SILC: SImple Lightweight CFB ⋆

Name: SILC v2

Designers/Submitters: Tetsu Iwata, Nagoya University, Japan
Kazuhiko Minematsu, NEC Corporation, Japan
Jian Guo, Nanyang Technological University, Singapore
Sumio Morioka, NEC Europe Ltd., United Kingdom
Eita Kobayashi, NEC Corporation, Japan

Contact Address: Tetsu Iwata, iwata@cse.nagoya-u.ac.jp

Date: August 29, 2015

⋆ This document was prepared based on [10,11,12] and [5,9].

1 Specification

SILC (which stands for SImple Lightweight CFB, and is pronounced as "silk") is a blockcipher mode of operation for authenticated encryption with associated data (AEAD), which is also called an authenticated cipher. SILC is built upon CLOC [10,11,12], 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.

1.1 Notation

We use the same notation as in [12], but we repeat the notation for completeness.

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 $(X[1], \ldots, X[x]) \xleftarrow{\ell} X$, which is defined as follows. If $X = \varepsilon$, then $x = 1$ and $X[1] \xleftarrow{\ell} X$, 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 SILC as byte-strings. Also we assume the big-endian format for all variables.

1.2 Algorithm and Parameters

We follow the description of CLOC [12].

SILC 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. SILC 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 $n \in \{64, 128\}$. We write $\mathrm{SILC}[E, \ell_N, \tau]$ for SILC 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. $\mathrm{SILC}[E, \ell_N, \tau] = (\text{SILC-}\mathcal{E}, \text{SILC-}\mathcal{D})$ consists of the encryption algorithm SILC-$\mathcal{E}$ and the decryption algorithm SILC-$\mathcal{D}$.

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

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

$\mathcal{K}_{\mathrm{SILC}} = \mathcal{K}_E$ is the key space, which is identical to the key space of the underlying blockcipher, $\mathcal{N}_{\mathrm{SILC}} = \mathbb{B}^{\ell_N/8}$ is the nonce space, $\mathcal{A}_{\mathrm{SILC}} = \mathbb{B}^*$ is the associated data space, $\mathcal{M}_{\mathrm{SILC}} = \mathbb{B}^*$ is the plaintext space, $\mathcal{CT}_{\mathrm{SILC}} = \mathcal{C}_{\mathrm{SILC}} \times \mathcal{T}_{\mathrm{SILC}}$ is the ciphertext space, where $\mathcal{C}_{\mathrm{SILC}} = \mathbb{B}^*$ and $\mathcal{T}_{\mathrm{SILC}} = \mathbb{B}^{\tau/8}$ is the tag space, and $\perp \notin \mathcal{M}_{\mathrm{SILC}}$ is the distinguished reject symbol. We write $(C, T) \leftarrow \text{SILC-}\mathcal{E}_K(N, A, M)$ and $M \leftarrow \text{SILC-}\mathcal{D}_K(N, A, C, T)$ or $\perp \leftarrow \text{SILC-}\mathcal{D}_K(N, A, C, T)$. We make a restriction that the maximum lengths of A, M, and C are all $2^{n/2} - 1$ bytes.

[⋆] In SILC v1, the requirement was $1 \leq \ell_N \leq n - 1$, and this was updated to handle `param` in SILC v2.

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

Algorithm $\mathsf{HASH}_K(N, A)$	**Algorithm** $\mathsf{PRF}_K(V, C)$
1. $S_{\mathsf{H}}[0] \leftarrow E_K(\mathsf{zpp}(\mathtt{param} \,\|\, N))$	1. $S_{\mathsf{P}}[0] \leftarrow E_K(\mathsf{g}(V))$
2. **if** $\|A\| = 0$ **then**	2. **if** $\|C\| = 0$ **then**
3. $\quad V \leftarrow \mathsf{g}(S_{\mathsf{H}}[0] \oplus \mathsf{Len}(A))$ // $\mathsf{Len}(A) = 0^n$	3. $\quad U \leftarrow \mathsf{g}(S_{\mathsf{P}}[0] \oplus \mathsf{Len}(C))$ // $\mathsf{Len}(C) = 0^n$
4. $\quad$ **return** V	4. $\quad T \leftarrow \mathsf{msb}_\tau(E_K(U))$
5. $(A[1], \ldots, A[a]) \xleftarrow{n} A$	5. $\quad$ **return** T
6. **for** $i \leftarrow 1$ **to** $a - 1$ **do**	6. $(C[1], \ldots, C[m]) \xleftarrow{n} C$
7. $\quad S_{\mathsf{H}}[i] \leftarrow E_K(S_{\mathsf{H}}[i-1] \oplus A[i])$	7. **for** $i \leftarrow 1$ **to** $m - 1$ **do**
8. $S_{\mathsf{H}}[a] \leftarrow E_K(S_{\mathsf{H}}[a-1] \oplus \mathsf{zap}(A[a]))$	8. $\quad S_{\mathsf{P}}[i] \leftarrow E_K(S_{\mathsf{P}}[i-1] \oplus C[i])$
9. $V \leftarrow \mathsf{g}(S_{\mathsf{H}}[a] \oplus \mathsf{Len}(A))$	9. $S_{\mathsf{P}}[m] \leftarrow E_K(S_{\mathsf{P}}[m-1] \oplus \mathsf{zap}(C[m]))$
10. **return** V	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 $\mathsf{ENC}_K(V, M)$	**Algorithm** $\mathsf{DEC}_K(V, C)$
1. **if** $\|M\| = 0$ **then**	1. **if** $\|C\| = 0$ **then**
2. $\quad C \leftarrow \varepsilon$	2. $\quad M \leftarrow \varepsilon$
3. $\quad$ **return** C	3. $\quad$ **return** M
4. $(M[1], \ldots, M[m]) \xleftarrow{n} M$	4. $(C[1], \ldots, C[m]) \xleftarrow{n} C$
5. $S_{\mathsf{E}}[1] \leftarrow E_K(V)$	5. $S_{\mathsf{D}}[1] \leftarrow E_K(V)$
6. **for** $i \leftarrow 1$ **to** $m - 1$ **do**	6. **for** $i \leftarrow 1$ **to** $m - 1$ **do**
7. $\quad C[i] \leftarrow S_{\mathsf{E}}[i] \oplus M[i]$	7. $\quad M[i] \leftarrow S_{\mathsf{D}}[i] \oplus C[i]$
8. $\quad S_{\mathsf{E}}[i+1] \leftarrow E_K(\mathsf{fix1}(C[i]))$	8. $\quad S_{\mathsf{D}}[i+1] \leftarrow E_K(\mathsf{fix1}(C[i]))$
9. $C[m] \leftarrow \mathsf{msb}_{\|M[m]\|}(S_{\mathsf{E}}[m]) \oplus M[m]$	9. $M[m] \leftarrow \mathsf{msb}_{\|C[m]\|}(S_{\mathsf{D}}[m]) \oplus C[m]$
10. $C \leftarrow (C[1], \ldots, C[m])$	10. $M \leftarrow (M[1], \ldots, M[m])$
11. **return** C	11. **return** M

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

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

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

These subroutines are defined in Fig. 2, and illustrated in Fig. 3, Fig. 4, and Fig. 5. We also present equivalent figures in Fig. 6, Fig. 7, and Fig. 8. We note that ENC and DEC are the same as those in CLOC [10]. In HASH, the nonce N is padded with $\mathtt{param} \in \mathbb{B}$ which is an 8-bit constant that depends on the parameters, E, ℓ_N, and τ. See Sect. 1.3 and Sect. 1.4 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

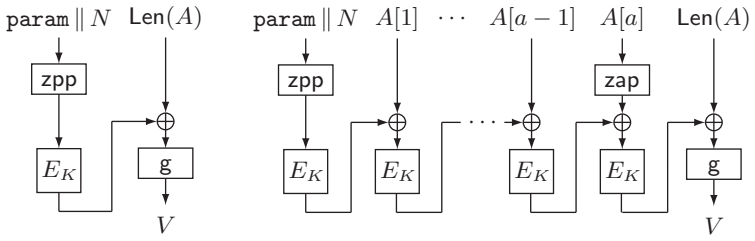

Fig. 3. $V \leftarrow \mathsf{HASH}_K(N, A)$ for $|A| = 0$ (left) and $|A| \geq 1$ (right)

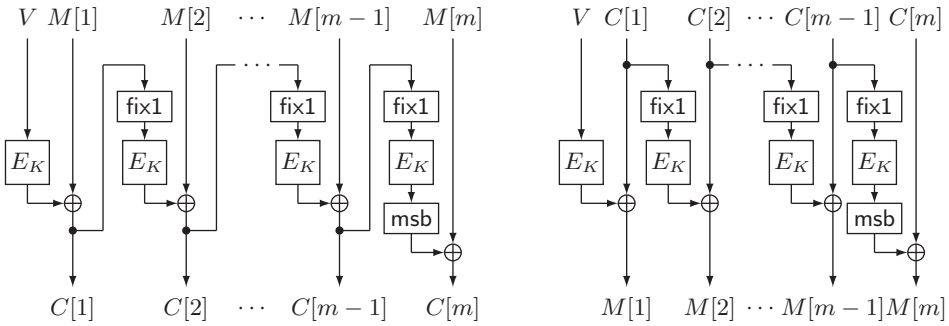

Fig. 4. $C \leftarrow \mathsf{ENC}_K(V, M)$ for $|M| \geq 1$ (left), and $\mathsf{DEC}_K(V, C)$ for $|C| \geq 1$ (right)

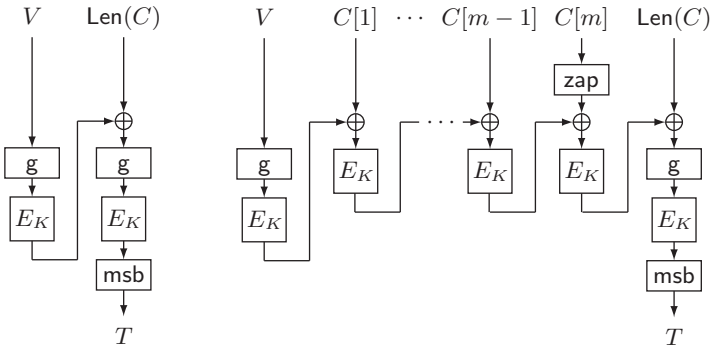

Fig. 5. $T \leftarrow \mathsf{PRF}_K(V, C)$ for $|C| = 0$ (left) and $|C| \geq 1$ (right)

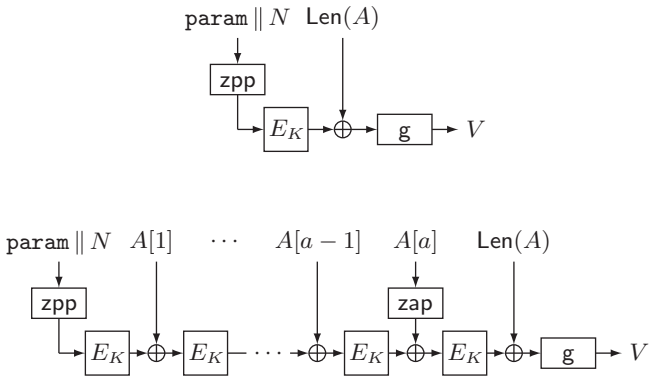

Fig. 6. $V \leftarrow \mathsf{HASH}_K(N, A)$ for $|A| = 0$ (top) and $|A| \geq 1$ (bottom)

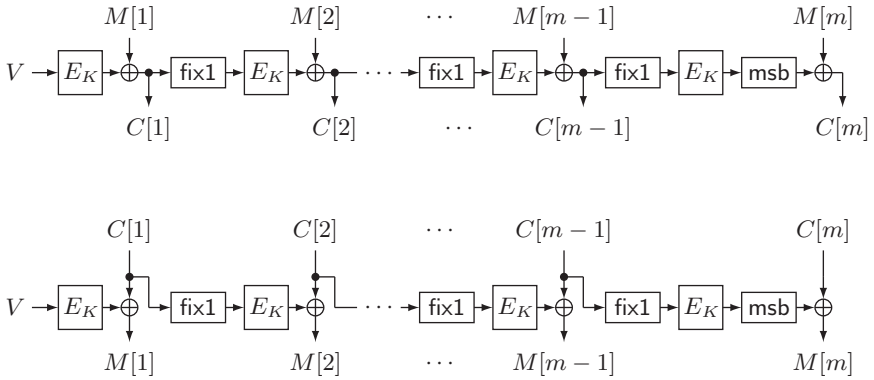

Fig. 7. $C \leftarrow \mathsf{ENC}_K(V, M)$ for $|M| \geq 1$ (top), and $\mathsf{DEC}_K(V, C)$ for $|C| \geq 1$ (bottom)

