CLOC: Authenticated Encryption for Short Input

Tetsu Iwata[1](✉), Kazuhiko Minematsu[2], Jian Guo[3], and Sumio Morioka[4]

[1] Nagoya University, Nagoya, Japan
iwata@cse.nagoya-u.ac.jp
[2] NEC Corporation, Tokyo, Japan
k-minematsu@ah.jp.nec.com
[3] Nanyang Technological University, Singapore, Singapore
ntu.guo@gmail.com
[4] NEC Europe Ltd., London, UK
s-morioka@ak.jp.nec.com

Abstract. We define and analyze the security of a blockcipher mode of operation, CLOC, for provably secure authenticated encryption with associated data. The design of CLOC aims at optimizing previous schemes, CCM, EAX, and EAX-prime, in terms of the implementation overhead beyond the blockcipher, the precomputation complexity, and the memory requirement. With these features, CLOC is suitable for handling short input data, say 16 bytes, without needing precomputation nor large memory. This property is especially beneficial to small microprocessors, where the word size is typically 8 bits or 16 bits, and there are significant restrictions in the size and the number of registers. CLOC uses a variant of CFB mode in its encryption part and a variant of CBC MAC in the authentication part. We introduce various design techniques in order to achieve the above mentioned design goals. We prove CLOC secure, in a reduction-based provable security paradigm, under the assumption that the blockcipher is a pseudorandom permutation. We also present our preliminary implementation results.

Keywords: CLOC · Blockcipher · Authenticated encryption with associated data · Security analysis · Efficiency analysis

1 Introduction

Background. An authenticated encryption with associated data scheme (AEAD) is a symmetric key cryptographic primitive that provides both confidentiality and integrity of plaintexts, and integrity of associated data. There are several ways of designing AEADs, and we focus on a design based on a blockcipher. CCM [39] was proposed by Whiting, Housley, and Ferguson for use within the IEEE 802.11 standard for Wireless LANs. It is adopted as NIST recommendation [16], and is broadly used in practice [9,20,21]. The mode is 2-pass, meaning that we run two algorithms, one for encryption and one for authentication. It is provably secure [25],

© International Association for Cryptologic Research 2015
C. Cid and C. Rechberger (Eds.): FSE 2014, LNCS 8540, pp. 149–167, 2015.
DOI: 10.1007/978-3-662-46706-0_8

but CCM suffers from a number of limitations, most notably it is not on-line; the encryption process cannot be started until knowing the whole input data. There are other issues in CCM [35], and EAX was proposed by Bellare, Rogaway, and Wagner to overcome these limitations [13]. EAX is included in ISO 19772 [9], and it has a number of attractive features; it is simple as it uses CMAC and CTR mode in a black-box manner, and it was designed by taking provable security into consideration. However, it has several implementation costs, and EAX-prime was designed by Moise, Beroset, Phinney, and Burns [31] to reduce the costs. It was designed to reduce the number of blockcipher calls both in precomputation and in processing the input data, to eliminate the key dependent constants, also called masks, to reduce memory requirement to store them, and to unify the associated data and the nonce, which contributes to reduce the memory requirement and the number of blockcipher calls as well. However, a practical attack was pointed out against EAX-prime [30], showing that it is not a secure AEAD. Later, Minematsu, Lucks, and Iwata proposed a variant of EAX called EAX$^+$, which has similar complexity as EAX-prime and is provably secure as EAX [29].

Presumably, though not clearly stated in the document [31], the most significant advantage of EAX-prime over original EAX (and CCM) is its efficient handling of *short input data* with small memory. As EAX-prime needs only one blockcipher call in precomputation whereas EAX needs three calls, EAX-prime gains the performance for short (say 16 bytes) input data, in particular if precomputation is difficult due to a limited amount of memory, or frequent key changes, or both. The performance for short input data is important for many practical applications, most notably for low-power wireless sensor networks, since messages are typically short to suppress the energy consumption of sensor nodes, which are usually battery-powered. For example, Zigbee [8] limits the maximum message length to be 127 bytes, and Bluetooth low energy limits the length to 47 bytes [4]. Another example is Electronic Product Code (EPC), which is a replacement of bar-code using RFID tags, and it typically has 96 bits [5].

Our Contributions. In this paper, we present a mode of operation, CLOC (which stands for Compact Low-Overhead CFB, and is pronounced as “clock”), to meet the demand. The design of CLOC aims at optimizing previous schemes, CCM, EAX, and EAX-prime, in terms of the implementation overhead beyond the blockcipher, the precomputation complexity, and the memory requirement. CLOC is sequential and its asymptotic performance (i.e. for long input data) is comparable to CCM, EAX, and EAX-prime. However, CLOC has a unique feature in its low overhead computation. CLOC works *without* any precomputation beyond the key scheduling of the blockcipher. Specifically, we do not need any blockcipher calls nor generating a key dependent table. This contributes to the improvement of the performance for short input data. For example, when the input data consists of 1-block nonce, 1-block associated data, and 1-block plaintext, CLOC needs 4 blockcipher calls, while we need 5 or 6 calls in CCM, 7 calls (where 3 out of 7 can be precomputed) in EAX, and 5 calls (where 1 out of 5 can be precomputed) in EAX-prime. We focus on provably secure schemes, but for comparison, there are lightweight AE schemes including ALE [15] and FIDES [14], where ALE needs

44 AES rounds which amount to 4.4 AES calls (10 out of 44 AES rounds can be precomputed), and FIDES needs 33 round function calls, where the round function is similar to that of AES but has larger state. This property of CLOC is particularly beneficial for embedded devices since the internal blockcipher is relatively slow due to limited computing power. Moreover, CLOC can be implemented using only two state blocks, i.e. the working memory of $2n$ bits with an n-bit blockcipher, except those needed for interfacing and blockcipher invocations. We do not aware of any provably secure AE mode with on-line capability to work with such a small amount of memory, and this property makes CLOC even suitable for small processors.

Important properties of CLOC can be summarized as follows.

1. It is a nonce-based authenticated encryption with associated data (AEAD).
2. It uses only the encryption of the blockcipher both for encryption and decryption.
3. It makes $\lceil |N|/n \rceil + \lceil |A|/n \rceil + 2\lceil |M|/n \rceil$ blockcipher calls for a nonce N, associated data A, and a plaintext M, when $|A| \geq 1$, where $|X|$ is the length of X in bits and n is the block length in bits of the blockcipher. No precomputation is needed. We note that in CLOC, $1 \leq |N| \leq n-1$ holds (hence we always have $\lceil |N|/n \rceil = 1$), and when $|A| = 0$, it needs $\lceil |N|/n \rceil + 1 + 2\lceil |M|/n \rceil$ blockcipher calls.
4. It works with two state blocks (i.e. $2n$ bits).

We introduce various design techniques in order to achieve the above mentioned design goals. We introduce *tweak functions* which are used to update the internal state at several points in the encryption and the decryption. While bit-wise operations, such as a constant multiplication over $\mathrm{GF}(2^n)$, are often employed in majority of previous schemes, considering the performance for small devices, we completely eliminate bit-wise operations. Instead, our tweak functions consist of word-wise permutations and xor's. As a result, each tweak function can be described by using a 4×4 binary matrix.

The use of word-wise permutations and xor's to update a mask or a key dependent constant was discussed in [22,29], and the approach was applied on CMAC and EAX. Here we use them directly to update the internal state, instead of updating a key dependent constant and xoring it to the state. This was employed for example in designs of MACs [32,40] using bit shift operations. The techniques introduced here seem to be worth for other areas, e.g., in designing MACs, and thus it may be of independent interest.

We also introduce bit-fixing functions. CFB mode leaks input and output pairs of the underlying blockcipher, which may result in the loss of security. We use the functions to logically separate the encryption part and the authentication part of CLOC.

With these techniques, we prove CLOC secure, in a reduction-based provable security paradigm, under the assumption that the blockcipher is a pseudorandom permutation. For security notions, CLOC fulfills the standard security notions for nonce-based AEADs, i.e., the privacy and the authenticity under nonce-respecting adversaries [34]. Furthermore, we prove that the authenticity notion holds even for

Table 1. Comparison of AE modes, for a-block associated data and m-block plaintext with one-block nonce, where $a \geq 1$

Property∘	CCM [16]	GCM [17]	EAX [13]	EAX-prime [31]	OCB3 [26]	CLOC
Calls	$a + 2m + 2$†	$m + 1$‡	$a + 2m + 1$	$a + 2m + 1$§	$a + m + 1$†	$a + 2m + 1$
Setup	0	1	3	1	1	0
On-line	No	Yes	Yes	Yes	Yes	Yes
Static AD	No	Yes	Yes	Yes	Yes	Yes
Parallel	No	Yes	No	No	Yes	No
Primitive	E	E, GHASH	E	E	E, D	E
PRIV/AUTH⋆	$O(2^{n/2})$[25]	$O(2^{n/2})$[24]	$O(2^{n/2})$[13]	$O(1)$[30]	$O(2^{n/2})$[26]	$O(2^{n/2})$
N-AUTH⋄	$\ll 2^{n/2}$[18,19]	$O(1)$[18]	$O(1)$[18]	$O(1)$[30]	$O(1)$[18]	$O(2^{n/2})$

∘ "Setup" shows the number of blockcipher calls for setup, "Static AD" shows if efficient handling of static associated data is possible, "Parallel" shows if the blockcipher calls are parallelizable, and "Primitive" shows the components of the mode. E is the encryption of the blockcipher and D is the decryption.
† May have additional one call
‡ Plus $a + m$ multiplications over $GF(2^n)$ for GHASH
§ Nonce and associated data are concatenated to form a 2-block "cleartext"
⋆ Attack workload of nonce-respecting adversaries to break the privacy notion or the authenticity notion
⋄ Attack workload of nonce-reusing adversaries to break the authenticity notion

nonce-reusing adversaries, where only a small number of schemes achieve this goal, and most of known modes do fail to provide [18]. See Table 1 for a brief comparison of CLOC to other AEADs.

