Integrity Analysis of Authenticated Encryption Based on Stream Ciphers *

Kazuya Imamura[1], Kazuhiko Minematsu[2], and Tetsu Iwata[3]

[1] Nagoya University, Japan, k_imamur@echo.nuee.nagoya-u.ac.jp
[2] NEC Corporation, Japan, k-minematsu@ah.jp.nec.com
[3] Nagoya University, Japan, iwata@cse.nagoya-u.ac.jp

Abstract. We study the security of authenticated encryption based on a stream cipher and a universal hash function. We consider ChaCha20-Poly1305 and generic constructions proposed by Sarkar, where the generic constructions include 14 AEAD (authenticated encryption with associated data) schemes and 3 DAEAD (deterministic AEAD) schemes. In this paper, we analyze the integrity of these schemes both in the standard INT-CTXT notion and in the RUP (releasing unverified plaintext) setting called INT-RUP notion. We present INT-CTXT attacks against 3 out of the 14 AEAD schemes and 1 out of the 3 DAEAD schemes. We then show INT-RUP attacks against 1 out of the 14 AEAD schemes and the 2 remaining DAEAD schemes. We next show that ChaCha20-Poly1305 is provably secure in the INT-RUP notion. Finally, we show that 4 out of the remaining 10 AEAD schemes are provably secure in the INT-RUP notion.

Keywords: authenticated encryption, stream cipher, universal hash function, provable security, integrity, releasing unverified plaintext

1 Introduction

Background. An authenticated encryption (AE) scheme is a symmetric encryption primitive where the goal is to achieve both privacy and integrity of plaintexts. Examples of AE include GCM [11], CCM [19], and EAX [6], and they are widely used in practice. There are several ways to construct AE, and the construction by the generic composition (GC), which was formalized by Bellare and Namprempre [3], is to combine existing primitives, one for encryption and the other for authentication, to obtain AE. The security notion for integrity, called INT-CTXT, requires that an adversary is unable to produce a ciphertext that is accepted in verification, where the adversary has access to an encryption oracle. Authenticated encryption with associated data (AEAD) was formalized in [15], where associated data (AD) is the input that is authenticated but not encrypted. Nonce-based encryption was formalized in [16], where a nonce is the input of the scheme which is supposed to be used only once, meaning that it is not repeated. Implementation of a nonce is non-trivial in practice, and a repeat of a nonce in AEAD is often devastating. To address this issue, deterministic authenticated encryption (DAE) was formalized in [17]. More precisely, DAEAD is DAE that supports AD, which is AE that remains secure without the use of a nonce and does not leak information about a plaintext from a ciphertext, except for the repetition of the input. In this sense DAEAD has the nonce-reuse misuse resistance, but on a downside, DAEAD requires off-line computation. The GC in [3] was refined by Namprempre, Rogaway, and Shrimpton [12] by explicitly treating the use of a nonce.

Another direction of GC was put forward by Sarkar [18], where a stream cipher is used for encryption and a universal hash function is used for authentication. In [18], a total of 17 AEAD/DAEAD schemes are proposed. There are 14 AEAD schemes, called $\mathsf{AEAD}\text{-}\{1, 2, 2\mathrm{a}, 2\mathrm{b}, 3, 4, 4\mathrm{a}, 4\mathrm{b}, 5, 6, 6\mathrm{a}, 7, 8, 8\mathrm{a}\}$, and 3 DAEAD schemes, called $\mathsf{DAEAD}\text{-}\{1, 2, 2\mathrm{a}\}$. It was proved that all these schemes achieve both privacy and integrity under the assumption that the stream cipher is a pseudo-random function (PRF) and that the hash function is a universal hash function.

Related AEAD which we call ChaCha20-Poly1305 was proposed by Nir and Langley [13]. A stream cipher ChaCha20 [8] is used for encryption and a universal hash function Poly1305 [7] is used for authentication, which were designed by Bernstein. ChaCha20-Poly1305 is practically used in IETF protocols [13]. The scheme is similar to one of the GC called AEAD-2b of [18], but there is a subtle difference and it does

*A proceedings version of this paper appears in [10]. The final publication is available at Springer via http://dx.doi.org/10.1007/978-3-319-47422-9_15. This is the full version.

Table 1. INT-CTXT and INT-RUP security of AEAD and DAEAD schemes. The mark ✓ means secure, ✗ means insecure, (✗) follows from the INT-CTXT result, and ? remains open.

Scheme	INT-CTXT	INT-RUP
ChaCha20-Poly1305	✓ ([14])	✓ (Theorem 1)
AEAD-1	✓ ([18, Theorem 20])	✓ (Theorem 2)
AEAD-2	✓ ([18, Theorem 20])	✓ (Theorem 2)
AEAD-2a	✗ (Sect. 4.1)	(✗)
AEAD-2b	✓ ([18, Theorem 20])	✗ (Sect. 4.2)
AEAD-3	✓ ([18, Theorem 20])	✓ (Theorem 2)
AEAD-4	✓ ([18, Theorem 20])	✓ (Theorem 2)
AEAD-4a	✗ (Sect. 4.1)	(✗)
AEAD-4b	✗ (Sect. 4.1)	(✗)
AEAD-5	✓ ([18, Theorem 20])	?
AEAD-6	✓ ([18, Theorem 20])	?
AEAD-6a	✓ ([18, Theorem 20])	?
AEAD-7	✓ ([18, Theorem 20])	?
AEAD-8	✓ ([18, Theorem 20])	?
AEAD-8a	✓ ([18, Theorem 20])	?
DAEAD-1	✓ ([18, Theorem 21])	✗ (Sect. 4.2)
DAEAD-2	✓ ([18, Theorem 21])	✗ (Sect. 4.2)
DAEAD-2a	✗ (Sect. 4.1)	(✗)

not exactly follow the composition. Procter [14] proved that ChaCha20-Poly1305 achieves both privacy and authenticity in the model of [4] under the assumption that ChaCha20 block function is a PRF.

Another security notion called the releasing unverified plaintext (RUP) was formalized by Andreeva et al. [1]. This notion is motivated to cover the situation in which there is not enough memory in decryption devices to store the entire decrypted plaintext and decrypted plaintexts are immediately required in real time. The corresponding integrity notion is called INT-RUP, and the goal of an adversary under the INT-RUP notion is to produce a new ciphertext which is accepted in the verification, where the adversary has access to the oracle that returns unverified plaintexts. We remark that the notion is often referred to as the decryption-misuse setting.

Our Contributions. In this paper, we study the integrity of AEAD and DAEAD based on a stream cipher and a universal hash function in the standard INT-CTXT notion and in the decryption-misuse, INT-RUP notion.

Our results are summarized in Table 1. We first show that there are INT-CTXT attacks against 4 out of 17 schemes in [18], invalidating the original INT-CTXT security claims. In addition to this, we show INT-RUP attacks against 3 out of the 17 schemes, showing a sort of tightness of the original INT-CTXT claims. All our attacks need only a few queries, and are hence practical. Specifically, we show INT-CTXT attacks against $\mathsf{AEAD}\text{-}\{2\mathrm{a}, 4\mathrm{a}, 4\mathrm{b}\}$ and $\mathsf{DAEAD}\text{-}2\mathrm{a}$, and INT-RUP attacks against $\mathsf{AEAD}\text{-}2\mathrm{b}$ and $\mathsf{DAEAD}\text{-}\{1, 2\}$. We note that INT-RUP security is not claimed in [18], as [18] predates [1].

A universal hash function, or more precisely an almost XOR universal (AXU) hash function, is used in these schemes, and our observation is that the definition of an AXU hash function does not exclude a case where it has a fixed point, which is the input X and the output Y of the hash function H such that $\mathsf{H}_L(X) = Y$ holds independent of the key L. Our INT-CTXT attacks against $\mathsf{AEAD}\text{-}\{2\mathrm{a}, 4\mathrm{a}, 4\mathrm{b}\}$ and $\mathsf{DAEAD}\text{-}2\mathrm{a}$, and INT-RUP attacks against $\mathsf{DAEAD}\text{-}\{1, 2\}$ make use of the existence of the fixed point. The INT-RUP attack against $\mathsf{AEAD}\text{-}2\mathrm{b}$ is based on a different observation. We show that an adversary can recover the hash key from the unverified plaintext and hence break the INT-RUP security with probability 1. The attacks are described in Sect. 4. We remark that our attacks imply the existence of a universal hash function that makes these schemes insecure, and the attacks do not imply the non-existence of a universal hash function that makes the schemes secure.

Next, we show that ChaCha20-Poly1305 is INT-RUP secure under the same assumption as Procter. While ChaCha20-Poly1305 is similar to $\mathsf{AEAD}\text{-}2\mathrm{b}$, there is a difference in the order of the generation of a hash key and a keystream, and this small difference results in the difference in INT-RUP security.

Finally, we show that $\mathsf{AEAD}\text{-}\{1,2,3,4\}$ are INT-RUP secure under the assumption that a stream cipher is a PRF. Our security bounds of these schemes are shown in Sect. 5.

2 Preliminaries

2.1 Notation

We write $\{0,1\}^*$ for the set of all finite bit strings, and for an integer $l \geq 0$, we write $\{0,1\}^l$ for all the l-bit strings. We write ε for the empty string. For $X \in \{0,1\}^*$, $|X|$ is its length in bits. For $X \in \{0,1\}^*$ and an integer l such that $|X| \geq l$, $\mathsf{msb}_l(X)$ denotes the most significant (the leftmost) l bits of X, and $\mathsf{lsb}_l(X)$ denotes the least significant (the rightmost) l bits of X. For $X, Y \in \{0,1\}^*$, their concatenation is written as $X \parallel Y$. The bit string of m zeros is written as $0^m \in \{0,1\}^m$, and m ones is written as $1^m \in \{0,1\}^m$. For a finite set $\mathcal{X}$, we write $X \xleftarrow{\$} \mathcal{X}$ for a procedure of assigning X an element sampled uniformly at random from $\mathcal{X}$.

2.2 AEAD and DAEAD

Authenticated Encryption with Associated Data (AEAD) [3,15]. The goal of AEAD is to achieve both privacy and integrity of a plaintext, and integrity of associated data. We consider that AEAD consists of three deterministic algorithms, and let $\mathsf{AEAD} = (\mathsf{AEAD.Enc}, \mathsf{AEAD.Dec}, \mathsf{AEAD.Ver})$. Let $K \in \mathcal{K}$ be the underlying secret key that fixes the three algorithms, where $\mathcal{K}$ is the key space. The encryption algorithm $\mathsf{AEAD.Enc}_K$ takes input a nonce N, associated data A, and a plaintext M, and outputs a ciphertext C and a tag T. The decryption algorithm $\mathsf{AEAD.Dec}_K$ takes input N, A, C, and T, and always outputs M. The verification algorithm $\mathsf{AEAD.Ver}_K$ takes input N, A, C, and T, and outputs $\top$ or $\perp$, where $\top$ means that the verification is accepted, and $\perp$ means that the verification is rejected. The correctness requirement must be satisfied, that is, the following requirements are satisfied.

$$\begin{cases} \mathsf{AEAD.Dec}_K(N, A, \mathsf{AEAD.Enc}_K(N, A, M)) = M \\ \mathsf{AEAD.Ver}_K(N, A, \mathsf{AEAD.Enc}_K(N, A, M)) = \top \end{cases}$$

Deterministic AEAD (DAEAD) [17]. DAEAD is AEAD that does not require a nonce. Let $\mathsf{DAEAD} = (\mathsf{DAEAD.Enc}, \mathsf{DAEAD.Dec}, \mathsf{DAEAD.Ver})$, where the encryption algorithm $\mathsf{DAEAD.Enc}_K$ takes input A and M, and outputs C and T, the decryption algorithm $\mathsf{DAEAD.Dec}_K$ takes input A, C, and T, and outputs M, and the verification algorithm $\mathsf{DAEAD.Ver}_K$ takes input A, C, and T, and outputs $\top$ or $\perp$. As in AEAD, the following correctness requirement must be satisfied.

$$\begin{cases} \mathsf{DAEAD.Dec}_K(A, \mathsf{DAEAD.Enc}_K(A, M)) = M \\ \mathsf{DAEAD.Ver}_K(A, \mathsf{DAEAD.Enc}_K(A, M)) = \top \end{cases}$$

2.3 Security Definitions

Ciphertext Integrity. For AEAD and DAEAD, privacy and integrity are the main two security notions. In this paper, we focus on the latter, and describe two notions called INT-CTXT and INT-RUP. INT-CTXT is a standard, classical notion that captures the integrity of ciphertext under chosen ciphertext attacks. INT-RUP considers a more powerful adversary that has access to an oracle that returns unverified plaintexts. We note that INT-RUP is a stronger notion than INT-CTXT, and if a scheme is INT-RUP secure, then it is also INT-CTXT secure.

Definition 1 (INT-CTXT Advantage [3,4]). *Let $\mathcal{A}$ be an adversary that has access to two oracles $\mathsf{AEAD.Enc}_K$ and $\mathsf{AEAD.Ver}_K$. Then we define the INT-CTXT advantage of $\mathcal{A}$ against AEAD as*

$$\mathbf{Adv}^{\text{int-ctxt}}_{\mathsf{AEAD}}(\mathcal{A}) \stackrel{\text{def}}{=} \Pr[\mathcal{A}^{\mathsf{AEAD.Enc}_K, \mathsf{AEAD.Ver}_K} \text{ forges}],$$

where $K \xleftarrow{\$} \mathcal{K}$ and $\mathcal{A}$ forges *is the event that $\mathsf{AEAD.Ver}_K$ returns $\top$ to $\mathcal{A}$. We assume that $\mathcal{A}$ does not repeat a query, and if $\mathcal{A}$ receives a response (C, T) for an encryption query (N, A, M), then $\mathcal{A}$ does not subsequently make a verification query (N, A, C, T). We assume that $\mathcal{A}$ is nonce-respecting with respect to encryption queries, that is, if (N_i, A_i, M_i) denotes the i-th encryption query, then it holds that $N_i \neq N_{i'}$ for any $i \neq i'$.*

We note that $\mathcal{A}$ may repeat a nonce within verification queries, may reuse a nonce used for an encryption query as a nonce for a subsequent verification query, and may reuse a nonce used for a verification query as a nonce for a subsequent encryption query.

