Security Analysis of SFrame

Takanori Isobe[1,2,3], Ryoma Ito[2], Kazuhiko Minematsu[4]

[1] University of Hyogo, Japan.
takanori.isobe@ai.u-hyogo.ac.jp
[2] National Institute of Information and Communications Technology, Japan.
itorym@nict.go.jp
[3] PRESTO, Japan Science and Technology Agency, Japan.
[4] NEC Corporation, Japan.
k-minematsu@nec.com

Abstract. As people become more and more privacy conscious, the need for end-to-end encryption (E2EE) has become widely recognized. We study herein the security of SFrame, an E2EE mechanism recently proposed to the Internet Engineering Task Force for video/audio group communications over the Internet. Despite being a quite recent project, SFrame is going to be adopted by a number of real-world applications. We inspect the original specification of SFrame and find critical issues that will lead to impersonation (forgery) attacks with a practical complexity by a malicious group member. We also investigate the several publicly available SFrame implementations and confirm that this issue is present in these implementations.

Keywords: End-to-End Encryption · SFrame · Authenticated Encryption · Signature · Impersonation

1 Introduction

End-to-end encryption (E2EE) is a technology that ensures the secrecy and the authenticity of communications from the intermediaries between the communicating parties. When E2EE is deployed in a communication application over the Internet, even the servers that facilitate communications cannot read or tamper the messages between the users of this application.

Due to the numerous pieces of evidence of massive surveillance, most notably by the case of Edward Snowden, E2EE has received significant attention and has been deemed as a key feature for protecting users' privacy and integrity for a wide range of communication applications. This also holds for the video calling/meeting applications, such as Zoom[1] or Webex[2]. The end-to-end security of video group meeting applications has been actively studied, and various approaches to E2EE have been proposed. Studying the security of E2EE systems in practice is also a hot topic, as shown by [9, 16, 18, 19, 43].

[1] https://zoom.us/
[2] https://www.webex.com

In this study, we investigate SFrame, which is one such approach aiming to provide E2EE over the Internet. Technically, SFrame is a mechanism for encrypting real-time communication (RTC) traffic in an end-to-end manner. RTC (or WebRTC, an RTC protocol between web browsers) is a popular protocol used by video/audio communication, and SFrame has been carefully designed to suppress communication overheads that would be introduced when E2EE is deployed. In 2020, it was proposed to the the Internet Engineering Task Force (IETF) by a team of Google and CoSMo Software engineers (*i.e.*, Omara, Uberti, Gouaillard, and Murillo) as a form of Internet draft [36]. Despite being a quite recent proposal, it has quickly gained much attention. One can find a large variety of ongoing plans to adopt SFrame as a crucial component for E2EE, including major proprietary software to open-source applications, such as Google Duo [35], Cisco Webex [7, 8], and Jitsi Meet [22, 44].

1.1 Our Contributions

We looked into the original specification of SFrame [36] and made several observations. Most notably, we found an issue regarding the use of authenticated encryption with associated data (AEAD) and signature algorithm. The specification [36] defines two AEAD algorithms, namely a generic composition of AES-CTR and HMAC-SHA256, which is dubbed as AES-CM-HMAC, and AES-GCM for encrypting video/audio packets. We will show an *impersonation* (*forgery*) attack by a malicious group member who owns a shared group key for the specified AEAD algorithm. The attack complexity depends on the AEAD algorithm. More specifically, for AES-CM-HMAC, the complexity depends on the tag length, while for AES-GCM, the complexity is negligible for any tag length. We observe that AES-CM-HMAC is specified with particularly short tags, such as 4 or 8 bytes, making the attack complexity practical.

Our results are solely based on the Internet draft [36] and publicly available source codes [7, 22, 45], and we have not implemented the proposed attacks to verify their feasibility. It is difficult to implement the proposed attacks because the SFrame specification is still a draft version, and no product that implements the current version of SFrame [36] has actually been deployed. Accordingly, instead of implementing the proposed attacks, we discussed with the designers to confirm the feasibility of the proposed attacks.

The specification remains abstract at some points and may be subject to change. Moreover, the real-world implementation often does not strictly follow what was specified in [36]. Hence, this issue does not immediately refer to the practical attacks against the existing E2EE video communication applications that adopt SFrame. Nevertheless, considering the practicality of our attacks, we think there is a need to improve the current SFrame specification. An overview of our security analysis is presented below.

AEAD security. In Section 4.1, we study the classical AEAD security (*i.e.*, confidentiality and integrity) of the SFrame encryption scheme. While SFrame adopts existing, well-analyzed AEAD schemes, they are used in a way different from what standard security analysis assumes; hence, the existing AEAD security proofs do not necessarily carry over to the entire protocol. Despite this discrepancy, we show that the encryption schemes defined by SFrame are provably secure in the context of a standard AEAD.

Impersonation against AES-CM-HMAC with Short Tags. In Section 4.2, we show an impersonation attack on AES-CM-HMAC with short tags by a malicious group member. This attack exploits a vulnerability of a very short tag length. The malicious group member owns a shared group key; thus, she can precompute multiple ciphertext/tag pairs from any input set and store them into a precomputation table. Subsequently, she can forge by intercepting a target message frame and replacing the ciphertext in that frame with a properly selected ciphertext from the precomputation table. For example, when the tag length is 4 bytes, she can practically perform an impersonation attack with a success probability of almost one by preparing 2^{32} precomputation tables in advance.

Security of AES-CM-HMAC with Long Tags. In Section 4.3, we discuss the security of AES-CM-HMAC with long tags. We show that AES-CM-HMAC with long tags is secure against the impersonation attack proposed in Section 4.2. In more detail, we prove that AES-CM-HMAC is a second-ciphertext unforgeability (SCU) security, defined by Dodis *et al.* [10], and this SCU security covers the class of impersonation attacks described above (*i.e.*, forging a ciphertext using the knowledge of the secret key, such that the forged ciphertext has the same tag value as a previously observed ciphertext). Concretely, we show that the SCU security of AES-CM-HMAC depends on the security of SHA256, which is the underlying hash function of SFrame. SHA256 has an everywhere second-preimage resistance, defined by Rogaway and Shrimpton [32, 42]; hence, AES-CM-HMAC with long tags can be considered as the SCU-secure AEAD.

Impersonation against AES-GCM with Any Long Tags. In Section 4.4, we present an impersonation attack on AES-GCM with any long tags by a malicious group member. This attack exploits the vulnerability of the GHASH function linearity in the known key setting. The malicious group member who owns the GCM key and observes a legitimate GCM input/output set, including a tag, can create another distinct set with the same tag. The remaining value in this set, excluding the tag, can be chosen almost freely from the GHASH function linearity and the knowledge of the GCM key; thus, this attack works with negligible complexity, irrespective of the tag length, unlike the case of AES-CM-HMAC.

Authentication Key Recovery against AES-GCM with Short Tags. In Section 4.5, we consider an authentication key recovery attack on AES-GCM with short tags. This attack exploits the fact that no restriction has been provided regarding the NIST requirements on GCM usage with short tags. The available implementations of the original [45], Cisco Webex [7], and Jitsi Meet [22] have no restrictions regarding such requirements. When these available implementations employ the 4-byte tag, the authentication key is recovered with the data complexity of 2^{32}, which is practically available in the adversary.

Further Analysis for Recommended Crountermeasures. In Section 4.6, we recommend three countermeasures against all the proposed attacks: (1) for AES-CM-HMAC, long tags (*e.g.*, 16-byte tags), instead of short tags, especially 4-byte tags, should be used; (2) for AES-GCM, a signature should be computed over a whole frame; and (3) other ciphersuites that work as a secure encryptment scheme, such as hash function chaining (HFC) [10], should be deployed. Then, we further analyze from a performance and security perspective towards implementation of these countermeasures in Section 5, and clarify the following: (1) computing a signature over a whole frame has a disadvantage in terms of efficiency; (2) the use of HFC with long tags in SFrame is not a problem in terms of performance and security; and (3) AEADs with a simple function in the known key setting should not be candidates for the ciphersuites in the E2EE applications unless a signature is computed over a whole frame.

1.2 Responsible Disclosure

In March 2021, we reported our results in this article to the SFrame designers via email and video conference. They acknowledged that our attacks are feasible under the existence of a malicious group member, quickly decided to remove the signature mechanism [13] and to extend tag calculation to cover nonces [12], and updated the specification in the Internet draft on March 29, 2021 [37]. They have a plan to review the SFrame specification and support the signature mechanism again in the future.

1.3 Organization of the Paper

The paper is organized as follows. Section 2 provides the specification of SFrame, including the underlying AEAD, and a brief survey on its publicly available implementations; Section 3 describes the security goals of the recently proposed E2EE; Section 4 presents our analysis showing the impersonation attacks against SFrame; Section 5 presents several other observations, followed by our recommendations; and Section 6 concludes this study.

2 SFrame

2.1 Specification

Overview. SFrame is a group communication protocol for end-to-end encryption (E2EE) used by video/audio meeting systems. It involves multiple users and a (media) server that mediates communication between users. They are connected via the server, and the communication between a user and the server is protected by a standard Internet client–server encryption protocol, specifically the Datagram Transport Layer Security-Secure Real-time Transport Protocol (DTLS-SRTP).

SFrame is specified in the Internet draft [36], but does not specify the key exchange protocol between the parties, and the choice is left to the implementors. In practice, Signal protocol [38], Olm protocol [29], or Message Layer Security (MLS) protocol [3] could be used. With SFrame, users encrypt/decrypt video and audio frames prior to RTP packetization. A generic RTP packetizer splits the encrypted frame into one or more RTP packets and adds an original SFrame header to the beginning of the first packet and an authentication tag to the end of the last packet. The SFrame header contains a signature flag S, a key ID number KID, and a counter value CTR for a nonce used for encryption/decryption.

Cryptographic Protocol. Suppose there is a group of users, G. All users in G first perform a predetermined key exchange protocol, as suggested above, and share multiple group keys $K_{\mathsf{base}}^{\mathsf{KID}}$ associated with the key ID number KID, which is called *base key* in the original specification [36]. In addition, each user establishes a digital signature key pair, $(K_{\mathrm{sig}}, K_{\mathrm{verf}})$.

An E2EE session for SFrame uses a single ciphersuite that consists of the following primitives:

– a hash function used for key derivation, tag generation, and hashing signature inputs (*e.g.*, SHA256 and SHA512);
– an authenticated encryption with associated data (AEAD) [31, 39] used for frame encryption (*e.g.*, AES-GCM [1, 17] and AES-CM-HMAC); the authentication tag may be truncated; and
– an optional signature algorithm (*e.g.*, EdDSA over Ed25519 [5,6] and ECDSA over P-521 [15, 23]).

