{T}$ and queries b^d times to $\mathsf{AE}.\mathcal{E}$. To simulate q' invocations of UT, $\mathcal{A}'_{\mathtt{ma}}$ queries $q'd$ times to $\mathsf{MAC}.\mathcal{T}$ and queries q' times to $\mathsf{AE}.\mathcal{E}$. The query to VT may invoke *replay queries* to $\mathsf{MAC}.\mathcal{V}$ and $\mathsf{AE}.\mathcal{D}$. A replay query means that $\mathcal{A}'_{\mathtt{ma}}$ queries (N, M, T) (resp. (N, C, T)) to $\mathsf{MAC}.\mathcal{V}$ (resp. $\mathsf{AE}.\mathcal{D}$) while (N, M) (resp. (N, M) s.t. $(C, T) = \mathsf{AE}.\mathcal{E}(N, M)$) has been queried to $\mathsf{MAC}.\mathcal{T}$ (resp. $\mathsf{AE}.\mathcal{E}$). Such queries are prohibited in the security notions of MAC and AE whereas they may appear in the simulation of unforgeability game since the game captures $\mathcal{A}'$ performing a replay attack. To avoid replay queries, we suppose $\mathcal{A}'_{\mathtt{ma}}$ stores input/output of $\mathsf{MAC}.\mathcal{T}$ and $\mathsf{AE}.\mathcal{E}$ invoked in IT and UT. Let $\mathtt{List}_{\mathsf{MAC}}$ and $\mathtt{List}_{\mathsf{AE}}$ be the set of such tuples. Specifically, elements of $\mathtt{List}_{\mathsf{MAC}}$ and $\mathtt{List}_{\mathsf{AE}}$ are defined as $(\mathsf{add}(u^\sigma) \,\|\, \mathsf{ctr}(u^\sigma), \mathsf{ctr}(\mathsf{ch}_1(u^\sigma)) \,\|\, \cdots \,\|\, \mathsf{ctr}(\mathsf{ch}_b(u^\sigma)), \mathsf{tag}(u^\sigma))$ and $(\mathsf{add}(\mathsf{leaf}(i)^\sigma) \,\|\, \mathsf{ctr}(\mathsf{leaf}(i)^\sigma), C[i], \mathsf{tag}(\mathsf{leaf}(i)^\sigma))$, where u is a node of a tree construction σ and $i \in [b^d]$. When $\mathcal{A}'$ invokes replay queries to $\mathsf{MAC}.\mathcal{V}$ and $\mathsf{AE}.\mathcal{D}$, $\mathcal{A}'_{\mathtt{ma}}$ can notice it by checking $\mathtt{List}_{\mathsf{MAC}}$ and $\mathtt{List}_{\mathsf{AE}}$, and can simulate the response of $\mathsf{MAC}.\mathcal{V}$ and $\mathsf{AE}.\mathcal{D}$ by just returning $\top$. Regarding the queries except for replay queries, $\mathcal{A}'_{\mathtt{ma}}$ can simulate them by querying $\mathsf{MAC}.\mathcal{V}$ and $\mathsf{AE}.\mathcal{D}$ in the same manner as Alg. 6. For the simulation of VT, $\mathcal{A}'_{\mathtt{ma}}$ queries at most d times to $\mathsf{MAC}.\mathcal{V}$ and queries at most one time to $\mathsf{AE}.\mathcal{D}$. From the above discussion, we obtain

$$\Pr[\mathcal{A}'^{\mathsf{IT},\mathsf{UT},\mathsf{VT}} \to 1] = \Pr[\mathcal{A}'^{\mathcal{A}'^{\mathsf{MAC}.\mathcal{T},\mathsf{AE}.\mathcal{E},\mathsf{MAC}.\mathcal{V},\mathsf{AE}.\mathcal{D}}_{\mathtt{ma}}} \to 1], \tag{4}$$

where $\mathcal{A}'^{\mathcal{A}'^{\mathsf{MAC}.\mathcal{T},\mathsf{AE}.\mathcal{E},\mathsf{MAC}.\mathcal{V},\mathsf{AE}.\mathcal{D}}_{\mathtt{ma}}}$ means that $\mathcal{A}'$ queries to $\mathcal{A}'_{\mathtt{ma}}$ pretending to be functions IT, UT, VT.

We define new adversary $\mathcal{A}''_{\mathtt{ma}}$ querying $\mathsf{MAC}.\mathcal{T}$, $\mathsf{AE}.\mathcal{E}$, and $\perp_{\mathsf{T}}$ to simulate IT, UT, and $\perp_{\mathsf{T}}$. $\mathcal{A}''_{\mathtt{ma}}$ can simulate IT and UT by employing $\mathsf{MAC}.\mathcal{T}$ and $\mathsf{AE}.\mathcal{E}$ in the same way that $\mathcal{A}'_{\mathtt{ma}}$ dose. For $\perp_{\mathsf{T}}$, $\mathcal{A}''_{\mathtt{ma}}$ only needs to mediate $\mathcal{A}'$'s query to $\perp_{\mathsf{T}}$. Thus, we obtain the following equation.

$$\Pr[\mathcal{A}'^{\mathsf{IT},\mathsf{UT},\perp_{\mathsf{T}}} \to 1] = \Pr[\mathcal{A}'^{\mathcal{A}''^{\mathsf{MAC}.\mathcal{T},\mathsf{AE}.\mathcal{E},\perp_{\mathsf{T}}}_{\mathtt{ma}}} \to 1]. \tag{5}$$

From (4) and (5), we obtain

$$\begin{aligned}\mathbf{Adv}^{\mathtt{uft}}_{\mathsf{PAT2}}(\mathcal{A}') = |\Pr[\mathcal{A}'^{\mathcal{A}'^{\mathsf{MAC}.\mathcal{T},\mathsf{AE}.\mathcal{E},\mathsf{MAC}.\mathcal{V},\mathsf{AE}.\mathcal{D}}_{\mathtt{ma}}} \to 1] \\ - \Pr[\mathcal{A}'^{\mathcal{A}''^{\mathsf{MAC}.\mathcal{T},\mathsf{AE}.\mathcal{E},\perp_{\mathsf{T}}}_{\mathtt{ma}}} \to 1]|.\end{aligned} \tag{6}$$

We can evaluate the upper bound of (6) by adding the following three inequation's left sides and right sides, respectively, then obtain (3).