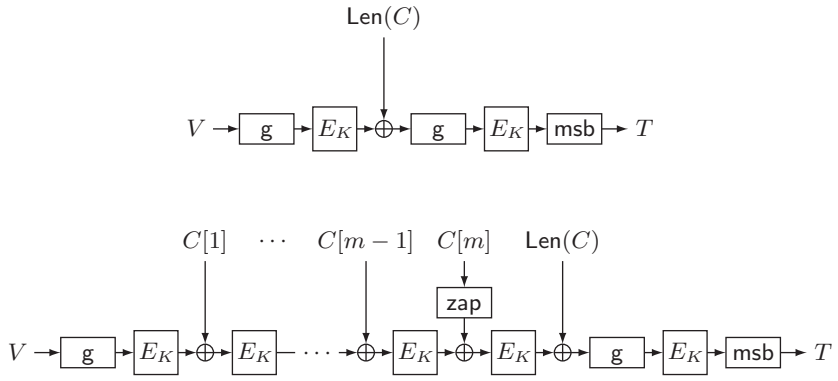

Fig. 8. $T \leftarrow \mathsf{PRF}_K(V, C)$ for $|C| = 0$ (top) and $|C| \geq 1$ (bottom)

only if the input is the empty string). For $X \in \mathbb{B}^*$, $\mathsf{zpp}(X)$ is defined as

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

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 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. If $n = 128$, for $X \in \{0,1\}^n$, we let $(X[1], X[2], \ldots, X[16]) \xleftarrow{n/16} X$. Then $\mathsf{g}(X)$ is defined as

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

where $X[a,b]$ stands for $X[a] \oplus X[b]$. Similarly, if $n = 64$, we let $(X[1], X[2], \ldots, X[8]) \xleftarrow{n/8} X$ and define $\mathsf{g}(X)$ as

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

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

1.3 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 [5], and LED is a 64-bit blockcipher proposed by Guo et al. at CHES 2011 [9]. The specification of PRESENT is described in Appendix A, and that of LED is described in Appendix B.

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.

1.4 Recommended Parameter Sets

We specify the recommended parameter sets as follows.

- Parameter set 1, `aes128n12t8silcv2`: E = AES-128, $\ell_N = 96$ (12-byte nonce), $\tau = 64$ (8-byte tag)
- Parameter set 2, `aes128n8t8silcv2`: E = AES-128, $\ell_N = 64$ (8-byte nonce), $\tau = 64$ (8-byte tag)
- Parameter set 3, `present80n6t4silcv2`: E = PRESENT-80, $\ell_N = 48$ (6-byte nonce), $\tau = 32$ (4-byte tag)
- Parameter set 4, `led80n6t4silcv2`: E = LED-80, $\ell_N = 48$ (6-byte nonce), $\tau = 32$ (4-byte tag)

These are marked with the asterisk in Table 1.

Table 1. Definition of `param`. 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`
*	PRESENT-80	6	4	`0xc4`
	PRESENT-80	6	6	`0xc5`
	PRESENT-80	6	8	`0xc6`
	PRESENT-80	4	4	`0xd4`
	PRESENT-80	4	6	`0xd5`
	PRESENT-80	4	8	`0xd6`
*	LED-80	6	4	`0xc8`
	LED-80	6	6	`0xc9`
	LED-80	6	8	`0xca`
	LED-80	4	4	`0xd8`
	LED-80	4	6	`0xd9`
	LED-80	4	8	`0xda`

Table 2. Security goal for confidentiality (privacy)

Parameter set	`aes128n12t8silcv2`	`aes128n8t8silcv2`	`present80n6t4silcv2`	`led80n6t4silcv2`
Data	64	64	32	32
Time	128	128	80	80

2 Security Goals

The security goal of 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. These goals are the same as CLOC. We note that, as CLOC, SILC has no secret message number.

SILC has 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). The attack models are given in Sect. 3, which are the same as in CLOC [12].

Attack Workload. SILC has provable security bounds up to the standard birthday bound, based on the pseudorandomness of the underlying blockcipher. Table 2 and Table 3 are obtained 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 2, Data denotes σ_{priv} of our privacy theorem in Theorem 1, and this roughly suggests the number of data blocks that the adversary obtains. In Table 3, Data denotes σ_{auth} and Verify denotes q' of our authenticity theorem in Theorem 2, where σ_{auth} roughly suggests the number of data blocks that the adversary obtains, and q' denotes the number of decryption queries. In both 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 a small constant is neglected in these tables.

As in CLOC, the nonce cannot be repeated to maintain the privacy. However, the privacy of SILC is kept as long as the uniqueness of (A, N), a pair of associated data and a nonce, is maintained for all encryption queries. 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, PRESENT or LED, is not for general purpose applications, since the birthday bound for the block length of 64 bits is usually unacceptable for conventional data transmission, as pointed out by McGrew [18]. However, there are various practical applications that benefit from the low implementation cost even with the limited security guarantee. See [12, Sect. 2] for such examples.

Table 3. Security goal for integrity (authenticity)

Parameter set	`aes128n12t8silcv2`	`aes128n8t8silcv2`	`present80n6t4silcv2`	`led80n6t4silcv2`
Data	64	64	32	32
Verify	64	64	32	32
Time	128	128	80	80

3 Security Analysis

In this section, we define the security notions of a blockcipher and SILC, which are the same as in [12, Sect. 3], 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 pseudorandom permutation, or a PRP [16]. We say that P is a random permutation if $P \xleftarrow{\$} \mathrm{Perm}(n)$, and define

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

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 SILC[Perm$(n), \ell_N, \tau$] for SILC that uses P as E_K, and the encryption and decryption algorithms are written as SILC-$\mathcal{E}_P$ and SILC-$\mathcal{D}_P$.

Privacy Notion. We define the privacy notion for SILC$[E, \ell_N, \tau]$ = (SILC-$\mathcal{E}$, SILC-$\mathcal{D}$). 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 SILC encryption oracle, or a random-bits oracle. The encryption oracle takes $(N, A, M) \in \mathcal{N}_{\mathrm{SILC}} \times \mathcal{A}_{\mathrm{SILC}} \times \mathcal{M}_{\mathrm{SILC}}$ as input and returns $(C, T) \leftarrow$ SILC-$\mathcal{E}_K(N, A, M)$. The random-bits oracle, \$-oracle, takes $(N, A, M) \in \mathcal{N}_{\mathrm{SILC}} \times \mathcal{A}_{\mathrm{SILC}} \times \mathcal{M}_{\mathrm{SILC}}$ as input and returns a random string $(C, T) \xleftarrow{\$} \{0,1\}^{|M|+\tau}$. We define the privacy advantage as

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

where the first probability is taken over $K \xleftarrow{\$} \mathcal{K}_{\mathrm{SILC}}$ 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* 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 Appendix C. If we use a blockcipher E, which is secure in the sense of the PRP notion, instead of Perm(n), then the corresponding complexity theoretic result can be shown by a standard argument. See e.g. [1]. We note that the privacy of SILC is broken if the nonce is reused, but as in CLOC, it remains secure if the uniqueness of (A, N) is maintained.

Authenticity Notion. We next define the authenticity notion, 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 SILC encryption oracle and the SILC decryption oracle. The encryption oracle is defined as above. The decryption oracle takes $(N, A, C, T) \in \mathcal{N}_{\mathrm{SILC}} \times \mathcal{A}_{\mathrm{SILC}} \times \mathcal{C}_{\mathrm{SILC}} \times \mathcal{T}_{\mathrm{SILC}}$ as input and returns $M \leftarrow$ SILC-$\mathcal{D}_K(N, A, C, T)$ or $\perp \leftarrow$ SILC-$\mathcal{D}_K(N, A, C, T)$. The authenticity advantage is defined as

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

where the probability is taken over $K \xleftarrow{\$} \mathcal{K}_{\mathrm{SILC}}$ 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 $(A_i[1], \ldots, A_i[a_i]) \xleftarrow{n} A_i$, $(M_i[1], \ldots, M_i[m_i]) \xleftarrow{n} M_i$, $(A'_i[1], \ldots, A'_i[a'_i]) \xleftarrow{n} A'_i$, and $(C'_i[1], \ldots, C'_i[m'_i]) \xleftarrow{n} C'_i$. 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 *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* $\mathbf{Adv}^{\mathrm{auth}}_{\mathrm{SILC}[\mathrm{Perm}(n),\ell_N,\tau]}(\mathcal{A}) \le 5\sigma^2_{\mathrm{auth}}/2^n + q'/2^\tau$*, where* $\sigma_{\mathrm{auth}} = 3q + \sigma_A + 2\sigma_M + 3q' + \sigma_{A'} + \sigma_{C'}$*.*

A complete proof is presented in Appendix C. 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., [1].

4 Features

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 implies that SILC 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.

Advantages over AES-GCM. Compared with AES-GCM [19], the hardware implementation of SILC with AES can be smaller, since we avoid using a full Galois-Field (GF) multiplier. 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, SILC inherits the advantages of CLOC over GCM. That is, the provable security bound of SILC for authentication is better than that of GCM presented in [15]. In GCM, the existence of weak keys was pointed out [21], while weak keys are not known in SILC. Also, SILC provides 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 PRESENT and `LED`. 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 `aes128n12t8silcv2`, 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 `aes128n8t8silcv2`, we select $\ell_N = 64$ considering the data overhead, and this is suitable for applications where the nonce consists of a counter. For `present80n6t4silcv2` and `led80n6t4silcv2`, we select $\ell_N = 48$ by taking the half of 96 in `aes128n12t8silcv2`. For all cases, the tag length was chosen by taking the balance between the security and the data overhead.

Limitations. We list several limitations of SILC. 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. Also, 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. SILC also inherits limitations of CLOC. For long input data, SILC is not efficient as it needs two blockcipher calls per one plaintext block. The nonce length is fixed, which may be problematic in some applications. The four functions used in SILC, HASH, ENC, DEC, and HASH, are all sequential, but the blockcipher calls in ENC and PRF can be done in parallel. We also note that the parallelization is always possible for multiple messages [7,6].

5 Design Rationale

The designers have not hidden any weaknesses in this cipher.

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. Basically, SILC is a 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. In order to keep implementation overhead as small as possible, we choose CFB, since CBC needs the decryption of the blockcipher, and CTR or OFB requires additional state for counter or intermediate output block. Since a naive combination of CFB and CBC MAC does not work, and we do not want to use precomputed blockcipher outputs such as $L = E_K(0^n)$ used in EAX, as this requires additional memory state, we use 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 (a multiplication by two over $\mathrm{GF}(2^n)$), a common operation for many blockcipher modes [3,23]. 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 [20,24]. There is only one tweak function in SILC, which is different from CLOC [10] 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 both $n = 64$ and $n = 128$ by computing the corresponding matrix ranks over $\mathrm{GF}(2)$ as was done in [10].

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 128-bit block size we choose AES as the underlying blockcipher, because the security of AES has been extensively studied. For 64-bit block size we choose PRESENT and LED as the underlying blockcipher. As explained in Sect. 4, both ciphers are 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.

6 Intellectual Property

We claim no intellectual property (IP) rights associated to SILC, and are unaware of any relevant IP held by others. We note that the statement does not cover the internal blockcipher. Nanyang Technological University has a patent related to LED blockcipher: 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.