2 Preliminaries

Let $\{0,1\}^*$ be the set of all finite bit strings, including the empty string ε. For an integer $\ell \geq 0$, let $\{0,1\}^\ell$ be the set of all bit strings of ℓ bits. For $X, Y \in \{0,1\}^*$, we write $X \| Y$, (X, Y), or simply XY to denote their concatenation. For $\ell \geq 0$, we write $0^\ell \in \{0,1\}^\ell$ to denote the bit string that consists of ℓ zeros, and $1^\ell \in \{0,1\}^\ell$ to denote the bit string that consists of ℓ ones. For $X \in \{0,1\}^*$, $|X|$ is its length in bits, and for $\ell \geq 1$, $|X|_\ell = \lceil |X|/\ell \rceil$ is the length in ℓ-bit blocks. For $X \in \{0,1\}^*$ and $\ell \geq 0$ such that $|X| \geq \ell$, $\mathsf{msb}_\ell(X)$ is the most significant (the leftmost) ℓ bits of X. For instance we have $\mathsf{msb}_1(1100) = 1$ and $\mathsf{msb}_3(1100) = 110$. For $X \in \{0,1\}^*$ and $\ell \geq 1$, we write its partition into ℓ-bit blocks as $(X[1], \ldots, X[x]) \overset{\ell}{\leftarrow} X$, which is defined as follows. If $X = \varepsilon$, then $x = 1$ and $X[1] \overset{\ell}{\leftarrow} X$, where $X[1] = \varepsilon$. Otherwise $X[1], \ldots, X[x] \in \{0,1\}^*$ are unique bit strings such that $X[1] \| \cdots \| X[x] = X$, $|X[1]| = \cdots = |X[x-1]| = \ell$, and $1 \leq |X[x]| \leq \ell$. For a finite set $\mathcal{X}$, $X \overset{\$}{\leftarrow} \mathcal{X}$ means that X is chosen uniformly random from $\mathcal{X}$.

In what follows, we fix a block length n and a blockcipher $E : \mathcal{K}_E \times \{0,1\}^n \to \{0,1\}^n$, where $\mathcal{K}_E$ is a non-empty set of keys. Let $\mathrm{Perm}(n)$ be the set of all permutations over $\{0,1\}^n$. We write $E_K \in \mathrm{Perm}(n)$ for the permutation specified by $K \in \mathcal{K}_E$, and $C = E_K(M)$ for the ciphertext of plaintext $M \in \{0,1\}^n$ under key $K \in \mathcal{K}_E$.

3 Specification of CLOC

CLOC takes three parameters, a blockcipher $E : \mathcal{K}_E \times \{0,1\}^n \to \{0,1\}^n$, a nonce length ℓ_N, and a tag length τ. We require $1 \leq \ell_N \leq n - 1$ and $1 \leq \tau \leq n$. We also

require that $n/4$ is an integer. We write $\mathrm{CLOC}[E, \ell_N, \tau]$ for CLOC that is parameterized by E, ℓ_N, and τ, and we often omit the parameters if they are irrelevant or they are clear from the context. $\mathrm{CLOC}[E, \ell_N, \tau] = (\mathrm{CLOC}\text{-}\mathcal{E}, \mathrm{CLOC}\text{-}\mathcal{D})$ consists of the encryption algorithm CLOC-$\mathcal{E}$ and the decryption algorithm CLOC-$\mathcal{D}$.

CLOC-$\mathcal{E}$ and CLOC-$\mathcal{D}$ have the following syntax.

$$\begin{cases} \mathrm{CLOC}\text{-}\mathcal{E} : \mathcal{K}_{\mathrm{CLOC}} \times \mathcal{N}_{\mathrm{CLOC}} \times \mathcal{A}_{\mathrm{CLOC}} \times \mathcal{M}_{\mathrm{CLOC}} \to \mathcal{CT}_{\mathrm{CLOC}} \\ \mathrm{CLOC}\text{-}\mathcal{D} : \mathcal{K}_{\mathrm{CLOC}} \times \mathcal{N}_{\mathrm{CLOC}} \times \mathcal{A}_{\mathrm{CLOC}} \times \mathcal{CT}_{\mathrm{CLOC}} \to \mathcal{M}_{\mathrm{CLOC}} \cup \{\perp\} \end{cases}$$

$\mathcal{K}_{\mathrm{CLOC}} = \mathcal{K}_E$ is the key space, which is identical to the key space of the underlying blockcipher, $\mathcal{N}_{\mathrm{CLOC}} = \{0,1\}^{\ell_N}$ is the nonce space, $\mathcal{A}_{\mathrm{CLOC}} = \{0,1\}^*$ is the associated data space, $\mathcal{M}_{\mathrm{CLOC}} = \{0,1\}^*$ is the plaintext space, $\mathcal{CT}_{\mathrm{CLOC}} = \mathcal{C}_{\mathrm{CLOC}} \times \mathcal{T}_{\mathrm{CLOC}}$ is the ciphertext space, where $\mathcal{C}_{\mathrm{CLOC}} = \{0,1\}^*$ and $\mathcal{T}_{\mathrm{CLOC}} = \{0,1\}^\tau$ is the tag space, and $\perp \notin \mathcal{M}_{\mathrm{CLOC}}$ is the distinguished reject symbol. We write $(C, T) \leftarrow \mathrm{CLOC}\text{-}\mathcal{E}_K(N, A, M)$ and $M \leftarrow \mathrm{CLOC}\text{-}\mathcal{D}_K(N, A, C, T)$ or $\perp \leftarrow \mathrm{CLOC}\text{-}\mathcal{D}_K(N, A, C, T)$, where $(C, T) \in \mathcal{CT}_{\mathrm{CLOC}}$ is a ciphertext, and we also call $C \in \mathcal{C}_{\mathrm{CLOC}}$ a ciphertext.

CLOC-$\mathcal{E}$ and CLOC-$\mathcal{D}$ are defined in Fig. 1. In these algorithms, we use four subroutines, HASH, PRF, ENC, and DEC. They have the following syntax.

$$\begin{cases} \mathsf{HASH} : \mathcal{K}_{\mathrm{CLOC}} \times \mathcal{N}_{\mathrm{CLOC}} \times \mathcal{A}_{\mathrm{CLOC}} \to \{0,1\}^n \\ \mathsf{PRF} : \mathcal{K}_{\mathrm{CLOC}} \times \{0,1\}^n \times \mathcal{C}_{\mathrm{CLOC}} \to \mathcal{T}_{\mathrm{CLOC}} \\ \mathsf{ENC} : \mathcal{K}_{\mathrm{CLOC}} \times \{0,1\}^n \times \mathcal{M}_{\mathrm{CLOC}} \to \mathcal{C}_{\mathrm{CLOC}} \\ \mathsf{DEC} : \mathcal{K}_{\mathrm{CLOC}} \times \{0,1\}^n \times \mathcal{C}_{\mathrm{CLOC}} \to \mathcal{M}_{\mathrm{CLOC}} \end{cases}$$

These subroutines are defined in Fig. 2, and illustrated in Figs. 3, 4, and 5. In the figures, i is the identity function, and $\mathsf{i}(X) = X$ for all $X \in \{0,1\}^n$. In the subroutines, we use the one-zero padding function $\mathsf{ozp} : \{0,1\}^* \to \{0,1\}^*$, the bit-fixing functions $\mathsf{fix0}, \mathsf{fix1} : \{0,1\}^* \to \{0,1\}^*$, and five tweak functions f_1, f_2, g_1, g_2, and h, which are functions over $\{0,1\}^n$.

The one-zero padding function ozp is used to adjust the length of an input string so that the total length becomes a positive multiple of n bits. For $X \in \{0,1\}^*$, $\mathsf{ozp}(X)$ is defined as $\mathsf{ozp}(X) = X$ if $|X| = \ell n$ for some $\ell \geq 1$, and $\mathsf{ozp}(X) = X \,\|\, 10^{n-1-(|X| \bmod n)}$ otherwise. We note that $\mathsf{ozp}(\varepsilon) = 10^{n-1}$, and we also note that, in general, the function is not invertible.

The bit-fixing functions $\mathsf{fix0}$ and $\mathsf{fix1}$ are used to fix the most significant bit of an input string to zero and one, respectively. For $X \in \{0,1\}^*$, $\mathsf{fix0}(X)$ is defined as $\mathsf{fix0}(X) = X \wedge 01^{|X|-1}$, and $\mathsf{fix1}(X)$ is defined as $\mathsf{fix1}(X) = X \vee 10^{|X|-1}$, where $\wedge$ and $\vee$ are the bit-wise AND operation, and the bit-wise OR operation, respectively.

The tweak function h is used in HASH if the most significant bit of $\mathsf{ozp}(A[1])$ is zero. We use f_1 and f_2 in HASH and PRF, where f_1 is used if the last input block is full (i.e., if $|A[a]| = n$ or $|C[m]| = n$) and f_2 is used otherwise. We use g_1 and g_2 in PRF, where we use g_1 if the second argument of the input is the empty string (i.e., $|C| = 0$), and otherwise we use g_2. Now for $X \in \{0,1\}^n$, let $(X[1], X[2], X[3], X[4]) \xleftarrow{n/4} X$. Then f_1, f_2, g_1, g_2, and h are defined as follows.