The INT-CTXT advantage for DAEAD is similarly defined as

$$\mathbf{Adv}^{\text{int-ctxt}}_{\mathsf{DAEAD}}(\mathcal{A}) \stackrel{\text{def}}{=} \Pr[\mathcal{A}^{\mathsf{DAEAD.Enc}_K, \mathsf{DAEAD.Ver}_K} \text{ forges}].$$

We assume that $\mathcal{A}$ does not repeat a query, and if $\mathcal{A}$ receives a response (C, T) for an encryption query (A, M), then $\mathcal{A}$ does not subsequently make a verification query (A, C, T). Since DAEAD does not take a nonce N as input, $\mathcal{A}$ has no nonce-respecting restriction.

Definition 2 (INT-RUP Advantage [1]). *Let $\mathcal{A}$ be an adversary that has access to three oracles $\mathsf{AEAD.Enc}_K$, $\mathsf{AEAD.Dec}_K$, and $\mathsf{AEAD.Ver}_K$. Then we define the INT-RUP advantage of $\mathcal{A}$ against* AEAD *as*

$$\mathbf{Adv}^{\text{int-rup}}_{\mathsf{AEAD}}(\mathcal{A}) \stackrel{\text{def}}{=} \Pr[\mathcal{A}^{\mathsf{AEAD.Enc}_K, \mathsf{AEAD.Dec}_K, \mathsf{AEAD.Ver}_K} \text{ forges}],$$

where $K \xleftarrow{\$} \mathcal{K}$ and $\mathcal{A}$ forges *is the event that $\mathsf{AEAD.Ver}_K$ returns $\top$ to $\mathcal{A}$. $\mathcal{A}$ does not repeat a query, and if $\mathcal{A}$ receives a response (C, T) for an encryption query (N, A, M), then $\mathcal{A}$ does not subsequently make a verification query (N, A, C, T). $\mathcal{A}$ is nonce-respecting with respect to encryption queries. However, a nonce can be repeated within decryption queries and within verification queries, and the same nonce can be reused across encryption, decryption, and verification queries.*

The INT-RUP advantage of DAEAD is defined as

$$\mathbf{Adv}^{\text{int-rup}}_{\mathsf{DAEAD}}(\mathcal{A}) \stackrel{\text{def}}{=} \Pr[\mathcal{A}^{\mathsf{DAEAD.Enc}_K, \mathsf{DAEAD.Dec}_K, \mathsf{DAEAD.Ver}_K} \text{ forges}].$$

As in the INT-CTXT definition, since DAEAD does not take a nonce N as input, $\mathcal{A}$ has no nonce-respecting restriction. However, we assume that $\mathcal{A}$ does not repeat a query, and if $\mathcal{A}$ receives a response (C, T) for an encryption query (A, M), then $\mathcal{A}$ does not subsequently make a verification query (A, C, T).

Pseudo-Random Function (PRF). Following [18], we consider a stream cipher as a function $\mathsf{SC}\colon \mathcal{K} \times \{0,1\}^n \to \{0,1\}^\ell$, where $\mathcal{K}$ is the set of keys, n denotes the length of IV in bits, and ℓ is a sufficiently large and fixed integer. For a key $K \in \mathcal{K}$, the corresponding function SC_K takes an IV $N \in \{0,1\}^n$ as input, and outputs the keystream $Z \leftarrow \mathsf{SC}_K(N) \in \{0,1\}^\ell$. Let $\mathrm{Rand}(n, \ell)$ be the set of all functions from $\{0,1\}^n$ to $\{0,1\}^\ell$, and let $\mathcal{A}$ be an adversary. Then we define the PRF-advantage of $\mathcal{A}$ against SC as

$$\mathbf{Adv}^{\text{prf}}_{\mathsf{SC}}(\mathcal{A}) \stackrel{\text{def}}{=} \Pr[K \xleftarrow{\$} \mathcal{K}\colon \mathcal{A}^{\mathsf{SC}_K} \Rightarrow 1] - \Pr[F \xleftarrow{\$} \mathrm{Rand}(n, \ell)\colon \mathcal{A}^F \Rightarrow 1],$$

where $\mathcal{A} \Rightarrow 1$ denotes the event that $\mathcal{A}$ outputs 1.

We note that in the above formalization, SC_K is a function with fixed-input length and fixed-output length, and we assume that the output of SC_K is always ℓ bits. However, in the actual usage of SC_K, we abuse the notation and for instance we write $C \leftarrow M \oplus \mathsf{SC}_K(N)$ to mean $C \leftarrow M \oplus \mathsf{msb}_{|M|}(\mathsf{SC}_K(N))$, or $R \parallel Z \leftarrow \mathsf{SC}_K(N)$, where $|R| = n$ and $|Z|$ is clear from the context (such as the length of the plaintext), to mean $Y \leftarrow \mathsf{SC}_K(N), R \leftarrow \mathsf{msb}_n(Y)$, and $Z \leftarrow \mathsf{lsb}_{|Z|}(\mathsf{msb}_{n+|Z|}(Y))$.

Hash Function. Let $\mathsf{H}\colon \mathcal{L} \times \mathcal{D}_{\mathsf{H}} \to \{0,1\}^n$ be a hash function, where $\mathcal{L}$ is a set of hash keys, $\mathcal{D}_{\mathsf{H}}$ denotes the domain, and n is the length of the output in bits. The function specified by $L \in \mathcal{L}$ is written as H_L.

Let $\{\mathsf{H}_L\}$ be a family of keyed hash functions. For any distinct $X', X \in \mathcal{D}_{\mathsf{H}}$ and any $Y \in \{0,1\}^n$, if the differential probability $\Pr[\mathsf{H}_L(X) \oplus \mathsf{H}_L(X') = Y]$ is at most ϵ, then H_L is defined to be an ϵ-almost-XOR-universal (ϵ-AXU) hash function, where the probability is taken over the choice of $L \xleftarrow{\$} \mathcal{L}$.

There are several examples of an ϵ-AXU hash function for small ϵ, and they include GHASH used in GCM [11] and Poly1305 [7]. For these hash functions, the key length is independent of the input length, and the key space is the set of bit strings of a fixed length. Following [18], we call this type of hash functions Type-I hash functions. There are other examples of an ϵ-AXU hash function where the key length can be as long as the input length, or even longer that that, including UMAC [9]. We call this type of hash functions Type-II hash functions.

We observe that the definition of an ϵ-AXU hash function does not exclude a case where the hash function has a fixed point. That is, there may exist $X \in \mathcal{D}_{\mathsf{H}}$ and $Y \in \{0,1\}^n$ such that $\mathsf{H}_L(X) = Y$ holds independently of the key L, since the requirement is about the differential probability, and the uniformity of a single input is irrelevant of the definition. Indeed, practical hash functions like GHASH and Poly1305 have a fixed point. For GHASH, it takes $(A, C) \in \{0,1\}^* \times \{0,1\}^*$ as input and outputs $Y \in \{0,1\}^n$, and it holds that $\mathrm{GHASH}_L(A, C) = Y$ with probability 1 for $(A, C) = (\varepsilon, \varepsilon)$ and $Y = 0^n$. Poly1305 has the same fixed point. We will exploit the existence of a fixed point in our attacks.

3 Schemes

In this section, we present the specifications of AEAD and DAEAD schemes that are proposed in [18], and ChaCha20-Poly1305 [13].

AEAD in [18]. Let fStr be an arbitrary fixed n-bit string. For instance fStr could be 0^n. AEAD schemes in [18] are specified by a stream cipher SC and a hash function H, and we write $\mathsf{AEAD}[\mathsf{SC}, \mathsf{H}]$ for AEAD that uses SC and H as parameters. We also write $\mathsf{AEAD}[\mathrm{Rand}(n, \ell), \mathsf{H}]$ for AEAD where we use a random function $F \xleftarrow{\$} \mathrm{Rand}(n, \ell)$ as the stream cipher SC_K. The encryption algorithms of the schemes are defined in Fig. 1. See Fig. 2 for the overall structure of the encryption algorithms. The decryption and verification algorithms are described in Appendix A. We note that these schemes have the convention on the length of the input. Specifically, the encryption algorithms take any plaintext M which is not empty, and $|M| = 0$ is not allowed [18].

We also note that $\mathsf{AEAD}\text{-}\{1, 2, 2\mathrm{a}, 2\mathrm{b}, 3, 4, 4\mathrm{a}, 4\mathrm{b}\}$ use H as a double-input hash function, but $\mathsf{AEAD}\text{-}\{5, 6, 6\mathrm{a}, 7, 8, 8\mathrm{a}\}$ use H as a hash function that can take both double-input and single-input. See [18] for more details on this matter.

ChaCha20-Poly1305 [13]. Let $\mathcal{K}_{\mathsf{CC}} = \{0,1\}^{256}$ and $\mathcal{K}_{\mathsf{Poly}} = \{0,1\}^{128} \times \{0,1\}^{128}$. We denote ChaCha20 block function by $\mathsf{CC}\colon \mathcal{K}_{\mathsf{CC}} \times \{0,1\}^{32} \times \{0,1\}^{96} \to \{0,1\}^{512}$, and denote Poly1305 authentication function by $\mathsf{Poly}\colon \mathcal{K}_{\mathsf{Poly}} \times \{0,1\}^* \to \{0,1\}^{128}$. The functions specified by $K \in \mathcal{K}_{\mathsf{CC}}$ and $(r, s) \in \mathcal{K}_{\mathsf{Poly}}$ are written as CC_K and $\mathsf{Poly}_{r,s}$, respectively. We write $\mathsf{CC\&Poly}$ for ChaCha20-Poly1305.

With these functions, the encryption algorithm of ChaCha20-Poly1305 is defined in Fig. 3. See Fig. 4 for the overall structure of the encryption algorithm. The decryption and verification algorithms are defined in Appendix A. See [7,8] for further details of the specifications of ChaCha20 and Poly1305.

Observe the similarity to AEAD-2b. $\mathsf{SC}_K(N)$ in AEAD-2b corresponds to $\mathsf{CC}_K(0, N), \mathsf{CC}_K(1, N), \ldots, \mathsf{CC}_K(\lceil |M|/512 \rceil, N)$, where (L, R) in AEAD-2b corresponds to (r, s) in ChaCha20-Poly1305. The difference is that L is taken from the rightmost bits of $\mathsf{SC}_K(N)$, thus the starting position can be moved depending on the length of M, while r is always taken from the same position.

DAEAD in [18]. The encryption algorithms of DAEAD schemes are defined in Fig. 5. See Fig. 6 for the overall structure. We note that the basic idea of DAEAD schemes follows the SIV construction in [17]. The decryption and verification algorithms are described in Appendix A.

4 Negative Results

In this section, we show that $\mathsf{AEAD}\text{-}\{2\mathrm{a}, 4\mathrm{a}, 4\mathrm{b}\}$ and DAEAD-2a are not INT-CTXT secure and that AEAD-2b and $\mathsf{DAEAD}\text{-}\{1, 2\}$ are not INT-RUP secure. Our forgery attacks against these schemes are presented in Fig. 7 and Fig. 8.

Before describing the details of our attacks, we present the following proposition showing that the fixed point can be "moved" to any desired point without changing the value of ϵ.

