CLOC and SILC [⋆]

Name:	CLOC and SILC v3
Designers/Submitters:	Tetsu Iwata, Nagoya University, Japan Kazuhiko Minematsu, NEC Corporation, Japan Jian Guo, Nanyang Technological University, Singapore Sumio Morioka, Interstellar Technologies, Japan Eita Kobayashi, NEC Corporation, Japan
Contact Address:	Tetsu Iwata, `iwata@cse.nagoya-u.ac.jp`
Date:	September 15, 2016

[⋆] This document was prepared based on [19,20,21,22,23,24,39,12,18].

1 Introduction

This document describes CLOC, Compact Low-Overhead CFB (pronounced as "clock"), and SILC, SImple Lightweight CFB (pronounced as "silk"), which are blockcipher modes of operation for authenticated encryption with associated data (AEAD).

The design of CLOC aims at being provably secure and optimizing the implementation overhead beyond the blockcipher, the precomputation complexity, and the memory requirement. CLOC handles short input data efficiently, and is suitable for use with embedded processors.

CLOC was presented in [19], and the main difference of our CAESAR submission from [19] is that the minimum data unit is defined to be a byte (8 bits) string, and we instantiate CLOC based on AES blockcipher for 16-byte block length and TWINE blockcipher [39] for 8-byte block length.

SILC is built upon CLOC, and the design of SILC aims at optimizing the hardware implementation cost of CLOC. SILC also maintains the provable security based on the pseudorandomness of the underlying blockcipher. SILC is suitable for use within constrained hardware devices, and we instantiate SILC based on AES blockcipher for 16-byte block length and PRESENT [12] and LED [18] for 8-byte block length.

There are many strategies in designing AEAD, and our focus is a blockcipher based scheme without using heavy operations like multiplication over a finite field. This line of research includes CCM [40], EAX [10], and EAX-prime [33], and CLOC and SILC improve these schemes in various ways.

2 Notation, Syntax, and Parameters

Notation. 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. We let $\mathbb{B} = \{0,1\}^8$ be the set of bytes (8-bit strings), and $\mathbb{B}^*$ be the set of all finite byte strings. For $X, Y \in \{0,1\}^*$, we write $X \| Y$, (X, Y), or 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 $1 \leq |X[x]| \leq \ell$.

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 $K \in \mathcal{K}_E$. Following the CAESAR call for submissions, we restrict all input and output variables of CLOC and SILC as byte-strings. Also we assume the big-endian format for all variables.

Syntax and Parameters. Let $\mathit{\Pi} \in \{\mathrm{CLOC}, \mathrm{SILC}\}$. $\mathit{\Pi}$ 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, where ℓ_N and τ are in bits. Here, a nonce corresponds to a public message number specified by the CAESAR call for submissions, and we may interchangeably use both names. $\mathit{\Pi}$ does not have the secret message number, i.e. it is always assumed to be of length zero. We require $1 \leq \ell_N \leq n - 9$ and $1 \leq \tau \leq n$, and assume that $\ell_N/8$ and $\tau/8$ are integers, and $n \in \{64, 96, 128\}$.[⋆] We write $\mathit{\Pi}[E, \ell_N, \tau]$ for $\mathit{\Pi}$ 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. $\mathit{\Pi}[E, \ell_N, \tau] = (\mathit{\Pi}\text{-}\mathcal{E}, \mathit{\Pi}\text{-}\mathcal{D})$ consists of the encryption algorithm $\mathit{\Pi}\text{-}\mathcal{E}$ and the decryption algorithm $\mathit{\Pi}\text{-}\mathcal{D}$.

$\mathit{\Pi}\text{-}\mathcal{E}$ and $\mathit{\Pi}\text{-}\mathcal{D}$ have the following syntax.

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

$\mathcal{K}_\Pi = \mathcal{K}_E$ is the key space, which is identical to the key space of the underlying blockcipher, $\mathcal{N}_\Pi = \mathbb{B}^{\ell_N/8}$ is the nonce space, $\mathcal{A}_\Pi = \mathbb{B}^*$ is the associated data space, $\mathcal{M}_\Pi = \mathbb{B}^*$ is the plaintext space, $\mathcal{CT}_\Pi = \mathcal{C}_\Pi \times \mathcal{T}_\Pi$ is the ciphertext space, where $\mathcal{C}_\Pi = \mathbb{B}^*$ and $\mathcal{T}_\Pi = \mathbb{B}^{\tau/8}$ is the tag space, and $\perp \notin \mathcal{M}_\Pi$ is

[⋆] $n = 96$ is added from version 3. An example for this case appears in Appendix D.

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

the distinguished reject symbol. We write $(C, T) \leftarrow \mathit{\Pi}\text{-}\mathcal{E}_K(N, A, M)$ and $M \leftarrow \mathit{\Pi}\text{-}\mathcal{D}_K(N, A, C, T)$ or $\perp \leftarrow \mathit{\Pi}\text{-}\mathcal{D}_K(N, A, C, T)$. For $\mathit{\Pi} = \text{SILC}$, we make a restriction that the maximum lengths of A, M, and C are all $2^{n/2} - 1$ bytes.[⋆⋆]

3 Specification of CLOC

In this section, we present the specification of $\mathit{\Pi} = \text{CLOC}$.

3.1 Algorithm of CLOC

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.

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

These subroutines are defined in Fig. 2, and illustrated in Fig. 3, Fig. 4, and Fig. 5. We also present equivalent figures in Fig. 6, Fig. 7, and Fig. 8. In the figures, i is the identity function, and $\mathsf{i}(X) = X$ for all $X \in \{0,1\}^n$. In HASH, the nonce N is padded with $\texttt{param} \in \mathbb{B}$ which is an 8-bit constant that depends on the parameters, E, ℓ_N, and τ. See Sect. 3.2 and Sect. 5 for the concrete values of $\texttt{param}$. In the subroutines, we use the one-zero padding function $\mathsf{ozp} : \mathbb{B}^* \rightarrow \mathbb{B}^*$, the bit-fixing functions $\mathsf{fix0}, \mathsf{fix1} : \mathbb{B}^* \rightarrow \mathbb{B}^*$, 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 $X \in \mathbb{B}^*$, $\mathsf{ozp}(X)$ 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 $X \in \mathbb{B}^*$, $\mathsf{fix0}(X)$ 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 one. 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 $(X[1], X[2], X[3], X[4]) \overset{n/4}{\leftarrow} X$. Then f_1, f_2, g_1, g_2, and h are defined as follows.