Algorithm CLOC-$\mathcal{E}_K(N, A, M)$	**Algorithm** CLOC-$\mathcal{D}_K(N, A, C, T)$
1. $V \leftarrow \mathsf{HASH}_K(N, A)$	1. $V \leftarrow \mathsf{HASH}_K(N, A)$
2. $C \leftarrow \mathsf{ENC}_K(V, M)$	2. $T^* \leftarrow \mathsf{PRF}_K(V, C)$
3. $T \leftarrow \mathsf{PRF}_K(V, C)$	3. **if** $T \neq T^*$ **then return** $\perp$
4. **return** (C, T)	4. $M \leftarrow \mathsf{DEC}_K(V, C)$
	5. **return** M

Fig. 1. Pseudocode of the encryption and the decryption algorithms of CLOC

$$\begin{cases} \mathsf{f}_1(X) = (X[1,3], X[2,4], X[1,2,3], X[2,3,4]) \\ \mathsf{f}_2(X) = (X[2], X[3], X[4], X[1,2]) \\ \mathsf{g}_1(X) = (X[3], X[4], X[1,2], X[2,3]) \\ \mathsf{g}_2(X) = (X[2], X[3], X[4], X[1,2]) \\ \mathsf{h}(X) = (X[1,2], X[2,3], X[3,4], X[1,2,4]) \end{cases}$$

Here $X[a,b]$ stands for $X[a] \oplus X[b]$ and $X[a,b,c]$ stands for $X[a] \oplus X[b] \oplus X[c]$.

Alternatively the tweak functions can be specified by a matrix. Let

$$\mathbf{M} = \begin{pmatrix} 0\,0\,0\,1 \\ 1\,0\,0\,1 \\ 0\,1\,0\,0 \\ 0\,0\,1\,0 \end{pmatrix} \tag{1}$$

be a 4×4 binary matrix, and let $\mathbf{M}^i$ for $i \geq 0$ be exponentiations of $\mathbf{M}$, where $\mathbf{M}^0$ denotes the identity matrix. Then we have $\mathsf{f}_1(X) = X \cdot \mathbf{M}^8$, $\mathsf{f}_2(X) = X \cdot \mathbf{M}$, $\mathsf{g}_1(X) = X \cdot \mathbf{M}^2$, $\mathsf{g}_2(X) = X \cdot \mathbf{M}$, and $\mathsf{h}(X) = X \cdot \mathbf{M}^4$, where $X = (X[1], X[2], X[3], X[4])$ is interpreted as a vector.

The design rationale for the tweak functions is explained in Sect. 4.

4 Design Rationale

Overall Structure. At abstract level CLOC is a straightforward combination of CFB mode and CBC MAC, where CBC MAC is called twice for processing associated data and a ciphertext, and CFB mode is called once to generate a ciphertext. However, when we want to achieve low-overhead computation and small memory consumption, we found that any other combination of a basic encryption mode and a MAC mode did not work. For instance, we could not use CTR mode or OFB mode, as they require one state block in processing a plaintext to hold a counter value or a blockcipher output. We then realized that combining CFB mode and CBC MAC was not an easy task. Since we avoid using two keys or using blockcipher pre-calls, such as $L = E_K(0^n)$ used in EAX, we could not computationally separate CFB mode and CBC MAC via input masking, such as Galois-field doubling ($2^i L$ for the i-th block, where $2L$ denotes the multiplication of 2 and L in $\mathrm{GF}(2^n)$) [13,33]. This implies that CFB mode leaks input and output pairs of

Algorithm $\mathsf{HASH}_K(N, A)$

1. $(A[1], \ldots, A[a]) \xleftarrow{n} A$
2. $S_{\mathsf{H}}[1] \leftarrow E_K(\mathsf{fix0}(\mathsf{ozp}(A[1])))$
3. **if** $\mathsf{msb}_1(\mathsf{ozp}(A[1])) = 1$ **then**
4. $S_{\mathsf{H}}[1] \leftarrow \mathsf{h}(S_{\mathsf{H}}[1])$
5. **if** $a \geq 2$ **then**
6.   **for** $i \leftarrow 2$ **to** $a - 1$ **do**
7. $S_{\mathsf{H}}[i] \leftarrow E_K(S_{\mathsf{H}}[i-1] \oplus A[i])$
8. $S_{\mathsf{H}}[a] \leftarrow E_K(S_{\mathsf{H}}[a-1] \oplus \mathsf{ozp}(A[a]))$
9. **if** $|A[a]| = n$ **then**
10. $V \leftarrow \mathsf{f}_1(S_{\mathsf{H}}[a] \oplus \mathsf{ozp}(N))$
11. **else** // $0 \leq |A[a]| \leq n - 1$
12. $V \leftarrow \mathsf{f}_2(S_{\mathsf{H}}[a] \oplus \mathsf{ozp}(N))$
13. **return** V

Algorithm $\mathsf{PRF}_K(V, C)$

1. **if** $|C| = 0$ **then**
2. $T \leftarrow \mathsf{msb}_\tau(E_K(\mathsf{g}_1(V)))$
3.   **return** T
4. $(C[1], \ldots, C[m]) \xleftarrow{n} C$
5. $S_{\mathsf{P}}[0] \leftarrow E_K(\mathsf{g}_2(V))$
6. **for** $i \leftarrow 1$ **to** $m - 1$ **do**
7. $S_{\mathsf{P}}[i] \leftarrow E_K(S_{\mathsf{P}}[i-1] \oplus C[i])$
8. **if** $|C[m]| = n$ **then**
9. $S_{\mathsf{P}}[m] \leftarrow E_K(\mathsf{f}_1(S_{\mathsf{P}}[m-1] \oplus C[m]))$
10. **else** // $1 \leq |C[m]| \leq n - 1$
11. $S_{\mathsf{P}}[m] \leftarrow E_K(\mathsf{f}_2(S_{\mathsf{P}}[m-1] \oplus \mathsf{ozp}(C[m])))$
12. $T \leftarrow \mathsf{msb}_\tau(S_{\mathsf{P}}[m])$
13. **return** T

Algorithm $\mathsf{ENC}_K(V, M)$

1. **if** $|M| = 0$ **then**
2. $C \leftarrow \varepsilon$
3.   **return** C
4. $(M[1], \ldots, M[m]) \xleftarrow{n} M$
5. $S_{\mathsf{E}}[1] \leftarrow E_K(V)$
6. **for** $i \leftarrow 1$ **to** $m - 1$ **do**
7. $C[i] \leftarrow S_{\mathsf{E}}[i] \oplus M[i]$
8. $S_{\mathsf{E}}[i+1] \leftarrow E_K(\mathsf{fix1}(C[i]))$
9. $C[m] \leftarrow \mathsf{msb}_{|M[m]|}(S_{\mathsf{E}}[m]) \oplus M[m]$
10. $C \leftarrow (C[1], \ldots, C[m])$
11. **return** C

Algorithm $\mathsf{DEC}_K(V, C)$

1. **if** $|C| = 0$ **then**
2. $M \leftarrow \varepsilon$
3.   **return** M
4. $(C[1], \ldots, C[m]) \xleftarrow{n} C$
5. $S_{\mathsf{D}}[1] \leftarrow E_K(V)$
6. **for** $i \leftarrow 1$ **to** $m - 1$ **do**
7. $M[i] \leftarrow S_{\mathsf{D}}[i] \oplus C[i]$
8. $S_{\mathsf{D}}[i+1] \leftarrow E_K(\mathsf{fix1}(C[i]))$
9. $M[m] \leftarrow \mathsf{msb}_{|C[m]|}(S_{\mathsf{D}}[m]) \oplus C[m]$
10. $M \leftarrow (M[1], \ldots, M[m])$
11. **return** M

Fig. 2. Subroutines used in the encryption and decryption algorithms of CLOC

the blockcipher calls, which can be freely used to guess or fake the internal chaining value of CBC MAC, leading to a break of the scheme. Lucks [28] proposed an AEAD scheme based on CFB mode, called CCFB. However, the problem is not relevant to CCFB due to the difference in the global structure. To overcome this obstacle in composition, we introduced the bit-fixing functions. Their role is to absolutely separate the input blocks of the blockcipher in CFB mode and *the first input block* of CBC MAC. This imposes the most significant one bit of the input of CBC MAC being fixed to 0, implying one-bit input loss. The set of five tweak functions, which is another tool we introduced in this paper, is used to compensate for this information loss. It also works to compensate the information loss caused by padding functions applied to the last input block to CBC MAC. A similar technique can be found in literature [32,40], however, the previous works only considered MACs and the tweak functions required bit operations.

In the following we explain the specific requirements for the tweak functions.

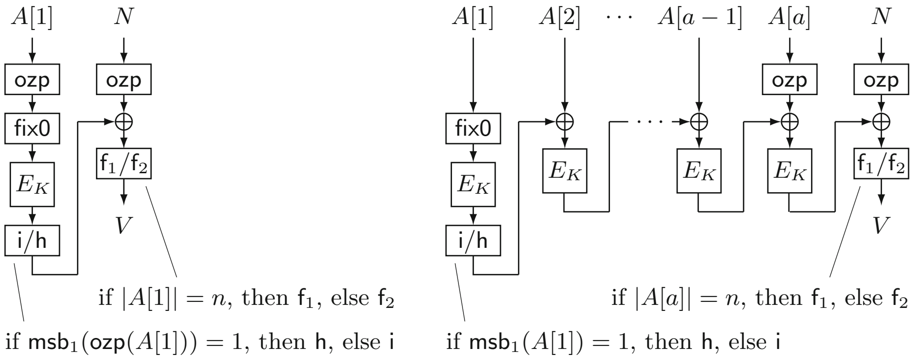

Fig. 3. $V \leftarrow \mathsf{HASH}_K(N, A)$ for $0 \le |A| \le n$ (left) and $|A| \ge n + 1$ (right)