$$\begin{aligned}&|\Pr[\mathcal{A}'^{\mathcal{A}'^{\mathsf{MAC}.\mathcal{T},\mathsf{AE}.\mathcal{E},\mathsf{MAC}.\mathcal{V},\mathsf{AE}.\mathcal{D}}_{\mathtt{ma}}} \to 1] \\ &\qquad - \Pr[\mathcal{A}'^{\mathcal{A}'^{\mathsf{MAC}.\mathcal{T},\mathsf{AE}.\mathcal{E},\mathsf{MAC}.\mathcal{V},\perp_{\mathsf{AE}}}_{\mathtt{ma}}} \to 1]| \\ &\quad \leq \mathbf{Adv}^{\mathtt{auth}}_{\mathsf{AE}}(\mathcal{A}^{\pm}_{\mathtt{ae}}),\end{aligned} \tag{7}$$

$$\begin{aligned}&|\Pr[\mathcal{A}'^{\mathcal{A}'^{\mathsf{MAC}.\mathcal{T},\mathsf{AE}.\mathcal{E},\mathsf{MAC}.\mathcal{V},\perp_{\mathsf{AE}}}_{\mathtt{ma}}} \to 1] \\ &\qquad - \Pr[\mathcal{A}'^{\mathcal{A}'^{\mathsf{MAC}.\mathcal{T},\mathsf{AE}.\mathcal{E},\perp_{\mathsf{MAC}},\perp_{\mathsf{AE}}}_{\mathtt{ma}}} \to 1]| \\ &\quad \leq \mathbf{Adv}^{\mathtt{mac}}_{\mathsf{MAC}}(\mathcal{A}_{\mathtt{mac}}),\end{aligned} \tag{8}$$

$$\begin{aligned}&|\Pr[\mathcal{A}'^{\mathcal{A}'^{\mathsf{MAC}.\mathcal{T},\mathsf{AE}.\mathcal{E},\perp_{\mathsf{MAC}},\perp_{\mathsf{AE}}}_{\mathtt{ma}}} \to 1] \\ &\qquad - \Pr[\mathcal{A}'^{\mathcal{A}''^{\mathsf{MAC}.\mathcal{T},\mathsf{AE}.\mathcal{E},\perp_{\mathsf{T}}}_{\mathtt{ma}}} \to 1]| = 0,\end{aligned} \tag{9}$$

where $\perp_{\mathsf{AE}}(\cdot,\cdot,\cdot)$ (resp. $\perp_{\mathsf{MAC}}(\cdot,\cdot,\cdot)$) is the function for decryption (resp. verification) queries to AE (resp. MAC), which always returns $\perp$ for any inputs. $\mathcal{A}^{\pm}_{\mathtt{ae}}$ is an authenticity adversary against AE performing $b^d + q'$ encryption queries and one decryption query. $\mathcal{A}_{\mathtt{mac}}$ is an adversary against MAC performing $(b^d - 1)/(b - 1) + q'd$ tagging queries and d verification queries. The remaining part of this section is devoted to the proof of (7), (8), and (9). We can prove (7) and (8) in the same manner, thus we omit the proof of (8). Suppose that $\mathcal{A}'_{\mathtt{ma}}$ eventually outputs a bit in her simulation of the unforgeability game and $\mathcal{A}'$ outputs the same bit as $\mathcal{A}'_{\mathtt{ma}}$. Then the left side of (7) is upper-bounded by the probability that $\mathcal{A}'_{\mathtt{ma}}$ querying $\mathsf{MAC}.\mathcal{T}$, $\mathsf{AE}.\mathcal{E}$, and $\mathsf{MAC}.\mathcal{V}$ successfully distinguishes $\mathsf{AE}.\mathcal{D}$ from $\perp_{\mathsf{AE}}$. Then it also can be upper-bounded by the probability that $\mathcal{A}'_{\mathtt{ma}}(K_M)$ querying $\mathsf{AE}.\mathcal{E}$ successfully distinguishes $\mathsf{AE}.\mathcal{D}$ from $\perp_{\mathsf{AE}}$. Without loss of generality, we can assume that this distinguishing probability

is equal to the probability that $\mathcal{A}'_{\mathrm{ma}}(K_M)$ querying $\mathsf{AE}.\mathcal{E}$ and $\mathsf{AE}.\mathcal{D}$ obtains something other than $\perp$ from $\mathsf{AE}.\mathcal{D}$ under the unforgeability game for ME trees. The authenticity adversary against AE, $\mathcal{A}^{\pm}_{\mathsf{ae}}$, can simulate the oracles that $\mathcal{A}'_{\mathrm{ma}}(K_M)$ queries because the query sequence of $\mathcal{A}'_{\mathrm{ma}}(K_M)$ respects the rule of authenticity game for AE schemes (performing nonce-respecting queries to $\mathsf{AE}.\mathcal{E}$ and not performing replay queries to $\mathsf{AE}.\mathcal{D}$). Thus, we obtain inequation (7).

We then prove (9). Recall that (idx', σ') is the tree verification query of $\mathcal{A}'$ and $(u_0^{\sigma'}, \ldots, u_d^{\sigma'})$ is the path of nodes from the root to the specified leaf. The left side of (9) can be seen as the probability that $\mathcal{A}'$ querying IT and UT obtains $\top$ from the tree verification oracle simulated by $\perp_{\mathsf{MAC}}$, $\perp_{\mathsf{AE}}$, $\mathtt{List}_{\mathsf{MAC}}$, and $\mathtt{List}_{\mathsf{AE}}$. This is the probability that all the data associated with $u_0^{\sigma'}, \ldots, u_d^{\sigma'}$ consist of the elements of $\mathtt{List}_{\mathsf{MAC}}$ and $\mathtt{List}_{\mathsf{AE}}$. We prove the probability is equal to zero in the following claim.

Claim 1: Either **(a)** or **(b)** described below must hold.

(a) There exists $i \in \{0, \ldots, d-1\}$ s.t. the tuple $(\mathsf{add}(u_i^{\sigma'}) \| \mathsf{ctr}(u_i^{\sigma'}), \mathsf{ctr}(\mathsf{ch}_1(u_i^{\sigma'})) \| \cdots \| \mathsf{ctr}(\mathsf{ch}_b(u_i^{\sigma'})), \mathsf{tag}(u_i^{\sigma'})) \notin \mathtt{List}_{\mathsf{MAC}}$.

(b) $(\mathsf{add}(u_d^{\sigma'}) \| \mathsf{ctr}(u_d^{\sigma'}), C^{\sigma'}[idx], \mathsf{tag}(u_d^{\sigma'})) \notin \mathtt{List}_{\mathsf{AE}}$.

This claim states that $\mathcal{A}'_{\mathrm{ma}}$ querying $\mathsf{MAC}.\mathcal{T}$, $\mathsf{AE}.\mathcal{E}$, $\perp_{\mathsf{MAC}}$ and $\perp_{\mathsf{AE}}$ has to query to $\perp_{\mathsf{MAC}}$ or $\perp_{\mathsf{AE}}$ in her simulation of the tree verification query. Thus, she always obtains $\perp$ from $\perp_{\mathsf{MAC}}$ or $\perp_{\mathsf{AE}}$ and she returns it to $\mathcal{A}'$. Note that HJ05 shows almost the same claim and its proof, and thus, we omit the proof of the claim.