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

[⋆⋆] This does not mean that CLOC does not have a restriction. See Sect. 7.

```
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.   $V \leftarrow \mathsf{f}_1(S_{\mathsf{H}}[a] \oplus \mathsf{ozp}(\mathtt{param} \,\|\, N))$
11. else                                  // $0 \leq |A[a]| \leq n - 1$
12.   $V \leftarrow \mathsf{f}_2(S_{\mathsf{H}}[a] \oplus \mathsf{ozp}(\mathtt{param} \,\|\, N))$
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

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.

3.2 Parameter Spaces

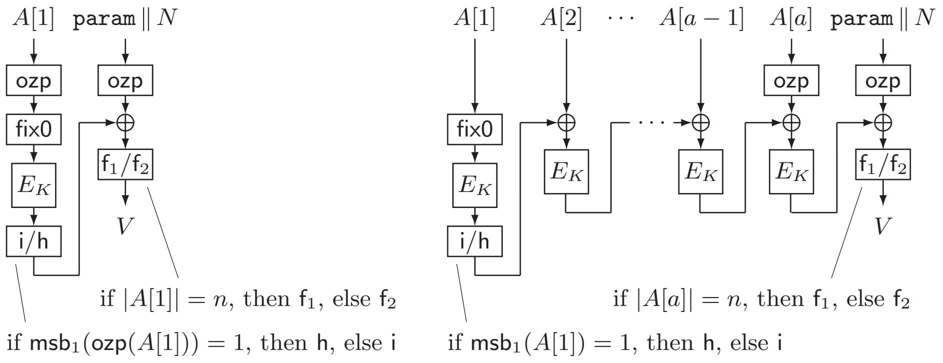
As the CAESAR submission we specify the parameter spaces of CLOC as follows.

– Blockcipher E: AES-128 (AES with 128-bit key), or TWINE-80 (TWINE with 80-bit key).
– Nonce length ℓ_N: For AES-128, $\ell_N \in \{64 \text{ bits (8 byte)}, 96 \text{ bits (12 bytes)}, 112 \text{ bits (14 bytes)}\}$, and for TWINE-80, $\ell_N \in \{32 \text{ bits (4 byte)}, 48 \text{ bits (6 bytes)}\}$.
– Tag length τ: For AES-128, $\tau \in \{32 \text{ bits (4 bytes)}, 64 \text{ bits (8 bytes)}, 96 \text{ bits (12 bytes)}, 128 \text{ bits (16 bytes)}\}$, and for TWINE-80, $\tau \in \{32 \text{ bits (4 bytes)}, 48 \text{ bits (6 bytes)}, 64 \text{ bits (8 bytes)}\}$.

TWINE is a 64-bit blockcipher proposed by Suzaki, Minematsu, Morioka, and Kobayashi at SAC 2012 [39]. The specification of TWINE is described in Appendix A.

The choice of the parameter determines the value of $\mathtt{param} \in \mathbb{B}$ which is concatenated to the nonce N in HASH. The definition of $\mathtt{param}$ is given in Table 1.

We note that CLOC is a blockcipher mode of operation and hence any reasonable blockcipher can be used. In order to meet a possible option, in Appendix D, we present how $\mathtt{param}$ for Simon and Speck [8] is specified when used with CLOC.



$A[1]$
$\mathtt{param} \,\|\, N$
ozp
fix0
E_K
i/h
$\mathsf{f}_1/\mathsf{f}_2$
V
if $|A[1]| = n$, then f_1, else f_2
if $\mathsf{msb}_1(\mathsf{ozp}(A[1])) = 1$, then h, else i
$A[2]$
$A[a-1]$
$A[a]$
if $|A[a]| = n$, then f_1, else f_2
if $\mathsf{msb}_1(A[1]) = 1$, then h, else i

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

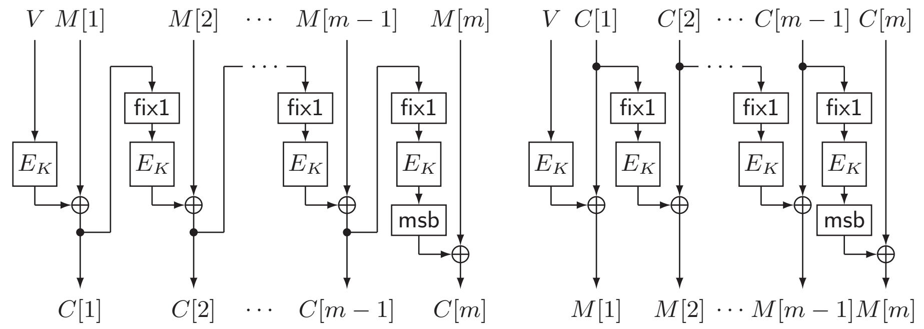
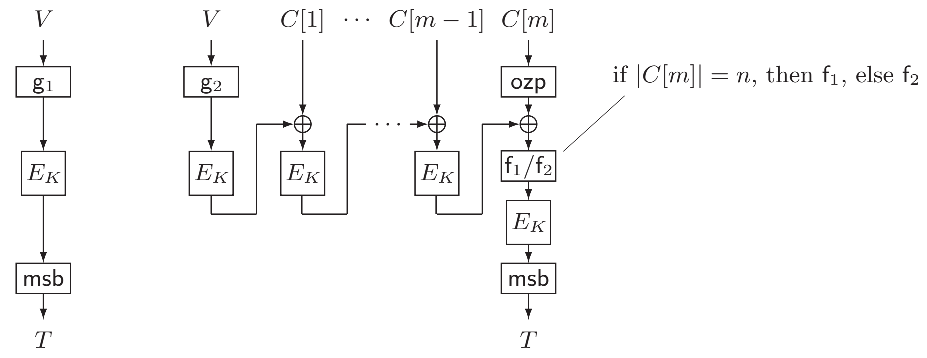


Fig. 4. $C \leftarrow \mathsf{ENC}_K(V, M)$ for $|M| \ge 1$ (left), and $\mathsf{DEC}_K(V, C)$ for $|C| \ge 1$ (right) for CLOC



V
g_1
g_2
E_K
msb
T
$C[1]$
$C[m-1]$
$C[m]$
ozp
$\mathsf{f}_1/\mathsf{f}_2$
if $|C[m]| = n$, then f_1, else f_2

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

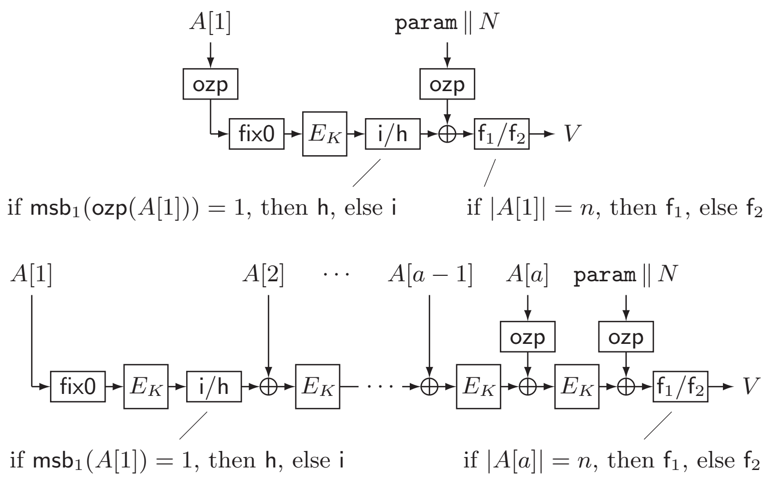

$A[1]$
$\mathtt{param} \| N$
ozp
ozp
fix0
E_K
i/h
$\mathsf{f}_1/\mathsf{f}_2$
V
if $\mathsf{msb}_1(\mathsf{ozp}(A[1])) = 1$, then h, else i
if $|A[1]| = n$, then f_1, else f_2
$A[1]$
$A[2]$
$\cdots$
$A[a-1]$
$A[a]$
$\mathtt{param} \| N$
ozp
ozp
fix0
E_K
i/h
E_K
E_K
E_K
$\mathsf{f}_1/\mathsf{f}_2$
V
if $\mathsf{msb}_1(A[1]) = 1$, then h, else i
if $|A[a]| = n$, then f_1, else f_2

Fig. 6. $V \leftarrow \mathsf{HASH}_K(N, A)$ for $0 \le |A| \le n$ (top) and $|A| \ge n+1$ (bottom) for CLOC

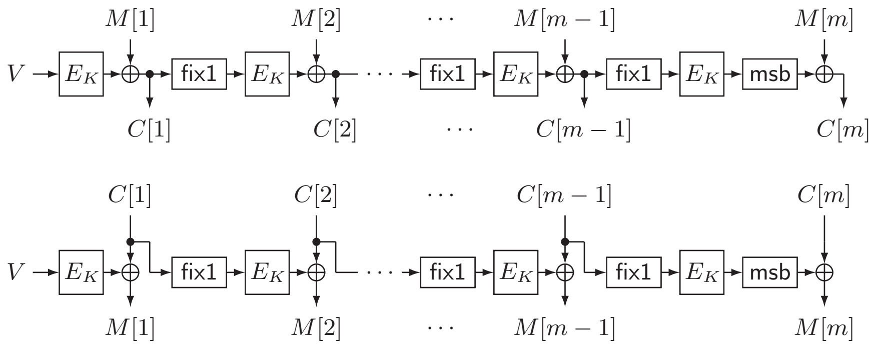


Fig. 7. $C \leftarrow \mathsf{ENC}_K(V, M)$ for $|M| \ge 1$ (top), and $\mathsf{DEC}_K(V, C)$ for $|C| \ge 1$ (bottom) for CLOC

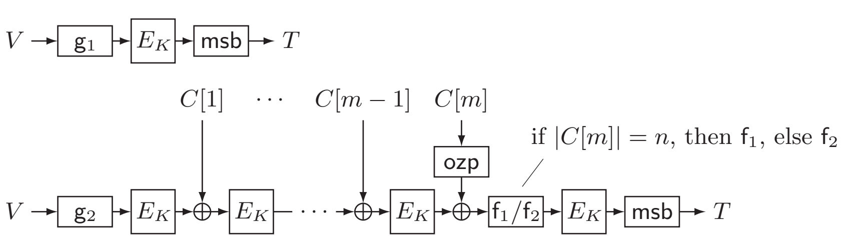


Fig. 8. $T \leftarrow \mathsf{PRF}_K(V, C)$ for $|C| = 0$ (top) and $|C| \ge 1$ (bottom) for CLOC

Table 1. Definition of `param` for CLOC. ℓ_N and τ are written in bytes, and `param` is in hex. The asterisk indicates the recommended parameter.

	E	ℓ_N	τ	`param`
*	AES-128	12	8	`0xc0`
	AES-128	12	12	`0xc1`
	AES-128	12	16	`0xc2`
	AES-128	12	4	`0xc3`
	AES-128	8	8	`0xd0`
	AES-128	8	12	`0xd1`
	AES-128	8	16	`0xd2`
	AES-128	8	4	`0xd3`
	AES-128	14	8	`0xe0`
	AES-128	14	12	`0xe1`
	AES-128	14	16	`0xe2`
	AES-128	14	4	`0xe3`

	E	ℓ_N	τ	`param`
*	TWINE-80	6	4	`0xcc`
	TWINE-80	6	6	`0xcd`
	TWINE-80	6	8	`0xce`
	TWINE-80	4	4	`0xdc`
	TWINE-80	4	6	`0xdd`
	TWINE-80	4	8	`0xde`

```
Algorithm SILC-$\mathcal{E}_K(N, A, M)$

1. $V \leftarrow \mathsf{HASH}_K(N, A)$
2. $C \leftarrow \mathsf{ENC}_K(V, M)$
3. $T \leftarrow \mathsf{PRF}_K(V, C)$
4. return $(C, T)$
```

```
Algorithm SILC-$\mathcal{D}_K(N, A, C, T)$

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

Fig. 9. Pseudocode of the encryption and the decryption algorithms of SILC

4 Specification of SILC

In this section, we present the specification of $\Pi =$ SILC.

4.1 Algorithm of SILC

SILC-$\mathcal{E}$ and SILC-$\mathcal{D}$ are defined in Fig. 9, which are the same as CLOC. In these algorithms, we use four subroutines, HASH, PRF, ENC, and DEC. They have the following syntax.

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

These subroutines are defined in Fig. 10, and illustrated in Fig. 11, Fig. 12, and Fig. 13. Equivalent figures are in Fig. 14, Fig. 15, and Fig. 16. We note that ENC and DEC are the same as those in CLOC. In HASH, the nonce N is padded with `param` $\in \mathbb{B}$ which is an 8-bit constant that depends on the parameters, E, ℓ_N, and τ. See Sect. 4.2 and Sect. 5 for the concrete values of `param`.

In the subroutines, we use the zero prepending function $\mathsf{zpp} : \mathbb{B}^* \to \mathbb{B}^*$, the zero appending function $\mathsf{zap} : \mathbb{B}^* \to \mathbb{B}^*$, the bit-fixing function $\mathsf{fix1} : \mathbb{B}^* \to \mathbb{B}^*$, the tweak function $\mathsf{g} : \{0,1\}^n \to \{0,1\}^n$, and the length encoding function $\mathsf{Len} : \mathbb{B}^* \to \{0,1\}^n$.

Both the zero prepending and appending functions are used to adjust the length of an input string so that the total length becomes a non-negative multiple of n bits (the output is the empty string if and only if the input is the empty string). For $X \in \mathbb{B}^*$, $\mathsf{zpp}(X)$ is defined as

$$\mathsf{zpp}(X) = \begin{cases} X & \text{if } |X| = \ell n \text{ for some } \ell \geq 0, \\ 0^{n-(|X| \bmod n)} \,\|\, X & \text{otherwise,} \end{cases}$$

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

 1. $S_\mathsf{H}[0] \leftarrow E_K(\mathsf{zpp}(\mathtt{param} \,\|\, N))$
 2. if $|A| = 0$ then
 3.    $V \leftarrow \mathsf{g}(S_\mathsf{H}[0] \oplus \mathsf{Len}(A))$   // $\mathsf{Len}(A) = 0^n$
 4.    return $V$
 5. $(A[1], \ldots, A[a]) \overset{n}{\leftarrow} A$
 6. for $i \leftarrow 1$ to $a - 1$ do
 7.    $S_\mathsf{H}[i] \leftarrow E_K(S_\mathsf{H}[i-1] \oplus A[i])$
 8. $S_\mathsf{H}[a] \leftarrow E_K(S_\mathsf{H}[a-1] \oplus \mathsf{zap}(A[a]))$
 9. $V \leftarrow \mathsf{g}(S_\mathsf{H}[a] \oplus \mathsf{Len}(A))$
10. return $V$
```

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

 1. $S_\mathsf{P}[0] \leftarrow E_K(\mathsf{g}(V))$
 2. if $|C| = 0$ then
 3.    $U \leftarrow \mathsf{g}(S_\mathsf{P}[0] \oplus \mathsf{Len}(C))$   // $\mathsf{Len}(C) = 0^n$
 4.    $T \leftarrow \mathsf{msb}_\tau(E_K(U))$
 5.    return $T$
 6. $(C[1], \ldots, C[m]) \overset{n}{\leftarrow} C$
 7. for $i \leftarrow 1$ to $m - 1$ do
 8.    $S_\mathsf{P}[i] \leftarrow E_K(S_\mathsf{P}[i-1] \oplus C[i])$
 9. $S_\mathsf{P}[m] \leftarrow E_K(S_\mathsf{P}[m-1] \oplus \mathsf{zap}(C[m]))$
10. $U \leftarrow \mathsf{g}(S_\mathsf{P}[m] \oplus \mathsf{Len}(C))$
11. $T \leftarrow \mathsf{msb}_\tau(E_K(U))$
12. return $T$
```

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

 1. if $|M| = 0$ then
 2.    $C \leftarrow \varepsilon$
 3.    return $C$
 4. $(M[1], \ldots, M[m]) \overset{n}{\leftarrow} M$
 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. $(C[1], \ldots, C[m]) \overset{n}{\leftarrow} C$
 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. 10. Subroutines used in the encryption and decryption algorithms of SILC

and $\mathsf{zap}(X)$ is defined as

$$\mathsf{zap}(X) = \begin{cases} X & \text{if } |X| = \ell n \text{ for some } \ell \geq 0, \\ X \,\|\, 0^{n-(|X| \bmod n)} & \text{otherwise.} \end{cases}$$

In general, they are not invertible functions.