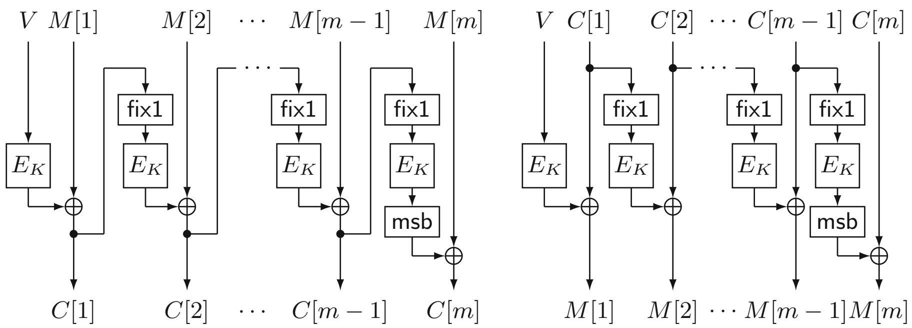

Fig. 4. $C \leftarrow \mathsf{ENC}_K(V, M)$ for $|M| \ge 1$ (left), and $\mathsf{DEC}_K(V, C)$ for $|C| \ge 1$ (right)

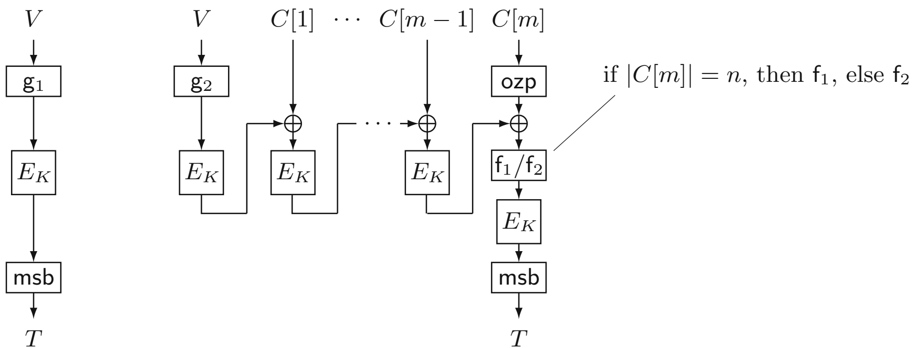

Fig. 5. $T \leftarrow \mathsf{PRF}_K(V, C)$ for $|C| = 0$ (left) and $|C| \ge 1$ (right)

$\mathsf{i} \oplus \mathsf{f}_1$	$\mathsf{i} \oplus \mathsf{f}_2\mathsf{h}$	$\mathsf{f}_1 \oplus \mathsf{f}_2\mathsf{h}$	$\mathsf{h} \oplus \mathsf{g}_2\mathsf{f}_1$	$\mathsf{g}_2\mathsf{f}_1 \oplus \mathsf{g}_1\mathsf{f}_2\mathsf{h}$
$\mathsf{i} \oplus \mathsf{g}_1\mathsf{f}_1$	$\mathsf{i} \oplus \mathsf{h}$	$\mathsf{f}_2 \oplus \mathsf{g}_1\mathsf{f}_1$	$\mathsf{h} \oplus \mathsf{f}_2$	$\mathsf{g}_2\mathsf{f}_1 \oplus \mathsf{f}_2\mathsf{h}$
$\mathsf{i} \oplus \mathsf{g}_1\mathsf{f}_1\mathsf{h}$	$\mathsf{i} \oplus \mathsf{g}_1$	$\mathsf{f}_2 \oplus \mathsf{g}_1\mathsf{f}_1\mathsf{h}$	$\mathsf{h} \oplus \mathsf{g}_1\mathsf{f}_2$	$\mathsf{g}_1\mathsf{f}_2 \oplus \mathsf{g}_2\mathsf{f}_1$
$\mathsf{i} \oplus \mathsf{g}_2\mathsf{f}_1$	$\mathsf{i} \oplus \mathsf{g}_2$	$\mathsf{f}_2 \oplus \mathsf{g}_2\mathsf{f}_1$	$\mathsf{h} \oplus \mathsf{g}_2\mathsf{f}_2$	$\mathsf{g}_1\mathsf{f}_2 \oplus \mathsf{g}_2\mathsf{f}_1\mathsf{h}$
$\mathsf{i} \oplus \mathsf{g}_2\mathsf{f}_1\mathsf{h}$	$\mathsf{f}_1 \oplus \mathsf{g}_1\mathsf{f}_1\mathsf{h}$	$\mathsf{f}_2 \oplus \mathsf{g}_2\mathsf{f}_1\mathsf{h}$	$\mathsf{g}_1\mathsf{f}_1 \oplus \mathsf{f}_1\mathsf{h}$	$\mathsf{g}_1\mathsf{f}_2 \oplus \mathsf{f}_1\mathsf{h}$
$\mathsf{i} \oplus \mathsf{f}_1\mathsf{h}$	$\mathsf{f}_1 \oplus \mathsf{g}_2\mathsf{f}_1\mathsf{h}$	$\mathsf{f}_2 \oplus \mathsf{f}_1\mathsf{h}$	$\mathsf{g}_1\mathsf{f}_1 \oplus \mathsf{g}_2\mathsf{f}_1\mathsf{h}$	$\mathsf{g}_1\mathsf{f}_2 \oplus \mathsf{g}_2\mathsf{f}_2\mathsf{h}$
$\mathsf{i} \oplus \mathsf{f}_2$	$\mathsf{f}_1 \oplus \mathsf{f}_2$	$\mathsf{f}_2 \oplus \mathsf{g}_1\mathsf{f}_2\mathsf{h}$	$\mathsf{g}_1\mathsf{f}_1 \oplus \mathsf{g}_2\mathsf{f}_2$	$\mathsf{g}_1\mathsf{f}_2 \oplus \mathsf{f}_2\mathsf{h}$
$\mathsf{i} \oplus \mathsf{g}_1\mathsf{f}_2$	$\mathsf{f}_1 \oplus \mathsf{g}_1\mathsf{f}_2$	$\mathsf{f}_2 \oplus \mathsf{g}_2\mathsf{f}_2\mathsf{h}$	$\mathsf{g}_1\mathsf{f}_1 \oplus \mathsf{g}_2\mathsf{f}_2\mathsf{h}$	$\mathsf{g}_2\mathsf{f}_2 \oplus \mathsf{g}_1\mathsf{f}_1\mathsf{h}$
$\mathsf{i} \oplus \mathsf{g}_1\mathsf{f}_2\mathsf{h}$	$\mathsf{f}_1 \oplus \mathsf{g}_1\mathsf{f}_2\mathsf{h}$	$\mathsf{g}_1 \oplus \mathsf{g}_2$	$\mathsf{g}_1\mathsf{f}_1 \oplus \mathsf{f}_2\mathsf{h}$	$\mathsf{g}_2\mathsf{f}_2 \oplus \mathsf{f}_1\mathsf{h}$
$\mathsf{i} \oplus \mathsf{g}_2\mathsf{f}_2$	$\mathsf{f}_1 \oplus \mathsf{g}_2\mathsf{f}_2$	$\mathsf{h} \oplus \mathsf{f}_1$	$\mathsf{g}_2\mathsf{f}_1 \oplus \mathsf{g}_1\mathsf{f}_1\mathsf{h}$	$\mathsf{g}_2\mathsf{f}_2 \oplus \mathsf{g}_1\mathsf{f}_2\mathsf{h}$
$\mathsf{i} \oplus \mathsf{g}_2\mathsf{f}_2\mathsf{h}$	$\mathsf{f}_1 \oplus \mathsf{g}_2\mathsf{f}_2\mathsf{h}$	$\mathsf{h} \oplus \mathsf{g}_1\mathsf{f}_1$	$\mathsf{g}_2\mathsf{f}_1 \oplus \mathsf{f}_1\mathsf{h}$	$\mathsf{g}_2\mathsf{f}_2 \oplus \mathsf{f}_2\mathsf{h}$

Fig. 6. Differential probability constraints of $\mathsf{f}_1, \mathsf{f}_2, \mathsf{g}_1, \mathsf{g}_2$, and h

Definition of $\mathsf{f}_1, \mathsf{f}_2, \mathsf{g}_1, \mathsf{g}_2$, *and* h. These functions are defined to meet the following properties. First, they have the additive property. That is, for any $\mathsf{z} \in \{\mathsf{f}_1, \mathsf{f}_2, \mathsf{g}_1, \mathsf{g}_2, \mathsf{h}\}$, we have $\mathsf{z}(X \oplus X') = \mathsf{z}(X) \oplus \mathsf{z}(X')$ for all $X, X' \in \{0,1\}^n$. Next, these functions are invertible over $\{0,1\}^n$. For any $\mathsf{z} \in \{\mathsf{f}_1, \mathsf{f}_2, \mathsf{g}_1, \mathsf{g}_2, \mathsf{h}\}$, we have $\mathsf{z} \in \mathrm{Perm}(n)$. Finally, they satisfy the differential probability constraints specified in Fig. 6. Let z be a function in Fig. 6. Then we require that, for any $Y \in \{0,1\}^n$, $\Pr[\mathsf{z}(K) = Y] = 1/2^n$, where the probability is taken over $K \stackrel{\$}{\leftarrow} \{0,1\}^n$. When z is of the form $\mathsf{z} = \mathsf{z}' \oplus \mathsf{z}''$, then $\mathsf{z}(K)$ stands for $\mathsf{z}'(K) \oplus \mathsf{z}''(K)$. When z is of the form $\mathsf{z} = \mathsf{z}'\mathsf{z}''$, then $\mathsf{z}(K)$ stands for $\mathsf{z}'(\mathsf{z}''(K))$. Recall that we define i as $\mathsf{i}(K) = K$.