7 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. 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)
2. Bellare, M., Rogaway, P.: The Security of Triple Encryption and a Framework for Code-Based Game-Playing Proofs. In: Vaudenay, S. (ed.) EUROCRYPT. Lecture Notes in Computer Science, vol. 4004, pp. 409–426. Springer (2006)
3. Bellare, M., Rogaway, P., Wagner, D.: The EAX Mode of Operation. In: Roy, B.K., Meier, W. (eds.) FSE. Lecture Notes in Computer Science, vol. 3017, pp. 389–407. Springer (2004)
4. Black, J., Rogaway, P.: CBC MACs for Arbitrary-Length Messages: The Three-Key Constructions. J. Cryptology 18(2), 111–131 (2005)
5. 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. Lecture Notes in Computer Science, vol. 4727, pp. 450–466. Springer (2007)
6. Bogdanov, A., Lauridsen, M., Tischhauser, E.: AES-Based Authenticated Encryption Modes in Parallel High-Performance Software. Cryptology ePrint Archive, Report 2014 (2014)
7. Bogdanov, A., Mendel, F., Regazzoni, F., Rijmen, V., Tischhauser, E.: ALE: AES-Based Lightweight Authenticated Encryption. Pre-proceedings of Fast Software Encryption 2013 (2013)
8. 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`
9. Guo, J., Peyrin, T., Poschmann, A., Robshaw, M.: The LED Block Cipher. In: Preneel, B., Takagi, T. (eds.) CHES. Lecture Notes in Computer Science, vol. 6917, pp. 326–341. Springer (2011), an updated version available `http://eprint.iacr.org/2012/600.pdf`
10. Iwata, T., Minematsu, K., Guo, J., Morioka, S.: CLOC: Authenticated Encryption for Short Input. Preproceedings of FSE 2014 (2014)
11. Iwata, T., Minematsu, K., Guo, J., Morioka, S.: CLOC: Authenticated Encryption for Short Input. Cryptology ePrint Archive, Report 2014 (2014), full version of FSE 2014 paper, `http://eprint.iacr.org/`
12. Iwata, T., Minematsu, K., Guo, J., Morioka, S.: CLOC: Compact Low-Overhead CFB. Submission to the CAESAR competition (2014), CLOC v1, March 15, 2014
13. Iwata, T., Minematsu, K., Guo, J., Morioka, S., Kobayashi, E.: CAESAR candidate SILC. Presentation at DIAC 2014 (2014), `http://2014.diac.cr.yp.to/schedule.html`
14. 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
15. Iwata, T., Ohashi, K., Minematsu, K.: Breaking and Repairing GCM Security Proofs. In: Safavi-Naini, R., Canetti, R. (eds.) CRYPTO. Lecture Notes in Computer Science, vol. 7417, pp. 31–49. Springer (2012)
16. Luby, M., Rackoff, C.: How to Construct Pseudorandom Permutations from Pseudorandom Functions. SIAM J. Comput. 17(2), 373–386 (1988)
17. 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`
18. McGrew, D.: Impossible Plaintext Cryptanalysis and Probable-Plaintext Collision Attacks of 64-bit Block Cipher Modes. Pre-proceedig of FSE 2013 (2013)
19. McGrew, D.A., Viega, J.: The Security and Performance of the Galois/Counter Mode (GCM) of Operation. In: Canteaut, A., Viswanathan, K. (eds.) INDOCRYPT. Lecture Notes in Computer Science, vol. 3348, pp. 343–355. Springer (2004)
20. Nandi, M.: Fast and Secure CBC-Type MAC Algorithms. In: Dunkelman, O. (ed.) FSE. Lecture Notes in Computer Science, vol. 5665, pp. 375–393. Springer (2009)
21. Procter, G., Cid, C.: On Weak Keys and Forgery Attacks against Polynomial-Based MAC Schemes. Preproceeding of FSE 2013 (2013)

22. Rogaway, P., Wagner, D.: A Critique of CCM. Cryptology ePrint Archive, Report 2003/070 (2003), http://eprint.iacr.org/
23. Rogaway, P.: Efficient Instantiations of Tweakable Blockciphers and Refinements to Modes OCB and PMAC. In: Lee, P.J. (ed.) ASIACRYPT. Lecture Notes in Computer Science, vol. 3329, pp. 16–31. Springer (2004)
24. Zhang, L., Wu, W., Zhang, L., Wang, P.: CBCR: CBC MAC with rotating transformations. SCIENCE CHINA Information Sciences 54(11), 2247–2255 (2011)

A PRESENT [5]

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 [5].

It consists of 31 rounds, and each of the 31 rounds consists of an xor operation of a round key K_i for $1 \leq i \leq 32$, 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 $K_i = \kappa_{63}^i \ldots \kappa_0^i$ for $1 \leq i \leq 32$ and current STATE $b_{63} \ldots b_0$, addRoundKey consists of the operation for $0 \leq j \leq 63$,

$$b_j \rightarrow b_j \oplus \kappa_j^i.$$

sBoxlayer. The S-box used in PRESENT is a 4-bit to 4-bit S-box $S : \{0,1\}^4 \rightarrow \{0,1\}^4$. 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 $b_{63} \ldots b_0$ is considered as sixteen 4-bit words $w_{15} \ldots w_0$ 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. $[\mathtt{k}_{19}\mathtt{k}_{18}\mathtt{k}_{17}\mathtt{k}_{16}\mathtt{k}_{15}] = [\mathtt{k}_{19}\mathtt{k}_{18}\mathtt{k}_{17}\mathtt{k}_{16}\mathtt{k}_{15}] \oplus \mathtt{round\_counter}$

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.

B LED [9]

LED [9] 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 $m_0\|m_1\|\dots\|m_{15}$, 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 A, to every nibble, i.e., $m_i = S(m_i)$ for $i = 0, \dots, 15$.

SHIFTROW shifts the i-th row to the left by i positions for $i = 0, \dots, 3$, 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

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

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 $r = 1, \dots, 48$ 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. 9.

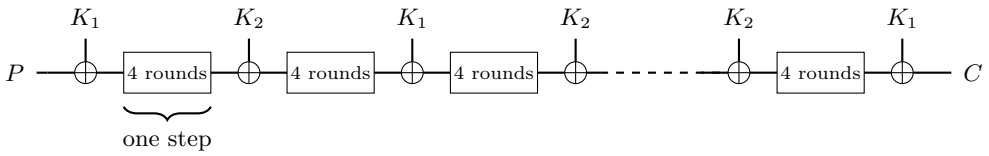

Fig. 9. Encryption of LED-80

C Security Proofs of SILC

We present a security proof of SILC. We note that the proof is similar to that of CLOC in [11]. We also note that the proof is presented using the specification of SILC v1 in [14]. This is because Theorem 1 and Theorem 2 hold both for SILC v1 and SILC v2, and the latter can be considered as the special case of SILC v1 by considering $\texttt{param} \,\|\, N$ as a nonce N in SILC v1.

PRP/PRF Switching. We first replace P in SILC[Perm$(n), \ell_N, \tau$] with a random function $R \xleftarrow{\$} \mathrm{Rand}(n)$. We then use the PRP/PRF switching lemma [2] to obtain

$$\mathbf{Adv}^{\mathrm{priv}}_{\mathrm{SILC}[\mathrm{Perm}(n),\ell_N,\tau]}(\mathcal{A}) \le \mathbf{Adv}^{\mathrm{priv}}_{\mathrm{SILC}[\mathrm{Rand}(n),\ell_N,\tau]}(\mathcal{A}) + \frac{0.5\sigma^2_{\mathrm{priv}}}{2^n}. \tag{1}$$

To see this, we observe that for a query (N_i, A_i, M_i), we need $\lceil |N_i|/n \rceil + \lceil |A_i|/n \rceil + 2\lceil |M_i|/n \rceil + 2 \le 3 + a_i + 2m_i$ calls of P in SILC-$\mathcal{E}_P$, and we have $\sum_{1 \le i \le q}(3 + a_i + 2m_i) \le 3q + \sigma_A + 2\sigma_M = \sigma_{\mathrm{priv}}$. We note that we have $\lceil |N_i|/n \rceil = 1$, $\lceil |A_i|/n \rceil \le a_i$, and $\lceil |M_i|/n \rceil \le m_i$. For the authenticity notion, without loss of generality, we assume that the decryption oracle, if $\mathcal{A}$ succeeds in forgery, returns a bit 1 instead of the plaintext since the returned value has no effect on the success probability of $\mathcal{A}$. Then for a decryption query (N'_j, A'_j, C'_j, T'_j), SILC-$\mathcal{D}_P$ makes $\lceil |N'_j|/n \rceil + \lceil |A'_j|/n \rceil + \lceil |C'_j|/n \rceil + 2 \le 3 + a'_j + m'_j$ calls of P, and from $\sum_{1 \le i \le q}(3 + a_i + 2m_i) + \sum_{1 \le j \le q'}(3 + a'_j + m'_j) \le 3q + \sigma_A + 2\sigma_M + 3q' + \sigma_{A'} + \sigma_{C'} = \sigma_{\mathrm{auth}}$, we obtain

$$\mathbf{Adv}^{\mathrm{auth}}_{\mathrm{SILC}[\mathrm{Perm}(n),\ell_N,\tau]}(\mathcal{A}) \le \mathbf{Adv}^{\mathrm{auth}}_{\mathrm{SILC}[\mathrm{Rand}(n),\ell_N,\tau]}(\mathcal{A}) + \frac{0.5\sigma^2_{\mathrm{auth}}}{2^n}. \tag{2}$$

In what follows, we evaluate $\mathbf{Adv}^{\mathrm{priv}}_{\mathrm{SILC}[\mathrm{Rand}(n),\ell_N,\tau]}(\mathcal{A})$ and $\mathbf{Adv}^{\mathrm{auth}}_{\mathrm{SILC}[\mathrm{Rand}(n),\ell_N,\tau]}(\mathcal{A})$.

Definition of $Q_1, \ldots, Q_7$. Let $R \xleftarrow{\$} \mathrm{Rand}(n)$ be a random function, and $K_1, K_2 \xleftarrow{\$} \{0,1\}^n$ be two independent and uniform random n-bit strings. We define seven functions $Q_1, \ldots, Q_7 : \{0,1\}^n \to \{0,1\}^n$ from R, K_1, and K_2 in Fig 10, and they are illustrated in Fig. 11. We write $Q = (Q_1, \ldots, Q_7)$.

$Q_1(X) = R(X) \oplus K_1$
$Q_2(X) = R(X \oplus K_1) \oplus K_1$
$Q_3(X) = R(\mathsf{g}(X \oplus K_1))$
$Q_4(X) = R(X)$
$Q_5(X) = R(\mathsf{g}(\mathsf{g}(X \oplus K_1))) \oplus K_2$
$Q_6(X) = R(X \oplus K_2) \oplus K_2$
$Q_7(X) = R(\mathsf{g}(X \oplus K_2))$

Fig. 10. Definition of $Q_1, \ldots, Q_7$

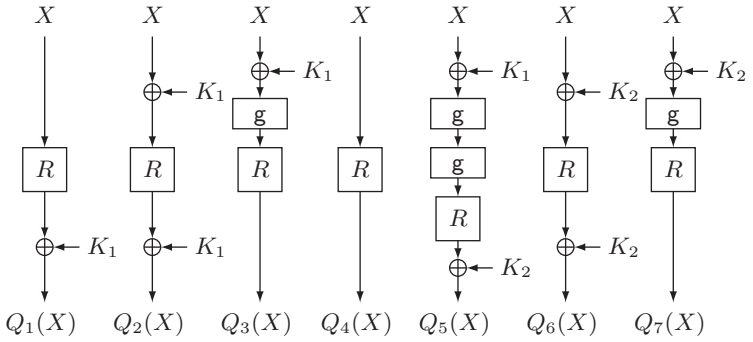

Fig. 11. $Q_1, \ldots, Q_7$

Algorithm SILC2-$\mathcal{E}_Q(N, A, M)$

1. **if** $|M| = 0$ **then**
2. $\quad C \leftarrow \varepsilon$
3. **else** // $|M| \geq 1$
4. $\quad S_{\mathsf{E}}[1] \leftarrow \mathsf{HASH2}_{Q_1,Q_2,Q_3}(N, A)$
5. $\quad C \leftarrow \mathsf{ENC2}_{Q_4}(S_{\mathsf{E}}[1], M)$
6. $T \leftarrow \mathsf{PRF2}_{Q_1,Q_2,Q_5,Q_6,Q_7}(N, A, C)$
7. **return** (C, T)

Algorithm SILC2-$\mathcal{D}_Q(N, A, C, T)$

1. $T^* \leftarrow \mathsf{PRF2}_{Q_1,Q_2,Q_5,Q_6,Q_7}(N, A, C)$
2. **if** $T \neq T^*$ **then return** $\perp$
3. **return** 1

Fig. 12. Pseudocode of the encryption and the decryption algorithms of SILC2

Algorithm $\mathsf{HASH2}_{Q_1,Q_2,Q_3}(N, A)$

1. $S_{\mathsf{H}}[0] \leftarrow Q_1(\mathsf{zpp}(N))$
2. **if** $|A| = 0$ **then**
3. $\quad S_{\mathsf{E}}[1] \leftarrow Q_3(S_{\mathsf{H}}[0] \oplus \mathsf{Len}(A))$ // $\mathsf{Len}(A) = 0^n$
4. $\quad$ **return** $S_{\mathsf{E}}[1]$
5. $(A[1], \ldots, A[a]) \xleftarrow{n} A$
6. **for** $i \leftarrow 1$ **to** $a - 1$ **do** // only for $a \geq 2$
7. $\quad S_{\mathsf{H}}[i] \leftarrow Q_2(S_{\mathsf{H}}[i-1] \oplus A[i])$
8. $S_{\mathsf{H}}[a] \leftarrow Q_2(S_{\mathsf{H}}[a-1] \oplus \mathsf{zap}(A[a]))$
9. $S_{\mathsf{E}}[1] \leftarrow Q_3(S_{\mathsf{H}}[a] \oplus \mathsf{Len}(A))$
10. **return** $S_{\mathsf{E}}[1]$