The original specification [36] specifically defines the following symmetric-key primitives for the ciphersuite:

– AES-GCM with a 128- or 256-bit key and no specified tag length; and
– AES-CM-HMAC, which is a combination of AES-CTR with a 128-bit key and HMAC-SHA256 with a 4- or 8-byte truncated authentication tag.

Figure 1 and Algorithm 1 show the media frame encryption flow in an E2EE session for SFrame using the abovementioned ciphersuites. When AES-GCM is adopted as the ciphersuite, AEAD.Encryption in Algorithm 1 is executed according to NIST SP 800-38D [11]. Before performing the AEAD encryption procedure by AES-GCM, HKDF [27] is used to generate the encryption key K_e^{KID} and the salt $salt^{\mathsf{KID}}$ for encrypting/decrypting media frames as follows:

$$\begin{aligned}\mathsf{SFrameSecret} &= \mathsf{HKDF}(K_{\mathsf{base}}^{\mathsf{KID}}, \text{'}\mathsf{SFrame10}\text{'}),\\ K_e^{\mathsf{KID}} &= \mathsf{HKDF}(\mathsf{SFrameSecret}, \text{'}\mathsf{key}\text{'}, \mathrm{KeyLen}),\\ salt^{\mathsf{KID}} &= \mathsf{HKDF}(\mathsf{SFrameSecret}, \text{'}\mathsf{salt}\text{'}, \mathrm{NonceLen}),\end{aligned}$$

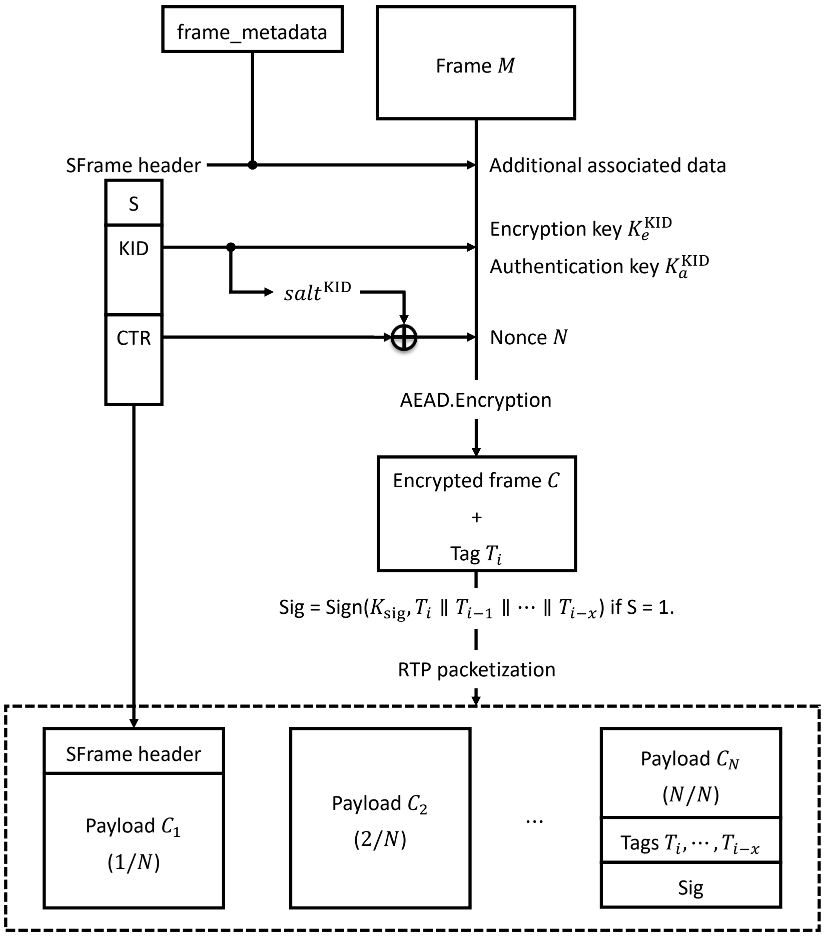
frame_metadata
Frame M
SFrame header
Additional associated data
S
KID
Encryption key K_e^{KID}
Authentication key K_a^{KID}
$salt^{\text{KID}}$
CTR
Nonce N
AEAD.Encryption
Encrypted frame C + Tag T_i
Sig = Sign($K_{\text{sig}}, T_i \parallel T_{i-1} \parallel \cdots \parallel T_{i-x}$) if S = 1.
RTP packetization
SFrame header
Payload C_1 ($1/N$)
Payload C_2 ($2/N$)
...
Payload C_N (N/N)
Tags $T_i, \cdots, T_{i-x}$
Sig

Fig. 1: Media frame encryption flow.

where KeyLen and NonceLen are the length (byte) of an encryption key and a nonce for the encryption algorithm, respectively. Each user then stores K_e^{KID} and $salt^{\mathsf{KID}}$, such as $\mathsf{KeyStore}[\mathsf{KID}] = (K_e^{\mathsf{KID}}, salt^{\mathsf{KID}})$. When AES-CM-HMAC is adopted as the ciphersuite, AEAD.Encryption in Algorithm 1 is executed according to Algorithm 2. Before performing AES-CM-HMAC, HKDF [27] is used as well as in the case of AES-GCM, albeit in a slightly different manner:

$$\begin{aligned}
\mathsf{AEADSecret} &= \mathsf{HKDF}(K_{\mathsf{base}}^{\mathsf{KID}}, \text{'}\mathsf{SFrame10\ AES\ CM\ AEAD}\text{'}),\\
K_e^{\mathsf{KID}} &= \mathsf{HKDF}(\mathsf{AEADSecret}, \text{'}\mathsf{key}\text{'}, \mathrm{KeyLen}),\\
K_a^{\mathsf{KID}} &= \mathsf{HKDF}(\mathsf{AEADSecret}, \text{'}\mathsf{auth}\text{'}, \mathrm{HashLen}),\\
salt^{\mathsf{KID}} &= \mathsf{HKDF}(\mathsf{AEADSecret}, \text{'}\mathsf{salt}\text{'}, \mathrm{NonceLen}),
\end{aligned}$$

Algorithm 1 Media frame encryption scheme

```
Input: S: signature flag, KID: key ID, CTR: counter value, frame_metadata: frame
    metadata, $M$: frame
Output: $C$: encrypted frame, $T$: authentication tag
 1: procedure ENCRYPTION(S, KID, CTR, frame_metadata, $M$)
 2:     if An AEAD encryption algorithm is AES-GCM then
 3:         $K_e^{\mathsf{KID}}, salt^{\mathsf{KID}} = \mathsf{KeyStore}[\mathsf{KID}]$
 4:     else
 5:         $K_e^{\mathsf{KID}}, K_a^{\mathsf{KID}}, salt^{\mathsf{KID}} = \mathsf{KeyStore}[\mathsf{KID}]$
 6:     end if
 7:     $ctr = \mathsf{encode}(\mathsf{CTR}, \mathrm{NonceLen})$        ▷ encode CTR as a big-endian of NonceLen.
 8:     $N = salt^{\mathsf{KID}} \oplus ctr$                              ▷ $N$ is a Nonce.
 9:     $\mathsf{header} = \mathsf{encode}(\mathsf{S}, \mathsf{KID}, \mathsf{CTR})$
10:     $\mathsf{aad} = \mathsf{header} + \mathsf{frame\_metadata}$           ▷ aad is an additional associated data.
11:     if an AEAD encryption algorithm is AES-GCM then
12:         $C, T = \mathrm{AEAD.ENCRYPTION}(K_e^{\mathsf{KID}}, N, \mathsf{aad}, M)$
13:     else
14:         $C, T = \mathrm{AEAD.ENCRYPTION}(K_e^{\mathsf{KID}}, K_a^{\mathsf{KID}}, N, \mathsf{aad}, M)$
15:     end if
16: end procedure
```

where HashLen is the output length (byte) of the hash function. Each user also stores the encryption key K_e^{KID}, the authentication key K_a^{KID}, and salt $salt^{\mathsf{KID}}$, such as $\mathsf{KeyStore}[\mathsf{KID}] = (K_e^{\mathsf{KID}}, K_a^{\mathsf{KID}}, salt^{\mathsf{KID}})$.

While an AEAD enables the detection of forgeries by an entity who does not own $K_{\mathsf{base}}^{\mathsf{KID}}$, it does not prevent the impersonation by a malicious group member who owns a shared group key. To detect such an impersonation, a common countermeasure is to attach a signature for each encrypted packet. This can incur a significant overhead both in time and bandwidth. SFrame addresses this problem by reducing the frequency and the input length of signature computations. That is, a signature Sig is computed over a list of authentication tags with a fixed size, $(T_i, T_{i-1}, \dots, T_{i-x})$, as follows:

$$\mathsf{Sig} = \mathsf{Sign}(K_{\mathrm{sig}}, T_i \parallel T_{i-1}, \parallel \cdots \parallel T_{i-x}),$$

where Sign denotes the signature function. This signature is appended to the end of the data comprising the SFrame header, the current encrypted payload, its corresponding authentication tag T_i, and the list of authentication tags $(T_{i-1}, \dots, T_{i-x})$ that correspond to the previously encrypted payload, such that any group user can verify the authenticity of the entire payload.

2.2 Available Implementations

We list some implementations of SFrame that are publicly available. Some of them do not strictly follow the original specification [36] and exhibit some varieties. In this study, we particularly focus on the specified AEAD schemes and the allowed

Algorithm 2 AEAD encryption by AES-CM-HMAC

```
Input: $K_a^{\mathsf{KID}}$: authentication key, aad: additional associated data, $C$: encrypted frame
Output: $T$: truncated authentication tag
1: procedure Tag.Generation($K_a^{\mathsf{KID}}$, aad, $C$)
2:     $\mathsf{aad}Len = \mathsf{encode}(len(\mathsf{aad}), 8)$    ▷ encode aad length as a big-endian of 8 bytes
3:     $D = \mathsf{aad}Len + \mathsf{aad} + C$
4:     $tag = \mathsf{HMAC}(K_a^{\mathsf{KID}}, D)$
5:     $T = \mathsf{trancate}(tag, \mathrm{TagLen})$
6: end procedure

Input: $K_e^{\mathsf{KID}}$, $K_a^{\mathsf{KID}}$, $N$: Nonce, aad, $M$: frame
Output: $C$, $T$
1: procedure AEAD.Encryption($K_e^{\mathsf{KID}}$, $K_a^{\mathsf{KID}}$, $N$, aad, $M$)
2:     $C$ = AES-CTR.Encryption($K_e^{\mathsf{KID}}, N, M$)
3:     $T$ = Tag.Generation($K_a^{\mathsf{KID}}$, aad, $C$)
4: end procedure
```

tag length in each of the implementation because this determines the complexity of our attack.