V. Implementation and Evaluation

We evaluate the hardware implementation of ELM using logic synthesis. We use AES-128, thus $n = 128$. Counter, address and tag is set to 64 bits, so $\alpha = \beta = \tau = n/2$. The 64-bit tag corresponds to the security level of SIT for a fair comparison. We focus on a high-throughput and area-time efficient architecture based on an unrolled and pipelined AES datapath, similar to that of SIT in [8], whose throughput is one block encryption per clock cycle. This high-throughput architecture suits the context of memory encryption. Note that our architecture can utilize other block ciphers and architectures (*e.g.*, round-based and byte-serial ones) in accordance with the optimization goals.

A. Hardware Architectures

Figure 4 shows the proposed hardware architecture of PXOR-MAC. The primary inputs consist of a block index, the number of branches $\mathtt{len}$ $(= b)$, nonce $(\mathsf{add}(u_i^{\sigma}) \| \mathsf{ctr}(u_i^{\sigma}))$, two n-bit keys, and an n-bit segmented plaintext block $(\mathsf{ctr}(\mathsf{ch}_{2j+1}(u_i^{\sigma})) \| \mathsf{ctr}(\mathsf{ch}_{2j+2}(u_i^{\sigma})))$ $(0 \le j \le b/2 - 2)$, and the primary output is given as tag. One plaintext block is fed to the hardware every clock cycle one after another and an encoding is completed with 11 clock cycles. In this architecture, the AES datapath is fully unrolled and pipelined. The pipeline registers are inserted at the boundaries of each round in order to increase the throughput. This enables the encryption of one plaintext block in one clock cycle with the frequency corresponding to the critical path of one round datapath.

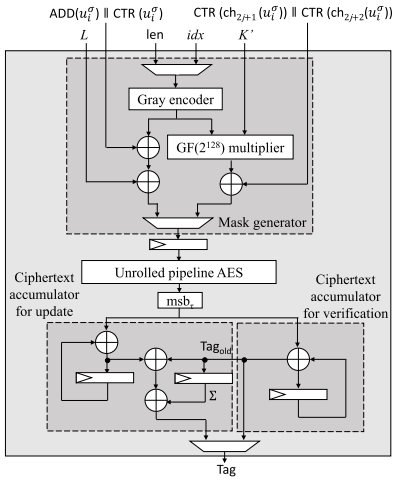


Fig. 4. Proposed MAC hardware architecture.

An up-to-date AES round datapath with a tower-field S-box presented in [25] is adopted for ELM (and SIT [8] for a comparison in this paper) in the following hardware implementation. A mask value for the input block to the AES core ($K' \cdot i$ in Alg. 2) is generated by the multiplication of a gray code (converted from a block index) and a key K' over GF(2^n). The conversion from a block index to a gray code is given by a combinational circuit and the generation of a mask value is implemented using a $(n \times \log b)$-bit GF multiplier. This multiplier generates mask values from all indices in a tree with b branches in one clock cycle. The mask value for the nonce ($N_o \oplus K' \cdot m \oplus L$ in Alg. 2) is computed as the sum of the last mask value and L without any GF multiplication. The accumulation in the tag generator is implemented by a feedback loop consisting of a bit-parallel-XOR (*i.e.*, GF($2^{n/2}$) adder) and registers, which realizes the **for** loop at lines 2–2 in Alg. 2.

Figure 5 shows the hardware architectures of the proposed AE, where one encryption core and one decryption core are utilized. Both cores are unrolled and pipelined similarly to the above MAC hardware to ensure high throughput. The encryption and decryption cores are separately implemented (without unifying them like [25], [26]) in order to perform a decryption in the pre-verification process and an encryption in the update process simultaneously for UT. The pre-mask and post-mask generators compute the mask values for the input and output of encryption/decryption, respectively. The proposed architecture utilizes two pre-mask generators and two post-mask generators for simultaneous encryption and decryption. The field doubling and tripling for mask value generation are achieved by combinational circuit blocks denoted by $\times 2$ and $\times 3$, which consist of four and 132 two-way XOR gates, respectively. The architecture can be implemented with less area than another one consisting of one pre-mask generator, one post-mask generator, and two 128-bit-wise first-in first-out (FIFO) buffers.[7] Plaintext accumulators obtain

[7]The mask value generated by the pre-mask generator should be retained for ten clock cycles for post-mask addition. This indicates that we require a (128×10)-bit register to implement one FIFO if we use AES as the block cipher, which consumes a larger area and power than the four mask generators in our architecture.

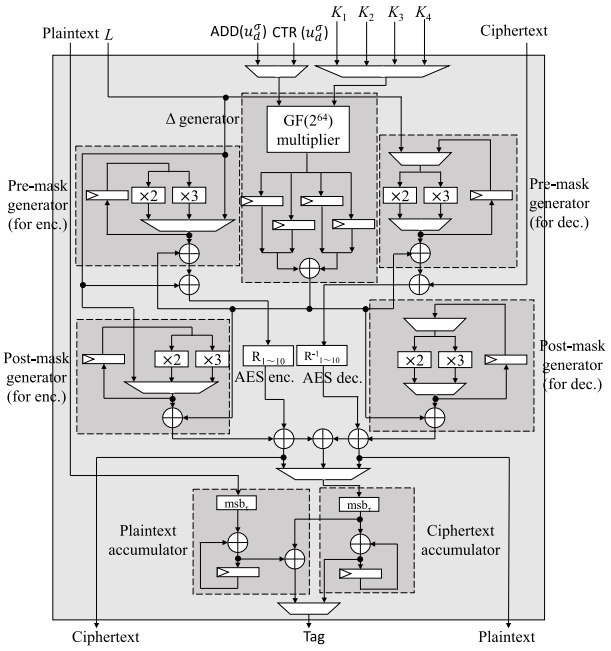


Fig. 5. Proposed AE hardware architecture.

TABLE II
CIRCUIT AREAS SYNTHESIZED WITH 4GHz TIMING CONSTRAINT [GE]

depth d	SIT	ELM
3	421,083.25	519,910.50
5	601,544.50	717,801.00
7	914,155.25	915,612.50

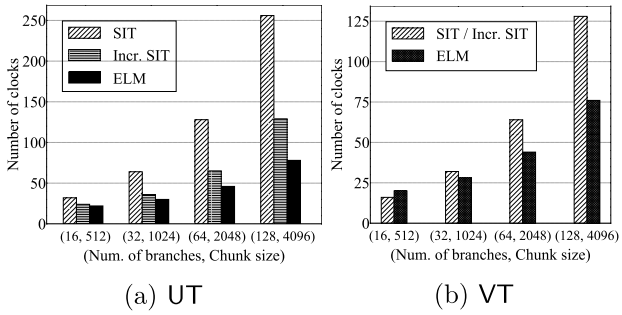


Fig. 6. Numbers of clock cycles to update and verify tag.

truncated plaintext blocks to generate the tag, which is given by a feedback loop with a bit-parallel-XOR (*i.e.*, GF($2^{n/2}$) adder). $\widetilde{E}_K^{N,0,0}(0^n)$ is added before outputting the tag.

The mask value Δ is generated from a nonce by the Δ generator module following Alg. 4, using a $((n/2) \times (n/2))$-bit GF multiplier with four clock cycles. The generated Δ is added to the mask values at the pre/post-mask generators.