Choosing Tweak Functions. Finding simple and word-wise tweak functions fulfilling all properties is not a trivial task. We start with matrix $\mathbf{M}$ of (1), which is invertible and has order 15 (i.e. $\mathbf{M}^{15} = \mathbf{M}^0$), and test all combinations of the form $(\mathsf{f}_1, \mathsf{f}_2, \mathsf{g}_1, \mathsf{g}_2, \mathsf{h}) = (i_1, \ldots, i_5) \in \{1, \ldots, 14\}^5$, where $i_1 = 2$ means $\mathsf{f}_1(X) = X \cdot \mathbf{M}^2$, using a computer. There are 864 candidates out of 537,824 fulfilling the differential probability constraints of Fig. 6. The complexity increases as the index of $\mathbf{M}$ grows, when we implement the tweak function by iterating $\mathbf{M}$, which seems suitable for hardware. For software we would directly implement $\mathbf{M}^i$ using a word-wise permutation and xor, and in this case we observe slight irregular, but similar phenomena (e.g. $\mathbf{M}^1$ needs one xor while $\mathbf{M}^3$ needs three xor's). Figure 7 shows $\mathbf{M}^i$ and the Feistel-like implementations using a word-wise permutation and xor. It shows that, except for $\mathbf{M}^5$ and $\mathbf{M}^{10}$, we have a simple implementation using at most four xor's. Based on these observations, we simply define the cost of computing $\mathbf{M}^i$ as i for $1 \le i \le 7$ and $15 - i$ for $8 \le i \le 14$, and define $f_{\mathrm{cost}}(i_1, \ldots, i_5)$ as

$$\left(i_1 \times \frac{1}{16} + i_2 \times \frac{15}{16}\right) \times 2 + i_4 + i_5 \times \frac{1}{2}.$$

This corresponds to the expected total cost for given $(i_1, \ldots, i_5)$, where associated data and a plaintext are assumed to be non-empty byte strings of random lengths (as we expect the standard use of CLOC is AEAD, not MAC), and we also assume that the most significant bit of the associated data is random. Then there remains

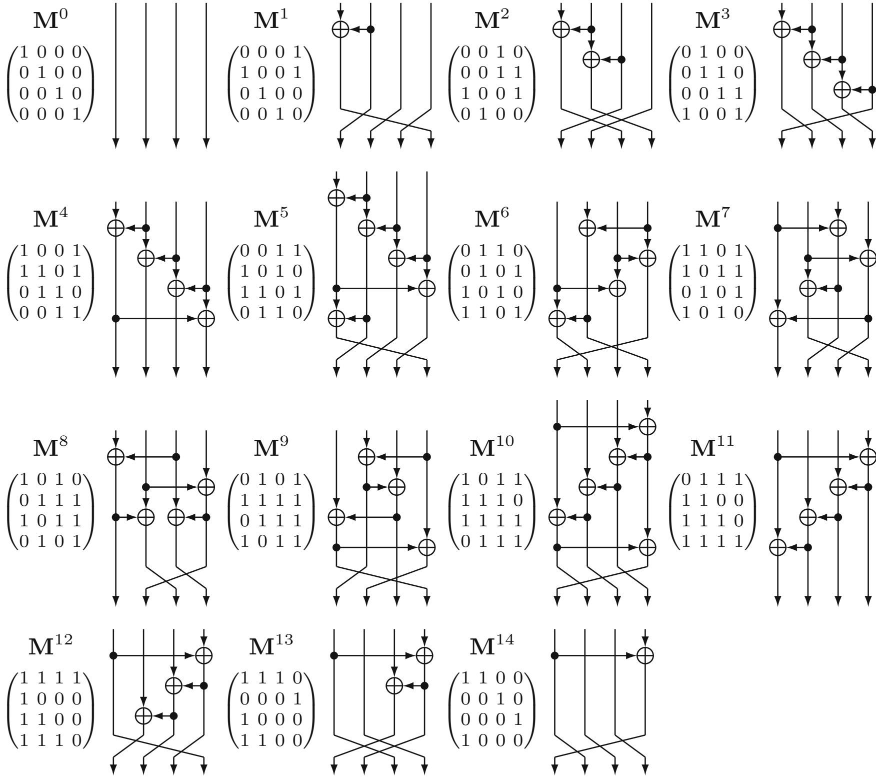

Fig. 7. Matrix exponentiations for the tweak functions

only two candidates giving the minimum value of f_{cost}, which are $(i_1, \ldots, i_5) = (8, 1, 2, 1, 4)$ and $(8, 1, 6, 1, 4)$. As smaller i_3 is better, we choose the former as the sole winner. We also tested other matrices, say the one replacing the forth column of $\mathbf{M}$ by the transposition of $(1, 0, 1, 0)$, but no better solution was found.

We note that $\mathbf{M}^8 = \mathbf{M}^2 \oplus \mathbf{M}^0$ and $\mathbf{M}^4 = \mathbf{M}^1 \oplus \mathbf{M}^0$ hold, implying that we have $\mathsf{f}_1(X) = \mathsf{g}_1(X) \oplus X$ and $\mathsf{h}(X) = \mathsf{f}_2(X) \oplus X = \mathsf{g}_2(X) \oplus X$, which may be useful in some implementations.

5 Security of CLOC

In this section, we define the security notions of a blockcipher and CLOC, and present our security theorems.

PRP Notion. We assume that the blockcipher $E : \mathcal{K}_E \times \{0,1\}^n \to \{0,1\}^n$ is a pseudo-random permutation, or a PRP [27]. We say that P is a random permutation if $P \stackrel{\$}{\leftarrow} \mathrm{Perm}(n)$, and define

$$\mathbf{Adv}_E^{\mathrm{prp}}(\mathcal{A}) \stackrel{\mathrm{def}}{=} \Pr\left[\mathcal{A}^{E_K(\cdot)} \Rightarrow 1\right] - \Pr\left[\mathcal{A}^{P(\cdot)} \Rightarrow 1\right],$$

where the first probability is taken over $K \xleftarrow{\$} \mathcal{K}_E$ and the randomness of $\mathcal{A}$, and the last is over $P \xleftarrow{\$} \mathrm{Perm}(n)$ and $\mathcal{A}$. We write $\mathrm{CLOC}[\mathrm{Perm}(n), \ell_N, \tau]$ for CLOC that uses P as E_K, and the encryption and decryption algorithms are written as CLOC-$\mathcal{E}_P$ and CLOC-$\mathcal{D}_P$. We also consider CLOC that uses a random function as E_K, which is naturally defined as the invertibility of E_K is irrelevant in the definition of CLOC. Let $\mathrm{Rand}(n)$ be the set of all functions from $\{0,1\}^n$ to $\{0,1\}^n$, and we say that R is a random function if $R \xleftarrow{\$} \mathrm{Rand}(n)$. We write $\mathrm{CLOC}[\mathrm{Rand}(n), \ell_N, \tau]$ for CLOC that uses R as E_K, and its encryption and decryption algorithms are written as CLOC-$\mathcal{E}_R$ and CLOC-$\mathcal{D}_R$.

Privacy Notion. We define the privacy notion for $\mathrm{CLOC}[E, \ell_N, \tau] = $ (CLOC-$\mathcal{E}$, CLOC-$\mathcal{D}$). This notion captures the indistinguishably of a nonce-respecting adversary in a chosen plaintext attack setting [34]. We consider an adversary $\mathcal{A}$ that has access to the CLOC encryption oracle, or a random-bits oracle. The encryption oracle takes $(N, A, M) \in \mathcal{N}_{\mathrm{CLOC}} \times \mathcal{A}_{\mathrm{CLOC}} \times \mathcal{M}_{\mathrm{CLOC}}$ as input and returns $(C, T) \leftarrow$ CLOC-$\mathcal{E}_K(N, A, M)$. The random-bits oracle, \$-oracle, takes $(N, A, M) \in \mathcal{N}_{\mathrm{CLOC}} \times \mathcal{A}_{\mathrm{CLOC}} \times \mathcal{M}_{\mathrm{CLOC}}$ as input and returns a random string $(C, T) \xleftarrow{\$} \{0,1\}^{|M|+\tau}$. We define the privacy advantage as

$$\mathbf{Adv}^{\mathrm{priv}}_{\mathrm{CLOC}[E,\ell_N,\tau]}(\mathcal{A}) \stackrel{\mathrm{def}}{=} \Pr\left[\mathcal{A}^{\mathrm{CLOC}\text{-}\mathcal{E}_K(\cdot,\cdot,\cdot)} \Rightarrow 1\right] - \Pr\left[\mathcal{A}^{\$(\cdot,\cdot,\cdot)} \Rightarrow 1\right],$$

where the first probability is taken over $K \xleftarrow{\$} \mathcal{K}_{\mathrm{CLOC}}$ and the randomness of $\mathcal{A}$, and the last is over the random-bits oracle and $\mathcal{A}$. We assume that $\mathcal{A}$ in the privacy game is nonce-respecting, that is, $\mathcal{A}$ does not make two queries with the same nonce.

Privacy Theorem. Let $\mathcal{A}$ be an adversary that makes q queries, and suppose that the queries are $(N_1, A_1, M_1), \ldots, (N_q, A_q, M_q)$. Then we define the total associated data length as $a_1 + \cdots + a_q$, and the total plaintext length as $m_1 + \cdots + m_q$, where $(A_i[1], \ldots, A_i[a_i]) \xleftarrow{n} A_i$ and $(M_i[1], \ldots, M_i[m_i]) \xleftarrow{n} M_i$. We have the following information theoretic result.

Theorem 1. *Let* $\mathrm{Perm}(n)$*,* ℓ_N*, and* τ *be the parameters of* CLOC*. Let* $\mathcal{A}$ *be an adversary that makes at most* q *queries, where the total associated data length is at most* σ_A*, and the total plaintext length is at most* σ_M*. Then we have* $\mathbf{Adv}^{\mathrm{priv}}_{\mathrm{CLOC}[\mathrm{Perm}(n),\ell_N,\tau]}(\mathcal{A}) \le 5\sigma_{\mathrm{priv}}^2/2^n$*, where* $\sigma_{\mathrm{priv}} = q + \sigma_A + 2\sigma_M$*.*

A proof overview is given in Sect. 6, and a complete proof is presented in [23, Appendix A]. If we use a blockcipher E, which is secure in the sense of the PRP notion, instead of $\mathrm{Perm}(n)$, then the corresponding complexity theoretic result can be shown by a standard argument. See e.g. [11]. We note that the privacy of CLOC is broken if the nonce is reused.