The original. There is a Javascript implementation by one of the designers of SFrame, Sergio Garcia Murillo [45]. It is based on webcrypt. In his implementation, it supports

– AES-CM-HMAC with a 4- or 10-byte tag, where the 4 (10)-byte tag is used for audio (video) packets.

Google Duo. Duo[3] is a video calling application developed by Google. For group calling, it adopts the Signal protocol as a key exchange mechanism and SFrame as the E2EE mechanism. One technical paper [35] was written by one of the coauthors, Emad Omera, of the original specification [36]. The source code is not available, but according to the technical paper, it supports

– AES-CM-HMAC.

The technical paper does not describe the tag length. Note that we confirmed that Google Duo does not currently use the signature feature.

Cisco Webex. Webex is a major video meeting application developed by Cisco. A recent whitepaper entitled "Zero-Trust Security for Webex White Paper" [8] describes the path to their goal referred to as the Zero-Trust Security and suggests the usage of the MLS protocol as a key exchange mechanism and SFrame as a media encryption to enhance the end-to-end security of Webex. The corresponding SFrame implementation is available at Github [7]. The repository maintainer warns that the specification is in progress. As of March 2021, it supports

– AES-GCM with a 128- or 256-bit key with a 16-byte tag; and

[3] `https://duo.google.com/about/`

– AES-CM-HMAC with a 4- or 8-byte tag.

Jitsi Meet. An open-source video communication application, called Jitsi Meet[4] was presented at FOSDEM 2021, a major conference for open-source projects[5]. Despite being a quite recent project, it is getting popularity as an open-source alternative to other major systems. It adopts SFrame with Olm protocol as the underlying key exchange protocol. The source code is available [22]. It supports

– AES-CM-HMAC with a 4- or 10-byte tag, where the 4 (10)-byte tag is used for audio (video) packets.

3 Adversary Models and Security Goals

3.1 Adversary Models

The designers did not define adversary models in the original specification [36]. We define the adversary models for our security analysis with reference to those defined by Isobe and Minematsu [19].

Definition 1. (Malicious User) *A malicious user, who is a legitimate user that does not possess a shared group key, tries to break one of the subsequently defined security goals of the other E2EE session by maliciously manipulating the protocol.*

Definition 2. (Malicious Group Member) *A malicious group member, who is a legitimate group member and possesses a shared group key, tries to break the subsequently defined security goals by deviating from the protocol.*

In addition, the *E2E adversary* is defined in [19], but we do not explain this definition because this adversary is out of the scope for our security analysis.

3.2 Security Goals of E2EE

In February 2021, the Internet draft entitled "Definition of End-to-end Encryption" was released [25]. This draft states that the fundamental features for E2EE require *authenticity*, *confidentiality*, and *integrity*, which are defined as follows:

Definition 3. (Authenticity) *A system provides message authenticity if the recipient is certain who sent the message, and the sender is certain who received it.*

Definition 4. (Confidentiality) *A system provides message confidentiality if only the sender and intended recipient(s) can read the message plaintext (i.e., messages are encrypted by the sender such that only the intended recipient(s) can decrypt them).*

[4] `https://meet.jit.si/`
[5] `https://fosdem.org/2021/schedule/`

Definition 5. (Integrity) *A system provides message integrity when it guarantees that messages has not been modified in transit (i.e., a recipient is assured that the message they have received is exactly what the sender intented to sent)*

In addition, *availability*, *deniability*, *forward secrecy*, and *post-compromise security* are defined in this draft as the optional/desirable features for enhancing the E2EE systems. We do not explain these definitions because these features are out of the scope for our security analysis.

3.3 Security Goals of AEAD for E2EE

Dodis *et al.* [10] proposed a new primitive, called *encryptment*, for the message franking scheme, which enables a cryptographically verifiable reporting of the malicious content in end-to-end encrypted messaging. In addition, they defined *confidentiality* and *second-ciphertext unforgeability* (SCU) as security goals to ensure the security level of the encryptment scheme.

Definition 6. (Second-ciphertext Unforgeability) *An adversary $\mathcal{A}$ is given $K \xleftarrow{\$} \mathcal{K}$, which means a randomly chosen key K from the key space $\mathcal{K}$ and is allowed to perform AEAD encryption/decryption in the local environment. Then, we define the second-ciphertext unforgeability (SCU) advantage of $\mathcal{A}$ against AEAD for E2EE as*

$$\begin{aligned}\mathbf{Adv}^{\mathsf{SCU}}_{\mathsf{AEAD}}(\mathcal{A}) = \Pr\big[&K \xleftarrow{\$} \mathcal{K} : \mathcal{A}(K) \to (N, A, C, N^*, A^*, C^*, T),\\ &\mathsf{Dec}(K, N, A, C, T) = M,\\ &\mathsf{Dec}(K, N^*, A^*, C^*, T) = M^*\\ &\textit{for some } M, M^* \neq \bot\big],\end{aligned}$$

where Dec denotes the decryption algorithm of AEAD; N and N^* denote nonces; A and A^* denote associated data; C and C^* denote ciphertexts; M and M^* denote plaintexts; T denotes a tag; and $\bot$ denotes a symbol representing a decryption failure.

The adversary in the SCU game is given with key; hence, this is not captured by the standard AEAD security notions of confidentiality and integrity [4, 39]. When there is a malicious group member in an E2EE application, she can actually work as a SCU adversary $\mathcal{A}$ by intercepting the target frame (N, A, C, T) because she knows the shared group key K.

3.4 Security Goals of Hash Functions

A secure hash function H typically has three fundamental properties: *preimage resistance*, *second-preimage resistance*, and *collision resistance*. We focus herein on two types of second-preimage resistance and define them with reference to [10, 32] as follows:

Definition 7. (Second-preimage Resistance) *Let $\mathcal{A}$ be an adversary attempting to find any second input with the same output as any specified input (i.e., for any given message $M \xleftarrow{\$} \mathcal{M}$, a randomly chosen message M from the message space $\mathcal{M}$ to find a second-preimage $M^* \neq M$ such that $H(M) = H(M^*)$). Then, we define the second-preimage (Sec) resistance advantage of $\mathcal{A}$ against H as*

$$\mathbf{Adv}_H^{\mathsf{Sec}}(\mathcal{A}) = \Pr\big[M \xleftarrow{\$} \mathcal{M}; M^* \leftarrow \mathcal{A} : \\ (M \neq M^*) \wedge \big(H(M) = H(M^*)\big)\big].$$

Definition 8. (Everywhere Second-preimage Resistance) *For a positive integer n, let $\{0,1\}^{\leq n}$ be a set of bit strings not longer than n. Let $\mathcal{M} = \{0,1\}^*$ and $\mathcal{Y} = \{0,1\}^n$. Suppose $H : \mathcal{K} \times \mathcal{M} \to \mathcal{Y}$ is a keyed hash function. Let $\mathcal{A}$ be an adversary against H to find a second preimage for the target input $M \in \mathcal{M}$ that is fixed with $|M| \leq \ell$. Then, we define the everywhere second-preimage (eSec) resistance advantage of $\mathcal{A}$ against H as*

$$\mathbf{Adv}_H^{\mathsf{eSec}[\leq \ell]}(\mathcal{A}) = \max_{M \in \{0,1\}^{\leq \ell}} \Big\{ \Pr\big[K \xleftarrow{\$} \mathcal{K}; M^* \leftarrow \mathcal{A}(K) : \\ (M \neq M^*) \wedge \big(H_K(M) = H_K(M^*)\big)\big] \Big\}.$$

The everywhere second-preimage or eSec resistance introduced by Rogaway and Shrimpton [42] is called a slight extension of a strong form of second-preimage resistance. In this study, we assumed the standard hash function (SHA2) as an instantiation of the keyed function by using IV as a key because standard security reduction is not possible, otherwise. (see [42]). For simplicity, we assume that this key is implicit and do not describe it in the proofs.

4 Security Analysis

4.1 Security of AEAD under SFrame

We first discuss the security of AEAD used by SFrame. Here, we view Algorithm 1 as an encryption of AEAD because viewing Algorithm 2 as a full-fledged AEAD does not make sense (see below). The keys are effectively contained by $\mathsf{KeyStore[KID]}$, and the nonce is CTR. The associated data are a tuple $(\mathsf{S}, \mathsf{KID}, \mathsf{frame\_metadata})$, and the plaintext is M.

In Algorithm 1, variable N is a sum of $\mathit{salt}^{\mathsf{KID}}$ and ctr (Line 8), where the former is essentially a part of key (via HKDF), and the latter is an encoded form of CTR. This N serves as nonce for the internal AEAD algorithm at Line 12/14. The data aad serves as AD for the internal AEAD and consists of header and $\mathsf{frame\_metadata}$, where the former contains an encoded form of $(\mathsf{S}, \mathsf{KID}, \mathsf{CTR})$. aad contains CTR equivalent to N; thus, if the internal AEAD is AES-CM-HMAC of Algorithm 2, HMAC takes the nonce (CTR) in addition to AD ($\mathsf{frame\_metadata}$) and the ciphertext C. In other words, the lack of $N = \mathit{salt}^{\mathsf{KID}} \oplus \mathit{ctr}$ is not a problem. Moreover, adding a pseudorandom value to the nonce of AES-CTR does not degrade security as long as that value is computationally independent of the AES-CTR key.

A slightly more formal analysis is given below. Algorithm 1 combined with AES-CM-HMAC can be interpreted as an encryption routine of the encryption-then-MAC AEAD construction. More specifically, it takes nonce $\widetilde{N} = \mathsf{CTR}$, associated data $\widetilde{A} = (\mathsf{S}, \mathsf{KID}, \mathsf{frame\_metadata})$, and plaintext M to produce the ciphertext C and the tag T:

$$C = \widetilde{\mathsf{Enc}}_K(\widetilde{N}, M),$$
$$T = \widetilde{\mathsf{MAC}}_{K'}(\widetilde{N}, \widetilde{A}, C),$$

where K and K' are derived via a master key with a key derivation function (HKDF); $\widetilde{\mathsf{Enc}}_K$ denotes the plain counter mode encryption with a pseudorandom offset to nonce (*i.e.*, $salt^{\mathsf{KID}}$ derived via HKDF); and $\widetilde{\mathsf{MAC}}_{K'}$ denotes the HMAC with a certain bijective input encoding. This means that Algorithm 1 is exactly reduced to the encryption-then-MAC generic composition (assuming HKDF as a PRF), whose security is proven when $\widetilde{\mathsf{Enc}}$ is IND-CPA secure, and $\widetilde{\mathsf{MAC}}$ is a PRF [26, 33]. Proving the latter claim is trivial. Algorithm 1 is secure under the standard assumptions that AES is a pseudorandom permutation, and HMAC is a PRF. Algorithm 2 itself is not a generically secure (*i.e.*, when nonce N and AD aad are independently chosen) AEAD because it ignores N in the tag computation. This issue was raised at the discussion in CFRG[6], and our analysis provides an answer for this.

