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 the security of SFrame, an E2EE mechanism recently proposed to IETF for video/audio group communications over the Internet. Although a quite recent project, SFrame is going to be adopted by a number of real-world applications. We inspected the original specification of SFrame. We found a critical issue that will lead to an impersonation (forgery) attack by a malicious group member with a practical complexity. We also investigated the several publicly-available SFrame implementations, and confirmed 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 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 evidences of massive surveillance, most notably by the case of Snowden, E2EE has received significant attentions and deemed as a key feature to protect users' privacy and integrity for a wide range of communication applications. This also holds for the video calling/meeting applications, such as Zoom[5] or Webex[6]. 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 [6, 12, 13, 14, 31].

[5] https://zoom.us/
[6] https://www.webex.com

In this article, we study SFrame, which is one such approach aiming to providing E2EE over the Internet. Technically, it is a mechanism to encrypt RTC (Real-Time Communication) 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 is carefully designed to suppress communication overheads that would be introduced when E2EE is deployed. It was proposed to IETF by a team of Google and CoSMo Software (Omara, Uberti, Gouaillard and Murillo) at 2020 as a form of Internet draft [26]. Although a quite recent proposal, it quickly gains lots of attentions. 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 [25], Cisco Webex [4, 5], and Jitsi Meet [15, 32].

1.1 Our Contributions.

We looked into the original specification of SFrame [26], 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 [26] defines two AEAD algorithms, namely a generic composition of AES-CTR and HMAC-SHA256, dubbed AES-CM-HMAC, and AES-GCM for encryption of video/audio packets. We 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, and 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. The following shows the overview of our security analysis.

AEAD security. In Section 4.1, we study the classical AEAD security (namely, confidentiality and integrity) of 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 encryption schemes defined by SFrame are provably secure in the context of 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 very short tag length. Since the malicious group member owns a shared group key, she can precompute multiple ciphertext/tag pairs from any input set, and store them into a precomputation table. After that, 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 second-ciphertext unforgeability (SCU) security, which was defined by Dodis *et al.* [7], and SCU security covers the class of impersonation attacks described above, i.e., forging a ciphertext using the knowledge of the secret key so 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. Since SHA256 has an everywhere second-preimage resistance, which was defined by Rogaway and Shrimpton [30], 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 show an impersonation attack on AES-GCM with any long tags by a malicious group member. This attack exploits a vulnerability of the linearity of GHASH function 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 is able to create another distinct set with the same tag. The remaining value in this set, excluding the tag, can be chosen almost freely from the linearity of GHASH function 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 there is no restriction regarding the NIST requirements on the usage of GCM with short tags. Actually, available implementations of the original [33], Cisco Webex [4], and Jitsi Meet [15] have no restriction 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 by the adversary.

Our results are based solely on the Internet draft [26] and publicly available source code [4,15,33], 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 [26] has actually been deployed. Accordingly, instead of implementing the proposed attacks, we discussed with the designers to confirm the feasibility of the proposed attacks.

Since the specification remains abstract at some points and may be subject to change, besides the real-world implementation often do not strictly follow what was specified in [26], this issue does not immediately mean 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 improvement of the current SFrame specification.

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 [10] and extend tag calculation to cover nonces [9], and updated the specification in the Internet draft on March 29, 2021 [27]. They have a plan to review the SFrame specification and support signature mechanism again in the future.

Organization of the Paper. The paper is organized as follows. Section 2 provides the specification of SFrame including the underlying AEAD, and also a brief survey on the publicly available implementations of SFrame. Section 3 describes the security goals of E2EE recently proposed. We present our analysis in Section 4 which shows impersonation attacks against SFrame. Several other observations are also made, followed by our recommendations. Section 5 concludes the article.

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 which mediates communication between users. They are connected via the server, and communication between a user and the server is protected by a standard Internet client-server encryption protocol, specifically Datagram Transport Layer Security-Secure Real-time Transport Protocol (DTLS-SRTP).

SFrame is specified in the Internet draft [26]. However, it does not specify the key exchange protocol between the parties and the choice is left to the implementors. In practice Signal protocol [28], Olm protocol [20], or Message Layer Security (MLS) protocol [2] 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 [26]. 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.

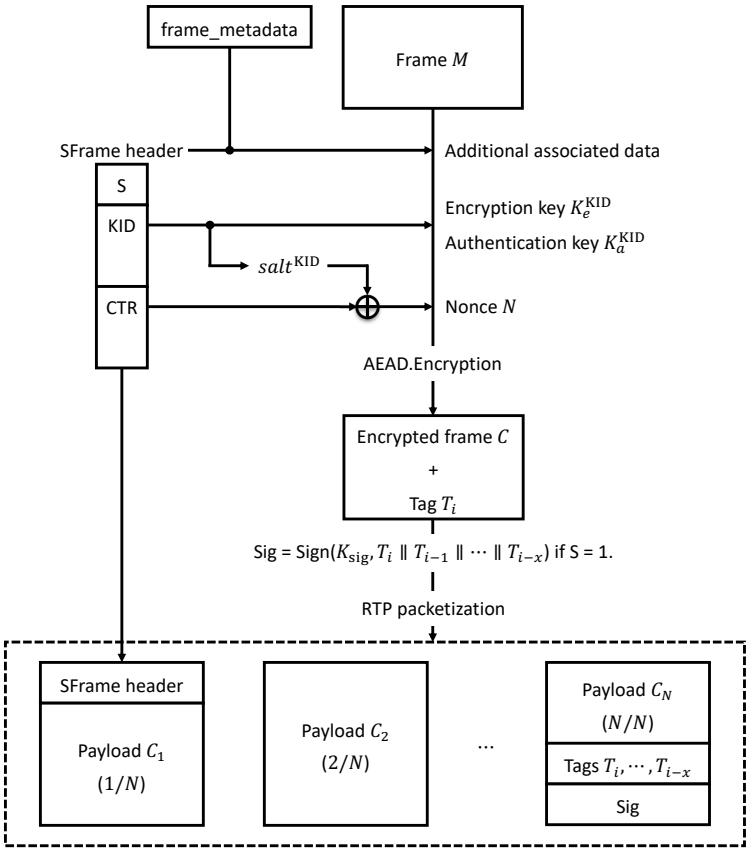

Fig. 1: Media frame encryption flow.