Authenticity Notion. We next define the authenticity notion, which captures the unforgeability of an adversary in a chosen ciphertext attack setting [34]. We consider a strong adversary that can repeat the same nonce multiple times. Let $\mathcal{A}$ be an adversary that has access to the CLOC encryption oracle and the CLOC

decryption oracle. The encryption oracle is defined as above. The decryption oracle takes $(N, A, C, T) \in \mathcal{N}_{\mathrm{CLOC}} \times \mathcal{A}_{\mathrm{CLOC}} \times \mathcal{C}_{\mathrm{CLOC}} \times \mathcal{T}_{\mathrm{CLOC}}$ as input and returns $M \leftarrow \text{CLOC-}\mathcal{D}_K(N, A, C, T)$ or $\bot \leftarrow \text{CLOC-}\mathcal{D}_K(N, A, C, T)$. The authenticity advantage is defined as

$$\mathbf{Adv}^{\mathrm{auth}}_{\mathrm{CLOC}[E,\ell_N,\tau]}(\mathcal{A}) \stackrel{\mathrm{def}}{=} \Pr\left[\mathcal{A}^{\text{CLOC-}\mathcal{E}_K(\cdot,\cdot,\cdot),\text{CLOC-}\mathcal{D}_K(\cdot,\cdot,\cdot,\cdot)} \text{ forges}\right],$$

where the probability is taken over $K \stackrel{\$}{\leftarrow} \mathcal{K}_{\mathrm{CLOC}}$ and the randomness of $\mathcal{A}$, and the adversary forges if the decryption oracle returns a bit string (other than $\bot$) for a query (N, A, C, T), but (C, T) was not previously returned to $\mathcal{A}$ from the encryption oracle for a query (N, A, M). The adversary $\mathcal{A}$ in the authenticity game is not necessarily nonce-respecting, and $\mathcal{A}$ can make two or more queries with the same nonce. Specifically, $\mathcal{A}$ can repeat using the same nonce for encryption queries, a nonce used for encryption queries can be used for decryption queries and vice-versa, and the same nonce can be repeated for decryption queries. Without loss of generality, we assume that $\mathcal{A}$ does not make trivial queries, i.e., if the encryption oracle returns (C, T) for a query (N, A, M), then $\mathcal{A}$ does not make a query (N, A, C, T) to the decryption oracle, and $\mathcal{A}$ does not repeat a query.

Authenticity Theorem. Let $\mathcal{A}$ be an adversary that makes q encryption queries and q' decryption queries. Let $(N_1, A_1, M_1), \ldots, (N_q, A_q, M_q)$ be the encryption queries, and $(N'_1, A'_1, C'_1, T'_1), \ldots, (N'_{q'}, A'_{q'}, C'_{q'}, T'_{q'})$ be the decryption queries. Then we define the total associated data length in encryption queries as $a_1 + \cdots + a_q$, the total plaintext length as $m_1 + \cdots + m_q$, the total associated data length in decryption queries as $a'_1 + \cdots + a'_{q'}$, and the total ciphertext length as $m'_1 + \cdots + m'_{q'}$, where $(A_i[1], \ldots, A_i[a_i]) \stackrel{n}{\leftarrow} A_i$, $(M_i[1], \ldots, M_i[m_i]) \stackrel{n}{\leftarrow} M_i$, $(A'_i[1], \ldots, A'_i[a'_i]) \stackrel{n}{\leftarrow} A'_i$, and $(C'_i[1], \ldots, C'_i[m'_i]) \stackrel{n}{\leftarrow} C'_i$. We have the following information theoretic result.

Theorem 2. *Let* $\mathrm{Perm}(n)$*,* ℓ_N*, and* τ *be the parameters of* CLOC*. Let* $\mathcal{A}$ *be an adversary that makes at most* q *encryption queries and at most* q' *decryption queries, where the total associated data length in encryption queries is at most* σ_A*, the total plaintext length is at most* σ_M*, the total associated data length in decryption queries is at most* $\sigma_{A'}$*, and the total ciphertext length is at most* $\sigma_{C'}$*. Then we have* $\mathbf{Adv}^{\mathrm{auth}}_{\mathrm{CLOC}[\mathrm{Perm}(n),\ell_N,\tau]}(\mathcal{A}) \le 5\sigma^2_{\mathrm{auth}}/2^n + q'/2^\tau$*, where* $\sigma_{\mathrm{auth}} = q + \sigma_A + 2\sigma_M + q' + \sigma_{A'} + \sigma_{C'}$*.*

A proof overview is given in Sect. 6, and a complete proof is presented in [23, Appendix A]. As in the privacy case, if we use a blockcipher E secure in the sense of the PRP notion, then we obtain the corresponding complexity theoretic result by a standard argument in, e.g., [11].

6 Overview of Security Proofs

PRP/PRF Switching. The first step is to replace the random permutation P in $\mathrm{CLOC}[\mathrm{Perm}(n), \ell_N, \tau]$ with a random function R, and use the PRP/PRF switching lemma [12] to obtain the followingdifferences.

$$\begin{cases}\mathbf{Adv}^{\mathrm{priv}}_{\mathrm{CLOC}[\mathrm{Perm}(n),\ell_N,\tau]}(\mathcal{A}) - \mathbf{Adv}^{\mathrm{priv}}_{\mathrm{CLOC}[\mathrm{Rand}(n),\ell_N,\tau]}(\mathcal{A}) \\ \mathbf{Adv}^{\mathrm{auth}}_{\mathrm{CLOC}[\mathrm{Perm}(n),\ell_N,\tau]}(\mathcal{A}) - \mathbf{Adv}^{\mathrm{auth}}_{\mathrm{CLOC}[\mathrm{Rand}(n),\ell_N,\tau]}(\mathcal{A})\end{cases}$$

Defining $Q_1,\ldots,Q_{26}$ *and* CLOC2. We define twenty six functions $Q_1,\ldots,Q_{26} : \{0,1\}^n \to \{0,1\}^n$ based on R, K_1, K_2, and K_3, where $K_1,K_2,K_3 \xleftarrow{\$} \{0,1\}^n$ are three independent random n-bit strings. We also define a modified version of $\mathrm{CLOC}[\mathrm{Rand}(n),\ell_N,\tau]$ called $\mathrm{CLOC2}[\ell_N,\tau]$, which uses $Q=(Q_1,\ldots,Q_{26})$ as oracles. Q and CLOC2 are designed so that CLOC-$\mathcal{E}_R$ and CLOC2-$\mathcal{E}_Q$ are the same algorithms, CLOC-$\mathcal{D}_R$ and CLOC2-$\mathcal{D}_Q$ are the same algorithms (except that CLOC2-$\mathcal{D}_Q$ is used for the verification only, and it does not output a plaintext even if the verification succeeds), and $Q_1,\ldots,Q_{26}$ are indistinguishable from $F_1,\ldots,F_{26}$, which are independent random functions. We then have

$$\begin{cases}\mathbf{Adv}^{\mathrm{priv}}_{\mathrm{CLOC}[\mathrm{Rand}(n),\ell_N,\tau]}(\mathcal{A}) = \mathbf{Adv}^{\mathrm{priv}}_{\mathrm{CLOC2}[\ell_N,\tau]}(\mathcal{A}), \\ \mathbf{Adv}^{\mathrm{auth}}_{\mathrm{CLOC}[\mathrm{Rand}(n),\ell_N,\tau]}(\mathcal{A}) = \mathbf{Adv}^{\mathrm{auth}}_{\mathrm{CLOC2}[\ell_N,\tau]}(\mathcal{A}),\end{cases}$$

and we show the distinguishing probability of $Q = (Q_1,\ldots,Q_{26})$ and $F = (F_1,\ldots,F_{26})$ in [23, Lemma 1]. However, the indistinguishability does not hold for arbitrary adversaries. We formalize an input-respecting adversary, and our indistinguishability result in [23, Lemma 1] holds only for these adversaries.

The three random strings, K_1, K_2, and K_3, are secret keys from the adversary's perspective, and we introduce them to show the indistinguishability between Q and F. For instance we know that the input $\mathsf{fix0}(\mathsf{ozp}(A[1]))$ to produce $S_{\mathsf{H}}[1]$ in $\mathsf{HASH}_K(N,A)$ (The 2nd line of $\mathsf{HASH}_K(N,A)$ in Fig. 2) never collides with the input $\mathsf{fix1}(C[i])$ to produce $S_{\mathsf{E}}[i+1]$ in $\mathsf{ENC}_K(V,M)$ (The 8th line of $\mathsf{ENC}_K(V,M)$ in Fig. 2), and hence we can safely assume that they are independent. Likewise, we show that the collision probability between $\mathsf{fix0}(\mathsf{ozp}(A[1]))$ and, say, $S_{\mathsf{H}}[i-1]\oplus A[i]$ in $\mathsf{HASH}_K(N,A)$ (The 7th line of $\mathsf{HASH}_K(N,A)$ in Fig. 2) is low, and the three random strings are introduced to help this argument.