4.2 Impersonation against AES-CM-HMAC with Short Tags

While the AEAD security of Algorithm 1 is sound, it does not necessarily mean the full E2EE security. In this section, we point out the risk of *impersonation* by a malicious group member who owns the group key. The impersonation attack implies that the scheme does not achieve the security goal of integrity in E2EE.

We simplify the model and stick to the standard AEAD notation, that is, the input is (N, A, M) for nonce N, associated data A, and plaintext M, and the output is (C, T) for ciphertext C and tag T. We also consider the case in which the signature is computed for each tag for simplicity. The notational discrepancies from Algorithms 1 and 2 do not change the essential procedure of our attacks. With this simplified model, each group member sends an encrypted frame to all other members, and this frame consists of an AEAD output (N, A, C, T) and a signature $\mathsf{Sig} = \mathsf{Sign}(K_{\text{sig}}, T)$ signed by the user's signing key K_{sig}. The encryption input is (N, A, M), and the frame encryption by AES-CM-HMAC is abstracted as follows:

$$\begin{aligned} &C \leftarrow \text{AES-CTR}(K_e^{\mathsf{KID}}, N, M), \\ &T \leftarrow \mathsf{truncate}(\text{HMAC-SHA256}(K_a^{\mathsf{KID}}, (N, A, C)), \tau), \end{aligned} \qquad (1)$$

[6] `https://mailarchive.ietf.org/arch/browse/cfrg/?q=SFrame`

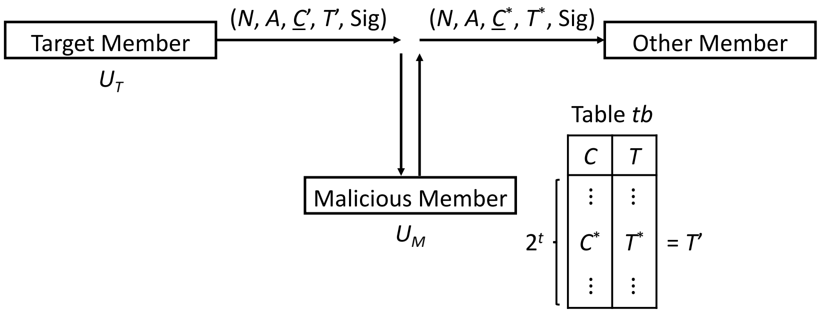

Fig. 2: Impersonation against AES-CM-HMAC with short tags. In the offline phase, a malicious group member U_M stores a set of (M, C, T) into the table tb. In the online phase, U_M intercepts a target frame $(N', A', C', T', \mathsf{Sig})$ sent by the target user U_T, searches a tuple (M^*, C^*, T^*) in tb, such that $T^* = T'$ and $C^* \neq C'$, replaces C' with C^* in the target frame, and sends $(N', A', C^*, T', \mathsf{Sig})$ to the other group members.

where τ denotes the tag length in bits. Note that N is included as a part of HMAC's input for the reason described at Section 4.1.

Suppose there is a communication group G containing a malicious group member U_M and another member U_T we call a target user. This U_M can mount a forgery attack (impersonation) by manipulating a frame sent by U_T. The forgery attack by U_M consists of offline and online phases.

In the offline phase, U_M determines (N, A, M), precomputes a set of (ciphertext,tag) tuples (C, T) by using K_e^{KID} and K_a^{KID}, which are known to all group members, and stores these into a table tb. Here, N and A are determined such that it is likely to be used by U_T (these information are public, and N is a counter; thus, this is practical).

In the online phase, the malicious group member observes the frames sent by U_T. If she finds the frame $(N, A, C', T', \mathsf{Sig})$, such that (C^*, T^*) is included in tb and $T^* = T'$, $C^* \neq C'$, she replaces C' in that frame with C^*. The signature Sig is computed over the tag T' that is not changed after the replacement; therefore, this manipulated frame will pass the verification. Figure 2 shows the overview of the attack. The details of attack procedures are given as follows:

Offline Phase.

1. U_M chooses the encryption input tuple (N, A, M).
2. U_M computes a ciphertext C and a τ-bit tag T for (N, A, M) following Eq. (1), where KID is set to point the target user.
3. U_M stores a set of (M, C, T) into the table tb.
4. U_M repeats step 1–3 2^t times with different messages.

Online Phase.

1. U_M intercepts a target frame $(N', A', C', T', \mathsf{Sig})$ sent by the target user, where $N' = N$ and $A' = A$.
2. U_M searches a tuple (M^*, C^*, T^*) in tb such that $T^* = T'$ and $C^* \neq C'$.
3. If U_M finds such a tuple, C' is replaced with C^* in the target frame, and $(N', A', C^*, T', \mathsf{Sig})$ is sent to the other group members.

The manipulated frame, including (C^*, T'), successfully passes the signature verification by other group members due to a tag collision, that is, no one can detect that the frame is manipulated by U_M, and the group members will accept M^* as a valid message from U_T. This is for the case where $x = 1$ (*i.e.*, each tag is independently signed by the signature key). It is naturally extend to the case where x is more than one, namely the case where a list of tags is signed altogether for efficiency.

To mount the attack described above, the adversary must intercept a legitimate message. In other words, the adversary may collude with an intermediate server or the E2E adversary, which is the central operating server. The practicality of this is beyond the scope of this article, but we remark that preventing a colluding attack with E2EE adversary is one of the fundamental goals of E2EE.

We note that the attack without intercept is also possible by creating a forged tuple $(N', A', C', T', \mathsf{Sig})$, such that $T' = T$ and $(N', A', C') \neq (N, A, C)$ by observing some legitimate tuples $(N, A, C, T, \mathsf{Sig})$ previously sent without corruption. Here, (N', A') is chosen; thus, it is likely to be used by U_T in the next frame, which is yet sent. This is essentially a reply of signature, and our guess as regards whether or not it is detected as replay depends on the actual system; hence, we keep it open. The cost of detecting a reply of a randomized algorithm is generally high because the receiver must keep the all random IVs used.

Complexity Evaluation. The computational cost to make the precomputation table tb in the offline phase is estimated as 2^t, and the success probability of Step 2 in the online phase is estimated as $2^{-\tau+t}$.

Practical Effects on SFrame. In the case of $\tau = 32$ (*i.e.*, 4-byte tags), if U_M prepares 2^{32} precomutation tables in the offline phase, the success probability is almost one. Thus, this forgery attack is practically feasible with a high success probability for the 4-byte tag. Moreover, in this attack, the adversary fully controls the decryption result (M^*) of the manipulated frame, except for 32 bits used for generating 2^{32} different tags in the offline phase.

To perform an actual attack on SFrame, the adversary must decide the target frame and set the target frame counter to the SFrame header file in M when generating tags in the offline phase because each SFrame header includes the frame counter to avoid replay attacks.

Even in the case of 8- and 10-byte tags, if U_M prepares 2^{56} tables, which is feasible by the nation-level adversary, the success probability is non-negligible at 2^{-8} and 2^{-24}, respectively.

4.3 Security of AES-CM-HMAC with Long Tags

We first discuss the security of AES-CM-HMAC with long tags (*e.g.*, 16-byte tags) against the impersonation attack described in Section 4.2. Even if a malicious group member prepares 2^{56} precomputation tables, it is infeasible because the success probability of the attack is 2^{-72}; therefore, AES-CM-HMAC with long tags can be secure against the impersonation attack proposed in Section 4.2.

We justify the abovementioned observation by showing the SCU security of AES-CM-HMAC with long tags. According to Algorithm 2, let D and D^* be (N, A, C) and (N^*, A^*, C^*) (see Line 3 in Tag.Generation procedure). Note that N is included in A (aad) as partial information (see Lines 7-10 in Algorithm 1). For simplicity, the tag generation by HMAC is abstracted as follows:

$$\mathrm{HMAC}(K_a^{\mathsf{KID}}, D) = H\big((K \oplus \mathit{opad}) \parallel H\big((K \oplus \mathit{ipad}) \parallel D\big)\big),$$

where H denotes a hash function (*e.g.*, SHA256 used in SFrame); *ipad* and *opad* denote fixed padding values; and K is generated from K_a^{KID} according to the padding rule in the HMAC algorithm (see [46] for details). The following theorem is simple to prove:

Theorem 1. *Let $\mathcal{A}$ be a SCU adversary against AES-CM-HMAC with the target encryption output being at most ℓ bits. Then, SCU advantage of $\mathcal{A}$ against AES-CM-HMAC is bounded as*

$$\mathbf{Adv}^{\mathsf{SCU}}_{\mathsf{AES\text{-}CM\text{-}HMAC}}(\mathcal{A}) < 2\mathbf{Adv}_H^{\mathsf{eSec}[\leq(\ell')]}(\mathcal{A}')$$

for some eSec adversary $\mathcal{A}'$ against H, which denotes the underlying SHA256 hash function, where $\ell' = \ell + 512$ (i.e., one block larger).

Proof. Let K be the key of HMAC. Thanks to the generic composition, we can assume that the adversary is given the key for the counter mode. The resulting game is that, given a transcript of encryption query (N, A, M, C, T) derived on K, the adversary is required to find a successful forgery (N^*, A^*, C^*, T) on K, such that $D^* \neq D$ (*i.e.*, $(N^*, A^*, C^*) \neq (N, A, C)$). Note that tag T is the output of HMAC taking K and $D = (N, A, C)$; thus, the plaintext M is not needed in the attack. Figure 3 illustrates this scenario, where IV denotes the initial hash value, $D = D_0 \parallel \ldots \parallel D_{l-1}$, $D^* = D_0^* \parallel \ldots \parallel D_{l-1}^*$, $S = H\big((K \oplus \mathit{ipad}) \parallel D\big)$, and $S^* = H\big((K \oplus \mathit{ipad}) \parallel D^*\big)$. D_i and D_i^* denote an input block to HMAC. The last block may need padding, but we simply ignore this (the analysis is pretty much the same). In this scenario, we consider the following two cases: $\mathcal{A}$ finds $S^* = S$ (Case 1), which implies $T = T^*$ or $S^* \neq S$ and $T = T^*$ (Case 2).