B. Performance Evaluation

We evaluate performance of the proposed architectures, on the basis of logic synthesis results. We assume that one MAC module is used at the top and each middle layer in the tree and one AE module is used at the lowest layer. Thus we use d MAC modules and one AE module, which fully exploits the parallelism of ELM. We investigate the best-case performance given a constraint in area and power (*i.e.*, the number of available MAC modules), and clarify the area-latency trade-offs from the evaluation results.

We use Synopsys Design Compiler I-2013.12-SP5 and Nangate 15nm Open Cell Library. The performance with $d = 3$, 5, and 7 are evaluated, following [8]. Table II lists the area obtained from the logic synthesis. We set the timing-constraint for the synthesis to the operating frequency of 4GHz, assuming modern high-end CPUs operating at 3GHz or faster. The synthesis result found no timing violation, thus the proposed architecture can be used even for modern high-end CPUs without degrading the system clock frequency. For comparison, Table II also lists the synthesis results of SIT (following [8]) implemented under the same conditions and assumption as above.

Table II shows that our implementation has a similar size as that of SIT, and the gap is closer as depth grows. The architectures for Flat-OCB require both encryption and decryption cores, which resulted in a larger area than the inverse-free AE of SIT. However, our MAC hardware needs only one AES encryption core, while SIT needs a GF(2^{64}) multiplier in addition to one AES encryption core.

Fig. 6 shows the numbers of clock cycles (*i.e.*, latency) of ELM and SIT. The tuple (Number of branches, Chunk size) determines the sizes of the covered memory region. Incremental SIT (Incr. SIT for short) indicates the evaluation result of SIT when UT is performed in an incremental manner. We stress that such an incremental update has not been shown in the literature including [8]; we do this for a fair comparison. Each clock cycle shown here is given by a larger one of either AE or MAC.

Fig. 6 shows that the advantage of ELM is greater as the covered region gets larger. One major reason is that ELM utilizes a n-bit block cipher for encryption, whereas SIT processes a plaintext in a $n/2$-bit-wise manner (*i.e.*, inner-product MAC over GF($2^{n/2}$)). More precisely, since the MAC module at each layer (and AE module) should process more bits for a larger parameter, the n-bit-wise computation of PXOR-MAC in ELM enables fewer calls of the underlying pseudorandom function than the $n/2$-bit-wise computation of SIT, which results in a lower latency of ELM. In addition, the number of cycles in the update process of AE is reduced by using a distinct decryption core to perform the pre-verification and update processes simultaneously. Note that SIT uses an Encrypt-then-MAC style AE composing counter mode encryption and an inner-product MAC. Since SIT does not utilize any decryption function and the MAC computation becomes critical for the latency, the latency of SIT cannot be reduced in the same manner as ours. The precise numbers of cycles for various tree parameters are shown in Appendix A. The results suggest that ELM is superior to SIT when covering a larger region. As the protected memory region becomes larger, the advantage of ELM increases significantly.

TABLE III
REQUIRED MEMORY SIZES

	SIT	ELM
on-chip	$320 + \max\{\ell, 64b\}$	952
off-chip	$113 \times \sum_{i=0}^{d} b^i - 56$	$112 \times \sum_{i=0}^{d} b^i - 56$

Table III lists the on-chip and off-chip memory sizes for each architecture. Here, both counter and tag lengths are 56 bits to be (perfectly) compliant with SIT. With respect to the on-chip storage, ELM requires four 64-bit round keys for Δ gen., a 128-bit L, a 128-bit plaintext/ciphertext processing key K, and a 56-bit $\mathsf{ctr}(u_{\mathsf{r}}^{\sigma})$. Here, the amounts of on-chip storage is constant regardless of parameters b and ℓ. In contrast, SIT needs to store the 2×128-bit keys and inner-product MAC for nonce processing and mask generation, depending on the size of the tree parameters. As a result, the on-chip size gets larger when b and ℓ are larger because the key length of inner-product MAC increases in proportion to the input block length (b). For example, the on-chip sizes of SIT are 768 and 8,448 bits when $b = 8$, $\ell = 512$ and $b = 128$, $\ell = 8{,}192$, respectively. On the other hand, the off-chip size of ours is comparable with that of SIT. For example, when $d = 3$ and $b = 8$, the off-chip sizes of SIT and ours are 66,049 and 65,464 bits, respectively.

VI. DISCUSSION

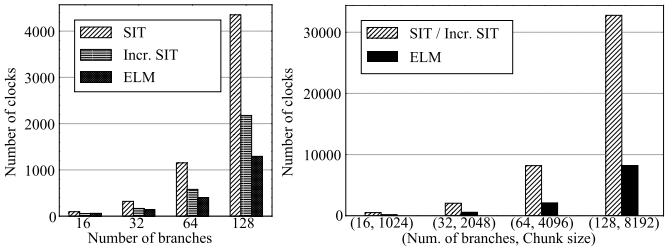
A. Design Optimizations

Our evaluations did not consider system constraints in order to demonstrate the scalability of our proposal. In practice, we would design the total hardware configuration depending on various system/architecture constraints, such as system clock frequency, memory size, available on-chip size, memory bandwidth, cache memory structure, and so on. The sizes of the MAC and AE modules should also be determined considering the above constraints.

A gap in latency between AE and MAC also leads to a loss of efficiency for ME tree because the entire latency is determined by a larger latency of either AE or MAC. While we evaluated the typical tree structures in Section V-B using power-of-two parameters, they should be determined such that the latencies of AE and MAC are well-balanced. In summary, when designing the ME tree and its hardware, we should first determine the optimal (*i.e.*, well-balanced) tree structure parameters for the required covered region. Then, we can mitigate the remaining gap in latency between AE and MAC on the basis of the above hardware optimization approach.

B. Application of Split Counter

ELM can use Split Counter [10], the up-to-date method to reduce the amount of counters for ME trees. Figure 7 shows the numbers of cycles when an overflow of minor counters occurs in MAC and AE, respectively. (See Appendix A for more details.) Here, we evaluate the cases of four different numbers of branches. Since the reset counter is always the same value, the encryption result of reset counter in MAC can be pre-computed for both SIT and our trees. Thus, ELM maintains superiority to SIT under the condition where an overflow occurs. We also found that the proposed AE is advantageous even with the split counter thanks to the simultaneous execution of pre-verification and update. In particular, when the number of branches increases, the proposed scheme becomes more advantageous in comparison with that without the split counter.



(a) in middle nodes (b) in leaf nodes

Fig. 7. Number of clocks when minor counter overflows.