Proposition 1. *Let $\widetilde{\mathsf{H}}_L\colon \mathcal{D}_{\mathsf{H}} \to \{0,1\}^n$ be a hash function, $\varphi\colon \mathcal{D}_{\mathsf{H}} \to \mathcal{D}_{\mathsf{H}}$ be an injective function, and $c \in \{0,1\}^n$ be a constant. Let $\mathsf{H}_L\colon \mathcal{D}_{\mathsf{H}} \to \{0,1\}^n$ be a hash function, where $\mathsf{H}_L(X) = \widetilde{\mathsf{H}}_L(\varphi(X)) \oplus c$. If $\{\widetilde{\mathsf{H}}_L\}$ is ϵ-AXU, then $\{\mathsf{H}_L\}$ is ϵ-AXU.*

$\mathsf{AEAD\text{-}1.Enc}_{K,L}(N, A, M)$

1. $R \parallel Z \leftarrow \mathsf{SC}_K(N)$
2. $C \leftarrow M \oplus \mathsf{msb}_{|M|}(Z)$
3. $T \leftarrow \mathsf{H}_L(A, C) \oplus R$
4. **return** (C, T)

$\mathsf{AEAD\text{-}2.Enc}_{K,K'}(N, A, M)$

1. $L \leftarrow \mathsf{SC}_K(K')$
2. $R \parallel Z \leftarrow \mathsf{SC}_K(N)$
3. $C \leftarrow M \oplus \mathsf{msb}_{|M|}(Z)$
4. $T \leftarrow \mathsf{H}_L(A, C) \oplus R$
5. **return** (C, T)

$\mathsf{AEAD\text{-}2a.Enc}_{K}(N, A, M)$

1. $K' \leftarrow \mathsf{msb}_n(\mathsf{SC}_K(\mathsf{fStr}))$
2. $L \leftarrow \mathsf{SC}_K(K')$
3. $R \parallel Z \leftarrow \mathsf{SC}_K(N)$
4. $C \leftarrow M \oplus \mathsf{msb}_{|M|}(Z)$
5. $T \leftarrow \mathsf{H}_L(A, C) \oplus R$
6. **return** (C, T)

$\mathsf{AEAD\text{-}2b.Enc}_{K}(N, A, M)$

1. $R \parallel S \leftarrow \mathsf{SC}_K(N)$
2. Parse S as $Z \parallel L$ where $|Z| = |M|$
3. $C \leftarrow M \oplus Z$
4. $T \leftarrow \mathsf{H}_L(A, C) \oplus R$
5. **return** (C, T)

$\mathsf{AEAD\text{-}3.Enc}_{K,L}(N, A, M)$

1. $R \parallel Z \leftarrow \mathsf{SC}_K(N)$
2. $C \leftarrow M \oplus \mathsf{msb}_{|M|}(Z)$
3. $T \leftarrow \mathsf{H}_L(A, M) \oplus R$
4. **return** (C, T)

$\mathsf{AEAD\text{-}4.Enc}_{K,K'}(N, A, M)$

1. $L \leftarrow \mathsf{SC}_K(K')$
2. $R \parallel Z \leftarrow \mathsf{SC}_K(N)$
3. $C \leftarrow M \oplus \mathsf{msb}_{|M|}(Z)$
4. $T \leftarrow \mathsf{H}_L(A, M) \oplus R$
5. **return** (C, T)

$\mathsf{AEAD\text{-}4a.Enc}_{K}(N, A, M)$

1. $K' \leftarrow \mathsf{msb}_n(\mathsf{SC}_K(\mathsf{fStr}))$
2. $L \leftarrow \mathsf{SC}_K(K')$
3. $R \parallel Z \leftarrow \mathsf{SC}_K(N)$
4. $C \leftarrow M \oplus \mathsf{msb}_{|M|}(Z)$
5. $T \leftarrow \mathsf{H}_L(A, M) \oplus R$
6. **return** (C, T)

$\mathsf{AEAD\text{-}4b.Enc}_{K}(N, A, M)$

1. $R \parallel S \leftarrow \mathsf{SC}_K(N)$
2. Parse S as $Z \parallel L$ where $|Z| = |M|$
3. $C \leftarrow M \oplus Z$
4. $T \leftarrow \mathsf{H}_L(A, M) \oplus R$
5. **return** (C, T)

$\mathsf{AEAD\text{-}5.Enc}_{K,L}(N, A, M)$

1. $V \leftarrow \mathsf{H}_L(A, N)$
2. $R \parallel Z \leftarrow \mathsf{SC}_K(V)$
3. $C \leftarrow M \oplus \mathsf{msb}_{|M|}(Z)$
4. $T \leftarrow \mathsf{H}_L(C) \oplus R$
5. **return** (C, T)

$\mathsf{AEAD\text{-}6.Enc}_{K,K'}(N, A, M)$

1. $L_1 \parallel L_2 \leftarrow \mathsf{SC}_K(K')$
2. $V \leftarrow \mathsf{H}_{L_1}(A, N)$
3. $R \parallel Z \leftarrow \mathsf{SC}_K(V)$
4. $C \leftarrow M \oplus \mathsf{msb}_{|M|}(Z)$
5. $T \leftarrow \mathsf{H}_{L_2}(C) \oplus R$
6. **return** (C, T)

$\mathsf{AEAD\text{-}6a.Enc}_{K}(N, A, M)$

1. $K' \leftarrow \mathsf{msb}_n(\mathsf{SC}_K(\mathsf{fStr}))$
2. $L_1 \parallel L_2 \leftarrow \mathsf{SC}_K(K')$
3. $V \leftarrow \mathsf{H}_{L_1}(A, N)$
4. $R \parallel Z \leftarrow \mathsf{SC}_K(V)$
5. $C \leftarrow M \oplus \mathsf{msb}_{|M|}(Z)$
6. $T \leftarrow \mathsf{H}_{L_2}(C) \oplus R$
7. **return** (C, T)

$\mathsf{AEAD\text{-}7.Enc}_{K,L}(N, A, M)$

1. $V \leftarrow \mathsf{H}_L(A, N)$
2. $R \parallel Z \leftarrow \mathsf{SC}_K(V)$
3. $C \leftarrow M \oplus \mathsf{msb}_{|M|}(Z)$
4. $T \leftarrow \mathsf{H}_L(M) \oplus R$
5. **return** (C, T)

$\mathsf{AEAD\text{-}8.Enc}_{K,K'}(N, A, M)$

1. $L_1 \parallel L_2 \leftarrow \mathsf{SC}_K(K')$
2. $V \leftarrow \mathsf{H}_{L_1}(A, N)$
3. $R \parallel Z \leftarrow \mathsf{SC}_K(V)$
4. $C \leftarrow M \oplus \mathsf{msb}_{|M|}(Z)$
5. $T \leftarrow \mathsf{H}_{L_2}(M) \oplus R$
6. **return** (C, T)

$\mathsf{AEAD\text{-}8a.Enc}_{K}(N, A, M)$

1. $K' \leftarrow \mathsf{msb}_n(\mathsf{SC}_K(\mathsf{fStr}))$
2. $L_1 \parallel L_2 \leftarrow \mathsf{SC}_K(K')$
3. $V \leftarrow \mathsf{H}_{L_1}(A, N)$
4. $R \parallel Z \leftarrow \mathsf{SC}_K(V)$
5. $C \leftarrow M \oplus \mathsf{msb}_{|M|}(Z)$
6. $T \leftarrow \mathsf{H}_{L_2}(M) \oplus R$
7. **return** (C, T)

Fig. 1. Pseudocode of the encryption algorithms of AEAD schemes [18]

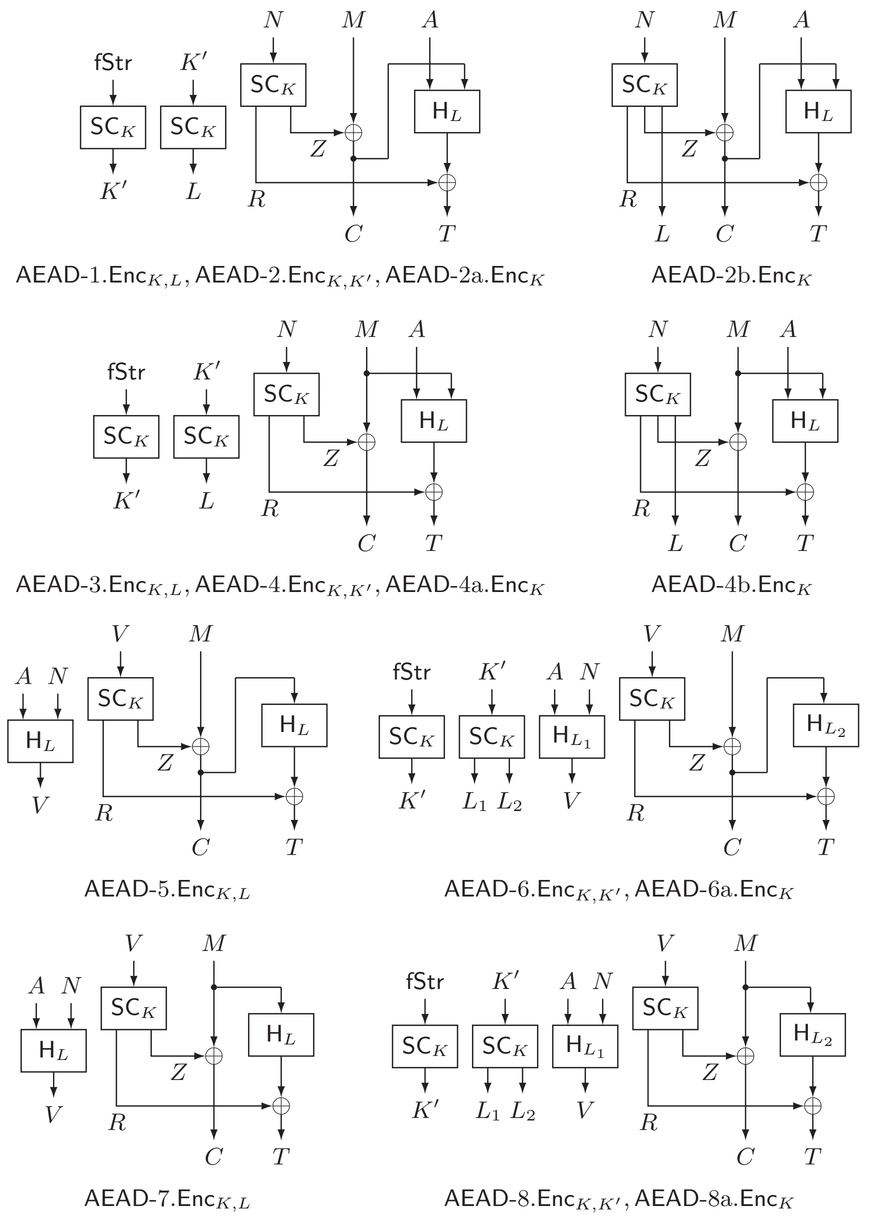

Fig. 2. Illustration of the encryption algorithms of AEAD schemes [18]. In AEAD-2 and AEAD-4, $L = \mathsf{SC}_K(K')$. In AEAD-2a and AEAD-4a, $L = \mathsf{SC}_K(\mathsf{msb}_n(\mathsf{SC}_K(\mathsf{fStr})))$. In AEAD-6 and AEAD-8, $L_1 \parallel L_2 = \mathsf{SC}_K(K')$. In AEAD-6a and AEAD-8a, $L_1 \parallel L_2 = \mathsf{SC}_K(\mathsf{msb}_n(\mathsf{SC}_K(\mathsf{fStr})))$.

$\mathsf{CC\&Poly.Enc}_K(N, A, M)$	$\mathsf{KSGen}_K(N, l)$	$\mathsf{Tag}_K(N, A, C)$
$Z \leftarrow \mathsf{KSGen}_K(N, \lvert M\rvert)$	$m \leftarrow \lceil l/512 \rceil$	$s \parallel r \leftarrow \mathsf{lsb}_{256}(\mathsf{CC}_K(0, N))$ where $\lvert r\rvert = \lvert s\rvert = 128$
$C \leftarrow M \oplus Z$	**for** $i \leftarrow 1$ **to** m **do**	$l_1 \leftarrow 128\lceil \lvert A\rvert/128 \rceil$
$T \leftarrow \mathsf{Tag}_K(N, A, C)$	$\quad Z[i] \leftarrow \mathsf{CC}_K(i, N)$	$l_2 \leftarrow l_1 + 128\lceil \lvert C\rvert/128 \rceil$
return (C, T)	$l^* \leftarrow l \bmod 512$	$l_3 \leftarrow l_2 + 64$
	$Z[m] \leftarrow \mathsf{lsb}_{l^*}(Z[m])$	$Y \leftarrow A$
	$Z \leftarrow \sum_{i=1}^{m} Z[i] \cdot 2^{512(i-1)}$	$Y \leftarrow Y + C \cdot 2^{l_1}$
	return Z	$Y \leftarrow Y + \lceil \lvert A\rvert/8 \rceil \cdot 2^{l_2}$
		$Y \leftarrow Y + \lceil \lvert C\rvert/8 \rceil \cdot 2^{l_3}$
		$T \leftarrow \mathsf{Poly}_{r,s}(Y)$
		return T

Fig. 3. Pseudocode of the encryption algorithm of ChaCha20-Poly1305. The arithmetics are usual integer addition and multiplication.

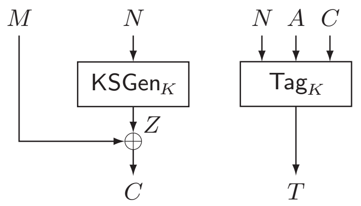

Fig. 4. Illustration of the encryption algorithm of ChaCha20-Poly1305

Proof. For any distinct $X, X' \in \mathcal{D}_\mathsf{H}$ and any $Y \in \{0,1\}^n$, $\varphi(X)$ and $\varphi(X')$ are distinct, and we have

$$\begin{aligned}\Pr[\mathsf{H}_L(X) \oplus \mathsf{H}_L(X') = Y] &= \Pr[\widetilde{\mathsf{H}}_L(\varphi(X)) \oplus c \oplus \widetilde{\mathsf{H}}_L(\varphi(X')) \oplus c = Y] \\ &= \Pr[\widetilde{\mathsf{H}}_L(\varphi(X)) \oplus \widetilde{\mathsf{H}}_L(\varphi(X')) = Y] \le \epsilon.\end{aligned}$$

Therefore, $\{\mathsf{H}_L\}$ is also ϵ-AXU. □

There exists an ϵ-AXU hash function $\widetilde{\mathsf{H}}_L$ such that $\widetilde{\mathsf{H}}_L(A, M) = 0^n$ for $(A, M) = (\epsilon, \epsilon)$, e.g., GHASH function in GCM and Poly1305, and we use the proposition to respect the non-empty plaintext convention of the schemes in [18].

We are now ready to present the details of our attacks.

4.1 AEAD-{2a, 4a, 4b} and DAEAD-2a Are Not INT-CTXT Secure

Attack against AEAD-2a. The hash function in AEAD-2a takes associated data A and a ciphertext C as input. Suppose that $\widetilde{\mathsf{H}}_L$ is an ϵ-AXU hash function such that $\widetilde{\mathsf{H}}_L \colon (\varepsilon, \varepsilon) \mapsto 0^n$. Let A_0 be any associated data and C_0 be any ciphertext such that $|C_0| = 1$, i.e., C_0 is a bit. We also assume that, given a hash key of length n bits, the adversary can compute the hash value for any input, e.g., Type-I hash functions like GHASH function. Define an injective function φ as follows.

$$\varphi(A, C) = \begin{cases} (\varepsilon, \varepsilon) & \text{if } (A, C) = (A_0, C_0) \\ (A_0, C_0) & \text{if } (A, C) = (\varepsilon, \varepsilon) \\ (A, C) & \text{otherwise} \end{cases}$$

Let $\mathsf{H}_L(A, C) = \widetilde{\mathsf{H}}_L(\varphi(A, C))$. Then H_L is an ϵ-AXU function from Proposition 1.