For Case 1, observe that $S = S^*$ means $H(K \oplus \mathit{ipad} \parallel D) = H(K \oplus \mathit{ipad} \parallel D^*)$; hence, a second preimage against the target input $K \oplus \mathit{ipad} \parallel D$ is obtained. For Case 2, when $S \neq S^*$ and $T = T^*$, the adversary finds a second preimage against the target (2-block, thus 1024-bit) input $K \oplus \mathit{opad} \parallel S$. Both cases are covered by

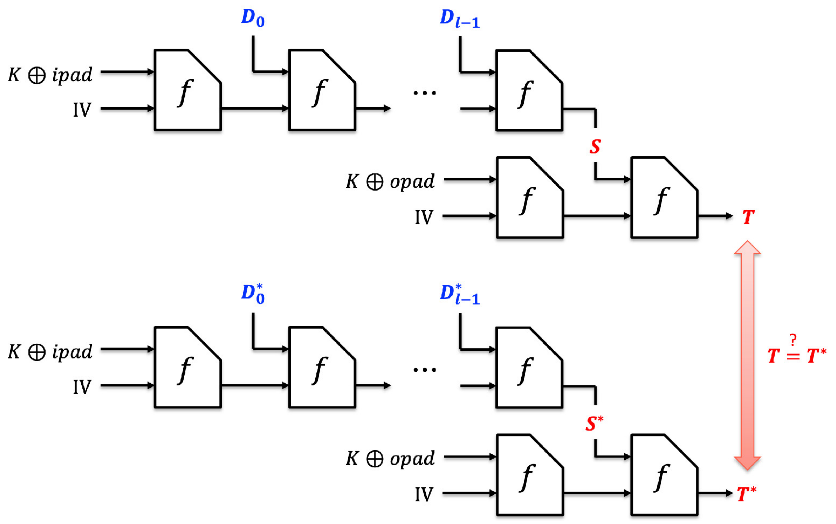

Fig. 3: SCU scenario against AES-CM-HMAC with long tags for the E2EE setting. In this scenario, given a transcript of encryption query (N, A, M, C, T) derived on K, the adversary $\mathcal{A}$ must find a successful forgery (N^*, A^*, C^*, T^*) on K, such that $T^* = T$ and $D^* \neq D$ (*i.e.*, $(N^*, A^*, C^*) \neq (N, A, C)$).

the eSec security of H; thus, we have

$$\begin{aligned}\mathbf{Adv}^{\mathsf{SCU}}_{\mathsf{AES\text{-}CM\text{-}HMAC}}(\mathcal{A}) &\leq \mathbf{Adv}^{\mathsf{eSec}[\leq(\ell')]}_{H}(\mathcal{A}') + \mathbf{Adv}^{\mathsf{eSec}[\leq 1024]}_{H}(\mathcal{A}') \\ &< 2\mathbf{Adv}^{\mathsf{eSec}[\leq(\ell')]}_{H}(\mathcal{A}'),\end{aligned}$$

which concludes the proof. □

Theorem 1 states that the SCU security of AES-CM-HMAC with long tags depends on the security of the underlying hash function. According to the Internet draft [36], SFrame adopts SHA256 as the hash function used in AES-CM-HMAC.

Second-preimage Security of SHA256. Ideally, an n-bit hash function provides an n-bit security level against second-preimage attacks. That is, we can find a second-preimage on SHA256 with a time complexity of 2^{256}. Khovratovich *et al.* [24] proposed a new concept of biclique as a technique for preimage attacks and applied it to the reduced-round SHA2 family. Their second-preimage attack on the reduced-round SHA256 performed up to 45 rounds (out of 64) with a time complexity of $2^{255.5}$ and a memory complexity of 2^6 words. Andreeva *et al.* [2] presented new generic second-preimage attacks on the basic Merkle–Damgård hash functions. Their best attack allowed us to find a second-preimage on the full SHA256 with a time complexity of 2^{173} and a memory complexity of 2^{83}; however, this attack required very long message blocks (*e.g.*, a 2^{118}-block message).

To the best of our knowledge, no study has yet been reported on a second-preimage attack that is more efficient than the above-described attacks; therefore, AES-CM-HMAC with long tags can be considered as the SCU-secure AEAD.

4.4 Impersonation against AES-GCM with Any Long Tags

The abovementioned impersonation attack is a generic attack, and the offline attack complexity depends on the tag length. If we use AES-GCM, a similar attack can easily be mounted without the offline phase because the adversary who owns the GCM key and observes a legitimate GCM output of (N, A, C, T) can create another distinct tuple of (N', A', C', T') with $T' = T$. The remaining $(N', A', C') \neq (N, A, C)$ can be chosen almost freely from the linearity of GHASH and the knowledge of the key. In particular, the attack works with a negligible complexity, irrespective of the tag length unlike the case of AES-CM-HMAC.

Once the adversary intercepts the legitimate tuple (N, A, C, T) created by GCM, it is trivial to compute (N', A', C', T'), such that $T' = T$ and $(N', A', C') \neq (N, A, C)$, for almost any choice of (N', A', C').

For example, suppose GCM with 96-bit nonce and 128-bit tag, which is one of the most typical settings. Given any GCM encryption output tuple (N, A, C, T) with 2-block $C = (C_1, C_2)$ and 1-block $A = A_1$, we have

$$\begin{aligned} T &= \mathsf{GHASH}(L, A \,\|\, C \,\|\, \mathsf{len}(A, C)) \oplus E_K(N \,\|\, 1_{32}) \\ &= A \cdot L^4 \oplus C_1 \cdot L^3 \oplus C_2 \cdot L^2 \oplus \mathsf{len}(A, C) \cdot L \oplus E_K(N \,\|\, 1_{32}), \\ C_1 &= E_K(N \,\|\, 2_{32}) \oplus M_1, \\ C_2 &= E_K(N \,\|\, 3_{32}) \oplus M_2, \end{aligned}$$

where $M = (M_1, M_2)$ is the plaintext. Here, $\mathsf{len}(A, C)$ is a 128-bit encoding of the lengths of A and C, and multiplications are over GF(2^{128}). $E_K(*)$ denotes the encryption by AES with key K and $L = E_K(0^{128})$, and i_{32} for a non-negative integer i denotes the 32-bit encoding of i. It is straightforward to create a valid tuple (N', A', C', T'), such that $T' = T$ and $(N', A', C') \neq (N, A, C)$ because we know K. Say, we first arbitrarily choose N' and A' and the fake plaintext block M'_1 to compute C'_1 and finally set C'_2 such that

$$C'_2 \cdot L^2 = T' \oplus A' \cdot L^4 \oplus C'_1 \cdot L^3 \oplus \mathsf{len}(A', C') \cdot L \oplus E_K(N' \,\|\, 1_{32})$$

holds. This will make the last decrypted plaintext block M'_2 random. It works even if the tag is truncated. That is, the malicious group member can impersonate other members, and the forged plaintext is almost arbitrary, except for the last block. We note that the plaintext is video or audio; hence, a tiny random block will not be recognized. This attack severely harms the integrity of group communication.

This difference from the case of AES-CM-HMAC is rooted in the authentication mechanism. While HMAC maintains a collision resistance once the key is known, GHASH with a known key is a simple function without any sort of known-key security.

Table 1: NIST requirements on the usage of GCM with short tags.

t	32						64					
L	2^1	2^2	2^3	2^4	2^5	2^6	2^{11}	2^{13}	2^{15}	2^{17}	2^{19}	2^{21}
q	2^{22}	2^{20}	2^{18}	2^{15}	2^{13}	2^{11}	2^{32}	2^{29}	2^{26}	2^{23}	2^{20}	2^{17}
c	2^{62}	2^{62}	2^{61}	2^{65}	2^{66}	2^{67}	2^{75}	2^{74}	2^{73}	2^{72}	2^{71}	2^{70}

4.5 Considerations on Authentication Key Recovery

The specification [36] appears to implicitly allow 4- and 8-byte tags with AES-GCM. In addition to the attacks described above, the use of short tags in GCM will lead to a complete recovery of the authentication key (*i.e.*, the key of GHASH) by a class of attacks, called reforging. This leads to a universal forgery.

Ferguson [14] first pointed out this attack, and Mattsson and Westerlund [30] further refined the attack and provided a concrete complexity estimation. According to [30], the security levels are only 62—67 bits and 70–75 bits for the 32-bit and 64-bit tags, respectively, even if we follow the NIST requirements on the usage of GCM with short tags (Table 1). In Table 1, L is the maximum combined length of A and C, and q is the maximum number of invocations of the authenticated decryption function. Table 1 also shows the required data complexity c for the authentication key recovery under each restriction of L and q. For example, for $L = 2^3$ and $q = 2^{18}$, the required data to recover the key of GHASH is 2^{61}.

If no restriction is provided regarding L and q, the authenticated key is recovered with the data complexity of 2^t because the complexity of the first forgery is dominated. Thus, for the 4-byte (= 32-bit) tag length, the authenticated key recovery is feasible with 2^{32} data complexity. The specification [36] seems to not explicitly mention the restrictions of q and L.

Practical Effects on SFrame. Upon checking the available implementations of the original [45], Cisco Webex [7], and Jitsi Meet [22], no restriction is found regarding L and q. In this case, for the 4-byte tag, the authenticated key is recovered with the data complexity of 2^{32}, which is practically made available by a malicious user.

4.6 Recommendations

We recommend the following from the vulnerabilities shown in Sections 4.2 to 4.5:

1. For AES-CM-HMAC, long tags (*e.g.*, 16-byte tags) instead of short tags, especially 4-byte tags, should be used.
2. For AES-GCM, a signature should be computed over not only a tag, but also a whole frame. In addition, the specification should clearly forbid short tags or refer to the NIST requirements on the usage of GCM with short tags.

Algorithm 3 EdDSA signature generation [5,6]

```
Input: $m$: message, $G$: generator, $d$: secret key, $v$: public key
Output: $(R, s)$: signature
 1: $h_0 h_1 \cdots h_{2b-1} = H(d)$                              ▷ $H$: hash function
 2: $a = 2^{b-2} + \sum_{3 \leq i \leq b-3} 2^i h_i$
 3: $r = H(h_b h_{b+1} \cdots h_{b-1}, m)$
 4: $R = rG$
 5: $s = r + H(R, v, m)a \bmod \ell$
```

Algorithm 4 ECDSA signature generation [23]

```
Input: $m$: message, $G$: generator, $n$: large prime number, $d$: secret key
Output: $(r, s)$: signature
 1: $k = [1, n-1]$
 2: $(x, y) = kG$
 3: $r = x \bmod n$
 4: if $r = 0$ then
 5:     go to Step 1.
 6: end if
 7: $s = (H(m) + dr)/k \bmod n$                                   ▷ $H$: hash function
 8: if $s = 0$ then
 9:     go to Step 1.
10: end if
```

3. As discussed in Section 3, switch to other ciphersuites that work as a secure encryptment scheme, such as HFC [10], with a sufficiently long tag is another option.