VII. CONCLUSION

We have presented ELM, a new memory encryption scheme with tree-based authentication. Unlike many recent proposals from computer architecture perspective, we focus on the internal MAC and AE modes, including their interactions, to reduce the entire read/write latency. ELM combines fully parallelizable MAC and AE modes and utilizes the incremental property of the MAC mode. Our AE mode is similar to OCB, however has a better decryption latency and it can be of independent interest as a stand-alone AE mode. We provide provable security results for these components as well as the whole scheme. Since Intel SGX's scheme (SIT) is a representative work on the same direction, we instantiated ELM using the same AES and compared ELM with SIT, and presented preliminary hardware implementations. The results showed that ELM achieves significantly lower latency, while keeping the comparable implementation size of SIT. Several future directions can be considered, as follows:

A. Other Instantiations

A low-latency (tweakable) block cipher such as PRINCE [27] or QARMA [28] will significantly improve both latency and size of ELM. Using multiple primitives of possibly different block sizes may further boost the performance.

B. Side-Channel Attacks

Cryptographic hardware frequently needs to be resistant against side-channel attacks (SCAs). Design and evaluation of SCA-resistant hardware architecture for ELM is an interesting future topic. The use of a SCA-resistant AES implementation will thwart typical SCAs trying to recover the AES key. For example, we can use a low-latency masked round-based AES [29]. However, it will significantly reduce throughput

TABLE IV
NUMBERS OF CLOCKS TO UPDATE AND VERIFY TAG

b	ℓ	Update			Verify		Covered region [Byte]		
		SIT	Incr. SIT	ELM	SIT / Incr. SIT	ELM	$d=3$	$d=5$	$d=7$
8	512	**20**	**20**	21	**14**	18	32K	2M	134M
	1,024	32	32	**25**	**18**	22	65K	4M	268M
	2,048	64	64	**33**	32	**30**	131K	8M	536M
	4,096	128	128	**49**	64	**46**	262K	16M	1G
	8,192	256	256	**81**	128	**78**	524K	33M	2G
16	512	32	**20**	22	**16**	20	262K	67M	17G
	1,024	32	32	**25**	**18**	22	524K	134M	34G
	2,048	64	64	**33**	32	**30**	1M	268M	68G
	4,096	128	128	**49**	64	**46**	2M	536M	137G
	8,192	256	256	**81**	128	**78**	4M	1G	274G
32	512	64	33	**30**	32	**28**	2M	2G	2T
	1,024	64	33	**30**	32	**28**	4M	4G	4T
	2,048	64	64	**33**	32	**30**	8M	8G	8T
	4,096	128	128	**49**	64	**46**	16M	17G	17T
	8,192	256	256	**81**	128	**78**	33M	34G	35T
64	512	128	65	**46**	64	**44**	16M	68G	281T
	1,024	128	65	**46**	64	**44**	33M	137G	562T
	2,048	128	65	**46**	64	**44**	67M	274G	1P
	4,096	128	128	**49**	64	**46**	134M	549G	2P
	8,192	256	256	**81**	128	**78**	268M	1T	4P
128	512	256	129	**78**	128	**76**	134M	2T	36P
	1,024	256	129	**78**	128	**76**	268M	4T	72P
	2,048	256	129	**78**	128	**76**	536M	8T	144P
	4,096	256	129	**78**	128	**76**	1G	17T	288P
	8,192	256	256	**81**	128	**78**	2G	35T	576P

due to a large area overhead and on-the-fly random number generation.

The usage of a (first-order) masking-friendly lightweight (tweakable) block cipher such as PRESENT [30] or Skinny [31] would be a practical alternative with less area overhead and no on-the-fly randomness.

Furthermore, it would be interesting to design leakage-resilient TBC/permutation-based AE (*e.g.*, [32], [33]) and MAC that enable low-latency operation and are suitable to be used with ELM.

APPENDIX A
DETAILED PERFORMANCE EVALUATION

Table IV of page 15 shows the numbers of clock cycles (*i.e.*, latency) and the size of protected memory region (namely, the covered region) of ELM and SIT in details for various tree parameters. Each clock cycle shown here is given by a larger one of either AE or MAC. For example, if our ME tree has eight branches $b = 8$ and handles 512-bit blocks $\ell = 512$ [bit] at the leaf (or lowest) node, ELM requires 18 and 21 clock cycles for MAC and AE, respectively; hence the clock cycle of this ME tree is given as 21 clock cycles in the table. The bold-face characters in each row highlight the scheme that achieved the lowest latency (minimal clock cycles) under the parameter condition of the row. The parameters used in Fig. 6 in Section V-B are underlined in the chunk size column in Table IV. The table shows the results of all tree structures comprehensively, where some rows hatched in gray indicate better clock cycles than those in white in terms of the latency required for the covered regions. For example, a tree with $b = 8$ and $\ell = 4{,}096$ requires a larger latency and a smaller covered region than that with $b = 16$ and $\ell = 512$, and therefore there is no reason to use the former tree rather than the latter. Such meaningless parameters are caused by the gap in latency between AE and MAC, as discussed in Section VI.

In addition, we show the numbers of clock cycles in details for the cases that the split counter is applied. The split counter is a method to reduce the amount of counters stored in an off-chip for ME trees [10]. It uses two types of counters: major and minor ones. A major counter is shared by the children nodes of the node of interest (or a parent node), and each child node is equipped with a minor counter. In other words, in an ME tree with split counter, children nodes having the same parent node share the upper bits of the same major counter. Here, we should point out that overflows of the minor counters frequently occur, since each minor counter is given with a small bit length. When such an overflow occurs, the corresponding major counter is incremented and all the minor counters associated with it are reset to zero. Accordingly, we need to update all the tags where the nonces are reset. The tag update with the split counter requires b times tag updates at the layer where a major counter is incremented (*i.e.*, the overflow of minor counter occurs), which is non-trivial in the update process (*i.e.*, UT). In ELM, b counters are originally used as the input for the plaintext part of tag generation by the MAC algorithm, that is, $b \times n/2$ bits should be verified by MAC. More precisely, let `ctr` be the counter of the parent

TABLE V
NUMBER OF CLOCKS WHEN MINOR COUNTER IN MIDDLE NODES OVERFLOWS

b	SIT	Incr. SIT	ELM
8	32	26	29
16	96	57	61
32	320	167	141
64	1,152	579	397
128	4,352	2,177	1,293

TABLE VI
NUMBER OF CLOCKS WHEN MINOR COUNTER IN LEAF NODES OVERFLOWS

b	ℓ	SIT/ Incr. SIT	ELM
8	512	136	49
	1,024	260	81
	2,048	512	147
	4,096	1,024	273
	8,192	2,048	529
16	512	136	49
	1,024	516	145
	2,048	1,024	275
	4,096	2,048	529
	8,192	4,096	1,041
32	512	520	145
	1,024	1,028	273
	2,048	2,048	531
	4,096	4,096	1,041
	8,192	8,192	2,065
64	512	1,032	273
	1,024	2,052	529
	2,048	4,096	1,043
	4,096	8,192	2,065
	8,192	16,384	4,113
128	512	2,056	529
	1,024	4,100	1,041
	2,048	8,192	2,067
	4,096	16,384	4,113
	8,192	32,768	8,209

node and let $\mathtt{ctr}'_1, \mathtt{ctr}'_2, \ldots \mathtt{ctr}'_b$ be the counters of children nodes without the split counter, where b is the number of branches in the tree structure. In ELM, a tag T is generated as

$$T = \mathsf{PXOR\text{-}MAC}.\mathcal{T}_{K,K'}\Big((\mathsf{add} \parallel \mathtt{ctr}), (\mathtt{ctr}'_1 \parallel \mathtt{ctr}'_2 \parallel \cdots \parallel \mathtt{ctr}'_b)\Big),$$

where $\mathsf{PXOR\text{-}MAC}.\mathcal{T}_{K,K'}(N, M)$ calculates a tag from a nonce N and a plaintext M. Each counter is given by $n/2$ bits and $b \times n/2$ bits should be encrypted.