Algorithm $\mathsf{ENC2}_{Q_4}(S_{\mathsf{E}}[1], M)$ // $|M| \geq 1$

1. $(M[1], \ldots, M[m]) \xleftarrow{n} M$
2. **for** $i \leftarrow 1$ **to** $m - 1$ **do** // only for $m \geq 2$
3. $\quad C[i] \leftarrow S_{\mathsf{E}}[i] \oplus M[i]$
4. $\quad S_{\mathsf{E}}[i+1] \leftarrow Q_4(\mathsf{fix1}(C[i]))$
5. $C[m] \leftarrow \mathsf{msb}_{|M[m]|}(S_{\mathsf{E}}[m]) \oplus M[m]$
6. $C \leftarrow (C[1], \ldots, C[m])$
7. **return** C

Algorithm $\mathsf{PRF2}_{Q_1,Q_2,Q_5,Q_6,Q_7}(N, A, C)$

1. $S_{\mathsf{P}}[0] \leftarrow \mathsf{HASH2}'_{Q_1,Q_2,Q_5}(N, A)$
2. $T \leftarrow \mathsf{PRF2}'_{Q_6,Q_7}(S_{\mathsf{P}}[0], C)$
3. **return** T

Algorithm $\mathsf{HASH2}'_{Q_1,Q_2,Q_5}(N, A)$

1. $S_{\mathsf{H}}[0] \leftarrow Q_1(\mathsf{zpp}(N))$
2. **if** $|A| = 0$ **then**
3. $\quad S_{\mathsf{P}}[0] \leftarrow Q_5(S_{\mathsf{H}}[0] \oplus \mathsf{Len}(A))$ // $\mathsf{Len}(A) = 0^n$
4. $\quad$ **return** $S_{\mathsf{P}}[0]$
5. $(A[1], \ldots, A[a]) \xleftarrow{n} A$
6. **for** $i \leftarrow 1$ **to** $a - 1$ **do** // only for $a \geq 2$
7. $\quad S_{\mathsf{H}}[i] \leftarrow Q_2(S_{\mathsf{H}}[i-1] \oplus A[i])$
8. $S_{\mathsf{H}}[a] \leftarrow Q_2(S_{\mathsf{H}}[a-1] \oplus \mathsf{zap}(A[a]))$
9. $S_{\mathsf{P}}[0] \leftarrow Q_5(S_{\mathsf{H}}[a] \oplus \mathsf{Len}(A))$
10. **return** $S_{\mathsf{P}}[0]$

Algorithm $\mathsf{PRF2}'_{Q_6,Q_7}(S_{\mathsf{P}}[0], C)$

1. **if** $|C| = 0$ **then**
2. $\quad S_{\mathsf{P}}[1] \leftarrow Q_7(S_{\mathsf{P}}[0] \oplus \mathsf{Len}(C))$ // $\mathsf{Len}(C) = 0^n$
3. $\quad T \leftarrow \mathsf{msb}_\tau(S_{\mathsf{P}}[1])$
4. $\quad$ **return** T
5. $(C[1], \ldots, C[m]) \xleftarrow{n} C$
6. **for** $i \leftarrow 1$ **to** $m - 1$ **do** // only for $m \geq 2$
7. $\quad S_{\mathsf{P}}[i] \leftarrow Q_6(S_{\mathsf{P}}[i-1] \oplus C[i])$
8. $S_{\mathsf{P}}[m] \leftarrow Q_6(S_{\mathsf{P}}[m-1] \oplus \mathsf{zap}(C[m]))$
9. $S_{\mathsf{P}}[m+1] \leftarrow Q_7(S_{\mathsf{P}}[m] \oplus \mathsf{Len}(C))$
10. $T \leftarrow \mathsf{msb}_\tau(S_{\mathsf{P}}[m+1])$
11. **return** T

Fig. 13. Subroutines used in the encryption and decryption algorithms of SILC2

Definition of SILC2. We next present a definition of SILC2$[\ell_N, \tau]$, which is the same algorithm as SILC$[\mathrm{Rand}(n), \ell_N, \tau]$, but is represented in a different way. SILC2$[\ell_N, \tau]$ is based on Q, and the en-

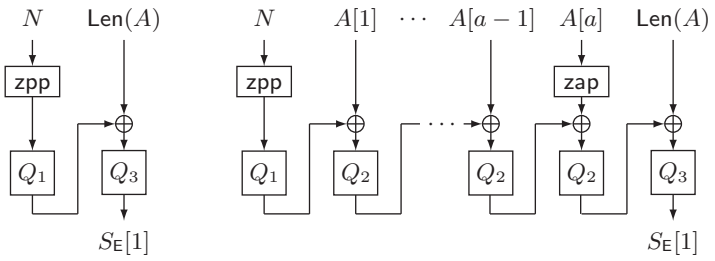

Fig. 14. $S_{\mathsf{E}}[1] \leftarrow \mathsf{HASH2}_{Q_1,Q_2,Q_3}(N, A)$ for $|A| = 0$ (left) and $|A| \geq 1$ (right). We note that $\mathsf{Len}(A) = 0^n$ holds in the left figure.

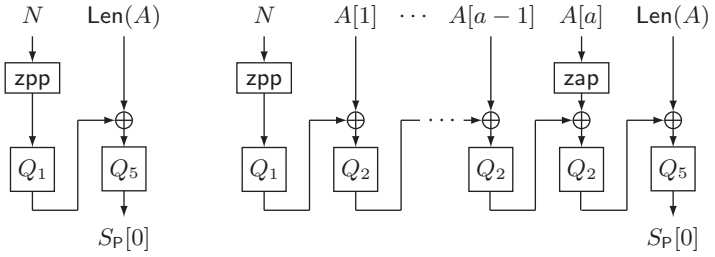

Fig. 15. $S_{\mathsf{P}}[0] \leftarrow \mathsf{HASH2}'_{Q_1,Q_2,Q_5}(N, A)$ for $|A| = 0$ (left) and $|A| \geq 1$ (right). This function is used as a subroutine in PRF2, together with PRF2′, to generate a tag T. $\mathsf{Len}(A) = 0^n$ holds in the left figure.

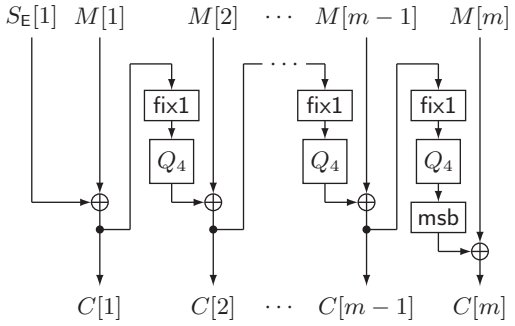

Fig. 16. $C \leftarrow \mathsf{ENC2}_{Q_4}(S_{\mathsf{E}}[1], M)$ for $|M| \geq 1$

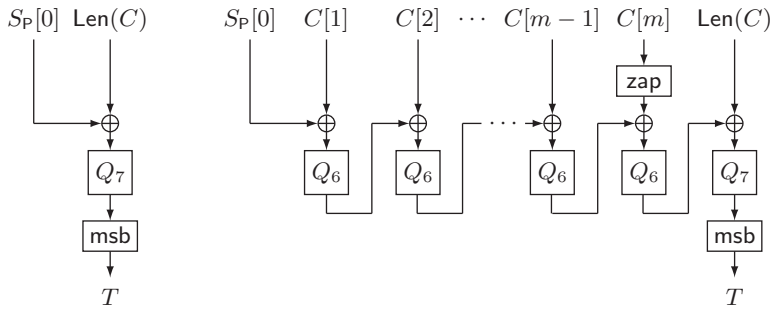

Fig. 17. $\mathsf{PRF2}'_{Q_6,Q_7}(S_{\mathsf{P}}[0], C)$ for $|C| = 0$ (left) and $|C| \geq 1$ (right). PRF2′ is used as a subroutine in PRF2, together with HASH2′, to generate a tag T. $\mathsf{Len}(C) = 0^n$ holds in the left figure.

cryption algorithm SILC2-$\mathcal{E}$ and the decryption algorithm SILC2-$\mathcal{D}$ are presented in Fig. 12, and the subroutines are presented in Fig. 13. We also show figures of subroutines $\mathsf{HASH2}$, $\mathsf{HASH2}'$, $\mathsf{ENC2}$, and $\mathsf{PRF2}$ used in these algorithms in Fig. 14, Fig. 15, Fig. 16, and Fig. 17. We write SILC2-$\mathcal{E}_Q$ and SILC2-$\mathcal{D}_Q$ as they are based on $Q = (Q_1, \ldots, Q_7)$, but we note that SILC2-$\mathcal{E}$ and SILC2-$\mathcal{D}$ take R, K_1, and K_2 as a key.

Let us describe how SILC2 works.

- $\mathsf{HASH2}$ takes N and A as input to output $R(V)$ of SILC (instead of V), which is subsequently used to encrypt the first plaintext block $M[1]$.
- Then $\mathsf{ENC2}$ takes the output from $\mathsf{HASH2}$ and M as input to output a ciphertext C.
- Finally we compute a tag T with $\mathsf{PRF2}$ from N, A, and C, which internally uses $\mathsf{HASH2}'$ and $\mathsf{PRF2}'$. $\mathsf{HASH2}'$ outputs $S_{\mathsf{P}}[0]$ from N and A by repeating the bulk of the computation which was already done in $\mathsf{HASH2}$. We note that $S_{\mathsf{P}}[0]$ in SILC2 corresponds to $S_{\mathsf{P}}[0] \oplus K_2$ in SILC, and the only difference between $\mathsf{HASH2}$ and $\mathsf{HASH2}'$ is the functions used to process the last input block. Finally, the output of $\mathsf{HASH2}'$, $S_{\mathsf{P}}[0]$, is used in $\mathsf{PRF2}'$, together with C, to compute the tag T.

From the construction, we claim that SILC-$\mathcal{E}_R$ and SILC2-$\mathcal{E}_Q$ are exactly the same algorithms, and furthermore, SILC-$\mathcal{D}_R$ and SILC2-$\mathcal{D}_Q$ are the same algorithms, since the random strings K_1 and K_2 are all canceled. We have

$$\begin{cases} \mathbf{Adv}^{\mathrm{priv}}_{\mathrm{SILC}[\mathrm{Rand}(n),\ell_N,\tau]}(\mathcal{A}) = \mathbf{Adv}^{\mathrm{priv}}_{\mathrm{SILC2}[\ell_N,\tau]}(\mathcal{A}), \\ \mathbf{Adv}^{\mathrm{auth}}_{\mathrm{SILC}[\mathrm{Rand}(n),\ell_N,\tau]}(\mathcal{A}) = \mathbf{Adv}^{\mathrm{auth}}_{\mathrm{SILC2}[\ell_N,\tau]}(\mathcal{A}). \end{cases} \tag{3}$$

Indistinguishability of Q. Next, let $F_1, \ldots, F_7 \xleftarrow{\$} \mathrm{Rand}(n)$ be seven independent random functions, and we write $F = (F_1, \ldots, F_7)$. We show that $Q = (Q_1, \ldots, Q_7)$ is indistinguishable from $F = (F_1, \ldots, F_7)$. Let $\mathcal{B}$ be an adversary. We define its advantage of distinguishing between Q and F as

$$\mathbf{Adv}^{\mathrm{ind}}_{Q}(\mathcal{B}) \stackrel{\mathrm{def}}{=} \Pr\left[\mathcal{B}^{Q_1(\cdot),\ldots,Q_7(\cdot)} \Rightarrow 1\right] - \Pr\left[\mathcal{B}^{F_1(\cdot),\ldots,F_7(\cdot)} \Rightarrow 1\right],$$

where the first probability is taken over $R \xleftarrow{\$} \mathrm{Rand}(n)$, $K_1, K_2 \xleftarrow{\$} \{0,1\}^n$, and the randomness of $\mathcal{B}$, and the last one is taken over $F_1, \ldots, F_7 \xleftarrow{\$} \mathrm{Rand}(n)$ and the randomness of $\mathcal{B}$. The adversary makes queries of the form $(j, X) \in \{1, \ldots, 7\} \times \{0,1\}^n$, and receives $Q_j(X)$ or $F_j(X)$. We say that the adversary is input-respecting if $\mathsf{msb}_1(X) = 0$ holds for all queries with $j = 1$, and $\mathsf{msb}_1(X) = 1$ holds for all queries with $j = 4$. Without loss of generality, we assume that $\mathcal{B}$ does not repeat a query. We have the following lemma.

Lemma 1. *Let $\mathcal{B}$ be an input-respecting adversary that makes at most q queries. Then* $\mathbf{Adv}^{\mathrm{ind}}_{Q}(\mathcal{B}) \leq 0.5q^2/2^n$.