The bit-fixing function $\mathsf{fix1}$ is the same as the one in CLOC, and is used to fix the most significant bit of an input string to one. For $X \in \mathbb{B}^*$, $\mathsf{fix1}(X)$ is defined as $\mathsf{fix1}(X) = X \vee 10^{|X|-1}$, where $\vee$ denotes the bit-wise OR operation.

The length encoding function $\mathsf{Len} : \mathbb{B}^* \to \{0,1\}^n$ is used to encode the input length (in bytes) in HASH and PRF. For $X \in \mathbb{B}^*$, it is defined as $\mathsf{Len}(X) = \mathsf{str}_n(|X|_8)$, where $\mathsf{str}_n(|X|_8)$ is the standard encoding of $|X|_8$ (the byte length of X) into an n-bit string. For example, when $X = \varepsilon$, we have $\mathsf{Len}(X) = 0^n$, and when $|X|_8 = 5$, we have $\mathsf{Len}(X) = 0^{n-4} \,\|\, 0101$. As the maximum lengths of A, M, and C are all $2^{n/2} - 1$ bytes, the most significant $n/2$ bits of $\mathsf{Len}(X)$ in HASH and PRF are fixed to $0^{n/2}$.

The tweak function $\mathsf{g} : \{0,1\}^n \to \{0,1\}^n$ is used in HASH and PRF.

- For $n = 128$ and $X \in \{0,1\}^n$, $\mathsf{g}(X)$ is defined as

$$\mathsf{g}(X) = (X[2], X[3], \ldots, X[16], X[1,2]),$$

  where $(X[1], X[2], \ldots, X[16]) \overset{n/16}{\leftarrow} X$ and $X[a,b]$ stands for $X[a] \oplus X[b]$.
- For $n = 96$, we let $(X[1], X[2], \ldots, X[12]) \overset{n/12}{\leftarrow} X$ and define $\mathsf{g}(X)$ as

$$\mathsf{g}(X) = (X[2], X[3], \ldots, X[12], X[1,2]).$$

- Similarly, if $n = 64$, we let $(X[1], X[2], \ldots, X[8]) \overset{n/8}{\leftarrow} X$ and define $\mathsf{g}(X)$ as

$$\mathsf{g}(X) = (X[2], X[3], \ldots, X[8], X[1,2]).$$

g can be interpreted as one byte left shift with the rightmost output byte being the xor of the leftmost two input bytes.

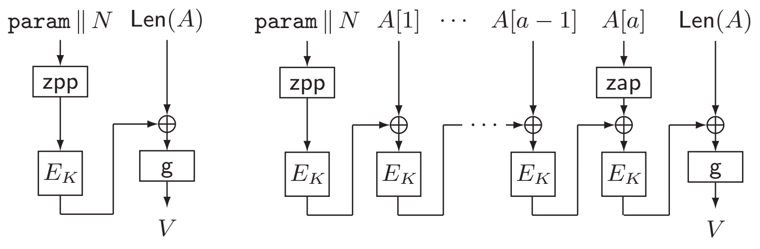


Fig. 11. $V \leftarrow \mathsf{HASH}_K(N, A)$ for $|A| = 0$ (left) and $|A| \geq 1$ (right) for SILC

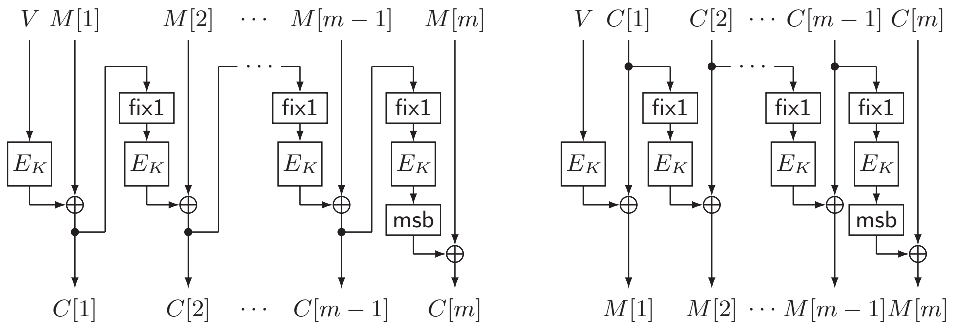


Fig. 12. $C \leftarrow \mathsf{ENC}_K(V, M)$ for $|M| \geq 1$ (left), and $\mathsf{DEC}_K(V, C)$ for $|C| \geq 1$ (right) for SILC

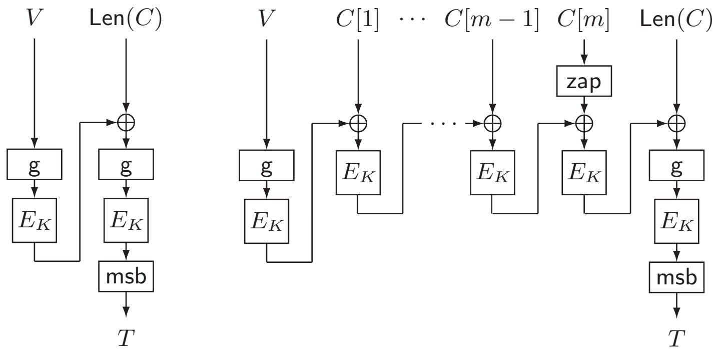


Fig. 13. $T \leftarrow \mathsf{PRF}_K(V, C)$ for $|C| = 0$ (left) and $|C| \geq 1$ (right) for SILC

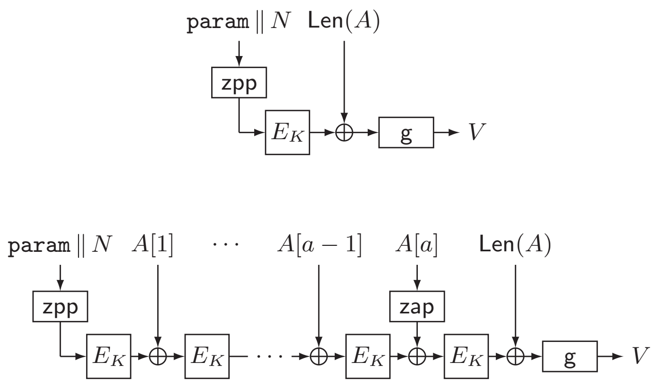


Fig. 14. $V \leftarrow \mathsf{HASH}_K(N, A)$ for $|A| = 0$ (top) and $|A| \geq 1$ (bottom) for SILC

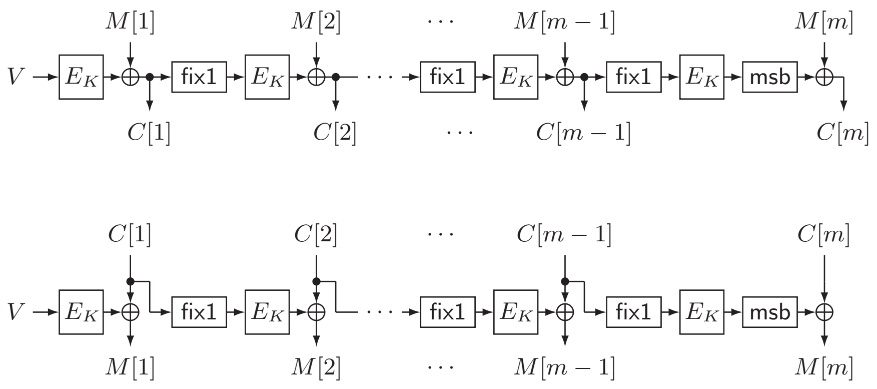


Fig. 15. $C \leftarrow \mathsf{ENC}_K(V, M)$ for $|M| \geq 1$ (top), and $\mathsf{DEC}_K(V, C)$ for $|C| \geq 1$ (bottom) for SILC

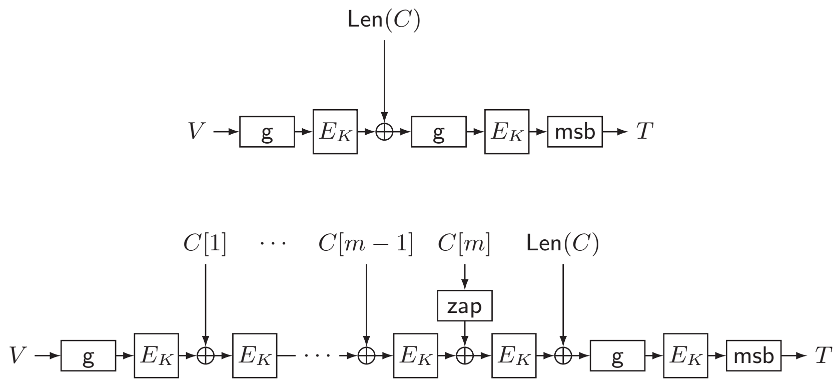


Fig. 16. $T \leftarrow \mathsf{PRF}_K(V, C)$ for $|C| = 0$ (top) and $|C| \geq 1$ (bottom) for SILC

Table 2. Definition of `param` for SILC. ℓ_N and τ are written in bytes, and `param` is in hex. The asterisk indicates the recommended parameter.

	E	ℓ_N	τ	`param`		E	ℓ_N	τ	`param`
*	AES-128	12	8	`0xc0`	*	PRESENT-80	6	4	`0xc4`
	AES-128	12	12	`0xc1`		PRESENT-80	6	6	`0xc5`
	AES-128	12	16	`0xc2`		PRESENT-80	6	8	`0xc6`
	AES-128	12	4	`0xc3`		PRESENT-80	4	4	`0xd4`
	AES-128	8	8	`0xd0`		PRESENT-80	4	6	`0xd5`
	AES-128	8	12	`0xd1`		PRESENT-80	4	8	`0xd6`
	AES-128	8	16	`0xd2`	*	LED-80	6	4	`0xc8`
	AES-128	8	4	`0xd3`		LED-80	6	6	`0xc9`
	AES-128	14	8	`0xe0`		LED-80	6	8	`0xca`
	AES-128	14	12	`0xe1`		LED-80	4	4	`0xd8`
	AES-128	14	16	`0xe2`		LED-80	4	6	`0xd9`
	AES-128	14	4	`0xe3`		LED-80	4	8	`0xda`

4.2 Parameter Spaces

As the CAESAR submission we specify the parameter spaces of SILC as follows.

- Blockcipher E: AES-128 (AES with 128-bit key), or PRESENT-80 (PRESENT with 80-bit key), or LED-80 (LED with 80-bit key).
- Nonce length ℓ_N: For AES-128, $\ell_N \in \{64 \text{ bits (8 byte)}, 96 \text{ bits (12 bytes)}, 112 \text{ bits (14 bytes)}\}$, and for PRESENT-80 and LED-80, $\ell_N \in \{32 \text{ bits (4 byte)}, 48 \text{ bits (6 bytes)}\}$.
- Tag length τ: For AES-128, $\tau \in \{32 \text{ bits (4 bytes)}, 64 \text{ bits (8 bytes)}, 96 \text{ bits (12 bytes)}, 128 \text{ bits (16 bytes)}\}$, and for PRESENT-80 and LED-80, $\tau \in \{32 \text{ bits (4 bytes)}, 48 \text{ bits (6 bytes)}, 64 \text{ bits (8 bytes)}\}$.