5 Discussions

This section discusses the advantages of the countermeasures recommended in Section 4.6. We have already described the advantage of the first countermeasure in Section 4.2. We will now discuss the advantages of the second and third countermeasures. We will also confirm the validity of switching to other AEADs, such as AES-CCM [1,17,47], which combines CBC-MAC for message authentication with AES-CTR for frame encryption, and ChaCha20-Poly1305 [34], which combines Poly1305 for message authentication with ChaCha20 for frame encryption.

5.1 Performance Analysis of Signature Generation

This subsection examines the performance when a signature is computed over not only a tag, but also a whole frame. To this end, we first consider the signature algorithms used by SFrame, which are EdDSA over Ed25519 [5,6] and ECDSA over P-521 [15,23]. These signature generation algorithms are briefly described as Algorithms 3 and 4. These algorithms show that a message m (*i.e.*, a tag or a whole frame) is always (a part of) the input to the hash function H. Specifically, in EdDSA over Ed25519 and ECDSA over P-521, the hash function H containing the message m as (a part of) the input is executed twice (Lines 3 and 5 in Algorithm 3) and once (Line 7 in Algorithm 4), respectively. SFrame adopts SHA256 and SHA512 as the underlying hash functions. These hash functions have the following feature: the longer the message length, the more times the compression function is executed, that is, the longer the computational time. We consider herein the maximum length of a whole frame. The maximum length of an RTP packet is 1500 bytes. The RTP packet contains at least a 12-byte RTP header, an 8-byte UDP header, a 20-byte IP header, and a 14-byte Ethernet header. By excluding these headers, the maximum length of a whole frame is 1446 bytes. Therefore, given that the lengths of a tag and a whole frame (*i.e.*, a 4 or 16-byte tag and a 1446-byte frame) are significantly different, we consider that the difference in message lengths greatly influences the computational time for the signature generation.

Based on this consideration, we conducted experiments to compare the computational times for the underlying hash functions, SHA256 and SHA512, using different message block lengths, such as 4-byte tags, 16-byte tags, and 1446-byte frames. The following is our experimental environment: a macOS version 11.6 machine with 2.8 GHz CPU and 16.0 GB of main memory. In our experiments, we used the openssl command with the speed and -bytes options as shown in the following example:

```
openssl speed -bytes 4 sha256
```

We measured the computational time for SHA256 using 4-byte message blocks as the input. The publicly available source code [7] shows that the ciphersuites in SFrame are implemented using openssl; hence, we consider that the performance analysis using the openssl command is meaningful. After running the above command, we obtain its experimental result as follows:

```
Doing for 3s on 4 size blocks: 8875740 sha256's in 3.00s
```

This means that SHA256 using 4-byte message blocks has executed 8,875,740 times in 3 seconds. From the result, we can calculate the computational time for SHA256 using 4-byte message blocks per once as

$$3.0/8875740 = 0.0000003379\ldots \text{ (s)} \approx 0.338 \text{ (μs)}.$$

Similarly, we calculate the computational times for SHA256 and SHA512 using several different message block lengths. Given that a signature is computed over a list of multiple tags, the number of tags or frames in the list should be considered. Let x denote the number of tags or frames in the list and decide to use $x \in \{1, 2, 4, 8, 16, 32\}$ to compare the computational times for SHA256 and SHA512 using different message block lengths. Tables 2 and 3 show a comparison of the computational times for SHA256 and SHA512, using lists of 4-byte tags, lists of 16-byte tags, and lists of 1446-byte frames as the input message blocks, respectively. From these tables, we clarify the correctness of our assumption that

Table 2: Comparison of the computational times for SHA256 using lists of 4-byte tags, lists of 16-byte tags, and lists of 1446-byte frames as the input message blocks. x denotes the number of tags or frames in the lists.

x	4-byte tags		16-byte tags		1446-byte frames	
	Length (bytes)	Time (μs)	Length (bytes)	Time (μs)	Length (bytes)	Time (μs)
1	4	0.338	16	0.342	1446	2.762
2	8	0.338	32	0.346	2892	5.287
4	16	0.342	64	0.469	5784	10.130
8	32	0.346	128	0.554	11568	20.189
16	64	0.469	256	0.810	23136	39.881
32	128	0.554	512	1.234	46272	80.874

Table 3: Comparison of the computational times for SHA512 using lists of 4-byte tags, lists of 16-byte tags, and lists of 1446-byte frames as the input message blocks. x denotes the number of tags or frames in the lists.

x	4-byte tags		16-byte tags		1446-byte frames	
	Length (bytes)	Time (μs)	Length (bytes)	Time (μs)	Length (bytes)	Time (μs)
1	4	0.406	16	0.407	1446	2.074
2	8	0.405	32	0.406	2892	3.705
4	16	0.407	64	0.412	5784	7.133
8	32	0.406	128	0.603	11568	13.792
16	64	0.412	256	0.726	23136	27.403
32	128	0.603	512	1.018	46272	53.687

the difference in message lengths greatly influences the computational time for the signature generation. Note that we express the case of SHA512 in parentheses.

- When $x = 1$, the computational times for the EdDSA and ECDSA signature generations with SHA256 (SHA512) as the underlying hash function are approximately 4.840 (3.334) and 2.420 (1.667) microseconds slower when using a 1446-byte frame than when using a 16-byte tag as the input. x is the minimum value; hence, these are the minimum differences in terms of the influence on the computational time for the signature generation.
- When $x = 32$, the computational times for the EdDSA and ECDSA signature generations with SHA256 (SHA512) as the underlying hash function are approximately 159.280 (105.338) and 79.640 (52.669) microseconds slower when using a list of 1446-byte frames than when using a list of 16-byte tags

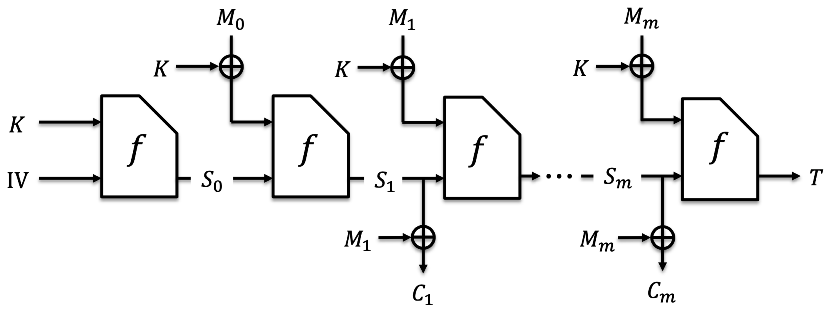

Fig. 4: Overall structure of the HFC scheme for a 1-block header $M_0 = (N, A)$ and an m-block plaintext $M = (M_1, \ldots, M_m)$, where f denotes the SHA256 or SHA512 compression function; K denotes a secret key; IV denotes a fixed constant value called an initialization vector; N denotes a nonce; A denotes an additional associated data; $S_0, \ldots, S_m$ denotes intermediate values; $C = (C_1, \ldots, C_m)$ denotes an m-block ciphertext; and T denotes a tag.

as the input, respectively. The differences in the computational time for the signature generation further increase as the number of 16-byte tags or 1446-byte frames in the list is increased.

As described in Section 4.4, we have clarified that AES-GCM with any long tags is vulnerable against an impersonation attack by a malicious group member. Therefore, if AES-GCM is used for frame encryption, we recommended that a signature is computed over a whole frame and not only a tag in terms of its security. In contrast, as discussed above, the SFrame implementors should be aware that this countermeasure has disadvantage in terms of efficiency.

5.2 Security Analysis of HFC under SFrame

This subsection discusses the security of HFC with long tags in the E2EE setting. HFC was proposed by Dodis *et al.* at CRYPTO 2018 [10] as a secure encryptment scheme for message franking. The overall structure of the HFC is shown in Figure 4, where f denotes the SHA256 or SHA512 compression function; K denotes a secret key; IV denotes a fixed constant value; M_0 denotes a 1-block header; $M = (M_1, \ldots, M_m)$ denotes an m-block plaintext; $S_1, \ldots, S_m$ denotes intermediate values; $C = (C_1, \ldots, C_m)$ denotes an m-block ciphertext; and T denotes a tag (refer to Section 6 in [10] for more details). For simplicity, we assume that the 1-block header M_0 contains a nonce N and an additional associated data A.

According to the existing study [10], Dodis *et al.* have already proven the SCU security of HFC in the message franking setting. However, due to some differences between message franking and E2EE settings, we consider that their proof cannot directly be applied to the proof of the SCU security of HFC in the

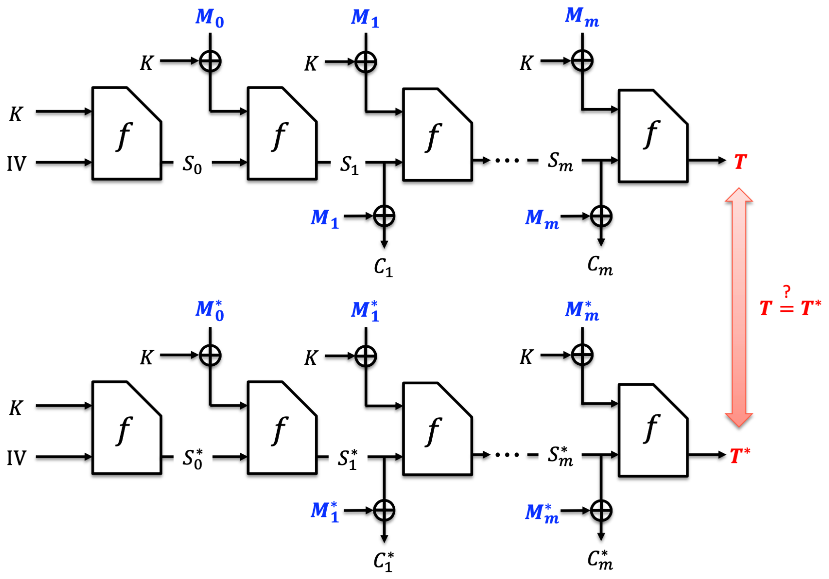

Fig. 5: SCU scenario against HFC with long tags for the E2EE setting. In this scenario, given a transcript of the encryption query (N, A, M, C, T) derived on K, the adversary $\mathcal{A}$ must find a successful forgery (N^*, A^*, M^*, T^*) on K, such that $T^* = T$ and $(N^*, A^*, M^*) \neq (N, A, M)$.