Defining CLOC3. We define another version of $\mathrm{CLOC}[\mathrm{Rand}(n),\ell_N,\tau]$ called $\mathrm{CLOC3}[\ell_N,\tau]$. It uses $F = (F_1,\ldots,F_{26})$ as oracles, and the encryption algorithm CLOC3-$\mathcal{E}_F$ and the decryption algorithm CLOC3-$\mathcal{D}_F$ are obtained from CLOC2-$\mathcal{E}_Q$ and CLOC2-$\mathcal{D}_Q$ by replacing $Q_1,\ldots,Q_{26}$ with $F_1,\ldots,F_{26}$, respectively. We use [23, Lemma 1] to obtain the following differences.

$$\begin{cases}\mathbf{Adv}^{\mathrm{priv}}_{\mathrm{CLOC2}[\ell_N,\tau]}(\mathcal{A}) - \mathbf{Adv}^{\mathrm{priv}}_{\mathrm{CLOC3}[\ell_N,\tau]}(\mathcal{A}) \\ \mathbf{Adv}^{\mathrm{auth}}_{\mathrm{CLOC2}[\ell_N,\tau]}(\mathcal{A}) - \mathbf{Adv}^{\mathrm{auth}}_{\mathrm{CLOC3}[\ell_N,\tau]}(\mathcal{A})\end{cases}$$

The simulations work with input-respecting adversaries, and hence [23, Lemma 1] is sufficient for our purpose.

Indistinguishability of $(\mathsf{HASH3},\mathsf{HASH3}',\mathsf{HASH3}'')$. We then consider three subroutines $\mathsf{HASH3}$, $\mathsf{HASH3}'$, and $\mathsf{HASH3}''$ in $\mathrm{CLOC3}[\ell_N,\tau]$. $\mathsf{HASH3}$ roughly corresponds to a function that computes $S_{\mathsf{E}}[1]$ from (N,A) in $\mathrm{CLOC}[E,\ell_N,\tau]$, i.e.,

$E_K(\mathsf{HASH}_K(N, A))$. $\mathsf{HASH3}'$ computes the tag T when $|C| = 0$, i.e., this function roughly corresponds to $\mathsf{msb}_\tau(E_K(\mathsf{g}_1(\mathsf{HASH}_K(N, A))))$. $\mathsf{HASH3}''$ computes $S_\mathsf{P}[0]$ from (N, A), which is used when $|C| \geq 1$, i.e., $E_K(\mathsf{g}_2(\mathsf{HASH}_K(N, A)))$. Then in [23, Lemma 2], we show that these functions are indistinguishable from three independent random functions $\mathsf{HASH4}$, $\mathsf{HASH4}'$, and $\mathsf{HASH4}''$.

Defining CLOC4. We define another version of CLOC[Rand$(n), \ell_N, \tau$], called CLOC4$[\ell_N, \tau]$. This is obtained by replacing $\mathsf{HASH3}$, $\mathsf{HASH3}'$, and $\mathsf{HASH3}''$ in CLOC3 with $\mathsf{HASH4}$, $\mathsf{HASH4}'$, and $\mathsf{HASH4}''$, respectively. We use [23, Lemma 2] to obtain the following differences.

$$\begin{cases} \mathbf{Adv}^{\mathrm{priv}}_{\mathrm{CLOC3}[\ell_N,\tau]}(\mathcal{A}) - \mathbf{Adv}^{\mathrm{priv}}_{\mathrm{CLOC4}[\ell_N,\tau]}(\mathcal{A}) \\ \mathbf{Adv}^{\mathrm{auth}}_{\mathrm{CLOC3}[\ell_N,\tau]}(\mathcal{A}) - \mathbf{Adv}^{\mathrm{auth}}_{\mathrm{CLOC4}[\ell_N,\tau]}(\mathcal{A}) \end{cases}$$

Indistinguishability of $\mathsf{PRF4}$. We then consider a subroutine called $\mathsf{PRF4}$ in CLOC4. This function outputs a tag T from (N, A, C), and internally uses $\mathsf{HASH4}'$, $\mathsf{HASH4}''$, F_{24}, F_{25}, and F_{26}. We show in [23, Lemma 3] that this function is indistinguishable from a random function $\mathsf{PRF5}$.

Defining CLOC5. We define our final version of CLOC[Rand$(n), \ell_N, \tau$], called CLOC5$[\ell_N, \tau]$, which is obtained from CLOC4 by replacing $\mathsf{PRF4}$ with $\mathsf{PRF5}$. This function is used in both encryption and decryption, and we obtain the following differences from [23, Lemma 3].

$$\begin{cases} \mathbf{Adv}^{\mathrm{priv}}_{\mathrm{CLOC4}[\ell_N,\tau]}(\mathcal{A}) - \mathbf{Adv}^{\mathrm{priv}}_{\mathrm{CLOC5}[\ell_N,\tau]}(\mathcal{A}) \\ \mathbf{Adv}^{\mathrm{auth}}_{\mathrm{CLOC4}[\ell_N,\tau]}(\mathcal{A}) - \mathbf{Adv}^{\mathrm{auth}}_{\mathrm{CLOC5}[\ell_N,\tau]}(\mathcal{A}) \end{cases}$$

Privacy and Authenticity of CLOC5. Finally, we analyze the privacy and the authenticity of CLOC5 in [23, Lemma 4]. The privacy result shows the upper bound on $\mathbf{Adv}^{\mathrm{priv}}_{\mathrm{CLOC5}[\ell_N,\tau]}(\mathcal{A})$, and the proof is reduced to bounding the collision probability among the input values of the random function which is used to encrypt plaintexts. The authenticity result shows the upper bound on $\mathbf{Adv}^{\mathrm{auth}}_{\mathrm{CLOC5}[\ell_N,\tau]}(\mathcal{A})$, and its proof is simple and the result is obtained from the fact that the adversary, even if the nonce is reused, has to guess the output of a random function $\mathsf{PRF5}$ for the input that was not queried before.

We finally obtain the proofs of Theorems 1 and 2 by combining the above differences between advantage functions.

7 Software Implementation

We first tested CLOC on a general-purpose CPU. It is interesting to note that the encryption process and tag generation can be done in parallel, which could speed up the overall computation by a factor close to 2 for long plaintexts, then the final speed could be close to that of encryption only in serial mode. To show that, we implemented CLOC instantiated with AES-128 using the AES new instruction set, and

tested against Intel processor, Core i5-3427U 1.80GHz (Ivy Bridge) [6]. It is known that Intel's AES instruction allows fast parallel processing (up to 4 or 8 blocks), and we used this technique for two parallel inputs to AES. The tested speed for long plaintexts (more than 2^{20} blocks) is around 4.9 cycles per byte (cpb), while AES-128 encrypts at a speed of 4.3 cpb in serial mode. In Table 2, we provide the test vectors.

We then tested CLOC on embedded software. We used an 8-bit microprocessor, Atmel AVR ATmega128 [2]. For comparison we also implemented EAX [13], EAX-prime [31], and OCB3 [26]. For OCB3 we used a byte-oriented code from [7]. OCB3 needs relatively large precomputation for GF doublings, but we modify the code so that the doublings are on-line, since large precomputation may not be suitable to handle short input data for microprocessors. We also considered GCM for comparison, however, recent studies show that GCM does not perform well on constrained devices (see e.g. [10,38]), hence we decided not to include it. All modes are written in C and combined with AES-128. Our AES code is taken from [3], which is written in assembler. AES runs at 156.7 cpb for encryption, 196.8 cpb for decryption, both without key scheduling, and the key scheduling runs at 1,979 cycles. Our codes are complied with Atmel Studio 6 available from [2]. Cycles counts are measured on the simulator of Atmel Studio 6. Table 3 shows the implementation result. ROM denotes the object size in bytes. The speed is measured based on the scenario of non-static associated data, i.e., we excluded key setup and other computations before processing associated data and a nonce, defined as "Init", and figures for Data b denote cycles per byte to process a b-byte plaintext. In EAX, "Init" includes the computation of $E_K(0^n)$, $E_K(0^{n-1}1)$, and $E_K(0^{n-2}10)$. The length of associated data is fixed to 16 bytes except for EAX-prime, and for EAX-prime, we use 32-byte "cleartext," which can be regarded as the combination of associated data and a nonce [31]. For OCB3 we also measured the decryption performance, whereas those of CLOC, EAX, and EAX-prime are almost the same as encryption, since CLOC, EAX and EAX-prime require only forward direction of the underlying blockcipher. The result shows a superior performance of CLOC for short input data, up to around 128 bytes, which would be sufficiently long for

Table 2. Test vector of CLOC instantiated with AES-128

	Length (bytes)	Value (in hex)
Key	16	`000102030405060708090a0b0c0d0e0f0`
Associated data	14	`ff0102030405060708090a0b0c0d`
Nonce	12	`00112233445566778899aabb`
Plaintext	30	`86012204ccebf09ad5305ea8967aebd0`
		`0dd9c05cbde9407ff1ef52f043a2`
Ciphertext	30	`ebd908c23eac555dee406434fb2cffd4`
		`e1bee4401002063e2d13cdf9df3b`
Tag	16	`6621dae27674aa6fbc303426824b2c05`

Table 3. Software implementation on ATmega128

	ROM (bytes)	RAM (bytes)	Init (cycles)	Speed (cycles/byte)					
				Data 16	32	64	96	128	256
CLOC	2980	362	1999	750.1	549.0	448.4	414.9	398.2	373.0
EAX	2772	402	12996	913.6	632.5	490.8	443.6	419.9	384.5
EAX-prime	2588	421	5102	908.7	638.7	496.6	449.3	425.6	390.1
OCB-E	5010	971	4956	1217.5	736.1	495.5	412.2	375.1	315.0
OCB-D	5010	971	4955	1252.2	773.4	534.0	451.2	414.3	354.4

low-power wireless networks, as we mentioned in Sect. 1. We also measure the RAM usage of the AVR implementations, using a public tool [41], based on data of 16 bytes. It is clear to see that CLOC requires much less RAM than OCB3.

8 Hardware Implementation

Although the primary focus of CLOC is embedded software, we also implemented CLOC on hardware to see basic performance figures. We used Altera FPGA, Cyclone IV GX (EP4CGX110DF31C7) [1], and implemented CLOC using AES-128. AES implementation is round-based, and the S-box of AES is based on a composite field [37]. For reference we also wrote EAX for the same device, using the same AES. Both CLOC and EAX use one AES core for encryption and authentication. In EAX implementation, all input masks are stored to registers. Table 4 shows the results. The size is measured by the number of logic elements (LEs). Our implementation is not optimized. Still, these figures show that CLOC has slightly smaller size and faster speed than EAX. Table 4 lacks other important modes, in particular OCB. A more comprehensive comparison and optimized implementation for short input data are interesting future topics.