PRESENT is a 64-bit blockcipher proposed by Bogdanov et al. at CHES 2007 [12], and LED is a 64-bit blockcipher proposed by Guo et al. at CHES 2011 [18]. The specification of PRESENT is described in Appendix B, and that of LED is described in Appendix C.

The choice of the parameter determines the value of `param` $\in \mathbb{B}$ which is concatenated to the nonce N in HASH. The definition of `param` is given in Table 2.

As with CLOC, SILC can be used with any reasonable blockcipher, and in Appendix D, we present how `param` for SIMON and SPECK [8] is specified when used with SILC.

5 Recommended Parameter Sets and Use Cases

We specify the recommended parameter sets as follows.

- Parameter set 1, `aes128n12t8clocv3`: E = AES-128, $\ell_N = 96$ (12-byte nonce), $\tau = 64$ (8-byte tag)
- Parameter set 2, `aes128n12t8silcv3`: E = AES-128, $\ell_N = 96$ (12-byte nonce), $\tau = 64$ (8-byte tag)
- Parameter set 3, `twine80n6t4clocv3`: E = TWINE-80, $\ell_N = 48$ (6-byte nonce), $\tau = 32$ (4-byte tag)
- Parameter set 4, `present80n6t4silcv3`: E = PRESENT-80, $\ell_N = 48$ (6-byte nonce), $\tau = 32$ (4-byte tag)
- Parameter set 5, `led80n6t4silcv3`: E = LED-80, $\ell_N = 48$ (6-byte nonce), $\tau = 32$ (4-byte tag)

These are marked with the asterisk in Table 1 and Table 2.

Following the CAESAR requirement, we identify prioritized targeted use cases as follows.

- Parameter set 1, `aes128n12t8clocv3`: Use Case 1: Lightweight applications
- Parameter set 2, `aes128n12t8silcv3`: Use Case 1: Lightweight applications
- Parameter set 3, `twine80n6t4clocv3`: Use Case 1: Lightweight applications
- Parameter set 4, `present80n6t4silcv3`: Use Case 1: Lightweight applications
- Parameter set 5, `led80n6t4silcv3`: Use Case 1: Lightweight applications

We note that although CLOC and SILC have some of the features listed in Use Case 3: Defense in depth, they are designed for lightweight applications.

Table 3. Security goal of CLOC for confidentiality (privacy)

Parameter set	`aes128n12t8clocv3`	`twine80n6t4clocv3`
Data	64	32
Time	128	80

Table 4. Security goal of CLOC for integrity (authenticity)

Parameter set	`aes128n12t8clocv3`	`twine80n6t4clocv3`
Data	64	32
Verify	64	32
Time	128	80

6 Security Goals

The security goal of CLOC and SILC is to provide the provable security in terms of confidentiality (or privacy) of plaintexts under nonce-respecting adversaries, and integrity (or authenticity) of plaintext, associated data, and nonce (public message number) under nonce-reusing adversaries. That is, to keep both confidentiality and integrity, the nonce of CLOC and SILC must be unique for all encryptions, and even if this condition is violated for some reason, say by a software error, CLOC and SILC retain the authenticity of sent messages, except for replays (which can be protected by some outer mechanism). Note that CLOC and SILC have no secret message number. CLOC and SILC have provable security guarantees both for confidentiality and integrity, up to the standard birthday bound of the block length of the underlying blockcipher, based on the assumption that the blockcipher is a pseudorandom permutation (PRP). That is, for the block length of n bits, the security is guaranteed provided that the attacker obtains $\sigma \ll 2^{n/2}$ blocks of data. A detailed explanation on the attack models and the provable security bounds are given in Sect. 7.

Attack Workload. We provide security bounds of CLOC and SILC in Sect. 7 based on the pseudorandomness of the underlying blockcipher. We obtain Tables 3, 4, 5, and 6 from these bounds. The variables in the tables denote the required workload of an adversary to break the cipher, in logarithm base 2. If one of the variables reaches the suggested number, then there is no security guarantee anymore, and the cipher can be broken. In Table 3 and Table 5, Data denotes σ_{priv} of our privacy theorems (Theorem 1 and Theorem 2), and this roughly suggests the number of data blocks that the adversary obtains. In Table 4 and Table 6, Data denotes σ_{auth} and Verify denotes q' of our authenticity theorems (Theorem 3 and Theorem 4), where σ_{auth} roughly suggests the number of data blocks that the adversary obtains, and q' denotes the number of decryption queries. In these tables, Time denotes the time complexity, which we assume to be equal to the bit length of the key of the underlying blockcipher. We note that small constant factors are neglected in these tables. For instance the privacy bound in Theorem 1 is $5\sigma_{\mathrm{priv}}^2/2^n$, and it becomes void if $\sigma_{\mathrm{priv}} \approx (2^n/5)^{1/2}$, which is slightly less than $2^{n/2}$.

We have already mentioned that the nonce cannot be repeated to maintain the privacy. As an additional security goal, we claim that the privacy of CLOC and SILC hold as long as the uniqueness of (A, N), a pair of associated data and a nonce, is maintained. That is, even if the nonce is reused, if the uniqueness is maintained as the pair, then the privacy bound still holds. We note that the authenticity holds in this setting as well, since it is maintained even if the nonce is reused.

On the Use of 64*-Bit Blockcipher.* We emphasize that the use of 64-bit blockciphers, TWINE, PRESENT, and `LED`, is not for general purpose applications. The birthday bound for the block length of 64 bits is usually unacceptable for conventional data transmission. As demonstrated by McGrew [30], it leaks information when the total data blocks reach about 32 Gbytes, if the key is not renewed. Moreover Bhargavan and Leurent [11] showed how TLS is attacked if 64-bit blockciphers are used and collisions are practical. However, the parameter sets with 64-bit blockciphers do not focus on commodity channels, e.g., the Internet, but they focus on networks where the data rate is significantly low, with short packet data, and sparse data transmission from edge devices. Many protocols for wireless sensor devices have a low-data rate to suppress the power consumption. For example, IEEE 802.15.4 has 20/40/250 Kbps [4], which is used as physical and data-link layers of popular sensor protocols, Zigbee and 6LoWPAN. Another

Table 5. Security goal of SILC for confidentiality (privacy)

Parameter set	`aes128n12t8silcv3`	`present80n6t4silcv3`	`led80n6t4silcv3`
Data	64	32	32
Time	128	80	80

Table 6. Security goal of SILC for integrity (authenticity)

Parameter set	`aes128n12t8silcv3`	`present80n6t4silcv3`	`led80n6t4silcv3`
Data	64	32	32
Verify	64	32	32
Time	128	80	80

example is Z-Wave, which has 9.6 or 40 Kbps [5]. If edge devices are powered by a small battery, sending 32 Gbytes for one battery is unlikely to be possible in the first place, and rekeying should occur with battery replacement. This naturally implies that the total data blocks sent from one device for its life time is small, hence it may keep the acceptable security even with a 64-bit blockcipher. For example, when the edge device sends data of 512 bytes for every thirty minutes for 10 years (which is exceptionally long for battery-powered sensor devices), the total data amount sent from the device for its life time is about 90 Mbytes. With a standard birthday bound with the block length of 64 bits, the security bound is still below 2^{-17}, which can be acceptable for such constrained devices. Note that this setting assumes only one key, and if the device can renew the key, say, for each year, the bound can be reduced to 2^{-23}.

Of course if the target network has a high-data rate with stable power source, we recommend to use the parameter sets with a 128-bit blockcipher.

7 Security Analysis

In this section, we define the security notions of a blockcipher and $\mathit{\Pi} \in \{\text{CLOC}, \text{SILC}\}$, and present our security theorems. The following descriptions are taken from [19], and they hold for all recommended parameter sets.

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

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

where the first probability is taken over $K \xleftarrow{\$} \mathcal{K}_E$ and the randomness of $\mathcal{A}$, and the last is over $P \xleftarrow{\$} \text{Perm}(n)$ and $\mathcal{A}$. We write $\mathit{\Pi}[\text{Perm}(n), \ell_N, \tau]$ for $\mathit{\Pi} \in \{\text{CLOC}, \text{SILC}\}$ that uses P as E_K, and the encryption and decryption algorithms are written as $\mathit{\Pi}\text{-}\mathcal{E}_P$ and $\mathit{\Pi}\text{-}\mathcal{D}_P$.

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

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

where the first probability is taken over $K \xleftarrow{\$} \mathcal{K}_{\mathit{\Pi}}$ 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 $(A_i[1], \ldots, A_i[a_i]) \overset{n}{\leftarrow} A_i$ and $(M_i[1], \ldots, M_i[m_i]) \overset{n}{\leftarrow} M_i$. We have the following information theoretic result for CLOC.

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* $\mathbf{Adv}^{\mathrm{priv}}_{\mathrm{CLOC}[\mathrm{Perm}(n),\ell_N,\tau]}(\mathcal{A}) \le 5\sigma^2_{\mathrm{priv}}/2^n$, *where* $\sigma_{\mathrm{priv}} = q + \sigma_A + 2\sigma_M$.

A complete proof is presented in [20, 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. [9].

We note that, in general, the privacy of CLOC is broken if the nonce is reused. However, as long as $(N_i, A_i) \ne (N_j, A_j)$ holds for all $1 \le i < j \le q$, the bound in Theorem 1 holds. This is because, in the proof in [20, Appendix A], the transition from CLOC to CLOC5 works without the nonce-respecting assumption, and $\mathsf{HASH5}$ and $\mathsf{PRF5}$ used in CLOC5 generate random and independent output values if $(N_i, A_i) \ne (N_j, A_j)$ holds for all $1 \le i < j \le q$.

For SILC, we have the following information theoretic result.

Theorem 2. *Let* $\mathrm{Perm}(n)$, ℓ_N, *and* τ *be the parameters of* SILC. *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* $\mathbf{Adv}^{\mathrm{priv}}_{\mathrm{SILC}[\mathrm{Perm}(n),\ell_N,\tau]}(\mathcal{A}) \le 5\sigma^2_{\mathrm{priv}}/2^n$, *where* $\sigma_{\mathrm{priv}} = 3q + \sigma_A + 2\sigma_M$.

A complete proof is presented in [24, Appendix C]. If we use a blockcipher E, which is secure in the sense of the PRP notion, then we obtain the corresponding complexity theoretic result. As in CLOC, the privacy of SILC is broken if the nonce is reused, but it remains secure if the uniqueness of (A, N) is maintained.

Authenticity Notion. We next define the authenticity notion for $\Pi[E, \ell_N, \tau] = (\Pi\text{-}\mathcal{E}, \Pi\text{-}\mathcal{D})$, which captures the unforgeability of an adversary in a chosen ciphertext attack setting. We consider a strong adversary that can repeat the same nonce multiple times. Let $\mathcal{A}$ be an adversary that has access to the encryption oracle $\Pi\text{-}\mathcal{E}$ and the decryption oracle $\Pi\text{-}\mathcal{D}$. The encryption oracle is defined as above. The decryption oracle takes $(N, A, C, T) \in \mathcal{N}_\Pi \times \mathcal{A}_\Pi \times \mathcal{C}_\Pi \times \mathcal{T}_\Pi$ as input and returns $M \leftarrow \Pi\text{-}\mathcal{D}_K(N, A, C, T)$ or $\perp \leftarrow \Pi\text{-}\mathcal{D}_K(N, A, C, T)$. The authenticity advantage is defined as

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

where the probability is taken over $K \overset{\$}{\leftarrow} \mathcal{K}_\Pi$ and the randomness of $\mathcal{A}$, and the adversary forges if the decryption oracle returns a bit string (other than $\perp$) 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 $(A_i[1], \ldots, A_i[a_i]) \overset{n}{\leftarrow} A_i$, $(M_i[1], \ldots, M_i[m_i]) \overset{n}{\leftarrow} M_i$, $(A'_i[1], \ldots, A'_i[a'_i]) \overset{n}{\leftarrow} A'_i$, and $(C'_i[1], \ldots, C'_i[m'_i]) \overset{n}{\leftarrow} C'_i$. We have the following information theoretic result for CLOC.