E2EE setting. Thus, we prove the SCU security of HFC with long tags in the E2EE setting through the following theorem.

Theorem 2. *Let $\mathcal{A}$ be the SCU adversary against HFC, with the target encryption output being at most ℓ bits. Then, SCU advantage of $\mathcal{A}$ against HFC is bounded as*

$$\mathbf{Adv}^{\mathsf{SCU}}_{\mathsf{HFC}}(\mathcal{A}) = \mathbf{Adv}^{\mathsf{eSec}[\leq(\ell')]}_{H}(\mathcal{A}')$$

for some eSec adversary $\mathcal{A}'$ against H, which denotes the underlying SHA256 hash function, where $\ell' = \ell + 512$ (i.e., one block larger).

Proof. The proof itself is similar to Theorem 1. We can assume that the adversary is given the key K of HFC in the E2EE setting. The resulting game is that, given a transcript of encryption query (N, A, M, C, T) derived on K, the adversary must find a successful forgery (N^*, A^*, M^*, T) on K, such that $(N^*, A^*, M^*) \neq (N, A, M)$. Note that the tag T is the output of HFC taking K and (N, A, M); thus, the ciphertext C is not needed in the attack. Figure 5 illustrates this scenario. The last block may need padding, but we simply ignore this (the analysis is pretty much the same).

Figure 4 depicts that the tag generation by HFC has the same Merkle–Damgård construction as SHA256; thus, if f is the SHA256 compression function, the tag generation by HFC can be regarded as equivalent to the SHA256 hash function, that is, $H(K \parallel M_0 \oplus K, \ldots, M_m \oplus K)$, where H denotes the SHA256 hash function. Therefore, in this scenario, the adversary must find a second preimage against the target input $M_0 \oplus K, \ldots, M_m \oplus K$ ($m+1$-block, thus $\ell + 512$ for SHA256) for $T = T^*$. This case is covered by the eSec security of H; thus, we have

$$\mathbf{Adv}_{\mathsf{HFC}}^{\mathsf{SCU}}(\mathcal{A}) = \mathbf{Adv}_{H}^{\mathsf{eSec}[\leq(\ell')]}(\mathcal{A}'),$$

which concludes the proof. □

Theorem 2 states that the SCU security of HFC with long tags depends on the security of the underlying hash function, such as SHA256. As discussed in Section 4.3, SHA256 has an eSec resistance; therefore, the HFC with long tags can be considered as the SCU-secure AEAD.

5.3 Performance Analysis of HFC

This subsection explores the HFC performance. To this end, by using the same experimental environment and tool (*i.e.*, the openssl command) described in Section 5.1, we conducted experiments to compare the computational times for HFC, AES-GCM, AES-CM-HMAC, and other AEADs, such as AES-CCM [1,17, 47] and ChaCha20-Poly1305 [34] that can be used with openssl. As described in Section 5.1, the publicly available source code [7] shows that the ciphersuites in SFrame are implemented using openssl. Therefore, if AES-CCM and ChaCha20-Poly1305 are superior in terms of performance and security in the E2EE setting, they may be hopeful ciphersuite candidates in SFrame.

We now have an issue to measure the computational times for the target AEADs. We can easily measure the computational times for AES-GCM, AES-CCM, and ChaCha20-Poly1305 with the openssl command, but cannot accurately measure the computational times for HCF and AES-CM-HMAC because they are not supported with openssl. To resolve this issue, we use SHA256 and HMAC-SHA256 instead of HFC and AES-CM-HMAC, respectively. As discussed in the proof of Theorem 2, we consider that the computational time for HFC can be regarded as equivalent to that of SHA256 because the tag generation by HFC can be regarded as equivalent to the SHA256 hash function. Moreover, we consider that the computational time for AES-CM-HMAC can be regarded as almost equivalent to that of HMAC-SHA256 because the computational time for AES-CM-HMAC significantly depends on that of HMAC-SHA256. In fact, when the input length is 1024 bytes, the computational times for AES-CTR and HMAC-SHA256 are 0.160 and 2.281 microseconds, respectively.

Table 4 presents a comparison of the computational times for HFC (actually, SHA256), AES-CM-HMAC (actually, HMAC-SHA256), AES-GCM, AES-CCM, and ChaCha20-Poly1305 using several different input message blocks. The table shows only a slight difference in the computational times between HFC and AES-CM-HMAC. HFC can be regarded as almost equivalent to AES-CM-HMAC in terms of performance and security; thus, the use of HFC with long tags in SFrame

Table 4: Comparison of the computational times for the target AEADs using several different input message blocks.

Input	Computational time (μs)				
length (bytes)	HFC	AES-CM-HMAC	AES-GCM	AES-CCM	ChaCha20-Poly1305
128	0.663	0.669	0.067	0.163	0.130
256	0.891	0.906	0.100	0.273	0.188
512	1.352	1.365	0.162	0.491	0.221
1024	2.284	2.281	0.260	0.936	0.408
2048	4.128	4.185	0.473	1.806	0.790
4096	7.834	8.022	0.890	3.587	1.554

is not a problem. Focusing on AES-GCM, its computational time is the fastest among the target AEADs. It should be noted, however, that it has a disadvantage in the computational time for signature generation when a signature is computed over a whole frame, as discussed in Section 5.1. In addition, the computational times for AES-CCM and ChaCha20-Poly1305 are more superior than those for HFC and AES-CM-HMAC; thus, they may be hopeful candidates for the ciphersuites in SFrame because the use of AES-CCM and ChaCha20-Poly1305 in SFrame is not a problem in terms of performance. The next subsection will discuss the security of AES-CCM and ChaCha20-Poly1305 in the E2EE setting.

5.4 Considerations on switching to other AEADs

This subsection discusses the security of AES-CCM [1, 17, 47] and ChaCha20-Poly1305 [34] in the E2EE setting. To conclude first, it is easy to perform the similar attack described in Section 4.4 against these AEADs with any long tags. An impersonation attack against ChaCha20-Poly1305 with any long tags can be performed in almost the same procedure as the case of AES-GCM in Section 4.4. Therefore, this subsection shows an impersonation attack against AES-CCM with any long tags.

The CCM encryption first executes the formatting function described in NIST SP 800-38C [1, Appendix A] to the input tuple (N, A, M) to produce the encoded blocks $B = (B_0, \ldots, B_r)$. Note that this formatting function is reversible, that is, the adversary obtains the legitimate input tuple (N, A, M) from the encoded blocks $B = (B_0, \ldots, B_r)$. In addition, N, A, and M are individually encoded (refer to [1] for more details).

As with the case of GCM described in Section 4.4, once the adversary who owns the key K intercepts a legitimate tuple (N, A, C, T) created by CCM, it is trivial to compute (N', A', C', T'), such that $T' = T$ and $(N', A', C') \neq (N, A, C)$, for almost any choice of (N', A', C').

For example, let $B = (B_0, B_1, B_2, B_3)$ be the encoded data corresponding to the legitimate input tuple (N, A, M) with 1-block nonce N, 1-block additional associated data A, and 2-block plaintext $M = (M_1, M_2)$. In this case, given any CCM encryption output tuple (N, A, C, T), we have

$$\begin{aligned} T &= E_K(E_K(E_K(E_K(B_0) \oplus B_1) \oplus B_2) \oplus B_3) \oplus E_K(ctr_0), \\ C_1 &= E_K(ctr_1) \oplus M_1, \\ C_2 &= E_K(ctr_2) \oplus M_2, \end{aligned}$$

where ctr_0, ctr_1, and ctr_2 are the counter values. Here, $E_K(*)$ denotes the encryption by AES with key K. It is straightforward to create a valid tuple (N', A', C', T'), such that $T' = T$ and $(N', A', C') \neq (N, A, C)$ (*i.e.*, $B' \neq B$) because we know K. Say, we first arbitrary choose a part of the encoded input tuple (B'_0, B'_1, B'_2) corresponding to a part of the fake input tuple (N', A', M'_1), and then set B'_3 corresponding to M'_2 such that

$$B'_3 = E_K^{-1}(T' \oplus E_K(ctr_0)) \oplus E_K(E_K(E_K(B'_0) \oplus B'_1) \oplus B'_2)$$

holds. Here, $E_K^{-1}(*)$ denotes the decryption by AES with key K. This will make the last decrypted plaintext block M'_2 randomly. It works even if the tag is truncated. Finally, we obtain the input tuple (N', A', M') from the encoded blocks $B' = (B'_0, B'_1, B'_2, B'_3)$ and compute

$$\begin{aligned} C'_1 &= E_K(ctr_1) \oplus M'_1, \\ C'_2 &= E_K(ctr_2) \oplus M'_2. \end{aligned}$$

That is, as with the case of GCM, the malicious group member can impersonate other members and the forged plaintext is almost arbitrary, except for the last block.

In summary, as with the case of GHASH, CBC-MAC and Poly1305 with a known key are also simple functions without any sort of known-key security; therefore, we conclude that AEADs with a simple function in the known key setting (*e.g.*, AES-GCM [1,17], AES-CCM [1,17,47], ChaCha20-Poly1305 [34], OCB [28,40,41], Deoxys [20,21], etc.) should not be candidates for the ciphersuites in the E2EE applications unless a signature is computed over a whole frame.

6 Conclusions

We showed herein our security analysis on SFrame, a recently proposed end-to-end encryption mechanism built on RTC, developed by Google and CoSMo Software and proposed to the IETF. SFrame is a young project, which will be adopted by a number of real-world products. Our results showed a practical risk of impersonation by a malicious group member. This problem is caused by the digital signature computed only on (a list of) AEAD tags, and the attack becomes practical when tags are short or when the used AEAD algorithm allows the creation of a collision on tags with the knowledge of the key. The former applies to the case of AES-CM-HMAC, while the latter applies to the case of AES-GCM. We also showed that AES-CM-HMAC with a long tag avoids this problem as it fulfills a “committing” property introduced by Dodis *et al.* [10]. Moreover, if correctly used by the upper layer, AES-CM-HMAC is a provably secure AEAD because it can be interpreted as a standard encryption-then-MAC generic composition. We notified our findings to the designers, and they acknowledged them and revised the specification, including the removal of the signature feature and a patch for the AEAD algorithm. Considering its quick deployment, we think SFrame should be more actively studied. We also hope that our work could help in its improvement.

Acknowledgments

We are grateful to the SFrame designers (Emad Omara, Justin Uberti, Alex Gouaillard, and Sergio Garcia Murillo) for the fruitful discussion and feedback about our findings. We would like to thank Shiguredo Inc. for helpful discussion about real-world applications of the end-to-end encryption. Takanori Isobe is supported by JST, PRESTO Grant Number JPMJPR2031, Grant-in-Aid for Scientific Research (B)(KAKENHI 19H02141) and SECOM science and technology foundation.