A proof is in Appendix D.

Definition of SILC3. Next, we define the third version SILC3$[\ell_N, \tau]$ of SILC$[\mathrm{Rand}(n), \ell_N, \tau]$. It uses $F = (F_1, \ldots, F_7)$, and the encryption algorithm SILC3-$\mathcal{E}$ and the decryption algorithm SILC3-$\mathcal{D}$ are obtained from SILC2-$\mathcal{E}$ and SILC2-$\mathcal{D}$ by using $F_1, \ldots, F_7$ instead of $Q_1, \ldots, Q_7$. Therefore, SILC3-$\mathcal{E}$ and SILC3-$\mathcal{D}$ take $F = (F_1, \ldots, F_7)$ as a key, and we write SILC3-$\mathcal{E}_F$ and SILC3-$\mathcal{D}_F$. We write the subroutines in SILC3-$\mathcal{E}_F$ and SILC3-$\mathcal{D}_F$ as $\mathsf{HASH3}$, $\mathsf{HASH3}'$, $\mathsf{ENC3}$, $\mathsf{PRF3}$, and $\mathsf{PRF3}'$, instead of $\mathsf{HASH2}$, $\mathsf{HASH2}'$, $\mathsf{ENC2}$, $\mathsf{PRF2}$, and $\mathsf{PRF2}'$. Lemma 1 gives

$$\begin{cases} \mathbf{Adv}^{\mathrm{priv}}_{\mathrm{SILC2}[\ell_N,\tau]}(\mathcal{A}) \leq \mathbf{Adv}^{\mathrm{priv}}_{\mathrm{SILC3}[\ell_N,\tau]}(\mathcal{A}) + 0.5\sigma^2_{\mathrm{priv}}/2^n, \\ \mathbf{Adv}^{\mathrm{auth}}_{\mathrm{SILC2}[\ell_N,\tau]}(\mathcal{A}) \leq \mathbf{Adv}^{\mathrm{auth}}_{\mathrm{SILC3}[\ell_N,\tau]}(\mathcal{A}) + 0.5\sigma^2_{\mathrm{auth}}/2^n. \end{cases} \tag{4}$$

Otherwise, we can construct an input-respecting adversary $\mathcal{B}$ that contradicts Lemma 1.

Algorithm SILC4-$\mathcal{E}_{\mathsf{HASH4},\mathsf{HASH4}',F_4,F_6,F_7}(N,A,M)$

1. **if** $|M|=0$ **then**
2. $\quad C \leftarrow \varepsilon$
3. **else** // $|M| \geq 1$
4. $\quad S_{\mathsf{E}}[1] \leftarrow \mathsf{HASH4}(N,A)$
5. $\quad C \leftarrow \mathsf{ENC4}_{F_4}(S_{\mathsf{E}}[1],M)$
6. $T \leftarrow \mathsf{PRF4}_{\mathsf{HASH4}',F_6,F_7}(N,A,C)$
7. **return** (C,T)

Algorithm SILC4-$\mathcal{D}_{\mathsf{HASH4},\mathsf{HASH4}',F_4,F_6,F_7}(N,A,C,T)$

1. $T^* \leftarrow \mathsf{PRF4}_{\mathsf{HASH4}',F_6,F_7}(N,A,C)$
2. **if** $T \neq T^*$ **then return** $\perp$
3. **return** 1

Fig. 18. Pseudocode of the encryption and the decryption algorithms of SILC4

Algorithm $\mathsf{ENC4}_{F_4}(S_{\mathsf{E}}[1],M)$ // $|M| \geq 1$

1. $(M[1],\ldots,M[m]) \xleftarrow{n} M$
2. **for** $i \leftarrow 1$ **to** $m-1$ **do** // only for $m \geq 2$
3. $\quad C[i] \leftarrow S_{\mathsf{E}}[i] \oplus M[i]$
4. $\quad S_{\mathsf{E}}[i+1] \leftarrow F_4(\mathsf{fix1}(C[i]))$
5. $C[m] \leftarrow \mathsf{msb}_{|M[m]|}(S_{\mathsf{E}}[m]) \oplus M[m]$
6. $C \leftarrow (C[1],\ldots,C[m])$
7. **return** C

Algorithm $\mathsf{PRF4}_{\mathsf{HASH4}',F_6,F_7}(N,A,C)$

1. $S_{\mathsf{P}}[0] \leftarrow \mathsf{HASH4}'(N,A)$
2. $T \leftarrow \mathsf{PRF4}'_{F_6,F_7}(S_{\mathsf{P}}[0],C)$
3. **return** T

Algorithm $\mathsf{PRF4}'_{F_6,F_7}(S_{\mathsf{P}}[0],C)$

1. **if** $|C|=0$ **then**
2. $\quad S_{\mathsf{P}}[1] \leftarrow F_7(S_{\mathsf{P}}[0] \oplus \mathsf{Len}(C))$ // $\mathsf{Len}(C)=0^n$
3. $\quad T \leftarrow \mathsf{msb}_\tau(S_{\mathsf{P}}[1])$
4. $\quad$ **return** T
5. $(C[1],\ldots,C[m]) \xleftarrow{n} C$
6. **for** $i \leftarrow 1$ **to** $m-1$ **do** // only for $m \geq 2$
7. $\quad S_{\mathsf{P}}[i] \leftarrow F_6(S_{\mathsf{P}}[i-1] \oplus C[i])$
8. $S_{\mathsf{P}}[m] \leftarrow F_6(S_{\mathsf{P}}[m-1] \oplus \mathsf{zap}(C[m]))$
9. $S_{\mathsf{P}}[m+1] \leftarrow F_7(S_{\mathsf{P}}[m] \oplus \mathsf{Len}(C))$
10. $T \leftarrow \mathsf{msb}_\tau(S_{\mathsf{P}}[m+1])$
11. **return** T

Fig. 19. Subroutines used in the encryption and decryption algorithms of SILC4

Indistinguishability of $(\mathsf{HASH3},\mathsf{HASH3}')$. Let $\mathsf{HASH4}$ and $\mathsf{HASH4}'$ be two independent random functions, where $\mathsf{HASH4},\mathsf{HASH4}' : \mathcal{N}_{\mathrm{SILC}} \times \mathcal{A}_{\mathrm{SILC}} \to \{0,1\}^n$. Our next task is to show that $(\mathsf{HASH3},\mathsf{HASH3}')$ is indistinguishable from $(\mathsf{HASH4},\mathsf{HASH4}')$. For an adversary $\mathcal{B}$, we define $\mathbf{Adv}^{\mathrm{ind}}_{\mathsf{HASH3},\mathsf{HASH3}'}(\mathcal{B})$ as

$$\mathbf{Adv}^{\mathrm{ind}}_{\mathsf{HASH3},\mathsf{HASH3}'}(\mathcal{B}) \stackrel{\mathrm{def}}{=} \Pr\left[\mathcal{B}^{\mathsf{HASH3}(\cdot,\cdot),\mathsf{HASH3}'(\cdot,\cdot)} \Rightarrow 1\right] - \Pr\left[\mathcal{B}^{\mathsf{HASH4}(\cdot,\cdot),\mathsf{HASH4}'(\cdot,\cdot)} \Rightarrow 1\right],$$

where the first probability is taken over F_1, F_2, F_3, F_5, and the randomness of $\mathcal{B}$, and the last is over the randomness of $\mathsf{HASH4}$, $\mathsf{HASH4}'$, and $\mathcal{B}$. The adversary makes queries of the form $(j,N,A) \in \{1,2\} \times \mathcal{N}_{\mathrm{SILC}} \times \mathcal{A}_{\mathrm{SILC}}$, and receives $\mathsf{HASH3}(N,A)$ or $\mathsf{HASH4}(N,A)$ if $j=1$, and $\mathsf{HASH3}'(N,A)$ or $\mathsf{HASH4}'(N,A)$ if $j=2$. If $\mathcal{B}$ makes q queries and the queries are $(j_1,N_1,A_1),\ldots,(j_q,N_q,A_q)$, then we define the total associated data length as $a_1+\cdots+a_q$, where $(A_i[1],\ldots,A_i[a_i]) \xleftarrow{n} A_i$. Without loss of generality, we assume that $\mathcal{B}$ does not repeat a query, but the same nonce can be repeated across different queries. We show the following lemma.

Lemma 2. *Let $\mathcal{B}$ be an adversary that makes at most q queries, where the total associated data length is at most σ_A. Then we have* $\mathbf{Adv}^{\mathrm{ind}}_{\mathsf{HASH3},\mathsf{HASH3}'}(\mathcal{B}) \leq 0.5q^2/2^n + (q+\sigma_A)^2/2^n$.

A proof is in Appendix E.

Definition of SILC4. Next, we define the fourth version $\mathrm{SILC4}[\ell_N,\tau]$ of $\mathrm{SILC}[\mathrm{Rand}(n),\ell_N,\tau]$. The encryption algorithm SILC4-$\mathcal{E}$ and the decryption algorithm SILC4-$\mathcal{D}$ are obtained from SILC3-$\mathcal{E}$ and SILC3-$\mathcal{D}$ by replacing $\mathsf{HASH3}$ and $\mathsf{HASH3}'$ with the random functions $\mathsf{HASH4}$ and $\mathsf{HASH4}'$, respectively. Therefore, SILC4-$\mathcal{E}$ and SILC4-$\mathcal{D}$ take $\mathsf{HASH4}$, $\mathsf{HASH4}'$, F_4, F_6, and F_7 as a key, and we write the subroutines as $\mathsf{ENC4}$, $\mathsf{PRF4}$, and $\mathsf{PRF4}'$, instead of $\mathsf{ENC3}$, $\mathsf{PRF3}$, and $\mathsf{PRF3}'$. For reference, we present the specification of SILC4 in Fig. 18 and the subroutines in Fig. 19.

From Lemma 2, we obtain

$$\begin{cases} \mathbf{Adv}^{\mathrm{priv}}_{\mathrm{SILC3}[\ell_N,\tau]}(\mathcal{A}) \leq \mathbf{Adv}^{\mathrm{priv}}_{\mathrm{SILC4}[\ell_N,\tau]}(\mathcal{A}) + 2q^2/2^n + 4(q+\sigma_A)^2/2^n, \\ \mathbf{Adv}^{\mathrm{auth}}_{\mathrm{SILC3}[\ell_N,\tau]}(\mathcal{A}) \leq \mathbf{Adv}^{\mathrm{auth}}_{\mathrm{SILC4}[\ell_N,\tau]}(\mathcal{A}) + 2(q+q')^2 + 4(q+\sigma_A+q'+\sigma_{A'})^2/2^n. \end{cases} \tag{5}$$

Algorithm SILC5-$\mathcal{E}_{\mathsf{HASH5},\mathsf{PRF5},F_4}(N,A,M)$	**Algorithm** $\mathsf{ENC5}_{F_4}(S_{\mathsf{E}}[1],M)$ // $\lvert M\rvert \geq 1$
1. **if** $\lvert M\rvert = 0$ **then** 2. $\quad C \leftarrow \varepsilon$ 3. **else** // $\lvert M\rvert \geq 1$ 4. $\quad S_{\mathsf{E}}[1] \leftarrow \mathsf{HASH5}(N,A)$ 5. $\quad C \leftarrow \mathsf{ENC5}_{F_4}(S_{\mathsf{E}}[1],M)$ 6. $T \leftarrow \mathsf{PRF5}(N,A,C)$ 7. **return** (C,T)	1. $(M[1],\ldots,M[m]) \xleftarrow{n} M$ 2. **for** $i \leftarrow 1$ **to** $m-1$ **do** // only for $m \geq 2$ 3. $\quad C[i] \leftarrow S_{\mathsf{E}}[i] \oplus M[i]$ 4. $\quad S_{\mathsf{E}}[i+1] \leftarrow F_4(\mathsf{fix1}(C[i]))$ 5. $C[m] \leftarrow \mathsf{msb}_{\lvert M[m]\rvert}(S_{\mathsf{E}}[m]) \oplus M[m]$ 6. $C \leftarrow (C[1],\ldots,C[m])$ 7. **return** C
Algorithm SILC5-$\mathcal{D}_{\mathsf{HASH5},\mathsf{PRF5},F_4}(N,A,C,T)$	
1. $T^* \leftarrow \mathsf{PRF5}(N,A,C)$ 2. **if** $T \neq T^*$ **then return** $\perp$ 3. **return** 1	

Fig. 20. Pseudocode of the encryption and the decryption algorithms of SILC5

To see (5), for privacy, suppose that $\mathcal{A}$ makes a query (N_i,A_i,M_i). If $|M_i|=0$, then $\mathcal{B}$ makes a query $(2,N_i,A_i)$ to obtain $S_{\mathsf{P}i}[0]$. Otherwise, $\mathcal{B}$ first makes a query $(1,N_i,A_i)$ to obtain $S_{\mathsf{E}i}[1]$, and then $(2,N_i,A_i)$ to obtain $S_{\mathsf{P}i}[0]$. For authenticity, $\mathcal{B}$ behaves as above for encryption queries. For a decryption query (N'_j,A'_j,C'_j,T'_j), $\mathcal{B}$ makes a query $(2,N'_j,A'_j)$ to obtain $S_{\mathsf{P}j}[0]$. We see that the total number of queries and the total number of associated data length of $\mathcal{B}$ are no more than the twice of those of $\mathcal{A}$.