Table 4. Hardware implementation. Throughput figures of CLOC and EAX are measured for 8-block plaintexts with one-block associated data.

	Size (LE)	Max. Freq. (MHz)	Throughput (Mbit/sec)
CLOC	5628	82.1	400.7
EAX	6453	61.3	342.2
AES Enc.	3175	98.7	971.7

9 Conclusions

We presented a blockcipher mode of operation called CLOC for authenticated encryption with associated data. It uses a variant of CFB mode in its encryption part and a variant of CBC MAC in the authentication part. The scheme efficiently

handles short input data without heavy precomputation nor large memory, and it is suitable for use in microprocessors. We proved CLOC secure, in a reduction-based provable security paradigm, under the assumption that the blockcipher is a pseudorandom permutation. We also presented our preliminary implementation results.

It would be interesting to see improved implementation results using possibly lightweight blockciphers.

Acknowledgments. The authors would like to thank the anonymous FSE 2014 reviewers for helpful comments. The work by Tetsu Iwata was carried out in part while visiting Nanyang Technological University, Singapore. The work by Jian Guo was supported by the Singapore National Research Foundation Fellowship 2012 (NRF-NRFF2012-06).

References

1. Altera Corporation. http://www.altera.com/
2. ATMEL Corporation. http://www.atmel.com/
3. AVR-Crypto-Lib. http://www.das-labor.org/wiki/AVR-Crypto-Lib/en/
4. Bluetooth low energy. http://www.bluetooth.com/Pages/Low-Energy.aspx/
5. Electronic Product Code (EPC) Tag Data Standard (TDS). http://www.epcglobalinc.org/standards/tds/
6. Intel Corporation. http://www.intel.com/
7. OCB News and Code. http://www.cs.ucdavis.edu/~rogaway/ocb/news/
8. ZigBee Alliance. http://www.zigbee.org/
9. Information Technology – Security Techniques – Authenticated Encryption, ISO/IEC 19772:2009. International Standard ISO/IEC 19772 (2009)
10. Bauer, G.R., Potisk, P., Tillich, S.: Comparing Block Cipher Modes of Operation on MICAz Sensor Nodes. In: Baz, D.E., Spies, F., Gross, T. (eds.) PDP, pp. 371–378. IEEE Computer Society (2009)
11. Bellare, M., Kilian, J., Rogaway, P.: The Security of the Cipher Block Chaining Message Authentication Code. J. Comput. Syst. Sci. **61**(3), 362–399 (2000)
12. Bellare, M., Rogaway, P.: The Security of Triple Encryption and a Framework for Code-Based Game-Playing Proofs. In: Vaudenay, S. (ed.) EUROCRYPT 2006. LNCS, vol. 4004, pp. 409–426. Springer, Heidelberg (2006)
13. Bellare, M., Rogaway, P., Wagner, D.: The EAX Mode of Operation. In: Roy and Meier [36], pp. 389–407
14. Bilgin, B., Bogdanov, A., Knežević, M., Mendel, F., Wang, Q.: Fides: Lightweight Authenticated Cipher with Side-Channel Resistance for Constrained Hardware. In: Bertoni, G., Coron, J.-S. (eds.) CHES 2013. LNCS, vol. 8086, pp. 142–158. Springer, Heidelberg (2013)
15. Bogdanov, A., Mendel, F., Regazzoni, F., Rijmen, V., Tischhauser, E.: ALE: AES-Based Lightweight Authenticated Encryption. In: Moriai, S. (ed.) FSE 2013. LNCS, vol. 8424, pp. 447–466. Springer, Heidelberg (2014)
16. Dworkin, M.: Recommendation for Block Cipher Modes of Operation: The CCM Mode for Authentication and Confidentiality, NIST Special Publication 800-38C (2004)
17. Dworkin, M.: Recommendation for Block Cipher Modes of Operation: Galois/Counter Mode (GCM) and GMAC, NIST Special Publication 800-38D (2007)

18. Fleischmann, E., Forler, C., Lucks, S.: McOE: A Family of Almost Foolproof On-Line Authenticated Encryption Schemes. In: Canteaut, A. (ed.) FSE 2012. LNCS, vol. 7549, pp. 196–215. Springer, Heidelberg (2012)
19. Fouque, P.-A., Martinet, G., Valette, F., Zimmer, S.: On the Security of the CCM Encryption Mode and of a Slight Variant. In: Bellovin, S.M., Gennaro, R., Keromytis, A.D., Yung, M. (eds.) ACNS 2008. LNCS, vol. 5037, pp. 411–428. Springer, Heidelberg (2008)
20. Housley, R.: Using Advanced Encryption Standard (AES) CCM Mode with IPsec Encapsulating Security Payload (ESP). IETF RFC 4309 (2005)
21. Housley, R.: Using AES-CCM and AES-GCM Authenticated Encryption in the Cryptographic Message Syntax (CMS). IETF RFC 5084 (2007)
22. Iwata, T., Minematsu, K.: Generating a Fixed Number of Masks with Word Permutations and XORs. DIAC: Directions in Authenticated Ciphers (2013). http://2013.diac.cr.yp.to/
23. Iwata, T., Minematsu, K., Guo, J., Morioka, S.: CLOC: Authenticated Encryption for Short Input. Cryptology ePrint Archive, Report 2014/157 (2014). Full version of this paper http://eprint.iacr.org/
24. Iwata, T., Ohashi, K., Minematsu, K.: Breaking and Repairing GCM Security Proofs. In: Safavi-Naini, R., Canetti, R. (eds.) CRYPTO 2012. LNCS, vol. 7417, pp. 31–49. Springer, Heidelberg (2012)
25. Jonsson, J.: On the Security of CTR + CBC-MAC. In: Nyberg, K., Heys, H. (eds.) SAC 2002. LNCS, vol. 2595, pp. 76–93. Springer, Heidelberg (2003)
26. Krovetz, T., Rogaway, P.: The Software Performance of Authenticated-Encryption Modes. In: Joux, A. (ed.) FSE 2011. LNCS, vol. 6733, pp. 306–327. Springer, Heidelberg (2011)
27. Luby, M., Rackoff, C.: How to Construct Pseudorandom Permutations from Pseudorandom Functions. SIAM J. Comput. **17**(2), 373–386 (1988)
28. Lucks, S.: Two-Pass Authenticated Encryption Faster than Generic Composition. In: Gilbert, H., Handschuh, H. (eds.) FSE 2005. LNCS, vol. 3557, pp. 284–298. Springer, Heidelberg (2005)
29. Minematsu, K., Lucks, S., Iwata, T.: Improved Authenticity Bound of EAX, and Refinements. In: Susilo, W., Reyhanitabar, R. (eds.) ProvSec 2013. LNCS, vol. 8209, pp. 184–201. Springer, Heidelberg (2013)
30. Minematsu, K., Lucks, S., Morita, H., Iwata, T.: Attacks and Security Proofs of EAX-Prime. In: Moriai, S. (ed.) FSE 2013. LNCS, vol. 8424, pp. 327–347. Springer, Heidelberg (2014). http://eprint.iacr.org/2012/018
31. Moise, A., Beroset, E., Phinney, T., Burns, M.: EAX' Cipher Mode, NIST Submission (May 2011). http://csrc.nist.gov/groups/ST/toolkit/BCM/documents/proposedmodes/eax-prime/eax-prime-spec.pdf
32. Nandi, M.: Fast and Secure CBC-Type MAC Algorithms. In: Dunkelman, O. (ed.) FSE 2009. LNCS, vol. 5665, pp. 375–393. Springer, Heidelberg (2009)
33. Rogaway, P.: Efficient Instantiations of Tweakable Blockciphers and Refinements to Modes OCB and PMAC. In: Lee, P.J. (ed.) ASIACRYPT 2004. LNCS, vol. 3329, pp. 16–31. Springer, Heidelberg (2004)
34. Rogaway, P.: Nonce-Based Symmetric Encryption. In: Roy and Meier [36], pp. 348–359
35. Rogaway, P., Wagner, D.: A Critique of CCM (2003). http://csrc.nist.gov/groups/ST/toolkit/BCM/documents/comments/800-38_Series-Drafts/CCM/RW_CCM_comments.pdf
36. Roy, B.K., Meier, W. (eds.): FSE 2004. LNCS, vol. 3017. Springer, Heidelberg (2004)

37. Satoh, A., Morioka, S., Takano, K., Munetoh, S.: A Compact Rijndael Hardware Architecture with S-Box Optimization. In: Boyd, C. (ed.) ASIACRYPT 2001. LNCS, vol. 2248, pp. 239–254. Springer, Heidelberg (2001)
38. Simplício Jr., M.A., de Oliveira, B.T., Barreto, P.S.L.M., Margi, C.B., Carvalho, T.C.M.B., Näslund, M.: Comparison of Authenticated-Encryption Schemes in Wireless Sensor Networks. In: Chou, C.T., Pfeifer, T., Jayasumana, A.P. (eds.) LCN, pp. 450–457. IEEE (2011)
39. Whiting, D., Housley, R., Ferguson, N.: Counter with CBC-MAC. Submission to NIST (2002). http://csrc.nist.gov/groups/ST/toolkit/BCM/modes_development.html
40. Zhang, L., Wu, W., Zhang, L., Wang, P.: CBCR: CBC MAC with Rotating Transformations. Sci. CHINA Inf. Sci. **54**(11), 2247–2255 (2011)
41. Zharkov, E.: EZSTACK: A tool to measure the RAM usage of AVR implementations http://home.comcast.net/~ezstack/ezstack.c