Theorem 3. *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 $\mathbf{Adv}^{\text{auth}}_{\text{CLOC}[\text{Perm}(n),\ell_N,\tau]}(\mathcal{A}) \le 5\sigma_{\text{auth}}^2/2^n + q'/2^\tau$, where $\sigma_{\text{auth}} = q + \sigma_A + 2\sigma_M + q' + \sigma_{A'} + \sigma_{C'}$.*

A complete proof is presented in [20, 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.

For SILC, we have the following information theoretic result.

Theorem 4. *Let* Perm(n), ℓ_N, *and* τ *be the parameters of* SILC. *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 $\mathbf{Adv}^{\text{auth}}_{\text{SILC}[\text{Perm}(n),\ell_N,\tau]}(\mathcal{A}) \le 5\sigma_{\text{auth}}^2/2^n + q'/2^\tau$, where $\sigma_{\text{auth}} = 3q + \sigma_A + 2\sigma_M + 3q' + \sigma_{A'} + \sigma_{C'}$.*

A proof is in [24, Appendix C]. It is standard to obtain the corresponding complexity theoretic result.

8 Features

Features of CLOC. CLOC has the following features.

1. It uses only the encryption of the blockcipher both for encryption and decryption, and does not use bit-wise operations, such as a constant multiplication over GF(2^n).
2. CLOC 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 $|A| \ge 1$. No precomputation other than the blockcipher key scheduling is needed. We note that in CLOC, $1 \le |N| \le n-1$ 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.
3. It works with two state blocks (i.e. $2n$ bits).
4. Both encryption and decryption can be processed in an online manner.[⋆⋆⋆]
5. Static associated data can be processed efficiently if the corresponding intermediate state value is stored.
6. For security, the privacy and authenticity are proved based on the PRP assumption of the blockcipher, assuming standard nonce-respecting adversaries. Moreover, the authenticity is proved with even stronger, nonce-reusing adversaries.

The second feature implies that a number of blockcipher calls required for processing short input data, say 16 or 32 bytes, is small. In particular, CLOC works without any precomputation of the blockcipher, say, computation of $E_K(0^n)$. The precomputation of CLOC is essentially the blockcipher key schedule, hence it can efficiently handle short input data even without precomputation. This feature is particularly desirable for low-power sensor networks, where messages are typically quite short and the devices have limited computational power. CLOC is designed to be used in embedded processors, but this feature may also be useful for powerful processors, for example when the key is frequently changed or when a large number of keys need to be processed. 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 [15], 7 calls (where 3 out of 7 can be precomputed) in EAX [10], and 5 calls (where 1 out of 5 can be precomputed) in EAX-prime [7], where the last one is insecure [32].

The first and the third features imply that CLOC works with small memory and its efficiency for processors with small words (say 8 or 16 bits). With these features CLOC is particularly suitable for embedded processors with severe ROM/RAM constraints. The last feature implies that CLOC provides standard security as a nonce-based AEAD, and in addition a level of security (i.e. authenticity only) even when the nonce is reused, unlike many previous nonce-based AEADs.

[⋆⋆⋆] The onlineness in decryption here merely means that the blockcipher can be called as soon as a block of data is received, and this does not mean that the verification step can be skipped.

Features of SILC. SILC has the following features.

1. It uses only the encryption of the blockcipher both for encryption and decryption.
2. It carefully avoids hardware-unfriendly operations as much as possible, e.g., conditional operation branching, which requires multiplexers in hardware, and dynamic change of data shift amount.
3. It makes $\lceil |N|/n \rceil + \lceil |A|/n \rceil + 2\lceil |M|/n \rceil + 2$ blockcipher calls for a nonce N, associated data A, and a plaintext M. No precomputation other than the blockcipher key scheduling is needed. As a result, no extra hardware register for storing the precomputed result is necessary. We note that in SILC, $1 \le |N| \le n-1$ holds (hence we always have $\lceil |N|/n \rceil = 1$).
4. The memory cost other than the blockcipher is low. It works with two state blocks (i.e. $2n$ bits) to store chaining blocks for encryption and authentication, plus a counter for storing the message length.
5. Both encryption and decryption can be processed in an online manner. [†]
6. For security, the privacy and authenticity are proved based on the PRP assumption of the blockcipher, assuming standard nonce-respecting adversaries. Moreover, the authenticity is proved with even stronger, nonce-reusing adversaries.

The first, second, and fourth features imply SILC's suitability for small hardware. SILC essentially consists a blockcipher encryption function E_K and other functions, zpp, zap, fix1, Len, and g. These functions are chosen by taking the hardware efficiency into account. For instance the 10^* padding function is commonly used in many blockcipher modes, but due to the operation branch depending on the input length, it imposes non-negligible increase in circuit gates compared with zpp or zap. At the cost of one additional blockcipher call for Len, the padding is significantly simplified. The last feature says that SILC has the same security property as CLOC.

Advantages over AES-GCM. Compared with AES-GCM [31], CLOC works efficiently for short input data on embedded processors, and the implementation of CLOC with AES can be smaller, as we do not use a full Galois-Field (GF) multiplier. In particular, AES-GCM is generally inefficient on embedded processors, since the GF multiplier is not fast (e.g. see [17]), while CLOC with AES can be efficiently implemented. For CLOC with TWINE, we expect even smaller implementation, at the cost of reduced security, which will be useful for ultimately tiny processors.

SILC also avoids using a GF multiplier, and the hardware implementation with AES can be smaller. In hardware, AES-GCM is generally fast, however, a fast GF multiplier requires a rather large number of gates, in addition to those needed for the AES encryption function. While SILC with AES can be efficiently implemented, it is also fast if AES is fast. For SILC with PRESENT or LED, we expect even smaller implementations with reduced power consumption, at the cost of reduced security which is reasonable for constrained hardware. The parameter set with PRESENT or LED would be beneficial to tiny devices, such as RFID or CPLD.

With respect to the security, the provable security bound of CLOC and SILC for authentication is better, since the bound of GCM has a term $q'(\ell_A + 1)/2^\tau$, which grows linearly with the block length ℓ_A of the associated data [25], while the corresponding term in CLOC is $q'/2^\tau$. This may have impact when τ is small. Furthermore, in GCM, the existence of weak keys was pointed out [35], while weak keys are not known in CLOC. Also, CLOC and SILC provide some level of security even if the nonce is reused.

Justifications of Parameter Sets. For the 128-bit blockcipher, we select AES for its excellent performance and extensively studied security. For the 64-bit blockcipher, we select TWINE for CLOC for its suitability for embedded processors (comparable speed to AES, smaller code size), and good performance even for high-end platforms with SIMD operations [39]. Notably, TWINE allows efficient processing of two blocks in parallel for a wide range of platforms, which is desirable for CLOC, since in CLOC, the encryption process and tag generation can be done in parallel. We select PRESENT and LED for SILC. Both ciphers can be implemented with small gate size, and in particular, PRESENT is selected for its high throughput, and LED is selected for its high security margin against various cryptanalysis.

For `aes128n12t8clocv3` and `aes128n12t8silcv3`, we select $\ell_N = 96$ from the current trend on the length of the nonce, and this is suitable, for instance, if a part of the nonce is randomly chosen and the other part consists of a counter. For `twine80n6t4clocv3`, `present80n6t4silcv3`, and `led80n6t4silcv3`, we select $\ell_N = 48$ by taking the half of 96 in `aes128n12t8clocv3` and `aes128n12t8silcv3`. For all cases, the tag length was chosen by taking the balance between the security and the data overhead.

[†] The same note as CLOC applies.

Limitations. We also list several limitations of CLOC and SILC. For long input data, CLOC and SILC are not efficient as they need two blockcipher calls per one plaintext block. The nonce length is fixed, which may be problematic in some applications, and the associated data is always processed before the plaintext blocks. The four functions used in CLOC and SILC, HASH, ENC, DEC, and HASH, are all sequential. However, the blockcipher calls in ENC and PRF can be done in parallel. We also note that the parallelization is always possible for multiple messages [14,13].

Due to the existence of five tweak functions in CLOC, the hardware implementation of CLOC does not show the smallest size compared to existing schemes, since the implementation of tweak functions requires many selectors.

SILC is designed to reduce the hardware gates of CLOC as much as possible, while maintaining the provable security based on the pseudorandomness of the underlying blockcipher, at the cost of constant increase in the number of blockcipher calls. It does not handle static associated data efficiently, as we first process a nonce and then associated data. We chose this order as the small hardware is the main target of SILC, and hence it is unlikely that we keep the intermediate state block to improve the efficiency.

9 Design Rationale

The designers have not hidden any weaknesses in this cipher.

Design Rationale of CLOC. The design rationale of CLOC is detailed in [19], and we repeat a part of it for completeness.

Our goal for CLOC is to provide an AEAD particularly efficient for processing short input data, while minimizing the memory consumption and precomputation outside the blockcipher. We mainly focus on constrained sensor networks, where each data packet is short. For example, Zigbee [6] limits the maximum packet length to 127 bytes, Bluetooth low energy limits to 47 bytes [1], and many previous proposals on sensor network security protocols, e.g., TinySec [26], defined similar limits, around 30 to 128 bytes. Another example is EPC tag, which is a replacement of bar-code using RFID and has typically 96 bits [2]. We here describe the design rationale of CLOC for achieving our goal.

At abstract level CLOC is a straightforward combination of CFB and CBC MAC, where CBC MAC is called twice for processing associated data and a ciphertext, and CFB 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 or OFB, 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 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 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 $\mathrm{GF}(2^n)$) [10,37]. This implies that CFB leaks input and output pairs of 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, 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 CFB 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 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 [34,41], however, the previous works only considered MACs and the tweak functions required bit operations.

See [19] for further details about the choice of the tweak functions.

Design Rationale of SILC. Our goal is to provide an AEAD particularly efficient for hardware, requiring a small number of gates other than the blockcipher implementation, that is, a small implementation overhead. For achieving hardware efficiency, we set our design strategy as follows.

– Construct data flow with minimized kinds/amount of functions, minimized flow branching and merging, which implies extra multiplexers and registers, and the use of same ordering of functions in different steps, which makes hardware sharing easy.

– Avoid functions not suitable to hardware, such as dynamic data shifting, which requires a barrel shifter, and integer operations etc.
– Avoid to use many pre-computed values, which consumes extra registers.

SILC is built upon CLOC, and inherits the overall structure. SILC is a combination of CFB and CBC MAC, where we use $\mathsf{fix1}$ and zpp functions to logically separate CFB and CBC MAC. Here, instead of zpp, any function that forces the first input bit to CBC MAC to zero would work, however, we choose zpp for its simplicity in hardware. This loses the capability of efficient handling of static associated data, but we think this is the right treading-off between the size and simplicity, considering our target (e.g. it is unlikely for small hardware to have a memory block and a control logic for caching static associated data).