Now in the attack in Fig. 7, the adversary receives (C_1, T_1) for the first encryption query $(N_1, A_1, M_1) = (\mathsf{fStr}, A_0, M_0)$, where $|M_0| = 1$. We see that $\Pr[C_1 = C_0]$ is approximately 1/2. If $C_1 \neq C_0$, then the adversary fails to make a forgery. If $C_1 = C_0$, from $K' = \mathsf{msb}_n(\mathsf{SC}_K(\mathsf{fStr}))$, $\varphi(A_0, C_0) = (\varepsilon, \varepsilon)$, and $\widetilde{\mathsf{H}}_L(\varepsilon, \varepsilon) = 0^n$, the adversary receives K' as the tag. Suppose that $K' \neq \mathsf{fStr}$. For the second encryption query $(N_2, A_2, M_2) = (K', A_0, M_0)$, the adversary receives (C_2, T_2). $\Pr[C_2 = C_0]$ is approximately 1/2, and if $C_2 \neq C_0$, then the adversary fails to make a forgery. If $C_2 = C_0$, from $L = \mathsf{SC}_K(K')$ and

$\mathsf{DAEAD\text{-}1.Enc}_{K,L}(A,M)$	$\mathsf{DAEAD\text{-}2.Enc}_{K,K'}(A,M)$	$\mathsf{DAEAD\text{-}2a.Enc}_{K}(A,M)$
1. $V \leftarrow \mathsf{H}_L(A,M)$	1. $L \leftarrow \mathsf{SC}_K(K')$	1. $K' \leftarrow \mathsf{msb}_n(\mathsf{SC}_K(\mathsf{fStr}))$
2. $T \leftarrow \mathsf{msb}_n(\mathsf{SC}_K(V))$	2. $V \leftarrow \mathsf{H}_L(A,M)$	2. $L \leftarrow \mathsf{SC}_K(K')$
3. $Z \leftarrow \mathsf{SC}_K(T)$	3. $T \leftarrow \mathsf{msb}_n(\mathsf{SC}_K(V))$	3. $V \leftarrow \mathsf{H}_L(A,M)$
4. $C \leftarrow M \oplus Z$	4. $Z \leftarrow \mathsf{SC}_K(T)$	4. $T \leftarrow \mathsf{msb}_n(\mathsf{SC}_K(V))$
5. **return** (C,T)	5. $C \leftarrow M \oplus Z$	5. $Z \leftarrow \mathsf{SC}_K(T)$
	6. **return** (C,T)	6. $C \leftarrow M \oplus Z$
		7. **return** (C,T)

Fig. 5. Pseudocode of the encryption algorithms of DAEAD schemes [18]

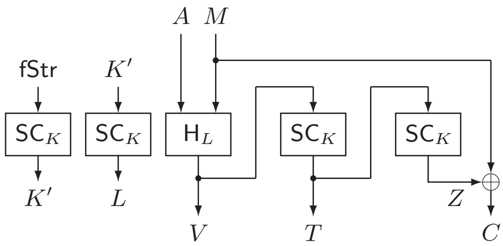

Fig. 6. Illustration the encryption algorithms of DAEAD schemes [18]. In DAEAD-2, $L = \mathsf{SC}_K(K')$. In DAEAD-2a, $L = \mathsf{SC}_K(\mathsf{msb}_n(\mathsf{SC}_K(\mathsf{fStr})))$.

$\mathsf{H}_L(A_0, C_0) = 0^n$, the adversary receives the first n bits of the hash key L, called L^*. If $K' = \mathsf{fStr}$, the hash key L^* is fStr. Therefore, the forgery (N^*, A^*, C^*, T^*), where $N^* = \mathsf{fStr}$, is accepted with probability approximately $1/4$.

Attack against AEAD-4a. In AEAD-4a, the hash function takes A and M as input. Let $\widetilde{\mathsf{H}}_L$ be an ϵ-AXU hash function such that $\widetilde{\mathsf{H}}_L\colon (\varepsilon,\varepsilon) \mapsto 0^n$. Given a hash key of length n bits, we assume that the adversary can compute the hash value for any input. Let A_0 be any associated data and M_0 be any non-empty plaintext that will be used in the attack. Define an injective function φ as follows.

$$\varphi(A,M) = \begin{cases} (\varepsilon,\varepsilon) & \text{if } (A,M) = (A_0,M_0) \\ (A_0,M_0) & \text{if } (A,M) = (\varepsilon,\varepsilon) \\ (A,M) & \text{otherwise} \end{cases} \tag{1}$$

Let $\mathsf{H}_L(A,M) = \widetilde{\mathsf{H}}_L(\varphi(A,M))$. Then H_L is an ϵ-AXU function from Proposition 1.

For the first encryption query $(N_1, A_1, M_1) = (\mathsf{fStr}, A_0, M_0)$, from $K' = \mathsf{msb}_n(\mathsf{SC}_K(\mathsf{fStr}))$, $\varphi(A_0, M_0) = (\varepsilon,\varepsilon)$, and $\widetilde{\mathsf{H}}_L(\varepsilon,\varepsilon) = 0^n$, the adversary receives K' as the tag. Suppose that $K' \neq \mathsf{fStr}$. For the second encryption query $(N_2, A_2, M_2) = (K', A_0, M_0)$, the adversary receives the first n bits of the hash key L, which we write L^*, and can compute the tag $T^* \leftarrow \mathsf{H}_{L^*}(A^*, M^*) \oplus K'$ without access to the encryption oracle. Here the length of M^* should be at most the length of M_0. If $\mathsf{fStr} = K'$, then the hash key L^* is fStr. Therefore the forgery (N^*, A^*, C^*, T^*), where $N^* = \mathsf{fStr}$ and $C^* \leftarrow M^* \oplus \mathsf{msb}_{|M^*|}(M_1 \oplus C_1)$, is accepted with probability 1.

Attack against AEAD-4b. The hash function in AEAD-4b takes A and M as input. Suppose that $\widetilde{\mathsf{H}}_L$ is an ϵ-AXU hash function such that $\widetilde{\mathsf{H}}_L\colon (\varepsilon,\varepsilon) \mapsto 0^n$. Let A_0 be any associated data and M_0 be any non-empty plaintext. Define an injective function φ as in (1). Let $\mathsf{H}_L(A,M) = \widetilde{\mathsf{H}}_L(\varphi(A,M))$. Then H_L is an ϵ-AXU function from Proposition 1. Let $N \in \{0,1\}^n$ be an arbitrary nonce.

For the encryption query $(N_1, A_1, M_1) = (N, A_0, M_0)$, since we define $R = \mathsf{msb}_n(\mathsf{SC}_K(N))$, $\varphi(A_0, M_0) = (\varepsilon,\varepsilon)$, and $\widetilde{\mathsf{H}}_L(\varepsilon,\varepsilon) = 0^n$, the adversary receives R as the tag. Observe that we have $(R, Z_0, L) = \mathsf{SC}_K(N)$. In the attack, we parse Z_0 as $Z_0 = Z^* \parallel L^*$, and use Z^* as the keystream and L^* as the hash key. Note that $Z_0 = M_0 \oplus C_0$ and hence the adversary can recover Z_0, and given L^*, the adversary can compute T^* for any (A^*, M^*). We remark that the length of L^* to compute Step 4 may depend on $|A^*|$ and $|M^*|$ if Type-II hash function is used, and the length of L^* can be arbitrarily long by using long

INT-CTXT attack against $\mathsf{AEAD\text{-}2a}$

1. $(C_1, T_1) \leftarrow \mathsf{AEAD\text{-}2a.Enc}_K(N_1, A_1, M_1)$ where $(N_1, A_1, M_1) \leftarrow (\mathsf{fStr}, A_0, M_0)$
2. **if** $C_1 = C_0$ **then**
3. $\quad K' \leftarrow T_1$
4. $\quad$ **if** $K' \neq \mathsf{fStr}$ **then**
5. $\qquad (C_2, T_2) \leftarrow \mathsf{AEAD\text{-}2a.Enc}_K(N_2, A_2, M_2)$ where $(N_2, A_2, M_2) \leftarrow (K', A_0, M_0)$
6. $\qquad$ **if** $C_2 = C_0$ **then**
7. $\qquad\quad L^* \leftarrow T_2$; $T^* \leftarrow \mathsf{H}_{L^*}(A^*, C^*) \oplus K'$
8. $\qquad\quad \top \leftarrow \mathsf{AEAD\text{-}2a.Ver}_K(N^*, A^*, C^*, T^*)$ where $N^* \leftarrow \mathsf{fStr}$
9. $\quad$ **else**
10. $\qquad L^* \leftarrow \mathsf{fStr}$; $T^* \leftarrow \mathsf{H}_{L^*}(A^*, C^*) \oplus \mathsf{fStr}$
11. $\qquad \top \leftarrow \mathsf{AEAD\text{-}2a.Ver}_K(N^*, A^*, C^*, T^*)$ where $N^* \leftarrow \mathsf{fStr}$

INT-CTXT attack against $\mathsf{AEAD\text{-}4a}$

1. $(C_1, K') \leftarrow \mathsf{AEAD\text{-}4a.Enc}_K(N_1, A_1, M_1)$ where $(N_1, A_1, M_1) \leftarrow (\mathsf{fStr}, A_0, M_0)$
2. **if** $K' \neq \mathsf{fStr}$ **then**
3. $\quad (C_2, L^*) \leftarrow \mathsf{AEAD\text{-}4a.Enc}_K(N_2, A_2, M_2)$ where $(N_2, A_2, M_2) \leftarrow (K', A_0, M_0)$
4. **else**
5. $\quad L^* \leftarrow K'$
6. $T^* \leftarrow \mathsf{H}_{L^*}(A^*, M^*) \oplus K'$ where $|M^*| \leq |M_0|$
7. $\top \leftarrow \mathsf{AEAD\text{-}4a.Ver}_K(N^*, A^*, M^* \oplus \mathsf{msb}_{|M^*|}(M_1 \oplus C_1), T^*)$ where $N^* \leftarrow \mathsf{fStr}$

INT-CTXT attack against $\mathsf{AEAD\text{-}4b}$

1. $(C_0, R) \leftarrow \mathsf{AEAD\text{-}4b.Enc}_K(N_1, A_1, M_1)$ where $(N_1, A_1, M_1) \leftarrow (N, A_0, M_0)$
2. $Z_0 \leftarrow M_0 \oplus C_0$
3. Parse Z_0 as $Z^* \parallel L^*$ where $|L^*|$ is the key length to compute Step 4
4. $T^* \leftarrow \mathsf{H}_{L^*}(A^*, M^*) \oplus R$
5. $\top \leftarrow \mathsf{AEAD\text{-}4b.Ver}_K(N^*, A^*, C^*, T^*)$ where $N^* \leftarrow N$ and $C^* \leftarrow M^* \oplus Z^*$

INT-CTXT attack against $\mathsf{DAEAD\text{-}2a}$

1. $(C_0, K') \leftarrow \mathsf{DAEAD\text{-}2a.Enc}_K(A_1, M_1)$ where $(A_1, M_1) \leftarrow (A_0, M_0)$
2. $L^* \leftarrow M_0 \oplus C_0$
3. Compute (A^*, M^*) such that $\mathsf{fStr} = \mathsf{H}_{L^*}(A^*, M^*)$ with L^*
4. $\top \leftarrow \mathsf{DAEAD\text{-}2a.Ver}_K(A^*, C^*, T^*)$ where $(C^*, T^*) \leftarrow (M^* \oplus \mathsf{msb}_{|M^*|}(L^*), K')$

Fig. 7. INT-CTXT attacks

M_0. For any $M^* \in \{0,1\}^{|Z^*|}$, the adversary can compute the tag T^*. Hence the forgery (N^*, A^*, C^*, T^*), where $N^* = N$ and $C^* = M^* \oplus Z^*$, is accepted with probability 1.

Attack against $\mathsf{DAEAD\text{-}2a}$. Suppose that $\widetilde{\mathsf{H}}_L$ is an ϵ-AXU hash function such that $\widetilde{\mathsf{H}}_L \colon (\varepsilon, \varepsilon) \mapsto 0^n$. We also assume that, given a hash key L^* and Y, we can compute some (A^*, M^*) such that $Y = \mathsf{H}_{L^*}(A^*, M^*)$. Let A_0 be arbitrary associated data and M_0 be a plaintext. Define an injective function φ as in (1). We have a restriction on $|M_0|$, which is discussed below. Let $\mathsf{H}_L(A, M) = \widetilde{\mathsf{H}}_L(\varphi(A, M)) \oplus \mathsf{fStr}$. Then H_L is ϵ-AXU from Proposition 1.

For the encryption query $(A_1, M_1) = (A_0, M_0)$, from $\varphi(A_0, M_0) = (\varepsilon, \varepsilon)$ and $\widetilde{\mathsf{H}}_L(\varepsilon, \varepsilon) = 0^n$, it follows that $\mathsf{H}_L(A_0, M_0) = \mathsf{fStr}$. From $K' = \mathsf{msb}_n(\mathsf{SC}_K(\mathsf{fStr}))$, the adversary receives K' as the tag. From $\mathsf{SC}_K(K') = L$ and $M_0 \oplus C_0$, it obtains the first $|M_0|$ bits of L. Let L^* be the value of $\mathsf{msb}_{|M_0|}(L)$. The length of L^* has to be long enough so that the adversary can compute (A^*, M^*) such that $\mathsf{fStr} = \mathsf{H}_{L^*}(A^*, M^*)$. Therefore, (A^*, C^*, T^*), where $(C^*, T^*) = (M^* \oplus \mathsf{msb}_{|M^*|}(L^*), K')$, is always accepted.