In contrast, we consider the tag generation when utilizing the split counter. Let $\mathtt{Mctr}$ and $\mathtt{mctr}$ be the major and minor counters of a parent node, respectively. Let $\mathtt{Mctr}'$ and $\mathtt{mctr}'_1, \mathtt{mctr}'_2, \ldots \mathtt{mctr}'_b$ be the major counter and minor counters of children nodes, respectively. In this case, unless the overflow of minor counters occurs, a tag is generated as

$$T = \mathsf{PXOR\text{-}MAC}.\mathcal{T}_{K,K'}\Big((\mathsf{add} \parallel \mathtt{Mctr} \parallel \mathtt{mctr}), (\mathtt{Mctr}' \parallel \mathtt{mctr}'_1 \parallel \cdots \parallel \mathtt{mctr}'_b)\Big).$$

Here, let s and t be the bit lengths of major and minor counters, respectively ($s + t = n/2$). When the split counter is used, the input is given with $s + bt$ bits. Since $s + bt \leq b(s + t)/2$, the input bit length of the MAC algorithm is reduced thanks to the split counter. In addition to the tag generation algorithm (*i.e.*, $\mathsf{PXOR\text{-}MAC}.\mathcal{T}$), VT (*i.e.*, $\mathsf{PXOR\text{-}MAC}.\mathcal{V}$) and UT (*i.e.*, $\mathsf{PXOR\text{-}MAC}.\mathcal{VU}$) algorithms are performed with the split counter as well. The tags of leaf (or lowest) nodes can also be generated, verified, and updated in a similar manner even when the split counter is applied.

As a result, Table V of page 15 and Table VI of page 16 show the numbers of clock cycles when the split counter is applied and an overflow occurs in MAC and AE, respectively. When the major counter is incremented due to the overflow of a minor counter, all minor counters associated with the major counter are reset to 0.

APPENDIX B
PROOF OF CLAIM 1

In this section, we show the proof of Claim 1. The literature [4] shows almost the same claim and its proof for the proposed ME tree without encryption of leaf nodes, however, the proof is a little bit complex. We recast it for the sake of ease to understand and the ME tree with encryption of leaf nodes.

If **(b)** occurs, the claim is simply proved. We need to see that if **(b)** does not occur, the case **(a)** must hold. First we discuss u_0 (*i.e.*, the root node). Let $(N_{u_0}, M_{u_0}, T_{u_0}) = (\mathsf{add}(u_0^{\sigma'}) \parallel \mathsf{ctr}(u_0^{\sigma'}),\ \mathsf{ctr}(\mathsf{ch}_1(u_0^{\sigma'})) \parallel \cdots \parallel \mathsf{ctr}(\mathsf{ch}_b(u_0^{\sigma'})), \mathsf{tag}(u_0^{\sigma'}))$. If $(N_{u_0}, M_{u_0}, T_{u_0}) \notin \mathtt{List}_{\mathsf{MAC}}$ holds, it means that **(a)** holds. Suppose that $(N_{u_0}, M_{u_0}, T_{u_0}) \in \mathtt{List}_{\mathsf{MAC}}$, and we obtain the following equation.

$$\begin{aligned}(N_{u_0}, M_{u_0}, T_{u_0}) &= (\mathsf{add}(u_0^{\tilde{\sigma}_{q'}}) \parallel \mathsf{ctr}(u_0^{\tilde{\sigma}_{q'}}), \\ &\quad \mathsf{ctr}(\mathsf{ch}_1(u_0^{\tilde{\sigma}_{q'}})) \parallel \cdots \parallel \mathsf{ctr}(\mathsf{ch}_b(u_0^{\tilde{\sigma}_{q'}})), \mathsf{tag}(u_0^{\tilde{\sigma}_{q'}})), \quad (10)\end{aligned}$$

which means that the data stored in $u_0^{\sigma'}$ is the same as that stored in $u_0^{\tilde{\sigma}_{q'}}$. This holds because N_{u_0} cannot be tampered by definition, and the element of $\mathtt{List}_{\mathsf{MAC}}$ including N_{u_0} is uniquely determined as (10) since nonces included in $\mathtt{List}_{\mathsf{MAC}}$ are distinct. Note that we also obtain $\mathsf{ctr}(u_1^{\sigma'}) = \mathsf{ctr}(u_1^{\tilde{\sigma}_{q'}})$ from (10).

Next, we discuss u_1. Let $(N_{u_1}, M_{u_1}, T_{u_1}) = (\mathsf{add}(u_1^{\sigma'}) \parallel \mathsf{ctr}(u_1^{\sigma'}), \mathsf{ctr}(\mathsf{ch}_1(u_1^{\sigma'})) \parallel \cdots \parallel \mathsf{ctr}(\mathsf{ch}_b(u_1^{\sigma'})), \mathsf{tag}(u_1^{\sigma'}))$. As well as the case of u_0, we can suppose that $(N_{u_1}, M_{u_1}, T_{u_1}) \in \mathtt{List}_{\mathsf{MAC}}$ since **(a)** occurs when $(N_{u_1}, M_{u_1}, T_{u_1}) \notin \mathtt{List}_{\mathsf{MAC}}$ holds. In the same manner as

the case of u_0, we obtain

$$\begin{aligned}&(N_{u_1}, M_{u_1}, T_{u_1})\\&\quad = (\mathsf{add}(u_1^{\tilde{\sigma}_{q'}}) \,\|\, \mathsf{ctr}(u_1^{\tilde{\sigma}_{q'}}),\\&\qquad \mathsf{ctr}(\mathsf{ch}_1(u_1^{\tilde{\sigma}_{q'}})) \,\|\, \cdots \,\|\, \mathsf{ctr}(\mathsf{ch}_b(u_1^{\tilde{\sigma}_{q'}})), \mathsf{tag}(u_1^{\tilde{\sigma}_{q'}})),\end{aligned}$$

since $\mathsf{add}(u_1^{\sigma'})$ cannot be tampered and $\mathsf{ctr}(u_1^{\sigma'}) = \mathsf{ctr}(u_1^{\tilde{\sigma}_{q'}})$ holds from (10).