For the tweak function, as in CLOC, we avoid using GF doubling, and instead, we have adopted the g function to reduce the hardware logic size. When implemented as combinational circuits, the g function is much simpler than the GF doubling because it consists of a static amount of shifting, which consumes no hardware resources, and a minimum amount of xors. The role of the g function is to tweak an input value of the blockcipher, and a similar technique can be found in the context of MAC [34,41]. There is only one tweak function in SILC, which is different from CLOC that has five tweak functions. This means the hardware implementation of SILC does not need many selectors. The tweak function is selected so that it satisfies the following conditions, which is needed for provable security. First, it is linear with respect to xor (i.e. $\mathsf{g}(X \oplus X') = \mathsf{g}(X) \oplus \mathsf{g}(X')$ holds for all $X, X' \in \{0,1\}^n$). Next, it is invertible over $\{0,1\}^n$. Finally, let $K \in \{0,1\}^n$ be uniform over $\{0,1\}^n$. Then, we require that the following functions are (close to) uniform over $\{0,1\}^n$.

$$\begin{cases} \mathsf{g}(K) \\ \mathsf{g}(K) \oplus K \\ \mathsf{g}(\mathsf{g}(K)) \\ \mathsf{g}(\mathsf{g}(K)) \oplus K \\ \mathsf{g}(\mathsf{g}(K)) \oplus \mathsf{g}(K) \end{cases}$$

It can be easily confirmed that our g function fulfills these conditions for $n = 64, 96, 128$ by computing the corresponding matrix ranks over GF(2) as was done in [19].

At the end of HASH and PRF, we use a simple padding function with additional length encoding. Though this always requires one additional blockcipher call compared to popular 10^* padding used by many blockcipher modes, the former is much more efficient in terms of the gate size. We remark that our padding scheme here is similar to the one used in GCM.

Selection of Blockciphers. For $n = 128$, we choose AES as the underlying blockcipher for both CLOC and SILC, because the security of AES has been extensively studied. For $n = 64$, we choose TWINE for CLOC as the underlying blockcipher, because of its low-resource implementation for embedded processors shown in [39]. The use of 64-bit blockciphers is particularly useful if input length is short, say a few bytes, which is in fact possible for ultimately constrained, single-purpose sensors such as energy harvester devices like [3], gas or water metering, etc. In such cases, a 128-bit blockcipher can be inefficient, since it is likely that we have more redundant output bits from the blockcipher that has to be discarded. For SILC, we choose PRESENT and LED as the underlying blockciphers. Both ciphers were chosen for their small hardware size, and we think PRESENT is useful when the application requires high throughput, and LED is useful when long-term security is required, where LED's high security margin will help.

10 Intellectual Property

We claim no intellectual property (IP) rights associated to CLOC nor SILC, and are unaware of any relevant IPs to CLOC or SILC held by others. The statement does not cover the internal blockcipher, and NEC Corporation (NEC) has pending patent applications related to TWINE blockcipher in CLOC: WO2011052585 and WO2011052587. In case that CLOC with TWINE blockcipher is included into the final portfolio, NEC is willing to provide to implementors, solely for the purpose of implementing CLOC, a royalty-free, non-exclusive license under the patents issuing on such patent applications, to the extent such patents are essential to implement CLOC as set forth in the final portfolio, provided said that implementor extends a reciprocal royalty-free license. Nanyang Technological University has a patent related to `LED` blockcipher in SILC: WO2012154129 A1.

If any of this information changes, the submitter will promptly (and within at most one month) announce these changes on the `crypto-competitions` mailing list.

11 Consent

The submitters hereby consent to all decisions of the CAESAR selection committee regarding the selection or non-selection of this submission as a second-round candidate, a third-round candidate, a finalist, a member of the final portfolio, or any other designation provided by the committee. The submitters understand that the committee will not comment on the algorithms, except that for each selected algorithm the committee will simply cite the previously published analyses that led to the selection of the algorithm. The submitters understand that the selection of some algorithms is not a negative comment regarding other algorithms, and that an excellent algorithm might fail to be selected simply because not enough analysis was available at the time of the committee decision. The submitters acknowledge that the committee decisions reflect the collective expert judgments of the committee members and are not subject to appeal. The submitters understand that if they disagree with published analyses then they are expected to promptly and publicly respond to those analyses, not to wait for subsequent committee decisions. The submitters understand that this statement is required as a condition of consideration of this submission by the CAESAR selection committee.

Acknowledgments. The work by Tetsu Iwata was carried out in part while visiting Nanyang Technological University, Singapore, and was supported in part by JSPS KAKENHI, Grant-in-Aid for Scientific Research (B), Grant Number 26280045. The work by Jian Guo was partially supported by the Singapore National Research Foundation Fellowship 2012 (NRF-NRFF2012-06).

References

1. Bluetooth low energy, `http://www.bluetooth.com/Pages/Low-Energy.aspx/`
2. Electronic Product Code (EPC) Tag Data Standard (TDS), `http://www.epcglobalinc.org/standards/tds/`
3. EnOcean, `http://www.enocean.com/`
4. IEEE 802.15.4, `http://en.wikipedia.org/wiki/IEEE_802.15.4/`
5. Z-Wave Alliance, `http://www.z-wavealliance.org/`
6. ZigBee Alliance, `http://www.zigbee.org/`
7. American National Standard Protocol Specification For Interfacing to Data Communication Networks. ANSI C12.22-2008. (2008)
8. Beaulieu, R., Shors, D., Smith, J., Treatman-Clark, S., Weeks, B., Wingers, L.: The SIMON and SPECK Families of Lightweight Block Ciphers. Cryptology ePrint Archive, Report 2013/404 (2013), `http://eprint.iacr.org/2013/404`
9. 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)
10. Bellare, M., Rogaway, P., Wagner, D.: The EAX Mode of Operation. In: Roy, B.K., Meier, W. (eds.) FSE 2004. LNCS, vol. 3017, pp. 389–407. Springer (2004)
11. Bhargavan, K., Leurent, G.: On the Practical (In-)Security of 64-bit Block Ciphers: Collision Attacks on HTTP over TLS and OpenVPN. In: ACM Conference on Computer and Communications Security. ACM (2016)
12. Bogdanov, A., Knudsen, L.R., Leander, G., Paar, C., Poschmann, A., Robshaw, M.J.B., Seurin, Y., Vikkelsoe, C.: PRESENT: An Ultra-Lightweight Block Cipher. In: Paillier, P., Verbauwhede, I. (eds.) CHES 2007. LNCS, vol. 4727, pp. 450–466. Springer (2007)
13. Bogdanov, A., Lauridsen, M.M., Tischhauser, E.: Comb to Pipeline: Fast Software Encryption Revisited. In: Leander, G. (ed.) FSE 2015. LNCS, vol. 9054, pp. 150–171. Springer (2015)
14. 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 (2013)
15. Dworkin, M.: Recommendation for Block Cipher Modes of Operation: The CCM Mode for Authentication and Confidentiality. NIST Special Publication 800-38C (2004)

16. Eichlseder, M.: Remark on variable tag lengths and OMD. A comment on the CAESAR mailing list (2014), https://groups.google.com/forum/#!forum/crypto-competitions
17. Gouvêa, C.P.L., López, J.: High Speed Implementation of Authenticated Encryption for the MSP430X Microcontroller. In: Hevia, A., Neven, G. (eds.) LATINCRYPT 2012. LNCS, vol. 7533, pp. 288–304. Springer (2012)
18. Guo, J., Peyrin, T., Poschmann, A., Robshaw, M.: The LED Block Cipher. In: Preneel, B., Takagi, T. (eds.) CHES 2011. LNCS, vol. 6917, pp. 326–341. Springer (2011), An updated version is available at http://eprint.iacr.org/2012/600
19. Iwata, T., Minematsu, K., Guo, J., Morioka, S.: CLOC: Authenticated Encryption for Short Input. In: Cid, C., Rechberger, C. (eds.) FSE 2014. LNCS, vol. 8540, pp. 149–167. Springer (2014)
20. Iwata, T., Minematsu, K., Guo, J., Morioka, S.: CLOC: Authenticated Encryption for Short Input. Cryptology ePrint Archive, Report 2014/157 (2014), Full version of FSE 2014 paper, http://eprint.iacr.org/2014/157
21. Iwata, T., Minematsu, K., Guo, J., Morioka, S.: CLOC: Compact Low-Overhead CFB. Submission to the CAESAR competition (2014), CLOC v1. March 15, 2014
22. Iwata, T., Minematsu, K., Guo, J., Morioka, S.: CLOC: Compact Low-Overhead CFB. Submission to the CAESAR competition (2015), CLOC v2. August 29, 2015
23. Iwata, T., Minematsu, K., Guo, J., Morioka, S., Kobayashi, E.: SILC: SImple Lightweight CFB. Submission to the CAESAR competition (2014), SILC v1. March 15, 2014
24. Iwata, T., Minematsu, K., Guo, J., Morioka, S., Kobayashi, E.: SILC: SImple Lightweight CFB. Submission to the CAESAR competition (2014), SILC v2. August 29, 2015
25. 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 (2012)
26. Karlof, C., Sastry, N., Wagner, D.: TinySec: a link layer security architecture for wireless sensor networks. In: Stankovic, J.A., Arora, A., Govindan, R. (eds.) SenSys 2004. pp. 162–175. ACM (2004)
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 (2005)
29. Manger, J.H.: [Cfrg] Attacker changing tag length in OCB. A discussion thread on Cfrg (2013), http://www.ietf.org/mail-archive/web/cfrg/current/msg03433.html
30. McGrew, D.: Impossible Plaintext Cryptanalysis and Probable-Plaintext Collision Attacks of 64-bit Block Cipher Modes. Pre-proceedig of FSE 2013 (2013)
31. McGrew, D.A., Viega, J.: The Security and Performance of the Galois/Counter Mode (GCM) of Operation. In: Canteaut, A., Viswanathan, K. (eds.) INDOCRYPT 2004. LNCS, vol. 3348, pp. 343–355. Springer (2004)
32. 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 (2013)
33. Moise, A., Beroset, E., Phinney, T., Burns, M.: EAX' Cipher Mode (May 2011). NIST Submission (2011), http://csrc.nist.gov/groups/ST/toolkit/BCM/documents/proposedmodes/eax-prime/eax-prime-spec.pdf
34. Nandi, M.: Fast and Secure CBC-Type MAC Algorithms. In: Dunkelman, O. (ed.) FSE 2009. LNCS, vol. 5665, pp. 375–393. Springer (2009)
35. Procter, G., Cid, C.: On Weak Keys and Forgery Attacks Against Polynomial-Based MAC Schemes. J. Cryptology 28(4), 769–795 (2015)
36. Rogaway, P., Wagner, D.: A Critique of CCM. Cryptology ePrint Archive, Report 2003/070 (2003), http://eprint.iacr.org/2003/070
37. 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 (2004)
38. Suzaki, T., Minematsu, K.: Improving the Generalized Feistel. In: Hong, S., Iwata, T. (eds.) FSE 2010. LNCS, vol. 6147, pp. 19–39. Springer (2010)
39. Suzaki, T., Minematsu, K., Morioka, S., Kobayashi, E.: TWINE : A Lightweight Block Cipher for Multiple Platforms. In: Knudsen, L.R., Wu, H. (eds.) SAC 2012. LNCS, vol. 7707, pp. 339–354. Springer (2012)
40. Whiting, D., Housley, R., Ferguson, N.: Counter with CBC-MAC (CCM). IETF RFC 3610 (2003)
41. Zhang, L., Wu, W., Zhang, L., Wang, P.: CBCR: CBC MAC with rotating transformations. SCIENCE CHINA Information Sciences 54(11), 2247–2255 (2011)

A TWINE **Blockcipher [39]**

We describe TWINE blockcipher by reusing materials from [39].