Comments. We note that, since the above schemes are not INT-CTXT secure, they are not INT-RUP secure, and these attacks contradict the claims in [18]. All the above attacks use the fixed point of the hash function. For example, given the security proof of $\mathsf{AEAD\text{-}2}$, it is tempting to claim that $\mathsf{AEAD\text{-}2a}$ is also secure. However, the dependence of the generation of K' and (R, Z) within the encryption algorithm allows the adversary to reproduce K' within encryption, and the fixed point of the hash function makes it

INT-RUP attack against $\mathsf{AEAD\text{-}2b}$

1. $(C,T) \leftarrow \mathsf{AEAD\text{-}2b.Enc}_K(N_1, A_1, M_1)$ where $(N_1, A_1, M_1) \leftarrow (N, A, M)$
2. Let c be an integer which is at least $|M|$ plus the hash key length of H
3. $Z' \leftarrow \mathsf{AEAD\text{-}2b.Dec}_K(N_1', A_1', C_1', T_1')$ where $(N_1', A_1', C_1', T_1') \leftarrow (N, A', 0^c, T')$
4. Parse Z' as $Z \parallel L$ where $|Z| = |M|$
5. $R \leftarrow \mathsf{H}_L(A, C) \oplus T$
6. Parse Z' as $Z^* \parallel L^*$ where $|Z^*| \le |Z|$
7. $T^* \leftarrow \mathsf{H}_{L^*}(A^*, C^*) \oplus R$ where $|C^*| = |Z^*|$
8. $\top \leftarrow \mathsf{AEAD\text{-}2b.Ver}_K(N^*, A^*, C^*, T^*)$ where $N^* \leftarrow N$

INT-RUP attack against $\mathsf{DAEAD}\text{-}\{1,2\}$

1. $M_1' \leftarrow \mathsf{DAEAD}\text{-}\{1,2\}.\mathsf{Dec}_K(A_1', C_1', T_1')$ where $(A_1', C_1', T_1') \leftarrow (A_1', C_1', V_0)$
2. $M_2' \leftarrow \mathsf{DAEAD}\text{-}\{1,2\}.\mathsf{Dec}_K(A_2', C_2', T_2')$ where $(A_2', C_2', T_2') \leftarrow (A_2', C_2', M_1' \oplus C_1')$
3. $\top \leftarrow \mathsf{DAEAD}\text{-}\{1,2\}.\mathsf{Ver}_K(A^*, C^*, T^*)$ where $(A^*, C^*, T^*) \leftarrow (A_0, M_0 \oplus M_2' \oplus C_2', T_2')$

Fig. 8. INT-RUP attacks

possible for the adversary to actually learn the value of K'. This type of discrepancy explains the success of the above attacks.

4.2 AEAD-2b and DAEAD-$\{1,2\}$ Are Not INT-RUP Secure

Attack against $\mathsf{AEAD\text{-}2b}$. Suppose that H_L is ϵ-AXU. The values $N \in \{0,1\}^n$, $A \in \{0,1\}^*$, and $M \in \{0,1\}^*$ can be arbitrarily chosen, where $|M| \neq 0$.

For the encryption query $(N_1, A_1, M_1) = (N, A, M)$, the adversary receives (C, T). For the decryption query $(N_1', A_1', C_1', T_1') = (N, A', 0^c, T')$, where $A' \in \{0,1\}^*$ and $T' \in \{0,1\}^n$ may be arbitrarily chosen, the adversary receives Z' as the plaintext. Z' can be parsed into the keystream Z and the hash key L. Then the adversary can compute $R = \mathsf{H}_L(A, C) \oplus T$ with L. We observe that Z' can also be parsed into another keystream Z^* and another hash key L^*. For any $A^* \in \{0,1\}^*$ and any $C^* \in \{0,1\}^{|Z^*|}$, the adversary can compute the tag as $T^* = \mathsf{H}_{L^*}(A^*, C^*) \oplus R$. Therefore, (N^*, A^*, C^*, T^*), where $N^* = N$, is accepted with probability 1. Note that this attack does not rely on the fixed point of the hash function.

Attacks against $\mathsf{DAEAD}\text{-}\{1,2\}$. Suppose that H_L is an ϵ-AXU hash function such that $\mathsf{H}_L(A_0, M_0) = V_0$ for any L, where $A_0 \in \{0,1\}^*$ and $M_0 \in \{0,1\}^n$ denote special input to produce the fixed point V_0. The values $A_1', A_2' \in \{0,1\}^*$ and $C_1', C_2' \in \{0,1\}^n$ can be arbitrarily chosen.

For the first decryption query $(A_1', C_1', T_1') = (A_1', C_1', V_0)$, the adversary receives the plaintext M_1'. Then the keystream is computed as $M_1' \oplus C_1'$. In fact, it is computed as the tag from $\mathsf{SC}_K(V_0)$. For the second decryption query $(A_2', C_2', T_2') = (A_2', C_2', M_1' \oplus C_1')$, the adversary receives the plaintext M_2'. For the verification query $(A^*, C^*, T^*) = (A_0, M_0 \oplus M_2' \oplus C_2', T_2')$, from $\mathsf{H}_L(A_0, M_0) = V_0$ and $T_2' = \mathsf{SC}_K(V_0)$, the forgery (A^*, C^*, T^*) is accepted with probability 1.

5 Positive Results

5.1 ChaCha20-Poly1305 Is INT-RUP Secure

Let $\mathcal{A}$ be an adversary. Suppose that $\mathcal{A}$ makes q encryption queries $(N_1, A_1, M_1), \ldots, (N_q, A_q, M_q)$, q' decryption queries $(N_1', A_1', C_1', T_1'), \ldots, (N_{q'}', A_{q'}', C_{q'}', T_{q'}')$, and q'' verification queries $(N_1'', A_1'', C_1'', T_1''), \ldots, (N_{q''}'', A_{q''}'', C_{q''}'', T_{q''}'')$. Define the maximum byte length of the message for the encryption queries and the verification queries as

$$16\left(\max_{\substack{1\le i\le q\\ 1\le j\le q''}}\left\{\left\lceil\frac{|A_i|}{128}\right\rceil + \left\lceil\frac{|M_i|}{128}\right\rceil\right\} \cup \left\{\left\lceil\frac{|A_j''|}{128}\right\rceil + \left\lceil\frac{|C_j''|}{128}\right\rceil\right\} + 1\right).$$

The security bound of ChaCha20-Poly1305 is given as follows. We note that we consider the case where CC_K is a random function and focus on the information theoretic case. However, it is standard to derive the corresponding complexity theoretic result. See for example [2].

Theorem 1. *Consider* CC&Poly*, where a random function* $F : \{0,1\}^{32} \times \{0,1\}^{128} \to \{0,1\}^{512}$ *is used as* CC_K*. Let* $\mathcal{A}$ *be an INT-RUP adversary that makes at most* q *encryption queries,* q' *decryption queries, and* q'' *verification queries, and the maximum byte length of the message for the encryption queries and the verification queries is at most* $\ell_{\max}$ *bytes. Then we have*

$$\mathbf{Adv}^{\text{int-rup}}_{\mathsf{CC\&Poly}}(\mathcal{A}) \leq q'' \frac{8\lceil \ell_{\max}/16 \rceil}{2^{106}}.$$

A proof is presented in Appendix B. We note that the INT-CTXT security was proved by Procter in [14][4]. The above theorem shows that the security does not change even if the adversary is given access to the decryption oracle. We see that the adversary learns the keystream $M_i \oplus C_i$ by making an encryption query (N_i, A_i, M_i). Intuitively, there is no additional information that the adversary can learn from the decryption oracle, since the decryption oracle simply allows the adversary to learn the keystream, which is already available from the encryption oracle.

5.2 AEAD-$\{1, 2, 3, 4\}$ Are INT-RUP Secure

The following theorem shows the security bounds of AEAD-$\{1, 2, 3, 4\}$. We focus on the information theoretic result, but the corresponding complexity theoretic result can be obtained in a standard way [2].

Theorem 2. *Let* $\mathrm{Rand}(n, \ell)$ *and* H *be the parameters of each AEAD scheme. Suppose that* $\{\mathsf{H}_L\}$ *is* ϵ*-AXU. Let* $\mathcal{A}$ *be an INT-RUP adversary that makes at most* q *encryption queries,* q' *decryption queries, and* q'' *verification queries. Then we have the following security bounds:*

$$\mathbf{Adv}^{\text{int-rup}}_{\mathsf{AEAD\text{-}1}[\mathrm{Rand}(n,\ell),\mathsf{H}]}(\mathcal{A}) \leq q''\epsilon, \tag{2}$$

$$\mathbf{Adv}^{\text{int-rup}}_{\mathsf{AEAD\text{-}3}[\mathrm{Rand}(n,\ell),\mathsf{H}]}(\mathcal{A}) \leq q''\epsilon, \tag{3}$$

$$\mathbf{Adv}^{\text{int-rup}}_{\mathsf{AEAD\text{-}2}[\mathrm{Rand}(n,\ell),\mathsf{H}]}(\mathcal{A}) \leq \frac{q + q' + q''}{2^n} + q''\epsilon, \text{ and} \tag{4}$$

$$\mathbf{Adv}^{\text{int-rup}}_{\mathsf{AEAD\text{-}4}[\mathrm{Rand}(n,\ell),\mathsf{H}]}(\mathcal{A}) \leq \frac{q + q' + q''}{2^n} + q''\epsilon. \tag{5}$$

A proof is presented in Appendix C.

6 Conclusions

In this paper, we analyzed the integrity of the authenticated encryption schemes that are based on stream ciphers and universal hash functions. Our attacks indicate that the use of fStr to reduce the number of secret keys requires careful handling in the security proof.

It would be interesting clarify the INT-RUP security of the remaining AEAD schemes shown as open in Table 1.

Acknowledgments. We thank the anonymous ProvSec 2016 reviewers and participants of Early Symmetric Crypto (ESC) 2015 for helpful comments. We also thank Palash Sarkar for insightful feedback on an earlier version of this paper. The work by Tetsu Iwata was supported in part by JSPS KAKENHI, Grant-in-Aid for Scientific Research (B), Grant Number 26280045, and was carried out in part while visiting Nanyang Technological University, Singapore.

[4] We remark that there is a minor gap in the proof in [14]. The proof introduces a hybrid (E^1, D^1) where the keystream is the output of a random function taking a nonce, and another hybrid (E^2, D^2) where the keystream is completely random for *both* encryption and decryption, and claims both hybrids are equivalent. This does not hold true in general since the keystream in a decryption query can be determined by an encryption query made before. However, as far as we see, the theorem statement stands.

References

1. Andreeva, E., Bogdanov, A., Luykx, A., Mennink, B., Mouha, N., Yasuda, K.: How to Securely Release Unverified Plaintext in Authenticated Encryption. In: Iwata, T., Sarkar, P. (eds.) ASIACRYPT 2014 (1). LNCS, vol. 8873, pp. 105–125. Springer (2014)
2. 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)
3. Bellare, M., Namprempre, C.: Authenticated Encryption: Relations among Notions and Analysis of the Generic Composition Paradigm. In: Okamoto, T. (ed.) ASIACRYPT 2000. LNCS, vol. 1976, pp. 531–545. Springer (2000)
4. Bellare, M., Rogaway, P.: Encode-Then-Encipher Encryption: How to Exploit Nonces or Redundancy in Plaintexts for Efficient Cryptography. In: Okamoto, T. (ed.) ASIACRYPT 2000. LNCS, vol. 1976, pp. 317–330. Springer (2000)
5. Bellare, M., Rogaway, P.: The Security of Triple Encryption and a Framework for Code-Based Game-Playing Proofs. In: Vaudenay, S. (ed.) EUROCRYPT 2006. LNCS, vol. 4004, pp. 409–426. Springer (2006)
6. Bellare, M., Rogaway, P., Wagner, D.: The EAX Mode of Operation. In: Roy, B.K., Meier, W. (eds.) FSE 2004. LNCS, vol. 3017, pp. 389–407. Springer (2004)
7. Bernstein, D.J.: The Poly1305-AES Message-Authentication Code. In: Gilbert, H., Handschuh, H. (eds.) FSE 2005. LNCS, vol. 3557, pp. 32–49. Springer (2005)
8. Bernstein, D.J.: ChaCha, a variant of Salsa20 (2008), `http://cr.yp.to/papers.html#chacha`, Document ID: 4027b5256e14b6796842e6d0f68b0b5e
9. Black, J., Halevi, S., Krawczyk, H., Krovetz, T., Rogaway, P.: UMAC: Fast and Secure Message Authentication. In: Wiener, M. (ed.) CRYPTO '99. LNCS, vol. 1666, pp. 216–233. Springer (1999)
10. Imamura, K., Minematsu, K., Iwata, T.: Integrity Analysis of Authenticated Encryption Based on Stream Ciphers. In: Chen, L., Han, J. (eds.) ProvSec 2016. LNCS, vol. 10005, pp. 257–276. Springer (2016)
11. McGrew, D.A., Viega, J.: The Security and Performance of the Galois/Counter Mode (GCM) of Operation. In: Canteaut, A., Viswanathan, K. (eds.) INDOCRYPT 2004. LNCS, vol. 3348, pp. 343–355. Springer (2004)
12. Namprempre, C., Rogaway, P., Shrimpton, T.: Reconsidering Generic Composition. In: Nguyen, P.Q., Oswald, E. (eds.) EUROCRYPT 2014. LNCS, vol. 8441, pp. 257–274. Springer (2014)
13. Nir, Y., Langley, A.: ChaCha20 and Poly1305 for IETF protocols. IRTF RFC 7539 (May 2015), `https://tools.ietf.org/html/rfc7539`
14. Procter, G.: A Security Analysis of the Composition of ChaCha20 and Poly1305. Cryptology ePrint Archive, Report 2014/613 (2014), `http://eprint.iacr.org/`
15. Rogaway, P.: Authenticated-Encryption with Associated-Data. In: Atluri, V. (ed.) ACM Conference on Computer and Communications Security. pp. 98–107. ACM (2002)
16. Rogaway, P.: Nonce-Based Symmetric Encryption. In: Roy, B.K., Meier, W. (eds.) FSE 2004. LNCS, vol. 3017, pp. 348–359. Springer (2004)
17. Rogaway, P., Shrimpton, T.: A Provable-Security Treatment of the Key-Wrap Problem. In: Vaudenay, S. (ed.) EUROCRYPT 2006. LNCS, vol. 4004, pp. 373–390. Springer (2006)
18. Sarkar, P.: Modes of operations for encryption and authentication using stream ciphers supporting an initialisation vector. Cryptography and Communications 6(3), 189–231 (2014)
19. Whiting, D., Housley, R., Ferguson, N.: Counter with CBC-MAC (CCM). Submission to NIST (2002), `http://csrc.nist.gov/`

A Definitions of Decryption and Verification Algorithms of AEAD and DAEAD Schemes

AEAD in [18]. Decryption algorithms of AEAD schemes are defined in Fig. 9. In each scheme, the mask R generated from the output of SC is never used, and the tag T, which is a part of the input, is not used. The associated data A is not used in $\mathsf{AEAD}\text{-}\{1, 2, 2a, 2b, 3, 4, 4a, 4b\}$.