Suppose that we repeat the same discussion as u_0 and u_1. For $2 \leq i \leq d-1$, we define $(N_{u_i}, M_{u_i}, T_{u_i}) = (\mathsf{add}(u_i^{\sigma'}) \,\|\, \mathsf{ctr}(u_i^{\sigma'}), \mathsf{ctr}(\mathsf{ch}_1(u_i^{\sigma'})) \,\|\, \cdots \,\|\, \mathsf{ctr}(\mathsf{ch}_b(u_i^{\sigma'})), \mathsf{tag}(u_i^{\sigma'}))$. When $(N_{u_i}, M_{u_i}, T_{u_i}) \in \mathtt{List}_{\mathsf{MAC}}$ for $0 \leq i \leq d-1$, we obtain the following equation.

$$\begin{aligned}&(N_{u_i}, M_{u_i}, T_{u_i})\\&\quad = (\mathsf{add}(u_i^{\tilde{\sigma}_{q'}}) \,\|\, \mathsf{ctr}(u_i^{\tilde{\sigma}_{q'}}),\\&\qquad \mathsf{ctr}(\mathsf{ch}_1(u_i^{\tilde{\sigma}_{q'}})) \,\|\, \cdots \,\|\, \mathsf{ctr}(\mathsf{ch}_b(u_i^{\tilde{\sigma}_{q'}})), \mathsf{tag}(u_i^{\tilde{\sigma}_{q'}})). \quad (11)\end{aligned}$$

Finally, we discuss u_d (*i.e.*, the leaf node). Let $(N_{u_d}, C_{u_d}, T_{u_d}) = (\mathsf{add}(u_d^{\sigma'}) \,\|\, \mathsf{ctr}(u_d^{\sigma'}), C^{\sigma'}[idx], \mathsf{tag}(u_d^{\sigma'}))$. Recall that we assumed that **(b)** did not occur, thus $(N_{u_d}, C_{u_d}, T_{u_d}) \in \mathtt{List}_{\mathsf{AE}}$. Here, $\mathsf{add}(u_d^{\sigma'}) = \mathsf{add}(u_d^{\tilde{\sigma}_q'})$ holds since $\mathsf{add}(\cdot)$ cannot be tampered, and $\mathsf{ctr}(u_d^{\sigma'}) = \mathsf{ctr}(u_d^{\tilde{\sigma}_q'})$ holds due to (11) when $i = d-1$. Thus, we have

$$\begin{aligned}&(N_{u_d}, C_{u_d}, T_{u_d})\\&\quad = (\mathsf{add}(u_d^{\tilde{\sigma}_{q'}}) \,\|\, \mathsf{ctr}(u_d^{\tilde{\sigma}_{q'}}), C^{\tilde{\sigma}_{q'}}[idx], \mathsf{tag}(u_d^{\tilde{\sigma}_{q'}})), \quad (12)\end{aligned}$$

since nonces included in $\mathtt{List}_{\mathsf{AE}}$ are distinct, hence the element of $\mathtt{List}_{\mathsf{AE}}$ including N_{u_d} is uniquely determined as (12).

From (11) and (12), we proved that the data stored in $u_i^{\sigma'}$ is the same as that stored in $u_i^{\tilde{\sigma}_{q'}}$ for all $i \in \{0, \ldots, d\}$, which is a forbidden query. Therefore, there must exist $i \in \{0, \ldots, d-1\}$ such that $(N_{u_i}, M_{u_i}, T_{u_i}) \notin \mathtt{List}_{\mathsf{MAC}}$. This concludes the claim.

REFERENCES

[1] M. Dworkin, *Recommendation for Block Cipher Modes of Operation: The XTS-AES Mode for Confidentiality on Storage Devices*, document NIST Special Publication (SP) 800-38E, 2010.

[2] J. A. Halderman *et al.*, "Lest we remember: Cold boot attacks on encryption keys," in *Proc. USENIX Secur. Symp.*, 2008, pp. 45–60.

[3] R. C. Merkle, "A digital signature based on a conventional encryption function," in *CRYPTO'87* (Lecture Notes in Computer Science), vol. 293, C. Pomerance, Ed. Berlin, Germany: Springer, Aug. 1988, pp. 369–378.

[4] W. E. Hall and C. S. Jutla, "Parallelizable authentication trees," in *SAC 2005* (Lecture Notes in Computer Science), vol. 3897, B. Preneel and S. Tavares, Eds. Berlin, Germany: Springer, Aug. 2006, pp. 95–109.

[5] B. Rogers, S. Chhabra, M. Prvulovic, and Y. Solihin, "Using address independent seed encryption and bonsai Merkle trees to make secure processors OS- and performance-friendly," in *Proc. 40th Annu. IEEE/ACM Int. Symp. Microarchitecture (MICRO)*, Dec. 2007, pp. 183–196.

[6] R. Elbaz, D. Champagne, R. B. Lee, L. Torres, G. Sassatelli, and P. Guillemin, "TEC-tree: A low-cost, parallelizable tree for efficient defense against memory replay attacks," in *CHES 2007* (Lecture Notes in Computer Science), vol. 4727, P. Paillier and I. Verbauwhede, Eds. Berlin, Germany: Springer, Sep. 2007, pp. 289–302.

[7] S. Gueron, "Memory encryption for general-purpose processors," *IEEE Secur. Privacy*, vol. 14, no. 6, pp. 54–62, Nov. 2016.

[8] S. Gueron, "A memory encryption engine suitable for general purpose processors," Cryptol. ePrint Arch., Tech. Rep. 2016/204, 2016. [Online]. Available: https://eprint.iacr.org/2016/204

[9] M. Dworkin, *Recommendation for Block Cipher Modes of Operation: Galois/Counter Mode (GCM) and GMAC*, document NIST Special Publication 800-38D, 2007.

[10] C. Yan, D. Englender, M. Prvulovic, B. Rogers, and Y. Solihin, "Improving cost, performance, and security of memory encryption and authentication," in *Proc. 33rd Int. Symp. Comput. Archit. (ISCA)*, May 2006, pp. 179–190.

[11] M. Taassori, A. Shafiee, and R. Balasubramonian, "VAULT: Reducing paging overheads in SGX with efficient integrity verification structures," in *Proc. ASPLOS*. New York, NY, USA: ACM, 2018, pp. 665–678.

[12] G. Saileshwar, P. J. Nair, P. Ramrakhyani, W. Elsasser, J. A. Joao, and M. K. Qureshi, "Morphable counters: Enabling compact integrity trees for low-overhead secure memories," in *Proc. 51st Annu. IEEE/ACM Int. Symp. Microarchitecture (MICRO)*, Oct. 2018, pp. 416–427.

[13] M. Bellare, O. Goldreich, and S. Goldwasser, "Incremental cryptography: The case of hashing and signing," in *CRYPTO'94* (Lecture Notes in Computer Science), vol. 839, Y. Desmedt, Ed. Berlin, Germany: Springer, Aug. 1994, pp. 216–233.

[14] M. Bellare and D. Micciancio, "A new paradigm for collision-free hashing: Incrementality at reduced cost," in *EUROCRYPT'97* (Lecture Notes in Computer Science), vol. 1233, W. Fumy, Ed. Berlin, Germany: Springer, May 1997, pp. 163–192.