Data Processing Part. TWINE is a 64-bit blockcipher with 80 or 128-bit keys. We write TWINE-80 or TWINE-128 to denote the key length. We here focus on TWINE-80, which is used in CLOC. The global structure of TWINE is a variant of Type-2 generalized Feistel structure (GFS) with 16 nibbles (i.e. 4-bit sub-blocks). A round function of TWINE consists of a nonlinear layer using 4-bit S-boxes and a diffusion layer, which is a permutation on 16 nibbles. The diffusion layer of TWINE is not a cyclic shift and is chosen to provide a better diffusion property than the cyclic shift, according to the result of Suzaki and Minematsu [38]. This round function is iterated for 36 times for both key lengths, where the diffusion layer of the last round is omitted. For $i = 1, \ldots, 36$, the i-th round uses a 32-bit round key, RK^i, which is derived from the 80-bit secret key, K, using the key schedule.

The data processing part essentially consists of a 4-bit S-box, denoted by S, and a permutation π over the indexes of 4-bit nibbles. That is, we have $\pi : \{0, \ldots, 15\} \to \{0, \ldots, 15\}$, where the j-th sub-block is mapped to the $\pi[j]$-th sub-block. Fig. 17 shows the encryption procedure, TWINE.Enc, and the decryption procedure, TWINE.Dec, using the derived round keys. Fig. 17 also shows S-box S, and the permutation π and its inverse. In all figures of this section, a variable X may have a subscript (i) to express its length, i.e., X may be written as $X_{(|X|)}$, for clearness. The round function is also illustrated in Fig. 19.

Key Schedule Part. The key schedule produces $RK_{(32\times36)}$ from the 80-bit secret key K. It is also a variant of GFS with nibbles using the same S-box as data processing part. The key schedule uses 6-bit round constants, $CON^i_{(6)} = CON^i_{H(3)} \| CON^i_{L(3)}$ for $i = 1$ to 35. Fig. 18 shows the pseudocode of the key schedule, and Fig. 20 illustrates the key schedule for one round. In Fig. 18 Rot$i(x)$ means i-bit left cyclic shift of x. We remark that CON^i corresponds to 2^i in GF(2^6) with primitive polynomial $z^6 + z + 1$. The values of CON^i are also listed at Fig. 18.

We provide a test vector in Table 7.

```
Algorithm TWINE.Enc($P_{(64)}, RK_{(32\times36)}, C_{(64)}$)
 1. $X^1_{0(4)} \| X^1_{1(4)} \| \ldots \| X^1_{15(4)} \leftarrow P$
 2. $RK^1_{(32)} \| \ldots \| RK^{36}_{(32)} \leftarrow RK_{(32\times36)}$
 3. for $i = 1$ to 35 do
 4.     $RK^i_{0(4)} \| RK^i_{1(4)} \| \ldots \| RK^i_{7(4)} \leftarrow RK^i_{(32)}$
 5.     for $j = 0$ to 7 do
 6.         $X^i_{2j+1} \leftarrow S(X^i_{2j} \oplus RK^i_j) \oplus X^i_{2j+1}$
 7.     for $h = 0$ to 15 do
 8.         $X^{i+1}_{\pi[h]} \leftarrow X^i_h$
 9. for $j = 0$ to 7 do
10.         $X^{36}_{2j+1} \leftarrow S(X^{36}_{2j} \oplus RK^{36}_j) \oplus X^{36}_{2j+1}$
11. $C \leftarrow X^{36}_0 \| X^{36}_1 \| \ldots \| X^{36}_{15}$
```

```
Algorithm TWINE.Dec($C_{(64)}, RK_{(32\times36)}, P_{(64)}$)
 1. $X^{36}_{0(4)} \| X^{36}_{1(4)} \| \ldots \| X^{36}_{15(4)} \leftarrow C$
 2. $RK^1_{(32)} \| \ldots \| RK^{36}_{(32)} \leftarrow RK_{(32\times36)}$
 3. for $i = 36$ to 2 do
 4.     $RK^i_{0(4)} \| RK^i_{1(4)} \| \ldots \| RK^i_{7(4)} \leftarrow RK^i_{(32)}$
 5.     for $j = 0$ to 7 do
 6.         $X^i_{2j+1} \leftarrow S(X^i_{2j} \oplus RK^i_j) \oplus X^i_{2j+1}$
 7.     for $h = 0$ to 15 do
 8.         $X^{i-1}_{\pi^{-1}[h]} \leftarrow X^i_h$
 9. for $j = 0$ to 7 do
10.         $X^1_{2j+1} \leftarrow S(X^1_{2j} \oplus RK^1_j) \oplus X^1_{2j+1}$
11. $P \leftarrow X^1_0 \| X^1_1 \| \ldots \| X^1_{15}$
```

x	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
$S(x)$	C	0	F	A	2	B	9	5	8	3	D	7	1	E	6	4

h	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
$\pi[h]$	5	0	1	4	7	12	3	8	13	6	9	2	15	10	11	14
$\pi^{-1}[h]$	1	2	11	6	3	0	9	4	7	10	13	14	5	8	15	12

Fig. 17. Data processing part of TWINE (top) with S-box S (middle) and permutation π (bottom)

Table 7. A test vector of TWINE-80 in hexadecimal notation

key (80 bits)	00112233 44556677 8899
plaintext	01234567 89ABCDEF
ciphertext	7C1F0F80 B1DF9C28

Algorithm TWINE.KeySchedule-80$(K_{(80)}, RK_{(32\times 36)})$

```
1.  $WK_{0(4)} \| WK_{1(4)} \| \ldots \| WK_{19(4)} \leftarrow K$
2.  for $r = 1$ to 35 do
3.      $RK^r_{(32)} \leftarrow WK_1 \| WK_3 \| WK_4 \| WK_6 \| WK_{13} \| WK_{14} \| WK_{15} \| WK_{16}$
4.      $WK_1 \leftarrow WK_1 \oplus S(WK_0)$
5.      $WK_4 \leftarrow WK_4 \oplus S(WK_{16})$
6.      $WK_7 \leftarrow WK_7 \oplus 0 \| CON^r_H$
7.      $WK_{19} \leftarrow WK_{19} \oplus 0 \| CON^r_L$
8.      $WK_0 \| \ldots \| WK_3 \leftarrow \mathrm{Rot4}(WK_0 \| \ldots \| WK_3)$
9.      $WK_0 \| \ldots \| WK_{19} \leftarrow \mathrm{Rot16}(WK_0 \| \ldots \| WK_{19})$
10. $RK^{36}_{(32)} \leftarrow WK_1 \| WK_3 \| WK_4 \| WK_6 \| WK_{13} \| WK_{14} \| WK_{15} \| WK_{16}$
11. $RK \leftarrow RK^1 \| RK^2 \| \ldots \| RK^{36}$
```

i	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
CON^i	01	02	04	08	10	20	03	06	0C	18	30	23	05	0A	14	28	13	26
i	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	
CON^i	0F	1E	3C	3B	35	29	11	22	07	0E	1C	38	33	25	09	12	24	

Fig. 18. Key schedule of TWINE-80. S-box S is the same as Fig. 17.

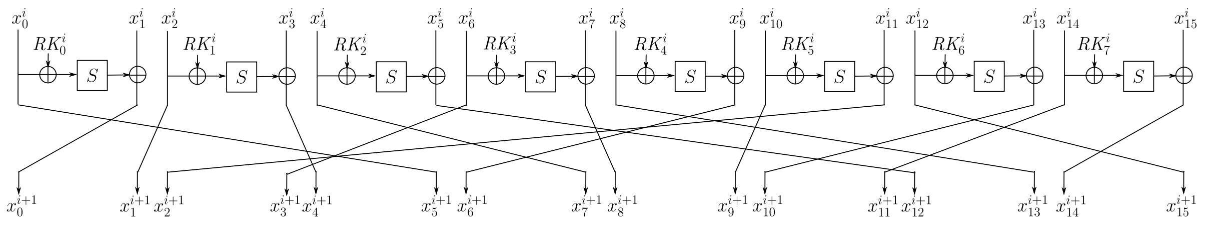


Fig. 19. Round function of TWINE

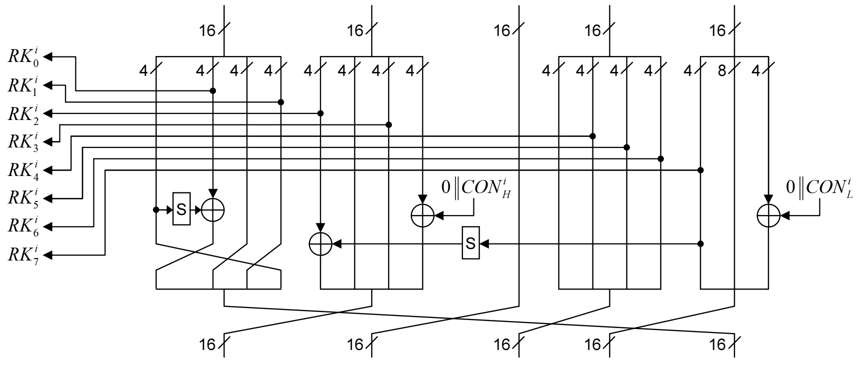


Fig. 20. 80-bit key schedule

B PRESENT [12]

PRESENT is a blockcipher with 80-bit or 128-bit keys, and employs the SP-network. We describe the 80-bit key version, which we write PRESENT-80, using the materials in [12].

It consists of 31 rounds, and each of the 31 rounds consists of an xor operation of a round key K_i for $1 \le i \le 32$, where K_{32} is used for post-whitening, a linear bitwise permutation, and a non-linear substitution layer. The non-linear layer uses a single 4-bit S-box S which is applied 16 times in parallel in each round. The cipher is described in the following pseudocode.

```
1. generateRoundKeys()
2. for $i \leftarrow 1$ to 31 do
```

```
3.     addRoundKey(STATE, $K_i$)
4.     sBoxLayer(STATE)
5.     pLayer(STATE)
6.  end for
7.  addRoundKey(STATE, $K_{32}$)
```

Throughout this section, we number bits from zero with bit zero on the right of a block or word. Each stage is specified below.

addRoundKey. Given round key $K_i = \kappa^i_{63} \dots \kappa^i_0$ for $1 \leq i \leq 32$ and current STATE $b_{63} \dots b_0$, addRoundKey consists of the operation for $0 \leq j \leq 63$,

$$b_j \to b_j \oplus \kappa^i_j.$$

sBoxlayer. The S-box used in PRESENT is a 4-bit to 4-bit S-box $S : \{0,1\}^4 \to \{0,1\}^4$. The following table shows the input and output of the S-box in hexadecimal notation.

x	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
$S[x]$	C	5	6	B	9	0	A	D	3	E	F	8	4	7	1	2

For sBoxLayer the current STATE $b_{63} \dots b_0$ is considered as sixteen 4-bit words $w_{15} \dots w_0$ where $w_i = b_{4*i+3} \,||\, b_{4*i+2} \,||\, b_{4*i+1} \,||\, b_{4*i}$ for $0 \leq i \leq 15$ and the output nibble $S[w_i]$ provides the update state values in the obvious way.

pLayer. The bit permutation used in PRESENT is given by the following table. Bit i of STATE is moved to bit position $P(i)$.

i	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
$P(i)$	0	16	32	48	1	17	33	49	2	18	34	50	3	19	35	51
i	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
$P(i)$	4	20	36	52	5	21	37	53	6	22	38	54	7	23	39	55
i	32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47
$P(i)$	8	24	40	56	9	25	41	57	10	26	42	58	11	27	43	59
i	48	49	50	51	52	53	54	55	56	57	58	59	60	61	62	63
$P(i)$	12	28	44	60	13	29	45	61	14	30	46	62	15	31	47	63

The key schedule. PRESENT can take keys of either 80 or 128 bits. In the 80-bit key version, the user-supplied key is stored in a key register K and represented as $k_{79}k_{78} \dots k_0$. At round i the 64-bit round key $K_i = \kappa_{63}\kappa_{62} \dots \kappa_0$ consists of the 64 leftmost bits of the current contents of register K. Thus at round i we have that:

$$K_i = \kappa_{63}\kappa_{62} \dots \kappa_0 = k_{79}k_{78} \dots k_{16}.$$

After extracting the round key K_i, the key register $K = k_{79}k_{78} \dots k_0$ is updated as follows.