Indistinguishability of PRF4. The syntax of PRF4 is $\mathsf{PRF4}:\mathcal{N}_{\mathrm{SILC}}\times\mathcal{A}_{\mathrm{SILC}}\times\mathcal{C}_{\mathrm{SILC}}\to\mathcal{T}_{\mathrm{SILC}}$, and it takes HASH4′, F_6, and F_7 as a key. Let PRF5 be a random function with the same syntax as PRF4. We show that PRF4 is indistinguishable from PRF5. Let $\mathcal{B}$ be an adversary, and we define $\mathbf{Adv}^{\mathrm{ind}}_{\mathsf{PRF4}}(\mathcal{B})$ as

$$\mathbf{Adv}^{\mathrm{ind}}_{\mathsf{PRF4}}(\mathcal{B}) \stackrel{\mathrm{def}}{=} \Pr\left[\mathcal{B}^{\mathsf{PRF4}(\cdot,\cdot,\cdot)}\Rightarrow 1\right] - \Pr\left[\mathcal{B}^{\mathsf{PRF5}(\cdot,\cdot,\cdot)}\Rightarrow 1\right],$$

where the first probability is taken over the randomness of HASH4′, F_6, F_7, and $\mathcal{B}$, and the last is over the randomness of PRF5 and $\mathcal{B}$. Suppose that $\mathcal{B}$ makes q queries, and let $(N_1,A_1,C_1),\ldots,(N_q,A_q,C_q)$ be the queries. Then we define the total ciphertext length as $m_1+\cdots+m_q$, where $(C_i[1],\ldots,C_i[m_i]) \leftarrow C_i$. The same nonce can be repeated across different queries, but without loss of generality, we assume that $\mathcal{B}$ does not repeat a query. We show the following lemma.

Lemma 3. *Let $\mathcal{B}$ be an adversary that makes at most q queries, where the total ciphertext length is at most σ_C. Then we have $\mathbf{Adv}^{\mathrm{ind}}_{\mathsf{PRF4}}(\mathcal{B}) \leq 0.5q^2/2^n + (q+\sigma_C)^2/2^n$.*

A proof is in Appendix F.

Definition of SILC5. We next define the fifth and the final version SILC5$[\ell_N,\tau]$ of SILC[Rand$(n),\ell_N,\tau$]. We note that SILC5$[\ell_N,\tau]$ is almost the same as CLOC5$[\ell_N,\tau]$ in [10,11]. The encryption algorithm SILC5-$\mathcal{E}$ and the decryption algorithm SILC5-$\mathcal{D}$ are obtained from SILC4-$\mathcal{E}$ and SILC4-$\mathcal{D}$, respectively, by using a random function PRF5 instead of PRF4. We write HASH5 for HASH4, and ENC5 for ENC4. SILC5-$\mathcal{E}$ and SILC5-$\mathcal{D}$ take HASH5, PRF5, and F_4 as a key, where $\mathsf{HASH5}:\mathcal{N}_{\mathrm{SILC}}\times\mathcal{A}_{\mathrm{SILC}}\to\{0,1\}^n$, $\mathsf{PRF5}:\mathcal{N}_{\mathrm{SILC}}\times\mathcal{A}_{\mathrm{SILC}}\times\mathcal{C}_{\mathrm{SILC}}\to\mathcal{T}_{\mathrm{SILC}}$, and $F_4:\{0,1\}^n\to\{0,1\}^n$ are all random functions. We present the complete specification in Fig. 20. From Lemma 3, we obtain

$$\begin{cases}\mathbf{Adv}^{\mathrm{priv}}_{\mathrm{SILC4}[\ell_N,\tau]}(\mathcal{A}) \leq \mathbf{Adv}^{\mathrm{priv}}_{\mathrm{SILC5}[\ell_N,\tau]}(\mathcal{A}) + 0.5q^2/2^n + (q+\sigma_M)^2/2^n,\\ \mathbf{Adv}^{\mathrm{auth}}_{\mathrm{SILC4}[\ell_N,\tau]}(\mathcal{A}) \leq \mathbf{Adv}^{\mathrm{auth}}_{\mathrm{SILC5}[\ell_N,\tau]}(\mathcal{A}) + 0.5(q+q')^2/2^n + (q+\sigma_M+q'+\sigma_{C'})^2/2^n.\end{cases} \tag{6}$$

We have (6), since for privacy, if $\mathcal{A}$ makes a query (N_i,A_i,M_i), then $\mathcal{B}$ obtains a ciphertext C_i using its own randomness, and makes a query (N_i,A_i,C_i) to obtain a tag T_i. For authenticity, $\mathcal{B}$ behaves as above for encryption queries. For a decryption query (N'_j,A'_j,C'_j,T'_j), $\mathcal{B}$ makes a query (N'_j,A'_j,C'_j) to obtain a candidate tag.

Privacy and Authenticity of SILC5. We have the following lemma on the privacy and the authenticity of SILC5.

Lemma 4. *We have* $\mathbf{Adv}^{\mathrm{priv}}_{\mathrm{SILC5}[\ell_N,\tau]}(\mathcal{A}) \le \sigma_M^2/2^n$ *and* $\mathbf{Adv}^{\mathrm{auth}}_{\mathrm{SILC5}[\ell_N,\tau]}(\mathcal{A}) \le q'/2^\tau$.

A proof is the same as that of [11, Lemma 4]. We present the proof in Appendix G for completeness.

Proof (of Theorem 1). We now show the proof of Theorem 1. From (1), (3), (4), (5), (6), and Lemma 4, we obtain

$$\mathbf{Adv}^{\mathrm{priv}}_{\mathrm{SILC}[\mathrm{Perm}(n),\ell_N,\tau]}(\mathcal{A}) \le \frac{\sigma_{\mathrm{priv}}^2}{2^n} + \frac{2.5q^2}{2^n} + \frac{4(q+\sigma_A)^2}{2^n} + \frac{(q+\sigma_M)^2}{2^n} + \frac{\sigma_M^2}{2^n} \le \frac{5\sigma_{\mathrm{priv}}^2}{2^n}.$$

The last inequality follows from $\sigma_{\mathrm{priv}} = 3q + \sigma_A + 2\sigma_M$. □

Proof (of Theorem 2). We show the proof of Theorem 2. From (2), (3), (4), (5), (6), and Lemma 4, we obtain

$$\begin{aligned}\mathbf{Adv}^{\mathrm{auth}}_{\mathrm{SILC}[\mathrm{Perm}(n),\ell_N,\tau]}(\mathcal{A}) &\le \frac{\sigma_{\mathrm{auth}}^2}{2^n} + \frac{2.5(q+q')^2}{2^n} + \frac{4(q+\sigma_A+q'+\sigma_{A'})^2}{2^n} + \frac{(q+\sigma_M+q'+\sigma_{C'})^2}{2^n} + \frac{q'}{2^\tau} \\ &\le \frac{5\sigma_{\mathrm{auth}}^2}{2^n} + \frac{q'}{2^\tau},\end{aligned}$$

since $\sigma_{\mathrm{auth}} = 3q + \sigma_A + 2\sigma_M + 3q' + \sigma_{A'} + \sigma_{C'}$. □

D Proof of Lemma 1

Without loss of generality, we assume that $\mathcal{B}$ makes exactly q queries. Let $(j_1, X_1), \ldots, (j_q, X_q)$ be the queries. Now suppose that $\mathcal{B}$ interacts with $Q = (Q_1, \ldots, Q_7)$. We say that a bad event occurs and write $\mathcal{B}^{Q_1(\cdot),\ldots,Q_7(\cdot)}$ sets bad, if there exist two distinct queries (j, X), $(j', X') \in \{(j_1, X_1), \ldots, (j_q, X_q)\}$ such that $I(j, X) = I(j', X')$, where $I(j, X)$ denotes the input value of R for a query $Q_j(X)$. That is, $I(1, X), \ldots, I(7, X)$ are defined as follows.

$$\begin{cases} I(1,X) = X \\ I(2,X) = X \oplus K_1 \\ I(3,X) = \mathsf{g}(X \oplus K_1) \\ I(4,X) = X \\ I(5,X) = \mathsf{g}(\mathsf{g}(X \oplus K_1)) \\ I(6,X) = X \oplus K_2 \\ I(7,X) = \mathsf{g}(X \oplus K_2) \end{cases}$$

The absence of the bad event implies that the responses that $\mathcal{B}$ receives from the oracles are uniform and independent random bit strings, since the output values of R are all independent. We have

$$\mathbf{Adv}^{\mathrm{ind}}_{Q}(\mathcal{B}) \le \Pr\left[\mathcal{B}^{Q_1(\cdot),\ldots,Q_7(\cdot)} \text{ sets } \mathsf{bad}\right]. \tag{7}$$

Furthermore, from the argument above, we may without loss of generality assume that the adversary is non-adaptive, and hence we now fix all queries $(j_1, X_1), \ldots, (j_q, X_q)$ made by $\mathcal{B}$, and evaluate the right hand side of (7) based on the randomness of K_1 and K_2. Let $(j, X), (j', X') \in \{(j_1, X_1), \ldots, (j_q, X_q)\}$ be two distinct queries. If $j = j'$, then we have $X \neq X'$, and $I(j, X) = I(j', X')$ never holds from the invertibility of g. We next consider the case $1 \le j < j' \le 7$, and we evaluate

$$\Pr[I(j, X) = I(j', X')] \tag{8}$$

in 21 cases in Fig. 21. We note that when $(j, j') = (1, 4)$, we have $(8) = \Pr[0^n = X \oplus X'] = 0$ since our adversary is input-respecting and hence $\mathsf{msb}_1(X) = 0$ and $\mathsf{msb}_1(X') = 1$ hold. For other cases, we have $(8) = 1/2^n$ from the construction of g.

(j, j')	(8)
$(1,2)$	$\Pr[K_1 = X \oplus X'] = 1/2^n$
$(1,3)$	$\Pr[K_1 = \mathsf{g}^{-1}(X) \oplus X'] = 1/2^n$
$(1,4)$	$\Pr[0^n = X \oplus X'] = 0$
$(1,5)$	$\Pr[K_1 = \mathsf{g}^{-1}(\mathsf{g}^{-1}(X)) \oplus X'] = 1/2^n$
$(1,6)$	$\Pr[K_2 = X \oplus X'] = 1/2^n$
$(1,7)$	$\Pr[K_2 = \mathsf{g}^{-1}(X) \oplus X'] = 1/2^n$
$(2,3)$	$\Pr[K_1 \oplus \mathsf{g}(K_1) = X \oplus \mathsf{g}(X')] = 1/2^n$
$(2,4)$	$\Pr[K_1 = X \oplus X'] = 1/2^n$
$(2,5)$	$\Pr[K_1 \oplus \mathsf{g}(\mathsf{g}(K_1)) = X \oplus \mathsf{g}(\mathsf{g}(X'))] = 1/2^n$
$(2,6)$	$\Pr[K_1 = X \oplus X' \oplus K_2] = 1/2^n$
$(2,7)$	$\Pr[K_1 = X \oplus \mathsf{g}(X' \oplus K_2)] = 1/2^n$
$(3,4)$	$\Pr[K_1 = X \oplus \mathsf{g}^{-1}(X')] = 1/2^n$
$(3,5)$	$\Pr[K_1 \oplus \mathsf{g}(K_1) = X \oplus \mathsf{g}(X')] = 1/2^n$
$(3,6)$	$\Pr[K_1 = X \oplus \mathsf{g}^{-1}(X' \oplus K_2)] = 1/2^n$
$(3,7)$	$\Pr[K_1 = X \oplus X' \oplus K_2] = 1/2^n$
$(4,5)$	$\Pr[K_1 = \mathsf{g}^{-1}(\mathsf{g}^{-1}(X)) \oplus X'] = 1/2^n$
$(4,6)$	$\Pr[K_2 = X \oplus X'] = 1/2^n$
$(4,7)$	$\Pr[K_2 = \mathsf{g}^{-1}(X) \oplus X'] = 1/2^n$
$(5,6)$	$\Pr[K_1 = X \oplus \mathsf{g}^{-1}(\mathsf{g}^{-1}(X' \oplus K_2))] = 1/2^n$
$(5,7)$	$\Pr[K_1 = X \oplus \mathsf{g}^{-1}(X' \oplus K_2)] = 1/2^n$
$(6,7)$	$\Pr[K_2 \oplus \mathsf{g}(K_2) = X \oplus \mathsf{g}(X')] = 1/2^n$