References

1. NIST SP 800-38C, Recommendation for Block Cipher Modes of Operation: the CCM Mode for Authentication and Confidentiality (2007), U.S.Department of Commerce/National Institute of Standards and Technology
2. Andreeva, E., Bouillaguet, C., Dunkelman, O., Fouque, P., Hoch, J.J., Kelsey, J., Shamir, A., Zimmer, S.: New Second-Preimage Attacks on Hash Functions. J. Cryptol. **29**(4), 657–696 (2016)
3. Barnes, R., Beurdouche, B., Millican, J., Omara, E., Cohn-Gordon, K., Robert, R.: The Messaging Layer Security (MLS) Protocol. `https://tools.ietf.org/html/draft-ietf-mls-protocol-10` (October 2020)
4. Bellare, M., Namprempre, C.: Authenticated Encryption: Relations among Notions and Analysis of the Generic Composition Paradigm. In: Okamoto, T. (ed.) Advances in Cryptology - ASIACRYPT 2000, 6th International Conference on the Theory and Application of Cryptology and Information Security, Kyoto, Japan, December 3-7, 2000, Proceedings. Lecture Notes in Computer Science, vol. 1976, pp. 531–545. Springer (2000). https://doi.org/10.1007/3-540-44448-3_41, `https://doi.org/10.1007/3-540-44448-3_41`
5. Bernstein, D.J., Duif, N., Lange, T., Schwabe, P., Yang, B.: High-Speed High-Security Signatures. In: Preneel, B., Takagi, T. (eds.) Cryptographic Hardware and Embedded Systems - CHES 2011 - 13th International Workshop, Nara, Japan, September 28 - October 1, 2011. Proceedings. Lecture Notes in Computer Science, vol. 6917, pp. 124–142. Springer (2011). https://doi.org/10.1007/978-3-642-23951-9_9, `https://doi.org/10.1007/978-3-642-23951-9_9`

6. Bernstein, D.J., Duif, N., Lange, T., Schwabe, P., Yang, B.: High-speed high-security signatures. J. Cryptogr. Eng. **2**(2), 77–89 (2012). https://doi.org/10.1007/s13389-012-0027-1, `https://doi.org/10.1007/s13389-012-0027-1`
7. Cisco Systems: SFrame (2020), `https://github.com/cisco/sframe`
8. Cisco Systems: Zero-Trust Security for Webex White Paper (2021), `https://www.cisco.com/c/en/us/solutions/collateral/collaboration/white-paper-c11-744553.pdf`
9. Cohn-Gordon, K., Cremers, C., Dowling, B., Garratt, L., Stebila, D.: A Formal Security Analysis of the Signal Messaging Protocol. J. Cryptol. **33**(4), 1914–1983 (2020)
10. Dodis, Y., Grubbs, P., Ristenpart, T., Woodage, J.: Fast Message Franking: From Invisible Salamanders to Encryptment. In: Shacham, H., Boldyreva, A. (eds.) CRYPTO 2018. LNCS, vol. 10991, pp. 155–186. Springer (2018)
11. Dworkin, M.:
12. Emad Omara: Extend Tag Calculation to Cover Nonce #59 (2021), `https://github.com/eomara/sframe/pull/59`
13. Emad Omara: Remove Signature #58 (2021), `https://github.com/eomara/sframe/pull/58`
14. Ferguson, N.: Authentication Weaknesses in GCM. Comments submitted to NIST Modes of Operation Process (2005), `http://csrc.nist.gov/groups/ST/toolkit/BCM/documents/comments/CWC-GCM/Ferguson2.pdf`
15. Gallagher, P.: Digital signature standard (DSS). Federal Information Processing Standards Publications, volume FIPS **186** (2013)
16. Garman, C., Green, M., Kaptchuk, G., Miers, I., Rushanan, M.: Dancing on the Lip of the Volcano: Chosen Ciphertext Attacks on Apple iMessage. In: Holz, T., Savage, S. (eds.) USENIX Security 2016. pp. 655–672. USENIX Association (2016)
17. Housley, R.: Using AES-CCM and AES-GCM Authenticated Encryption in the Cryptographic Message Syntax (CMS). RFC **5084**, 1–11 (2007). https://doi.org/10.17487/RFC5084, `https://doi.org/10.17487/RFC5084`
18. Isobe, T., Ito, R.: Security Analysis of End-to-End Encryption for Zoom Meetings. IEEE Access **9**, 90677–90689 (2021)
19. Isobe, T., Minematsu, K.: Breaking message integrity of an end-to-end encryption scheme of LINE. In: López, J., Zhou, J., Soriano, M. (eds.) ESORICS 2018. LNCS, vol. 11099, pp. 249–268. Springer (2018)
20. Jean, J., Nikolic, I., Peyrin, T.: Tweaks and Keys for Block Ciphers: The TWEAKEY Framework. In: ASIACRYPT (2). Lecture Notes in Computer Science, vol. 8874, pp. 274–288. Springer (2014)
21. Jean, J., Nikolic, I., Peyrin, T., Seurin, Y.: The Deoxys AEAD Family. J. Cryptol. **34**(3), 31 (2021). https://doi.org/10.1007/s00145-021-09397-w, `https://doi.org/10.1007/s00145-021-09397-w`
22. Jitsi: Jitsi Meet API library (2020), `https://github.com/jitsi/lib-jitsi-meet/`
23. Johnson, D., Menezes, A., Vanstone, S.A.: The Elliptic Curve Digital Signature Algorithm (ECDSA). Int. J. Inf. Sec. **1**(1), 36–63 (2001). https://doi.org/10.1007/s102070100002, `https://doi.org/10.1007/s102070100002`
24. Khovratovich, D., Rechberger, C., Savelieva, A.: Bicliques for Preimages: Attacks on Skein-512 and the SHA-2 Family. In: Canteaut, A. (ed.) FSE 2012. LNCS, vol. 7549, pp. 244–263. Springer (2012)
25. Knodel, M., Baker, F., Kolkman, O., Celi, S., Grover, G.: Definition of End-to-end Encryption. `https://datatracker.ietf.org/doc/draft-knodel-e2ee-definition/` (February 2021)

26. Krawczyk, H.: The Order of Encryption and Authentication for Protecting Communications (or: How Secure Is SSL?). In: Kilian, J. (ed.) CRYPTO 2001. LNCS, vol. 2139, pp. 310–331. Springer (2001)
27. Krawczyk, H., Eronen, P.: HMAC-based Extract-and-Expand Key Derivation Function (HKDF). Internet Engineering Task Force - IETF, Request for Comments **5869** (May 2010)
28. Krovetz, T., Rogaway, P.: The Software Performance of Authenticated-Encryption Modes. In: FSE. Lecture Notes in Computer Science, vol. 6733, pp. 306–327. Springer (2011)
29. Matrix.org Foundation.: Olm: A Cryptographic Ratchet (2016), `https://gitlab.matrix.org/matrix-org/olm/-/blob/master/docs/olm.md`
30. Mattsson, J., Westerlund, M.: Authentication Key Recovery on Galois/Counter Mode (GCM). In: Pointcheval, D., Nitaj, A., Rachidi, T. (eds.) AFRICACRYPT 2016. LNCS, vol. 9646, pp. 127–143. Springer (2016)
31. McGrew, D.A.: An Interface and Algorithms for Authenticated Encryption. Internet Engineering Task Force - IETF, Request for Comments **5116** (January 2008)
32. Menezes, A.J., Oorschot, P.C.V., Vanstone, S.A.: Handbook of Applied Cryptography. CRC press (1996)
33. 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)
34. Nir, Y., Langley, A.: ChaCha20 and Poly1305 for IETF Protocols. RFC **8439**, 1–46 (2018). https://doi.org/10.17487/RFC8439, `https://doi.org/10.17487/RFC8439`
35. Omara, E.: Google Duo End-to-End Encryption Overview - Technical Paper (2020), `https://www.gstatic.com/duo/papers/duo_e2ee.pdf`
36. Omara, E., Uberti, J., Gouaillard, A., Murillo, S.G.: Secure Frame (SFrame). `https://tools.ietf.org/html/draft-omara-sframe-01` (November 2020)
37. Omara, E., Uberti, J., Gouaillard, A., Murillo, S.G.: Secure Frame (SFrame). `https://tools.ietf.org/html/draft-omara-sframe-02` (March 2021)
38. Open Whisper Systems.: Signal Github Repository (2017), `https://github.com/WhisperSystems/`
39. Rogaway, P.: Authenticated-encryption with associated-data. In: Atluri, V. (ed.) Proceedings of the 9th ACM Conference on Computer and Communications Security, CCS 2002, Washington, DC, USA, November 18-22, 2002. pp. 98–107. ACM (2002). https://doi.org/10.1145/586110.586125, `https://doi.org/10.1145/586110.586125`
40. Rogaway, P.: Efficient Instantiations of Tweakable Blockciphers and Refinements to Modes OCB and PMAC. In: ASIACRYPT. pp. 16–31 (2004)
41. Rogaway, P., Bellare, M., Black, J., Krovetz, T.: OCB: a block-cipher mode of operation for efficient authenticated encryption. In: ACM Conference on Computer and Communications Security. pp. 196–205. ACM (2001)
42. Rogaway, P., Shrimpton, T.: Cryptographic Hash-Function Basics: Definitions, Implications, and Separations for Preimage Resistance, Second-Preimage Resistance, and Collision Resistance. In: Roy, B.K., Meier, W. (eds.) FSE 2004. LNCS, vol. 3017, pp. 371–388. Springer (2004)
43. Rösler, P., Mainka, C., Schwenk, J.: More is Less: On the End-to-End Security of Group Chats in Signal, WhatsApp, and Threema. In: 2018 IEEE European Symposium on Security and Privacy (EuroS&P). pp. 415–429. IEEE (2018)
44. SaÍ Ibarra Corretgé: The road to End-to-End Encryption in Jitsi Meet (2021), `https://fosdem.org/2021/schedule/event/e2ee/attachments/slides/4435/export/events/attachments/e2ee/slides/4435/E2EE.pdf`

45. Sergio Garcia Murillo: SFrame.js (2020), `https://github.com/medooze/sframe`
46. Turner, J.M.: The Keyed-Hash Message Authentication Code (HMAC). Federal Information Processing Standards Publication **198**, 1 (2008)
47. Whiting, D., Housley, R., Ferguson, N.: Counter with CBC-MAC (CCM). RFC **3610**, 1–26 (2003). https://doi.org/10.17487/RFC3610, `https://doi.org/10.17487/RFC3610`