Verification algorithms of AEAD schemes are defined in Fig. 10. Since the hash function takes input a plaintext in $\mathsf{AEAD}\text{-}\{3, 4, 4a, 4b, 7, 8, 8a\}$, we use both R and Z.

ChaCha20-Poly1305 [13]. The decryption and verification algorithms are defined in Fig. 11. The functions KsGen_K and Tag_K are defined in Fig. 3.

DAEAD in [18]. Decryption and verification algorithms of DAEAD schemes are defined in Fig. 12. The keystream Z is generated by using the tag T. The associated data A is never used for the decryption.

$\mathsf{AEAD\text{-}1.Dec}_{K,L}(N,A,C,T)$ 1. $R \parallel Z \leftarrow \mathsf{SC}_K(N)$ 2. $M \leftarrow C \oplus Z$ 3. **return** M	$\mathsf{AEAD\text{-}2.Dec}_{K,K'}(N,A,C,T)$ 1. $R \parallel Z \leftarrow \mathsf{SC}_K(N)$ 2. $M \leftarrow C \oplus Z$ 3. **return** M
$\mathsf{AEAD\text{-}2a.Dec}_{K}(N,A,C,T)$ 1. $R \parallel Z \leftarrow \mathsf{SC}_K(N)$ 2. $M \leftarrow C \oplus Z$ 3. **return** M	$\mathsf{AEAD\text{-}2b.Dec}_{K}(N,A,C,T)$ 1. $R \parallel S \leftarrow \mathsf{SC}_K(N)$ 2. Parse S as $Z \parallel L$ where $\lvert Z\rvert = \lvert C\rvert$ 3. $M \leftarrow C \oplus Z$ 4. **return** M
$\mathsf{AEAD\text{-}3.Dec}_{K,L}(N,A,C,T)$ 1. $R \parallel Z \leftarrow \mathsf{SC}_K(N)$ 2. $C \leftarrow M \oplus Z$ 3. **return** M	$\mathsf{AEAD\text{-}4.Dec}_{K,K'}(N,A,C,T)$ 1. $R \parallel Z \leftarrow \mathsf{SC}_K(N)$ 2. $M \leftarrow C \oplus Z$ 3. **return** M
$\mathsf{AEAD\text{-}4a.Dec}_{K}(N,A,C,T)$ 1. $R \parallel Z \leftarrow \mathsf{SC}_K(N)$ 2. $M \leftarrow C \oplus Z$ 3. **return** M	$\mathsf{AEAD\text{-}4b.Dec}_{K}(N,A,C,T)$ 1. $R \parallel S \leftarrow \mathsf{SC}_K(N)$ 2. Parse S as $Z \parallel L$ where $\lvert Z\rvert = \lvert C\rvert$ 3. $M \leftarrow C \oplus Z$ 4. **return** M
$\mathsf{AEAD\text{-}5.Dec}_{K,L}(N,A,C,T)$ 1. $V \leftarrow \mathsf{H}_L(A,N)$ 2. $R \parallel Z \leftarrow \mathsf{SC}_K(V)$ 3. $M \leftarrow C \oplus Z$ 4. **return** M	$\mathsf{AEAD\text{-}6.Dec}_{K,K'}(N,A,C,T)$ 1. $L_1 \parallel L_2 \leftarrow \mathsf{SC}_K(K')$ 2. $V \leftarrow \mathsf{H}_{L_1}(A,N)$ 3. $R \parallel Z \leftarrow \mathsf{SC}_K(V)$ 4. $M \leftarrow C \oplus Z$ 5. **return** M
$\mathsf{AEAD\text{-}6a.Dec}_{K}(N,A,C,T)$ 1. $K' \leftarrow \mathsf{msb}_n(\mathsf{SC}_K(\mathsf{fStr}))$ 2. $L_1 \parallel L_2 \leftarrow \mathsf{SC}_K(K')$ 3. $V \leftarrow \mathsf{H}_{L_1}(A,N)$ 4. $R \parallel Z \leftarrow \mathsf{SC}_K(V)$ 5. $M \leftarrow C \oplus Z$ 6. **return** M	$\mathsf{AEAD\text{-}7.Dec}_{K,L}(N,A,C,T)$ 1. $V \leftarrow \mathsf{H}_L(A,N)$ 2. $R \parallel Z \leftarrow \mathsf{SC}_K(V)$ 3. $M \leftarrow C \oplus Z$ 4. **return** M
$\mathsf{AEAD\text{-}8.Dec}_{K,K'}(N,A,C,T)$ 1. $L_1 \parallel L_2 \leftarrow \mathsf{SC}_K(K')$ 2. $V \leftarrow \mathsf{H}_{L_1}(A,N)$ 3. $R \parallel Z \leftarrow \mathsf{SC}_K(V)$ 4. $M \leftarrow C \oplus Z$ 5. **return** M	$\mathsf{AEAD\text{-}8a.Dec}_{K}(N,A,C,T)$ 1. $K' \leftarrow \mathsf{msb}_n(\mathsf{SC}_K(\mathsf{fStr}))$ 2. $L_1 \parallel L_2 \leftarrow \mathsf{SC}_K(K')$ 3. $V \leftarrow \mathsf{H}_{L_1}(A,N)$ 4. $R \parallel Z \leftarrow \mathsf{SC}_K(V)$ 5. $M \leftarrow C \oplus Z$ 6. **return** M

Fig. 9. Pseudocode of the decryption algorithms of AEAD schemes [18]

$\mathsf{AEAD\text{-}1.Ver}_{K,L}(N, A, C, T)$

1. $R \parallel Z \leftarrow \mathsf{SC}_K(N)$
2. $T^* \leftarrow \mathsf{H}_L(A, C) \oplus R$
3. **if** $T^* = T$ **then return** $\top$
4. **return** $\perp$

$\mathsf{AEAD\text{-}2.Ver}_{K,K'}(N, A, C, T)$

1. $L \leftarrow \mathsf{SC}_K(K')$
2. $R \parallel Z \leftarrow \mathsf{SC}_K(N)$
3. $T^* \leftarrow \mathsf{H}_L(A, C) \oplus R$
4. **if** $T^* = T$ **then return** $\top$
5. **return** $\perp$

$\mathsf{AEAD\text{-}2a.Ver}_{K}(N, A, C, T)$

1. $K' \leftarrow \mathsf{msb}_n(\mathsf{SC}_K(\mathsf{fStr}))$
2. $L \leftarrow \mathsf{SC}_K(K')$
3. $R \parallel Z \leftarrow \mathsf{SC}_K(N)$
4. $T^* \leftarrow \mathsf{H}_L(A, C) \oplus R$
5. **if** $T^* = T$ **then return** $\top$
6. **return** $\perp$

$\mathsf{AEAD\text{-}2b.Ver}_{K}(N, A, C, T)$

1. $R \parallel S \leftarrow \mathsf{SC}_K(N)$
2. Parse S as $Z \parallel L$ where $|Z| = |C|$
3. $T^* \leftarrow \mathsf{H}_L(A, C) \oplus R$
4. **if** $T^* = T$ **then return** $\top$
5. **return** $\perp$

$\mathsf{AEAD\text{-}3.Ver}_{K,L}(N, A, C, T)$

1. $R \parallel Z \leftarrow \mathsf{SC}_K(N)$
2. $M \leftarrow C \oplus Z$
3. $T \leftarrow \mathsf{H}_L(A, M) \oplus R$
4. **if** $T^* = T$ **then return** $\top$
5. **return** $\perp$

$\mathsf{AEAD\text{-}4.Ver}_{K,K'}(N, A, C, T)$

1. $L \leftarrow \mathsf{SC}_K(K')$
2. $R \parallel Z \leftarrow \mathsf{SC}_K(N)$
3. $M \leftarrow C \oplus Z$
4. $T^* \leftarrow \mathsf{H}_L(A, M) \oplus R$
5. **if** $T^* = T$ **then return** $\top$
6. **return** $\perp$

$\mathsf{AEAD\text{-}4a.Ver}_{K}(N, A, C, T)$

1. $K' \leftarrow \mathsf{msb}_n(\mathsf{SC}_K(\mathsf{fStr}))$
2. $L \leftarrow \mathsf{SC}_K(K')$
3. $R \parallel Z \leftarrow \mathsf{SC}_K(N)$
4. $M \leftarrow C \oplus Z$
5. $T^* \leftarrow \mathsf{H}_L(A, M) \oplus R$
6. **if** $T^* = T$ **then return** $\top$
7. **return** $\perp$

$\mathsf{AEAD\text{-}4b.Ver}_{K}(N, A, C, T)$

1. $R \parallel S \leftarrow \mathsf{SC}_K(N)$
2. Parse S as $Z \parallel L$ where $|Z| = |C|$
3. $M \leftarrow C \oplus Z$
4. $T^* \leftarrow \mathsf{H}_L(A, M) \oplus R$
5. **if** $T^* = T$ **then return** $\top$
6. **return** $\perp$

$\mathsf{AEAD\text{-}5.Ver}_{K,L}(N, A, C, T)$

1. $V \leftarrow \mathsf{H}_L(A, N)$
2. $R \parallel Z \leftarrow \mathsf{SC}_K(V)$
3. $T^* \leftarrow \mathsf{H}_L(C) \oplus R$
4. **if** $T^* = T$ **then return** $\top$
5. **return** $\perp$

$\mathsf{AEAD\text{-}6.Ver}_{K,K'}(N, A, C, T)$

1. $L_1 \parallel L_2 \leftarrow \mathsf{SC}_K(K')$
2. $V \leftarrow \mathsf{H}_{L_1}(A, N)$
3. $R \parallel Z \leftarrow \mathsf{SC}_K(V)$
4. $T^* \leftarrow \mathsf{H}_{L_2}(C) \oplus R$
5. **if** $T^* = T$ **then return** $\top$
6. **return** $\perp$

$\mathsf{AEAD\text{-}6a.Ver}_{K}(N, A, C, T)$

1. $K' \leftarrow \mathsf{msb}_n(\mathsf{SC}_K(\mathsf{fStr}))$
2. $L_1 \parallel L_2 \leftarrow \mathsf{SC}_K(K')$
3. $V \leftarrow \mathsf{H}_{L_1}(A, N)$
4. $R \parallel Z \leftarrow \mathsf{SC}_K(V)$
5. $T^* \leftarrow \mathsf{H}_{L_2}(C) \oplus R$
6. **if** $T^* = T$ **then return** $\top$
7. **return** $\perp$

$\mathsf{AEAD\text{-}7.Ver}_{K,L}(N, A, C, T)$

1. $V \leftarrow \mathsf{H}_L(A, N)$
2. $R \parallel Z \leftarrow \mathsf{SC}_K(V)$
3. $M \leftarrow C \oplus Z$
4. $T^* \leftarrow \mathsf{H}_L(M) \oplus R$
5. **if** $T^* = T$ **then return** $\top$
6. **return** $\perp$

$\mathsf{AEAD\text{-}8.Ver}_{K,K'}(N, A, C, T)$

1. $L_1 \parallel L_2 \leftarrow \mathsf{SC}_K(K')$
2. $V \leftarrow \mathsf{H}_{L_1}(A, N)$
3. $R \parallel Z \leftarrow \mathsf{SC}_K(V)$
4. $M \leftarrow C \oplus Z$
5. $T^* \leftarrow \mathsf{H}_{L_2}(M) \oplus R$
6. **if** $T^* = T$ **then return** $\top$
7. **return** $\perp$

$\mathsf{AEAD\text{-}8a.Ver}_{K}(N, A, C, T)$

1. $K' \leftarrow \mathsf{msb}_n(\mathsf{SC}_K(\mathsf{fStr}))$
2. $L_1 \parallel L_2 \leftarrow \mathsf{SC}_K(K')$
3. $V \leftarrow \mathsf{H}_{L_1}(A, N)$
4. $R \parallel Z \leftarrow \mathsf{SC}_K(V)$
5. $M \leftarrow C \oplus Z$
6. $T^* \leftarrow \mathsf{H}_{L_2}(M) \oplus R$
7. **if** $T^* = T$ **then return** $\top$
8. **return** $\perp$