[15] M. Bellare, R. Guérin, and P. Rogaway, "XOR MACs: New methods for message authentication using finite pseudorandom functions," in *CRYPTO'95* (Lecture Notes in Computer Science), vol. 963, D. Coppersmith, Ed. Berlin, Germany: Springer, Aug. 1995, pp. 15–28.

[16] T. Krovetz and P. Rogaway, "The software performance of authenticated-encryption modes," in *FSE 2011* (Lecture Notes in Computer Science), vol. 6733, A. Joux, Ed. Berlin, Germany: Springer, Feb. 2011, pp. 306–327.

[17] M. Liskov, R. L. Rivest, and D. Wagner, "Tweakable block ciphers," in *CRYPTO 2002* (Lecture Notes in Computer Science), vol. 2442, M. Yung, Ed. Berlin, Germany: Springer, Aug. 2002, pp. 31–46.

[18] P. Rogaway, "Efficient instantiations of tweakable blockciphers and refinements to modes OCB and PMAC," in *ASIACRYPT 2004* (Lecture Notes in Computer Science), vol. 3329, P. J. Lee, Ed. Berlin, Germany: Springer, Dec. 2004, pp. 16–31.

[19] M. Bellare and C. Namprempre, "Authenticated encryption: Relations among notions and analysis of the generic composition paradigm," in *ASIACRYPT 2000* (Lecture Notes in Computer Science), vol. 1976, T. Okamoto, Ed. Berlin, Germany: Springer, Dec. 2000, pp. 531–545.

[20] W. E. Hall and C. S. Jutla, "Parallelizable authentication trees," *IACR Cryptol. ePrint Arch.*, vol. 2002, p. 190, Jan. 2002.

[21] T. Unterluggauer, M. Werner, and S. Mangard, "MEAS: Memory encryption and authentication secure against side-channel attacks," *J. Cryptograph. Eng.*, vol. 9, no. 2, pp. 137–158, Jun. 2019.

[22] M. Bellare, A. Desai, E. Jokipii, and P. Rogaway, "A concrete security treatment of symmetric encryption," in *Proc. IEEE 38th Annu. Symp. Found. Comp. Sci.*, Oct. 1997, pp. 394–403.

[23] K. Minematsu and T. Matsushima, "Generalization and extension of XEX* mode," *IEICE Trans. Fundam. Electron., Commun. Comput. Sci.*, vol. E92-A, no. 2, pp. 517–524, 2009.

[24] M. Bellare, O. Goldreich, and A. Mityagin, "The power of verification queries in message authentication and authenticated encryption," Cryptol. ePrint Arch., Tech. Rep. 2004/309, 2004. [Online]. Available: https://eprint.iacr.org/2004/309

[25] R. Ueno *et al.*, "High throughput/gate AES hardware architectures based on datapath compression," *IEEE Trans. Comput.*, vol. 69, no. 4, pp. 534–548, Apr. 2020.

[26] R. Ueno, S. Morioka, N. Homma, and T. Aoki, "A high throughput/gate AES hardware architecture by compressing encryption and decryption datapaths—Toward efficient CBC-mode implementation," in *CHES* (Lecture Notes in Computer Science), vol. 9813. Cham, Switzerland: Springer, 2016, pp. 538–558.

[27] J. Borghoff *et al.*, "PRINCE—A low-latency block cipher for pervasive computing applications—Extended abstract," in *ASIACRYPT 2012* (Lecture Notes in Computer Science), vol. 7658, X. Wang and K. Sako, Eds. Berlin, Germany: Springer, Dec. 2012, pp. 208–225.

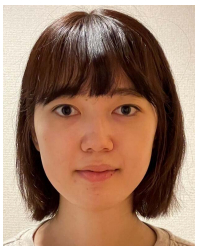
[28] R. Avanzi, "The QARMA block cipher family," *IACR Trans. Symm. Cryptol.*, vol. 2017, no. 1, pp. 4–44, 2017.

[29] P. Sasdrich, B. Bilgin, M. Hutter, and M. E. Marson, "Low-latency hardware masking with application to AES," *IACR Trans. Cryptograph. Hardw. Embedded Syst.*, vol. 2020, pp. 300–326, Mar. 2020.

[30] A. Bogdanov *et al.*, "PRESENT: An ultra-lightweight block cipher," in *Proc. 9th Int. Workshop Cryptograph. Hardw. Embedded Syst. (CHES)*, in Lecture Notes in Computer Science, vol. 4727, P. Paillier and I. Verbauwhede, Eds. Vienna, Austria: Springer, Sep. 2007, pp. 450–466.
[31] C. Beierle *et al.*, "The SKINNY family of block ciphers and its low-latency variant MANTIS," in *CRYPTO 2016, Part II Advances in Cryptology—CRYPTO 2016* (Lecture Notes in Computer Science), vol. 9815, M. Robshaw and J. Katz, Eds. Berlin, Germany: Springer, Aug. 2016, pp. 123–153.
[32] C. Dobraunig, M. Eichlseder, S. Mangard, F. Mendel, and T. Unterluggauer, "ISAP—Towards side-channel secure authenticated encryption," *IACR Trans. Symmetric Cryptol.*, vol. 2017, pp. 80–105, Mar. 2017.
[33] F. Berti, C. Guo, O. Pereira, T. Peters, and F.-X. Standaert, "TEDT: A leakage-resistant AEAD mode," *IACR TCHES*, vol. 2020, no. 1, pp. 256–320, 2019. [Online]. Available: https://tches.iacr.org/index.php/TCHES/article/view/8400

Akiko Inoue received the B.E. and M.E. degrees from Kyushu University in 2015 and 2017, respectively. She joined NEC in 2017. She works on design and analysis of symmetric-key ciphers and related systems. She received the Best Paper Award at CRYPTO 2019.

Kazuhiko Minematsu received the B.E., M.E., and D.S. degrees from Waseda University in 1996, 1998, and 2008, respectively. He joined NEC in 1998. He currently works as a Research Fellow. Since 2019, he has also been a Visiting Professor with Yokohama National University. His research interests are design and analysis of symmetric-key ciphers and its application systems. He received the best paper awards at FSE 2015 and CRYPTO 2019.



Maya Oda received the B.E. and M.S. degrees from Tohoku University in 2019 and 2021, respectively. Her research interests include hardware security, cryptographic implementation, and memory protection.

Rei Ueno (Member, IEEE) is currently an Assistant Professor with the Research Institute of Electrical Communication, Tohoku University. He is also working with the JST as a Researcher for a PRESTO Project. His research interests include arithmetic circuits, cryptographic implementations, formal verification, and hardware security.

Naofumi Homma (Senior Member, IEEE) received the B.E., M.S., and Ph.D. degrees from Tohoku University in 1997, 1999 and 2001, respectively. He is currently a Professor with the Research Institute of Electrical Communication, Tohoku University. He is also working as a Researcher with the JST CREST. His research interests include hardware security, computer arithmetic, and EDA methodology.