Fig. 21. Case analysis of (8)

Algorithm $\mathsf{HASH3}^*_{F_1,F_2}(N, A)$

1. $S_{\mathsf{H}}[0] \leftarrow F_1(\mathsf{zpp}(N))$
2. **if** $|A| = 0$ **then**
3. $\quad Y \leftarrow S_{\mathsf{H}}[0] \oplus \mathsf{Len}(A)$ // $\mathsf{Len}(A) = 0^n$
4. $\quad$ **return** Y
5. $(A[1], \ldots, A[a]) \xleftarrow{n} A$
6. **for** $i \leftarrow 1$ **to** $a - 1$ **do**
7. $\quad S_{\mathsf{H}}[i] \leftarrow F_2(S_{\mathsf{H}}[i-1] \oplus A[i])$
8. $S_{\mathsf{H}}[a] \leftarrow F_2(S_{\mathsf{H}}[a-1] \oplus \mathsf{zap}(A[a]))$
9. $Y \leftarrow S_{\mathsf{H}}[a] \oplus \mathsf{Len}(A)$
10. **return** Y

Fig. 22. Definition of $\mathsf{HASH3}^*$ used in the proof of Lemma 2

Finally, we evaluate the probability of the bad event. From the analysis above, for any two distinct queries (j, X), $(j', X') \in \{(j_1, X_1), \ldots, (j_q, X_q)\}$, we have $\Pr[I(j, X) = I(j', X')] \leq 1/2^n$. Therefore, we obtain

$$\Pr\left[\mathcal{B}^{Q_1(\cdot),\ldots,Q_7(\cdot)} \text{ sets } \mathsf{bad}\right] \leq \sum_{1 \leq i < i' \leq q} \frac{1}{2^n} \leq \frac{0.5q^2}{2^n}$$

as claimed. □

E Proof of Lemma 2

Without loss of generality, we assume that the adversary $\mathcal{B}$ makes exactly q queries, which are written as $(j_1, N_1, A_1), \ldots, (j_q, N_q, A_q)$. Suppose that $\mathcal{B}$ interacts with $\mathsf{HASH3}$ and $\mathsf{HASH3}'$. We say that a bad event occurs and write $\mathcal{B}^{\mathsf{HASH3}(\cdot,\cdot),\mathsf{HASH3}'(\cdot,\cdot)}$ sets bad, if there exist two distinct queries (j, N, A), $(j', N', A') \in \{(j_1, N_1, A_1), \ldots, (j_q, N_q, A_q)\}$ such that

– $j = j'$, and

– $\mathsf{HASH3}^*(N, A) = \mathsf{HASH3}^*(N', A')$,

where $\mathsf{HASH3}^*$ is defined in Fig. 22. It takes N and A as input, and $\mathsf{HASH3}^*$ outputs the input value of the last invocation of the random function in $\mathsf{HASH3}$ or $\mathsf{HASH3}'$. The absence of the bad event implies that the responses that $\mathcal{B}$ receives from the oracles are uniform and independent random bit strings. Therefore, we have

$$\mathbf{Adv}^{\mathrm{ind}}_{\mathsf{HASH3},\mathsf{HASH3}'}(\mathcal{B}) \leq \Pr\left[\mathcal{B}^{\mathsf{HASH3}(\cdot,\cdot),\mathsf{HASH3}'(\cdot,\cdot)} \text{ sets } \mathsf{bad}\right]. \tag{9}$$

We may now without loss of generality assume that the adversary is non-adaptive. We fix all queries $(j_1, N_1, A_1), \ldots, (j_q, N_q, A_q)$ made by $\mathcal{B}$, and evaluate the right hand side of (9) based on the randomness of F_1 and F_2. Let $(j, N, A), (j', N', A') \in \{(j_1, N_1, A_1), \ldots, (j_q, N_q, A_q)\}$ be two distinct queries such that $j = j'$. In what follows, we evaluate

$$\Pr\left[\mathsf{HASH3}^*(N, A) = \mathsf{HASH3}^*(N', A')\right]. \tag{10}$$

In order to evaluate (10), we introduce a lemma shown by Black and Rogaway [4]. Let $M, M' \in \{0,1\}^*$ be two distinct strings such that $|M| = \ell n$ and $|M'| = \ell' n$, where $\ell, \ell' \geq 1$. Let the partitions be $(M[1], \ldots, M[\ell]) \xleftarrow{n} M$ and $(M'[1], \ldots, M'[\ell']) \xleftarrow{n} M'$. Let $F \xleftarrow{\$} \mathrm{Rand}(n)$ be a random function. We define $\mathsf{CBC}_F(M)$ as $S[\ell]$, where $S[i] \leftarrow F(S[i-1] \oplus M[i])$ for $i = 1, \ldots, \ell$ and $S[0] = 0^n$. We define $\mathsf{CBC}_F(M')$ analogously. Let $\mathsf{COLL}_F(M, M')$ denote the event $\mathsf{CBC}_F(M) = \mathsf{CBC}_F(M')$. The following lemma shows the upper bound on the probability of $\mathsf{COLL}_F(M, M')$.

Lemma 5 ([4]). $\Pr\left[\mathsf{COLL}_F(M, M')\right] \leq \ell\ell'/2^n + \max\{\ell, \ell'\}/2^n$, *where the probability is taken over* $F \xleftarrow{\$} \mathrm{Rand}(n)$.

Now for two distinct queries $(j, N, A), (j', N', A') \in \{(j_1, N_1, A_1), \ldots, (j_q, N_q, A_q)\}$, let $(A[1], \ldots, A[a]) \xleftarrow{n} A$ and $(A'[1], \ldots, A'[a']) \xleftarrow{n} A'$ be the partition. For N and $A = (A[1], \ldots, A[a])$, we consider $M = (M[1], \ldots, M[\ell])$ defined as

$$M \leftarrow \begin{cases} F_1(\mathsf{zpp}(N)) \oplus \mathsf{Len}(A) & \text{if } |A| = 0, \\ (F_1(\mathsf{zpp}(N)) \oplus \mathsf{zap}(A[1]), \mathsf{Len}(A)) & \text{if } 1 \leq |A| \leq n, \\ (F_1(\mathsf{zpp}(N)) \oplus A[1], A[2], \ldots, A[a-1], \mathsf{zap}(A[a]), \mathsf{Len}(A)) & \text{if } |A| \geq n+1. \end{cases}$$

We note that $\ell = \lceil |A|/n \rceil + 1$ holds, and we define $M' = (M'[1], \ldots, M'[\ell'])$ from N' and $A' = (A'[1], \ldots, A'[a'])$ analogously.

Now we see that if $\mathsf{HASH3}^*(N, A) = \mathsf{HASH3}^*(N', A')$ holds, then $\mathsf{COLL}_F(M, M')$ holds, which is $\mathsf{CBC}_F(M) = \mathsf{CBC}_F(M')$, by setting $F \leftarrow F_2$. However, the converse may not be true since we may have $\mathsf{COLL}_F(M, M')$ even if $\mathsf{HASH3}^*(N, A) \neq \mathsf{HASH3}^*(N', A')$. Now we evaluate (10) in two cases, Case $N = N'$, and Case $N \neq N'$.

Case $N = N'$. We arbitrarily fix F_1. We have $M \neq M'$ from $A \neq A'$, and hence by using Lemma 5 with $F \leftarrow F_2$, we obtain

$$(10) \leq \Pr\left[\mathsf{COLL}_F(M, M')\right] \leq \frac{\ell\ell'}{2^n} + \frac{\max\{\ell, \ell'\}}{2^n},$$

where $\ell = \lceil |A|/n \rceil + 1$ and $\ell' = \lceil |A'|/n \rceil + 1$.

Case $N \neq N'$. We have

$$\begin{aligned}(10) &\leq \Pr\left[\mathsf{COLL}_F(M, M') \text{ and } M[1] = M'[1]\right] + \Pr\left[\mathsf{COLL}_F(M, M') \text{ and } M[1] \neq M'[1]\right] \\ &\leq \Pr\left[M[1] = M'[1]\right] + \Pr\left[\mathsf{COLL}_F(M, M') \mid M[1] \neq M'[1]\right],\end{aligned}$$

and we also have $\Pr\left[\mathsf{COLL}_F(M, M') \mid M[1] \neq M'[1]\right] \leq \ell\ell'/2^n + \max\{\ell, \ell'\}/2^n$, where $\ell = \lceil |A|/n \rceil + 1$ and $\ell' = \lceil |A'|/n \rceil + 1$, from Lemma 5. It remains to evaluate $\Pr\left[M[1] = M'[1]\right]$. $M[1]$ is obtained as $F_1(\mathsf{zpp}(N)) \oplus Z$, where $Z \in \{\mathsf{Len}(A), \mathsf{zap}(A[1]), A[1]\}$ depending on the length of A. Similarly, $M'[1] = F_1(\mathsf{zpp}(N')) \oplus Z'$, where $Z' \in \{\mathsf{Len}(A'), \mathsf{zap}(A'[1]), A'[1]\}$, and we are interested in the event $F_1(\mathsf{zpp}(N)) \oplus$

$Z = F_1(\mathsf{zpp}(N')) \oplus Z'$. Since $N \neq N'$, we have $\mathsf{zpp}(N) \neq \mathsf{zpp}(N')$, and hence we have $\Pr[M[1] = M'[1]] = 1/2^n$. Therefore, we have

$$(10) \leq \frac{1}{2^n} + \frac{\ell\ell'}{2^n} + \frac{\max\{\ell, \ell'\}}{2^n},$$

where $\ell = \lceil |A|/n \rceil + 1$ and $\ell' = \lceil |A'|/n \rceil + 1$.

Finally, we evaluate the probability of the bad event. From the analyses above, for any two distinct queries $(j, N, A), (j', N', A') \in \{(j_1, N_1, A_1), \ldots, (j_q, N_q, A_q)\}$, we have $(10) \leq 1/2^n + \ell\ell'/2^n + \max\{\ell, \ell'\}/2^n$ for both cases. We obtain

$$\begin{aligned}
\Pr\left[\mathcal{B}^{\mathsf{HASH3}(\cdot,\cdot),\mathsf{HASH3'}(\cdot,\cdot)} \text{ sets } \mathsf{bad}\right] &\leq \sum_{1 \leq i < i' \leq q} \frac{1}{2^n} + \frac{\ell_i \ell_{i'}}{2^n} + \frac{\max\{\ell_i, \ell_{i'}\}}{2^n} \\
&\leq \frac{0.5q^2}{2^n} + \frac{\left(\sum_{1 \leq i \leq q} \ell_i\right)^2}{2^n} \\
&\leq \frac{0.5q^2}{2^n} + \frac{(q + \sigma_A)^2}{2^n},
\end{aligned}$$

where $\ell_i = \lceil |A_i|/n \rceil + 1$ and $\ell_{i'} = \lceil |A_{i'}|/n \rceil + 1$, and the second last inequality follows from $\ell_i \leq a_i + 1$ and the proof of [4, Theorem 4]. □

F Proof of Lemma 3

For reference, we first present a figure of $T \leftarrow \mathsf{PRF4}_{\mathsf{HASH4'},F_6,F_7}(N, A, C)$ in Fig. 23. The proof is similar to that of Lemma 2 in Appendix. E.

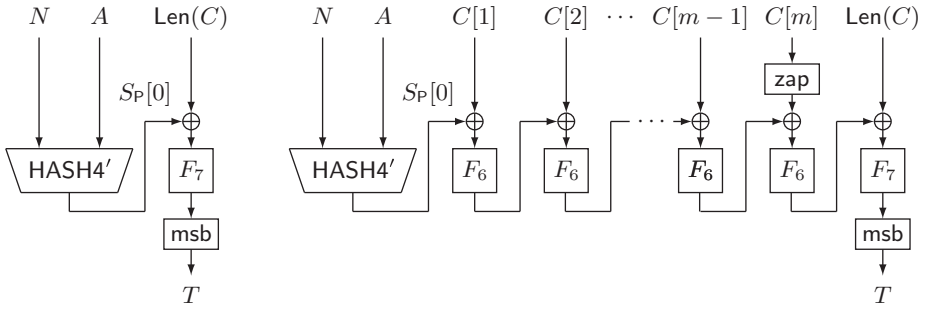

Fig. 23. $T \leftarrow \mathsf{PRF4}_{\mathsf{HASH4'},F_6,F_7}(N, A, C)$ for $|C| = 0$ (left), and $|C| \geq 1$ (right). We have $\mathsf{Len}(C) = 0^n$ in the left figure.