Fig. 10. Pseudocode of the verification algorithms of AEAD schemes [18]

$\mathsf{CC\&Poly.Dec}_K(N, A, C, T)$	$\mathsf{CC\&Poly.Ver}_K(N, A, C, T)$
1. $Z \leftarrow \mathsf{KSGen}_K(N, \|C\|)$ 2. $M \leftarrow C \oplus Z$ 3. **return** M	1. $T^* \leftarrow \mathsf{Tag}_K(N, A, C)$ 2. **if** $T^* = T$ **then return** $\top$ 3. **return** $\perp$

Fig. 11. Pseudocode of the decryption and verification algorithms of ChaCha20-Poly1305 [13]

$\mathsf{DAEAD\text{-}1.Dec}_{K,L}(A, C, T)$	$\mathsf{DAEAD\text{-}2.Dec}_{K,K'}(A, C, T)$	$\mathsf{DAEAD\text{-}2a.Dec}_K(A, C, T)$
1. $Z \leftarrow \mathsf{SC}_K(T)$ 2. $M \leftarrow C \oplus Z$ 3. **return** M	1. $Z \leftarrow \mathsf{SC}_K(T)$ 2. $M \leftarrow C \oplus Z$ 3. **return** M	1. $Z \leftarrow \mathsf{SC}_K(T)$ 2. $M \leftarrow C \oplus Z$ 3. **return** M
$\mathsf{DAEAD\text{-}1.Ver}_{K,L}(A, C, T)$	$\mathsf{DAEAD\text{-}2.Ver}_{K,K'}(A, C, T)$	$\mathsf{DAEAD\text{-}2a.Ver}_K(A, C, T)$
1. $Z \leftarrow \mathsf{SC}_K(T)$ 2. $M \leftarrow C \oplus Z$ 3. $V \leftarrow \mathsf{H}_L(A, M)$ 4. $T^* \leftarrow \mathsf{msb}_n(\mathsf{SC}_K(V))$ 5. **if** $T^* = T$ **then return** $\top$ 6. **return** $\perp$	1. $L \leftarrow \mathsf{SC}_K(K')$ 2. $Z \leftarrow \mathsf{SC}_K(T)$ 3. $M \leftarrow C \oplus Z$ 4. $V \leftarrow \mathsf{H}_L(A, M)$ 5. $T^* \leftarrow \mathsf{msb}_n(\mathsf{SC}_K(V))$ 6. **if** $T^* = T$ **then return** $\top$ 7. **return** $\perp$	1. $K' \leftarrow \mathsf{msb}_n(\mathsf{SC}_K(\mathsf{fStr}))$ 2. $L \leftarrow \mathsf{SC}_K(K')$ 3. $Z \leftarrow \mathsf{SC}_K(T)$ 4. $M \leftarrow C \oplus Z$ 5. $V \leftarrow \mathsf{H}_L(A, M)$ 6. $T^* \leftarrow \mathsf{msb}_n(\mathsf{SC}_K(V))$ 7. **if** $T^* = T$ **then return** $\top$ 8. **return** $\perp$

Fig. 12. Pseudocode of the decryption and verification algorithms of DAEAD schemes [18]

B Proof of Theorem 1

We evaluate $\mathbf{Adv}^{\text{int-rup}}_{\mathsf{CC\&Poly}}(\mathcal{A})$ following the game playing proof technique in [5]. Without loss of generality, we assume that $\mathcal{A}$ is deterministic and makes exactly q encryption queries, q' decryption queries, and q'' verification queries. Let (N_i, A_i, M_i) for $i = 1, \ldots, q$, $(N'_{i'}, A'_{i'}, C'_{i'}, T'_{i'})$ for $i' = 1, \ldots, q'$, and $(N''_j, A''_j, C''_j, T''_j)$ for $j = 1, \ldots, q''$ denote the queries. The internal variables are written analogously.

We define Game G_0 in Fig. 13. In Fig. 13, Game G_0 simulates the real oracles of ChaCha20-Poly1305 based on the random function F. Then we have

$$\mathbf{Adv}^{\text{int-rup}}_{\mathsf{CC\&Poly}}(\mathcal{A}) = \Pr[\mathcal{A}^{G_0} \text{ sets } \mathsf{forge}].$$

We next define Game G_1 in Fig. 14. Game G_1 simulates the oracles using the lazy sampling of F, where F is regarded as an array, and the array $F(X, Y)$ is initially undefined for all $(X, Y) \in \{0,1\}^{32} \times \{0,1\}^{96}$. Now since the function F produces the random values and the values are perfectly indistinguishable between Game G_0 and Game G_1, these games are identical. Hence

$$\Pr[\mathcal{A}^{G_0} \text{ sets } \mathsf{forge}] = \Pr[\mathcal{A}^{G_1} \text{ sets } \mathsf{forge}]. \tag{6}$$

We consider $\Pr[\mathcal{A}^{G_1} \text{ sets } \mathsf{forge}]$. In Fig. 14, the authentication keys in verification queries are generated independently of the keystreams in decryption queries, and hence there are two cases to consider. We denote the polynomial hash function in Poly1305 [7] by H_r. If for the j-th verification query, it holds that $N''_j \neq N_i$ for all i, then (r''_j, s''_j) is uniformly distributed and independent of (r_i, s_i). Hence

$$\Pr[T^*_j = T''_j] = \Pr[H_{r''_j}(Y''_j) + s''_j \bmod 2^{128} = T''_j] = \frac{1}{2^{128}}.$$

Suppose that for the j-th verification query, we have $N''_j = N_i$ for some i. Then it follows that $(r''_j, s''_j) = (r_i, s_i)$. The event $T^*_j = T''_j$ is equivalent to

$$H_{r_i}(Y''_j) - H_{r_i}(Y_i) \bmod 2^{128} = T''_j - T_i \bmod 2^{128}. \tag{7}$$

Now if $(A''_j, C''_j) = (A_i, C_i)$, then we necessarily have $T''_j \neq T_i$ and hence (7) cannot hold. Therefore let $(A''_j, C''_j) \neq (A_i, C_i)$. Then, since H_r is ϵ-AΔU [7, Sect. 3], meaning that it has a small differential probability with respect to modulo 2^{128}, we have

$$\Pr[T^*_j = T''_j] = \Pr[H_{r_i}(Y''_j) - H_{r_i}(Y_i) \bmod 2^{128} = T''_j - T_i \bmod 2^{128}] \leq \epsilon.$$

Game G_0
Initialize

$\mathsf{forge} \leftarrow \mathsf{false}$; $F \xleftarrow{\$} \{f \mid f \colon \{0,1\}^{32} \times \{0,1\}^{96} \to \{0,1\}^{512}\}$

Oracle Encrypt(N, A, M)

$Z \leftarrow$ **KSGen**$(N, |M|)$
$C \leftarrow M \oplus Z$
$T \leftarrow$ **Tag**(N, A, C)
return (C, T)

Oracle Decrypt(N, A, C, T)

$Z \leftarrow$ **KSGen**$(N, |C|)$
$M \leftarrow C \oplus Z$
return M

Oracle Verify(N, A, C, T)

$T^* \leftarrow$ **Tag**(N, A, C)
if $T^* = T$ **then** $\mathsf{forge} \leftarrow \mathsf{true}$; **return** $\top$
return $\perp$

Subroutine KSGen(N, l)

$m \leftarrow \lceil l/512 \rceil$
for $i \leftarrow 1$ **to** m **do**
    $Z[i] \leftarrow F(i, N)$
$Z[m] \leftarrow \mathsf{lsb}_{l \bmod 512}(Z[m])$
$Z \leftarrow \sum_{i=1}^{m} Z[i] \cdot 2^{512(i-1)}$
return Z

Subroutine Tag(N, A, C)

$s \parallel r \leftarrow \mathsf{lsb}_{256}(F(0, N))$ where $|r| = |s| = 128$
$l_1 \leftarrow 128\lceil |A|/128 \rceil$
$l_2 \leftarrow l_1 + 128\lceil |C|/128 \rceil$
$l_3 \leftarrow l_2 + 64$
$Y \leftarrow A$
$Y \leftarrow Y + C \cdot 2^{l_1}$
$Y \leftarrow Y + \lceil |A|/8 \rceil \cdot 2^{l_2}$
$Y \leftarrow Y + \lceil |C|/8 \rceil \cdot 2^{l_3}$
$T \leftarrow \mathsf{Poly}_{r,s}(Y)$
return T

Fig. 13. Game G_0 for the proof of Theorem 1

Game G_1

Initialize

$\mathsf{forge} \leftarrow \mathsf{false}$; $\mathcal{N} \leftarrow \emptyset$

Oracle Encrypt(N, A, M)

$Z \leftarrow \mathbf{KSGen2}(N, |M|)$
$C \leftarrow M \oplus Z$
$T \leftarrow \mathbf{Tag}(N, A, C)$
return (C, T)

Oracle Decrypt(N, A, C, T)

$Z \leftarrow \mathbf{KSGen2}(N, |C|)$
$M \leftarrow C \oplus Z$
return M

Oracle Verify(N, A, C, T)

$T^* \leftarrow \mathbf{Tag2}(N, A, C)$
if $T^* = T$ **then** $\mathsf{forge} \leftarrow \mathsf{true}$; **return** $\top$
return $\perp$

Subroutine KSGen2(N, l)

$m \leftarrow \lceil l/512 \rceil$
for $i \leftarrow 1$ **to** m **do**
  $Z[i] \xleftarrow{\$} \{0,1\}^{512}$
  if $(i, N) \in \mathcal{N}$ **then** $Z[i] \leftarrow F(i, N)$
  else $\mathcal{N} \leftarrow \mathcal{N} \cup \{(i, N)\}$
  $F(i, N) \leftarrow Z[i]$
$Z[m] \leftarrow \mathsf{lsb}_{l \bmod 512}(Z[m])$
$Z \leftarrow \sum_{i=1}^{m} Z[i] \cdot 2^{512(i-1)}$
return Z

Subroutine Tag2(N, A, C)

$U \parallel s \parallel r \xleftarrow{\$} \{0,1\}^{512}$ where $|r| = |s| = 128$
if $(0, N) \in \mathcal{N}$ **then** $U \parallel s \parallel r \leftarrow F(0, N)$
else $\mathcal{N} \leftarrow \mathcal{N} \cup \{(0, N)\}$
$F(0, N) \leftarrow U \parallel s \parallel r$
$l_1 \leftarrow 128 \lceil |A|/128 \rceil$
$l_2 \leftarrow l_1 + 128 \lceil |C|/128 \rceil$
$l_3 \leftarrow l_2 + 64$
$Y \leftarrow A$
$Y \leftarrow Y + C \cdot 2^{l_1}$
$Y \leftarrow Y + \lceil |A|/8 \rceil \cdot 2^{l_2}$
$Y \leftarrow Y + \lceil |C|/8 \rceil \cdot 2^{l_3}$
$T \leftarrow \mathsf{Poly}_{r,s}(Y)$
return T

Fig. 14. Game G_1. Keystreams and authentication keys are generated at random.

Therefore, for each $j = 1, \dots, q''$, we have $\Pr[T^*_j = T''_j] \le \epsilon$. Following [7, Sect. 3], $\epsilon = (8\lceil \ell_{\max}/16 \rceil)/2^{106}$. Hence we have

$$\Pr[\mathcal{A}^{G_1} \text{ sets } \mathsf{forge}] \le q'' \frac{8\lceil \ell_{\max}/16 \rceil}{2^{106}}. \tag{8}$$

The claimed bound is obtained from (6) and (8). □

C Proof of Theorem 2

C.1 Proof of (4)

We evaluate $\mathbf{Adv}^{\text{int-rup}}_{\mathsf{AEAD\text{-}2}[\mathrm{Rand}(n,\ell),\mathsf{H}]}(\mathcal{A})$ following [5], where $\mathsf{AEAD\text{-}2}[\mathrm{Rand}(n,\ell),\mathsf{H}]$ is $\mathsf{AEAD\text{-}2}$ that is based on a random function $F \xleftarrow{\$} \mathrm{Rand}(n,\ell)$, and $\mathcal{A}$ is a deterministic adversary that makes exactly q encryption queries, q' decryption queries, and q'' verification queries.

We define two games, Game G_0 and Game G_1, in Fig. 15, where Game G_1 includes the boxed statements and Game G_0 does not. Game G_1 simulates the $\mathsf{AEAD\text{-}2}$ encryption oracle in lines 3–10, the decryption oracle in lines 11–17, and the verification oracle in lines 18–25 using the lazy sampling of F. We initialize the two flags, bad and forge, to false. We let $\mathcal{N}$ be the set of the input values of F that have already been defined.

Game G_0 is obtained from Game G_1 by removing the boxed statements, and we see that Game G_0 and Game G_1 are identical until one of the flags gets set. Therefore, from the fundamental lemma of game playing, we have

$$\mathbf{Adv}^{\text{int-rup}}_{\mathsf{AEAD\text{-}2}[\mathrm{Rand}(n,\ell),\mathsf{H}]}(\mathcal{A}) \le \Pr[\mathcal{A}^{G_0} \text{ sets } \mathsf{bad} \text{ or } \mathsf{forge}].$$

We consider $\Pr[\mathcal{A}^{G_0} \text{ sets } \mathsf{bad} \text{ or } \mathsf{forge}]$. We write the q encryption queries as (N_i, A_i, M_i), q' decryption queries as $(N'_{i'}, A'_{i'}, C'_{i'}, T'_{i'})$, and q'' verification queries as $(N''_j, A''_j, C''_j, T''_j)$. Observe that Game G_0 always returns a random string of $|M_i| + n$ bits for the i-th encryption query based on the randomness R_i and Z_i chosen for this query, or based on $Z'_{i'}$ and $R'_{i'}$ chosen in the prior decryption query. We also see that it returns a random string of $|C'_{i'}|$ bits for the i'-th decryption query based on the randomness $Z'_{i'}$ chosen for this query, or based on Z_i in the prior encryption query. The verification oracle always returns $\perp$. From these observations, we may fix the queries and focus on the non-adaptive strategy.