1. $[\mathtt{k}_{79}\mathtt{k}_{78} \dots \mathtt{k}_1\mathtt{k}_0] = [\mathtt{k}_{18}\mathtt{k}_{17} \dots \mathtt{k}_{20}\mathtt{k}_{19}]$
2. $[\mathtt{k}_{79}\mathtt{k}_{78}\mathtt{k}_{77}\mathtt{k}_{76}] = S[\mathtt{k}_{79}\mathtt{k}_{78}\mathtt{k}_{77}\mathtt{k}_{76}]$
3. $[\mathtt{k}_{19}\mathtt{k}_{18}\mathtt{k}_{17}\mathtt{k}_{16}\mathtt{k}_{15}] = [\mathtt{k}_{19}\mathtt{k}_{18}\mathtt{k}_{17}\mathtt{k}_{16}\mathtt{k}_{15}] \oplus$ `round_counter`

Thus, the key register is rotated by 61 bit positions to the left, the left-most four bits are passed through the PRESENT S-box, and the `round_counter` value i is xor'ed with bits $\mathtt{k}_{19}\mathtt{k}_{18}\mathtt{k}_{17}\mathtt{k}_{16}\mathtt{k}_{15}$ of K with the least significant bit of `round_counter` on the right.

C LED [18]

LED [18] is a 64-bit lightweight blockcipher family designed by Guo et al. in 2011, consists of mainly two variants of 64-bit and 128-bit key, denoted as LED-64 and LED-128, respectively. The 64-bit plaintext m is split into 16 4-bit nibbles $m_0\|m_1\|\ldots\|m_{15}$, and can be represented in a square array as:

$$\begin{bmatrix} m_0 & m_1 & m_2 & m_3 \\ m_4 & m_5 & m_6 & m_7 \\ m_8 & m_9 & m_{10} & m_{11} \\ m_{12} & m_{13} & m_{14} & m_{15} \end{bmatrix}$$

LED is AES like, and every round function consists of 4 operations: SubByte, ShiftRow, MixColumn, and AddConstant.

SubByte applies the PRESENT S-box, as already described in Appendix B, to every nibble, i.e., $m_i = S(m_i)$ for $i = 0, \ldots, 15$.

ShiftRow shifts the i-th row to the left by i positions for $i = 0, \ldots, 3$, and the resulted matrix becomes

$$\begin{bmatrix} m_0 & m_1 & m_2 & m_3 \\ m_4 & m_5 & m_6 & m_7 \\ m_8 & m_9 & m_{10} & m_{11} \\ m_{12} & m_{13} & m_{14} & m_{15} \end{bmatrix} \leftarrow \begin{bmatrix} m_0 & m_1 & m_2 & m_3 \\ m_5 & m_6 & m_7 & m_4 \\ m_{10} & m_{11} & m_8 & m_9 \\ m_{15} & m_{12} & m_{13} & m_{14} \end{bmatrix}$$

MixColumn applies Galois-Field multiplication, with irreducible polynomial $f(x) = x^4 + x + 1$, of MDS matrix to each column. The MDS matrix is defined as

$$M = (A)^4 = \begin{pmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 4 & 1 & 2 & 2 \end{pmatrix}^4 = \begin{pmatrix} 4 & 1 & 2 & 2 \\ 8 & 6 & 5 & 6 \\ \mathtt{B} & \mathtt{E} & \mathtt{A} & 9 \\ 2 & 2 & \mathtt{F} & \mathtt{B} \end{pmatrix}.$$

Then for $i = 0, 1, 2, 3$,

$$\begin{bmatrix} m_{i+0} \\ m_{i+4} \\ m_{i+8} \\ m_{i+12} \end{bmatrix} = M \times \begin{bmatrix} m_{i+0} \\ m_{i+4} \\ m_{i+8} \\ m_{i+12} \end{bmatrix}.$$

AddConstant adds a round-dependent value rc and key-size dependent value ks (ks is an 8-bit representation of the master key size) to the state. The constant format is as follows.

$$\begin{bmatrix} 0 \oplus (\mathrm{ks}_7\|\mathrm{ks}_6\|\mathrm{ks}_5\|\mathrm{ks}_4) & (\mathrm{rc}_5\|\mathrm{rc}_4\|\mathrm{rc}_3) & 0 & 0 \\ 1 \oplus (\mathrm{ks}_7\|\mathrm{ks}_6\|\mathrm{ks}_5\|\mathrm{ks}_4) & (\mathrm{rc}_2\|\mathrm{rc}_1\|\mathrm{rc}_0) & 0 & 0 \\ 2 \oplus (\mathrm{ks}_3\|\mathrm{ks}_2\|\mathrm{ks}_1\|\mathrm{ks}_0) & (\mathrm{rc}_5\|\mathrm{rc}_4\|\mathrm{rc}_3) & 0 & 0 \\ 3 \oplus (\mathrm{ks}_3\|\mathrm{ks}_2\|\mathrm{ks}_1\|\mathrm{ks}_0) & (\mathrm{rc}_2\|\mathrm{rc}_1\|\mathrm{rc}_0) & 0 & 0 \end{bmatrix}$$

The values of $(\mathrm{rc}_5, \mathrm{rc}_4, \mathrm{rc}_3, \mathrm{rc}_2, \mathrm{rc}_1, \mathrm{rc}_0)$ for rounds $r = 1, \ldots, 48$ are shown below:

Rounds	Constants
1–24	01,03,07,0F,1F,3E,3D,3B,37,2F,1E,3C,39,33,27,0E,1D,3A,35,2B,16,2C,18,30
25–48	21,02,05,0B,17,2E,1C,38,31,23,06,0D,1B,36,2D,1A,34,29,12,24,08,11,22,04

Every 4 rounds are then grouped together to form a Step, and the key material is added in every step. In this proposal, we make use of LED-80, which follows LED-128. The 80-bit key is padded with '0's and then split into two 64-bit subkeys K_1 and K_2 (note K_1 and K_2 can be encoded in the same way as for plaintext), which are then added into the state alternatively in every one of the 12 steps, as shown in Fig. 21.

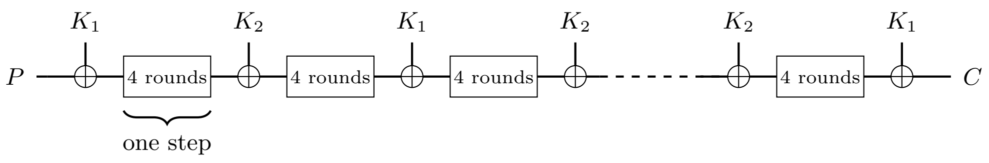


Fig. 21. Encryption of LED-80

Table 8. Values of `param` for CLOC and SILC with SIMON-n/k (left) and SPECK-n/k (right). τ is written in bits and `param` is in hex.

	SIMON-n/k				
τ	64/96	64/128	96/96	128/128	128/256
32	`0xa0`	`0xa1`	`0xa2`	`0xa3`	`0xa4`
48	`0xa5`	`0xa6`	`0xa7`	`0xa8`	`0xa9`
64	`0xaa`	`0xab`	`0xac`	`0xad`	`0xae`

	SPECK-n/k				
τ	64/96	64/128	96/96	128/128	128/256
32	`0xb0`	`0xb1`	`0xb2`	`0xb3`	`0xb4`
48	`0xb5`	`0xb6`	`0xb7`	`0xb8`	`0xb9`
64	`0xba`	`0xbb`	`0xbc`	`0xbd`	`0xbe`

D Using Other Blockciphers

CLOC and SILC are blockcipher modes of operation and any blockcipher can be used. In this appendix, we describe how other blockciphers can be used by taking examples of SIMON and SPECK in [8] as possible options (and hence the specifications of SIMON and SPECK are not included in this document. See [8]).

They are lightweight blockciphers, and SIMON is a family of 10 blockciphers depending on the block and key lengths. SPECK also consists of 10 blockciphers, and SIMON-n/k refers to SIMON with block length n bits and key length k bits. SPECK-n/k is analogously defined. We consider SIMON-n/k and SPECK-n/k for $(n, k) = (64, 96), (64, 128), (96, 96), (128, 128), (128, 256)$.

When we use SIMON-n/k or SPECK-n/k in CLOC or SILC, we let the nonce length be

$$\ell_N = \begin{cases} 96 & \text{if } n = 128, \\ 72 & \text{if } n = 96, \\ 48 & \text{if } n = 64 \end{cases}$$

in bits, $\tau \in \{32, 48, 64\}$ in bits, and the values of `param` are as specified in Table 8.

E Changes

E.1 Changes from CLOC v1 to CLOC v2

The specification of CLOC v2 uses `param` so that the encryption and decryption algorithms depend on the choice of the parameters, which are E, ℓ_N, and τ. This type of dependency was previously highlighted, e.g., in [16,29,36]. There are three things to note:

- The introduction of `param` does not mean that CLOC v2 handles variable length nonces nor variable length tags. All the parameters, E, ℓ_N, and τ, have to be fixed during the lifetime of the secret key.
- The introduction of `param` does not affect the provable security result of CLOC, since we may consider `param` $\|\, N$ as a nonce, and then the provable security results in [19] still hold.
- We also note that `param` does not remove the dependency to other blockcipher modes of operation. For instance the concurrent use (with the same secret key) of CLOC and ECB mode results in the loss of security. Similarly, CLOC and SILC cannot be used concurrently.

See [22, Appendix B] for the details of the specific part of the document that was updated.

E.2 Changes from SILC v1 to SILC v2

The introduction of `param` is the change from SILC v1 to SILC v2, and the same notes as CLOC v2 above apply to SILC v2. See [24, Appendix H] for the details of the specific part of the document that was updated.

E.3 Changes from CLOC v2 and SILC v2 to CLOC and SILC v3

- CLOC v2 and SILC v2 are integrated into a single submission document as "CLOC and SILC v3" following the request by CAESAR. Overall texts were revised for the integration.
- Two recommended parameter sets, `aes128n8t8clocv2` and `aes128n8t8silcv2`, were removed to narrow down the size of the recommended parameter space.
- We specified the order of the five recommended parameter sets.
- $n = 96$ is listed as a possible choice of the block length of the underlying blockcipher.
- We added Simon and Speck as possible options in Appendix D.
- There is no algorithmic change in the specifications of CLOC and SILC.