Without loss of generality, we assume that $\mathcal{B}$ makes exactly q queries, and we write the queries as $(N_1, A_1, C_1), \ldots, (N_q, A_q, C_q)$. Consider the case where $\mathcal{B}$ interacts with $\mathsf{PRF4}$, and we say that a bad event occurs and write $\mathcal{B}^{\mathsf{PRF4}(\cdot,\cdot,\cdot)}$ sets bad, if there exist two distinct queries $(N, A, C), (N', A', C') \in \{(N_1, A_1, C_1), \ldots, (N_q, A_q, C_q)\}$ such that $I = I'$, where I and I' denote the input values of F_7 for (N, A, C) and (N', A', C'), respectively. Specifically, $I = S_{\mathsf{P}}[0] \oplus \mathsf{Len}(C)$ when $|C| = 0$ (line 2 of Algorithm $\mathsf{PRF4}'_{F_6,F_7}(S_{\mathsf{P}}[0], C)$ in Fig. 19), and $I = S_{\mathsf{P}}[m] \oplus \mathsf{Len}(C)$ when $|C| \geq 1$ (line 9 of Algorithm $\mathsf{PRF4}'_{F_6,F_7}(S_{\mathsf{P}}[0], C)$ in Fig. 19). I' is similarly defined. The absence of the bad event implies that the responses that $\mathcal{A}$ receives are random bit strings. Therefore, we have

$$\mathbf{Adv}^{\mathrm{ind}}_{\mathsf{PRF4}}(\mathcal{B}) \leq \Pr\left[\mathcal{B}^{\mathsf{PRF4}(\cdot,\cdot,\cdot)} \text{ sets } \mathsf{bad}\right]. \tag{11}$$

We may assume that $\mathcal{B}$ is non-adaptive, so we now fix all queries $(N_1, A_1, C_1), \ldots, (N_q, A_q, C_q)$, and evaluate the right hand side of (11). Let $(N, A, C), (N', A', C') \in \{(N_1, A_1, C_1), \ldots, (N_q, A_q, C_q)\}$ be

two distinct queries, where $(C[1],\ldots,C[m]) \xleftarrow{n} C$ and $(C'[1],\ldots,C'[m']) \xleftarrow{n} C'$. For (N,A) and $C = (C[1],\ldots,C[m])$, define $M = (M[1],\ldots,M[\ell])$ as

$$M \leftarrow \begin{cases} \mathsf{HASH4}'(N,A) \oplus \mathsf{Len}(C) & \text{if } |C| = 0, \\ (\mathsf{HASH4}'(N,A) \oplus \mathsf{zap}(C[1]), \mathsf{Len}(C)) & \text{if } 1 \le |C| \le n, \\ (\mathsf{HASH4}'(N,A) \oplus C[1], C[2], \ldots, C[m-1], \mathsf{zap}(C[m]), \mathsf{Len}(C)) & \text{if } |C| \ge n+1. \end{cases}$$

We have $\ell = \lceil |C|/n \rceil + 1$, and we similarly define $M' = (M'[1],\ldots,M'[\ell'])$ from (N',A') and $C' = (C'[1],\ldots,C'[m'])$.

Now we see that if $I = I'$ holds, then we have $\mathsf{CBC}_F(M) = \mathsf{CBC}_F(M')$, i.e., we have $\mathsf{COLL}_F(M,M')$, by substituting $F \leftarrow F_2$. We next evaluate $\Pr[I = I']$ in two cases, Case $(N,A) = (N',A')$, and Case $(N,A) \neq (N',A')$.

Case $(N,A) = (N',A')$. We arbitrarily fix $\mathsf{HASH4}'$. Then we have $M \neq M'$ from $C \neq C'$, and we use Lemma 5 with $F \leftarrow F_2$ to see

$$\Pr[I = I'] \le \Pr\left[\mathsf{COLL}_F(M,M')\right] \le \frac{\ell\ell'}{2^n} + \frac{\max\{\ell,\ell'\}}{2^n},$$

where $\ell = \lceil |C|/n \rceil + 1$ and $\ell' = \lceil |C'|/n \rceil + 1$.

Case $(N,A) \neq (N',A')$. We proceed as follows.

$$\begin{aligned} \Pr[I = I'] &\le \Pr\left[\mathsf{COLL}_F(M,M') \text{ and } M[1] = M'[1]\right] + \Pr\left[\mathsf{COLL}_F(M,M') \text{ and } M[1] \neq M'[1]\right] \\ &\le \Pr\left[M[1] = M'[1]\right] + \Pr\left[\mathsf{COLL}_F(M,M') \mid M[1] \neq M'[1]\right] \end{aligned}$$

We first evaluate the last term. We use Lemma 5 to have $\Pr\left[\mathsf{COLL}_F(M,M') \mid M[1] \neq M'[1]\right] \le \ell\ell'/2^n + \max\{\ell,\ell'\}/2^n$, where $\ell = \lceil |C|/n \rceil + 1$ and $\ell' = \lceil |C'|/n \rceil + 1$. We next evaluate $\Pr\left[M[1] = M'[1]\right]$. $M[1]$ is obtained as $\mathsf{HASH4}'(N,A) \oplus Z$, where the value of $Z \in \{\mathsf{Len}(C), \mathsf{zap}(C[1]), C[1]\}$ depends on $|C|$, and $M'[1]$ is obtained as $\mathsf{HASH4}'(N',A') \oplus Z'$, where $Z' \in \{\mathsf{Len}(C'), \mathsf{zap}(C'[1]), C'[1]\}$. The event $M[1] = M'[1]$ is equivalent to $\mathsf{HASH4}'(N,A) \oplus Z = \mathsf{HASH4}'(N',A') \oplus Z'$, and since $(N,A) \neq (N',A')$, we have $\Pr\left[M[1] = M'[1]\right] = 1/2^n$. Therefore, we have

$$\Pr[I = I'] \le \frac{1}{2^n} + \frac{\ell\ell'}{2^n} + \frac{\max\{\ell,\ell'\}}{2^n},$$

where $\ell = \lceil |C|/n \rceil + 1$ and $\ell' = \lceil |C'|/n \rceil + 1$.

Now we are ready to evaluate the probability of the bad event. For any two distinct queries (N,A,C), $(N',A',C') \in \{(N_1,A_1,C_1),\ldots,(N_q,A_q,C_q)\}$, for both cases, we have $\Pr[I = I'] \le 1/2^n + \ell\ell'/2^n + \max\{\ell,\ell'\}/2^n$. Therefore, we obtain

$$\begin{aligned} \Pr\left[\mathcal{B}^{\mathsf{PRF4}(\cdot,\cdot,\cdot)} \text{ sets } \mathsf{bad}\right] &\le \sum_{1 \le i < i' \le q} \frac{1}{2^n} + \frac{\ell_i \ell_{i'}}{2^n} + \frac{\max\{\ell_i,\ell_{i'}\}}{2^n} \\ &\le \frac{0.5q^2}{2^n} + \frac{\left(\sum_{1 \le i \le q} \ell_i\right)^2}{2^n} \\ &\le \frac{0.5q^2}{2^n} + \frac{(q+\sigma_C)^2}{2^n}, \end{aligned}$$

where $\ell_i = \lceil |C_i|/n \rceil + 1$ and $\ell_{i'} = \lceil |C_{i'}|/n \rceil + 1$. The second last inequality uses $\ell_i \le m_i + 1$, where $(C_i[1],\ldots,C_i[m_i]) \xleftarrow{n} C_i$, and the proof of [4, Theorem 4]. □

G Proof of Lemma 4

Privacy of SILC5. First, we consider the privacy of SILC5, and assume that $\mathcal{A}$ interacts with SILC5-$\mathcal{E}$. Let (N_i,A_i,M_i) be the i-th query, and (C_i,T_i) be the response, where $(C_i[1],\ldots,C_i[m_i]) \xleftarrow{n} C_i$. Recall that if $|C_i| = 0$, then $m_i = 1$ and $C_i[1] = \varepsilon$. Let $\mathcal{I}_i$ be the set of input values of F_4 for the i-th query.

Specifically, we have $\mathcal{I}_i = \{\mathsf{fix1}(C_i[1]), \ldots, \mathsf{fix1}(C_i[m_i - 1])\}$. Note that $\mathcal{I}_i = \emptyset$ if $0 \leq |C_i| \leq n$. We say that a bad event occurs and write $\mathcal{A}^{\text{SILC5-}\mathcal{E}(\cdot,\cdot,\cdot)}$ sets bad if, for some $1 \leq i \leq q$ and $1 \leq j \leq m_i - 1$, we have

$$\mathsf{fix1}(C_i[j]) \in \mathcal{I}_1 \cup \cdots \cup \mathcal{I}_{i-1} \cup \{\mathsf{fix1}(C_i[1]), \ldots, \mathsf{fix1}(C_i[j-1])\}. \tag{12}$$

If (12) holds, then we say that $C_i[j]$ causes the bad event. The absence of the bad event implies that the responses that $\mathcal{A}$ receives are uniform random bit strings. We therefore have

$$\mathbf{Adv}^{\text{priv}}_{\text{SILC5}[\ell_N,\tau]}(\mathcal{A}) \leq \Pr\left[\mathcal{A}^{\text{SILC5-}\mathcal{E}(\cdot,\cdot,\cdot)} \text{ sets } \mathsf{bad}\right].$$

Now assume that $C_1[1], \ldots, C_1[m_1 - 1], \ldots, C_{i-1}[1], \ldots, C_{i-1}[m_{i-1} - 1], C_i[1], \ldots, C_i[j-1]$ do not cause the bad event. Then we see that $\mathsf{fix1}(C_i[j])$ is a uniform random string of $(n-1)$ bits. We also see that $C_1[1], \ldots, C_q[1]$ are all random bits from the nonce-respecting assumption on $\mathcal{A}$, and other values are random bits from the randomness of F_4. Therefore, we obtain the upper bound on $\Pr\left[\mathcal{A}^{\text{SILC5-}\mathcal{E}(\cdot,\cdot,\cdot)} \text{ sets } \mathsf{bad}\right]$ as

$$\sum_{1 \leq i \leq q} \sum_{1 \leq j \leq m_i - 1} \frac{m_1 - 1}{2^{n-1}} + \cdots + \frac{m_{i-1} - 1}{2^{n-1}} + \frac{j-1}{2^{n-1}} \leq \sum_{0 \leq \ell \leq \sigma_M - 1} \frac{\ell}{2^{n-1}} \leq \frac{\sigma_M^2}{2^n},$$

and therefore, we obtain $\mathbf{Adv}^{\text{priv}}_{\text{SILC5}[\ell_N,\tau]}(\mathcal{A}) \leq \sigma_M^2/2^n$.

Authenticity of SILC5. Finally, we analyze the authenticity of SILC5. Consider the j-th decryption query (N'_j, A'_j, C'_j, T'_j), and suppose that, prior to this decryption query, $\mathcal{A}$ made i encryption queries and obtained the responses. Let $(N_1, A_1, M_1, C_1, T_1), \ldots, (N_i, A_i, M_i, C_i, T_i)$ be the list of the queries and the responses. Now $(N'_j, A'_j, C'_j) \notin \{(N_1, A_1, C_1), \ldots, (N_i, A_i, C_i)\}$ holds, since otherwise $\mathcal{A}$ does not succeed. This implies that, each time $\mathcal{A}$ makes a decryption query, $\mathcal{A}$ has to guess the output value of $\mathsf{PRF5}$ for a new input value. Since $\mathcal{A}$ makes at most q' decryption queries, we have $\mathbf{Adv}^{\text{auth}}_{\text{SILC5}[\ell_N,\tau]}(\mathcal{A}) \leq q'/2^\tau$. □

H Changes

H.1 Changes from SILC v1 to SILC v2

The specification of SILC 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 [8,17,22]. The same notes as CLOC v2 apply here. We repeat them for completeness.

- The introduction of `param` does not mean that SILC 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 SILC, since we may consider $\texttt{param} \,\|\, N$ as a nonce, and then the provable security results in [13], also presented in Sect. 3, 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 SILC and ECB mode results in the loss of security. Similarly, SILC and CLOC cannot be used concurrently.

The following part of the document was updated.

- The condition on the nonce length was updated to $1 \leq \ell_N \leq n - 9$ to handle `param`.
- The first line in the definition of HASH in Fig. 2 was updated to concatenate `param` to N.
- Figures 3 and 6 were updated.
- Sect. 1.3 was updated. The parameter space was reduced so that different parameters can be encoded into `param`, and Table 1 was added.
- Security theorems were added in Sect. 3, and the proofs were presented in Appendices C, D, E, F, and G.
- We also made minor changes.