We first consider $\Pr[\mathcal{A}^{G_0} \text{ sets } \mathsf{bad}]$. The flag bad means that the secret key K' is equal to one of the nonces N_i, $N'_{i'}$, or N''_j, where $i = 1, \dots, q$, $i' = 1, \dots, q'$, and $j = 1, \dots, q''$. Hence we have

$$\Pr[\mathcal{A}^{G_0} \text{ sets } \mathsf{bad}] \le \frac{q + q' + q''}{2^n}. \tag{9}$$

Next, we consider $\Pr[\mathcal{A}^{G_0} \text{ sets } \mathsf{forge}]$. The flag forge means that a tag computed in the **Oracle Verify** is equal to a tag queried for the **Oracle Verify**. There are two cases to consider. If for the j-th verification query, $N''_j \ne N_i$ for all i, the value R''_j is uniformly distributed and independent of (R_i, Z_i) for all i. We note that whether $N''_j = N'_h$ for some h holds or not does not matter, since this only reveals Z''_j from the result of h-th decryption query. The value R''_j is also independent of the hash key L. Hence

$$\Pr[T^*_j = T''_j] = \Pr[R''_j = \mathsf{H}_L(A''_j, C''_j) \oplus T''_j] = \frac{1}{2^n}.$$

Suppose that for the j-th verification query, we have $N''_j = N_i$ for some i. Then the value (R''_j, Z''_j) is independent of (R_k, Z_k) for all $i \ne k$, and $R''_j = R_i$. Hence the event $T^*_j = T''_j$ is equivalent to

$$\mathsf{H}_L(A''_j, C''_j) \oplus \mathsf{H}_L(A_i, C_i) = T''_j \oplus T_i. \tag{10}$$

From the restriction on the adversary, we have $(N''_j, A''_j, C''_j, T''_j) \ne (N_i, A_i, C_i, T_i)$, and this implies $(A''_j, C''_j, T''_j) \ne (A_i, C_i, T_i)$. If $(A''_j, C''_j) = (A_i, C_i)$, then we have $T''_j \ne T_i$ and (10) cannot hold. If $(A''_j, C''_j) \ne (A_i, C_i)$, then

$$\Pr[T^*_j = T''_j] = \Pr[\mathsf{H}_L(A''_j, C''_j) \oplus \mathsf{H}_L(A_i, C_i) = T''_j \oplus T_i] \le \epsilon.$$

Therefore, for each $j = 1, \dots, q''$, we have $\Pr[T^*_j = T''_j] \le \epsilon$, and hence we obtain

$$\Pr[\mathcal{A}^{G_0} \text{ sets } \mathsf{forge}] \le q''\epsilon. \tag{11}$$

The claimed bound (4) is obtained from (9) and (11). □

Game G_0, $\boxed{\textbf{Game } G_1}$
Initialize

$\mathsf{bad} \leftarrow \mathsf{forge} \leftarrow \mathsf{false}$; $\mathcal{N} \leftarrow \emptyset$
$K' \xleftarrow{\$} \{0,1\}^n$; $L \xleftarrow{\$} \{0,1\}^\ell$; $F(K') \leftarrow L$

Oracle Encrypt(N, A, M)

$R \xleftarrow{\$} \{0,1\}^n$; $Z \xleftarrow{\$} \{0,1\}^{\ell-n}$
if $K' = N$ **then** $\mathsf{bad} \leftarrow \mathsf{true}$; $\boxed{R \,\|\, Z \leftarrow F(N)}$
else if $N \in \mathcal{N}$ **then** $R \,\|\, Z \leftarrow F(N)$
else $\mathcal{N} \leftarrow \mathcal{N} \cup \{N\}$
$F(N) \leftarrow R \,\|\, Z$
$C \leftarrow M \oplus \mathsf{msb}_{|M|}(Z)$
$T \leftarrow \mathsf{H}_L(A, C) \oplus R$
return (C, T)

Oracle Decrypt(N, A, C, T)

$R \xleftarrow{\$} \{0,1\}^n$; $Z \xleftarrow{\$} \{0,1\}^{\ell-n}$
if $K' = N$ **then** $\mathsf{bad} \leftarrow \mathsf{true}$; $\boxed{R \,\|\, Z \leftarrow F(N)}$
else if $N \in \mathcal{N}$ **then** $R \,\|\, Z \leftarrow F(N)$
else $\mathcal{N} \leftarrow \mathcal{N} \cup \{N\}$
$F(N) \leftarrow R \,\|\, Z$
$M \leftarrow C \oplus \mathsf{msb}_{|C|}(Z)$
return M

Oracle Verify(N, A, C, T)

$R \xleftarrow{\$} \{0,1\}^n$; $Z \xleftarrow{\$} \{0,1\}^{\ell-n}$
if $K' = N$ **then** $\mathsf{bad} \leftarrow \mathsf{true}$; $\boxed{R \,\|\, Z \leftarrow F(N)}$
else if $N \in \mathcal{N}$ **then** $R \,\|\, Z \leftarrow F(N)$
else $\mathcal{N} \leftarrow \mathcal{N} \cup \{N\}$
$F(N) \leftarrow R \,\|\, Z$
$T^* \leftarrow \mathsf{H}_L(A, C) \oplus R$
if $T^* = T$ **then** $\mathsf{forge} \leftarrow \mathsf{true}$; $\boxed{\textbf{return } \top}$
return $\bot$

Fig. 15. Game G_0 and G_1 for the proof of (4) in Theorem 2

Game G_0, $\boxed{\textbf{Game } G_1}$
Initialize

1. $\mathsf{forge} \leftarrow \mathsf{false}$; $\mathcal{N} \leftarrow \emptyset$
2. $L \xleftarrow{\$} \mathcal{L}$

Oracle Encrypt(N, A, M)

3. $R \xleftarrow{\$} \{0,1\}^n$; $Z \xleftarrow{\$} \{0,1\}^{\ell-n}$
4. **if** $N \in \mathcal{N}$ **then** $R \,\|\, Z \leftarrow F(N)$
5. **else** $\mathcal{N} \leftarrow \mathcal{N} \cup \{N\}$
6. $F(N) \leftarrow R \,\|\, Z$
7. $C \leftarrow M \oplus \mathsf{msb}_{|M|}(Z)$
8. $T \leftarrow \mathsf{H}_L(A, M) \oplus R$
9. **return** (C, T)

Oracle Decrypt(N, A, C, T)

10. $R \xleftarrow{\$} \{0,1\}^n$; $Z \xleftarrow{\$} \{0,1\}^{\ell-n}$
11. **if** $N \in \mathcal{N}$ **then** $R \,\|\, Z \leftarrow F(N)$
12. **else** $\mathcal{N} \leftarrow \mathcal{N} \cup \{N\}$
13. $F(N) \leftarrow R \,\|\, Z$
14. $M \leftarrow C \oplus \mathsf{msb}_{|C|}(Z)$
15. **return** M

Oracle Verify(N, A, C, T)

16. $R \xleftarrow{\$} \{0,1\}^n$; $Z \xleftarrow{\$} \{0,1\}^{\ell-n}$
17. **if** $N \in \mathcal{N}$ **then** $R \,\|\, Z \leftarrow F(N)$
18. **else** $\mathcal{N} \leftarrow \mathcal{N} \cup \{N\}$
19. $F(N) \leftarrow R \,\|\, Z$
20. $T^* \leftarrow \mathsf{H}_L(A, C) \oplus R$
21. **if** $T^* = T$ **then** $\mathsf{forge} \leftarrow \mathsf{true}$; $\boxed{\textbf{return } \top}$
22. **return** $\bot$

Fig. 16. Game G_0 and G_1 for the proof of (3) in Theorem 2

C.2 Proof of (3)

AEAD-3 has the hash key L as the secret key and hashes the plaintext instead of the ciphertext. As in the proof of (4), we assume that $\mathcal{A}$ is deterministic and makes exactly q encryption queries, q' decryption queries, and q'' verification queries.

Two games Game G_0 and Game G_1 are defined in Fig. 16, where Game G_1 includes the boxed statements and Game G_0 does not. Game G_1 simulates the AEAD-3 encryption oracle in lines 3–9, the decryption oracle in lines 10–15, and the verification oracle in lines 16–22 using the lazy sampling of F.

Game G_0 is obtained from Game G_1 by removing the boxed statements, and Game G_0 and Game G_1 are identical until the flag forge gets set, and we therefore have

$$\mathbf{Adv}^{\text{int-rup}}_{\mathsf{AEAD\text{-}3}[\mathrm{Rand}(n,\ell),\mathsf{H}]}(\mathcal{A}) \leq \Pr[\mathcal{A}^{G_0} \text{ sets } \mathsf{forge}]$$

from the fundamental lemma of game playing.

We consider $\Pr[\mathcal{A}^{G_0}$ sets forge]. The flag forge means that a tag computed in the **Oracle Verify** is equal to a tag queried for the **Oracle Verify**.

By following a similar argument to the proof of (4), we focus on the non-adaptive strategy and fix the q encryption queries (N_i, A_i, M_i), q' decryption queries $(N'_{i'}, A'_{i'}, C'_{i'}, T'_{i'})$, and q'' verification queries $(N''_j, A''_j, C''_j, T''_j)$.

We consider three cases. If for the j-th verification query, $N''_j \neq N_i$ for all i, and $N''_j \neq N'_{i'}$ for all i', the value (R''_j, Z''_j) is uniformly distributed and independent of (R_i, Z_i) for all i, and $Z'_{i'}$ for all i'. The value R''_j is also independent of the hash key L. Hence

$$\Pr[T^*_j = T''_j] = \Pr[R''_j = \mathsf{H}_L(A''_j, M''_j) \oplus T''_j] = \frac{1}{2^n}.$$

If for the j-th verification query, $N''_j \neq N_i$ for all i, and $N''_j = N'_{i'}$ for some i', then the keystreams Z''_j and $Z'_{i'}$ have an overlap. Suppose that $|M''_j| \leq |M'_{i'}|$. In this case, Z''_j is a prefix of $Z'_{i'}$, and let this value be z''_j. Then the event $T^*_j = T''_j$ is equivalent to

$$\mathsf{H}_L(A''_j, C''_j \oplus z''_j) \oplus R''_j = T''_j.$$

Now since R''_j is independent of z''_j, we have

$$\Pr[T^*_j = T''_j] = \Pr[\mathsf{H}_L(A''_j, C''_j \oplus z''_j) \oplus R''_j = T''_j] = \frac{1}{2^n}.$$

The other situation where $Z'_{i'}$ is a proper prefix of Z''_j can be shown in a similar way.

Suppose that for the j-th verification query, $N''_j = N_i$ for some i. Then the value (R''_j, Z''_j) is independent of (R_k, Z_k) for all $i \neq k$, and $R''_j = R_i$. The keystreams Z''_j and Z_i have an overlap. Suppose that $|M''_j| \leq |M_i|$, i.e., Z''_j is a prefix of Z_i. Note that the other case of $|M_j| < |M''_j|$ follows similarly. Let this value be z''_j. The event $T^*_j = T''_j$ is equivalent to

$$\mathsf{H}_L(A''_j, C''_j \oplus z''_j) \oplus \mathsf{H}_L(A_i, M_i) = T''_j \oplus T_i. \tag{12}$$

Now if $(A''_j, C''_j) = (A_i, C_i)$, it follows that $M''_j = M_i$ and $Z''_j = Z_i$. We have $T''_j \neq T_i$ and (12) cannot hold. If $(A''_j, C''_j) \neq (A_i, C_i)$, then $(A''_j, C''_j \oplus z''_j) = (A''_j, M''_j) \neq (A_i, M_i)$. Hence we have

$$\Pr[T^*_j = T''_j] = \Pr[\mathsf{H}_L(A''_j, C''_j \oplus z''_j) \oplus \mathsf{H}_L(A_i, M_i) = T''_j \oplus T_i] \leq \epsilon.$$

Therefore, for each $j = 1, \ldots, q''$, we have $\Pr[T^*_j = T''_j] \leq \epsilon$. It follows that

$$\Pr[\mathcal{A}^{G_0} \text{ sets } \mathsf{forge}] \leq q''\epsilon, \tag{13}$$

and the bound (3) follows from (13). □

C.3 Proof Outlines of (2) and (5)

We briefly discuss how other bounds of (2) and (5) are obtained.

We first consider the security bound of AEAD-1 in (2). While AEAD-2 derives L from $\mathsf{SC}_K(K')$, AEAD-1 uses independent key L. Thus the bound of AEAD-1 is derived from Appendix C.1 by removing the bad event regarding a collision between K' and other inputs to SC_K, which has probability $(q + q' + q'')/2^n$.

For AEAD-4 in (5), we see that it is similar to both AEAD-2 and AEAD-3. It derives the hash key from K' as in AEAD-2, and it hashes the plaintext instead of the ciphertext as in AEAD-3. We need to consider a collision between K' and other inputs to SC_K as a bad event. We obtain the bound (5) by following the proof of AEAD-3 in Appendix C.2 and adopting the evaluation of the probability of the bad event as in Appendix C.1.