- An authenticated encryption with associated data (AEAD) [22, 29] used for frame encryption, e.g., AES-GCM and AES-CM-HMAC. The authentication tag may be truncated.
- An optional signature algorithm, e.g., EdDSA over Ed25519 and ECDSA over P-521.

Specifically, the original specification [26] defines the following symmetric-key primitives for the ciphersuite:

- AES-GCM with a 128- or 256-bit key and no specified tag length.
- 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.

Fig. 1 and Alg. 1 show the media frame encryption flow in an E2EE session for SFrame using the above ciphersuites. When AES-GCM is adopted as the ciphersuite, AEAD.Encryption in Alg. 1 is executed according to NIST SP

Algorithm 1 Media frame encryption scheme

Input: S: signature flag, KID: key ID, CTR: counter value, $\mathsf{frame\_metadata}$: frame metadata, M: frame
Output: C: encrypted frame, T: authentication tag

procedure ENCRYPTION(S, KID, CTR, $\mathsf{frame\_metadata}$, M)
  if An AEAD encryption algorithm is AES-GCM **then**
    $K_e^{\mathsf{KID}}, salt^{\mathsf{KID}} = \mathsf{KeyStore}[\mathsf{KID}]$
  else
    $K_e^{\mathsf{KID}}, K_a^{\mathsf{KID}}, salt^{\mathsf{KID}} = \mathsf{KeyStore}[\mathsf{KID}]$
  end if
  $ctr = \mathsf{encode}(\mathsf{CTR}, \mathrm{NonceLen})$ ▷ encode CTR as a big-endian of NonceLen.
  $N = salt^{\mathsf{KID}} \oplus ctr$ ▷ N is a Nonce.
  $\mathsf{header} = \mathsf{encode}(\mathsf{S}, \mathsf{KID}, \mathsf{CTR})$
  $\mathsf{aad} = \mathsf{header} + \mathsf{frame\_metadata}$ ▷ aad is an additional associated data.
  if an AEAD encryption algorithm is AES-GCM **then**
    $C, T = \mathrm{AEAD.ENCRYPTION}(K_e^{\mathsf{KID}}, N, \mathsf{aad}, M)$
  else
    $C, T = \mathrm{AEAD.ENCRYPTION}(K_e^{\mathsf{KID}}, K_a^{\mathsf{KID}}, N, \mathsf{aad}, M)$
  end if
end procedure

800-38D [8]. Before performing by the AEAD encryption procedure by AES-GCM, HKDF [19] is used to generate the encryption key K_e^{KID} and the salt $salt^{\mathsf{KID}}$ for encrypting/decrypting media frames as follows:

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

where KeyLen and NonceLen are the length (byte) of an encryption key and a nonce for the encryption algorithm, respectively. Then, each user 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 Alg. 1 is executed according to Alg. 2. Before performing AES-CM-HMAC, HKDF [19] is used as well as the case of AES-GCM, however in a slightly different manner:

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

where HashLen is the output length (byte) of the hash function. Also, each user stores the encryption key K_e^{KID}, the authentication key K_a^{KID}, and the 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 to detect forgeries by an entity who does not own $K_{\mathsf{base}}^{\mathsf{KID}}$, it does not prevent from an impersonation by a malicious group member

Algorithm 2 AEAD encryption by AES-CM-HMAC

Input: K_a^{KID}: authentication key, aad: additional associated data, C: encrypted frame
Output: T: truncated authentication tag
1: **procedure** TAG.GENERATION(K_a^{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: $T = \mathsf{trancate}(tag, \text{TagLen})$
6: **end procedure**

Input: K_e^{KID}, K_a^{KID}, N: Nonce, aad, M: frame
Output: C, T
1: **procedure** AEAD.ENCRYPTION(K_e^{KID}, K_a^{KID}, N, aad, M)
2: $C =$ AES-CTR.ENCRYPTION(K_e^{KID}, N, M)
3: $T =$ TAG.GENERATION(K_a^{KID}, aad, C)
4: **end procedure**

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 input length of signature computations. Namely a signature Sig is computed over a list of authentication tags with a fixed size, $(T_i, T_{i-1}, \ldots, T_{i-x})$, as follows:

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

where Sign denotes the signature function. This signature is appended to the end of the data which consists of SFrame header, the current encrypted payload, its corresponding authentication tag T_i, and the list of authentication tags $(T_{i-1}, \ldots, T_{i-x})$ which correspond to the previously encrypted payload so 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 [26] and exhibit some varieties. In this article, we particularly focus on the specified AEAD schemes and the allowed tag length in each of the implementation since this determines the complexity of our attack.

The original. There is a Javascript implementation by one of the designers of SFrame (Sergio Garcia Murillo) [33]. It is based on webcrypt. In his implementation, it supports

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

Google Duo. Duo[7] is a video calling application developed by Google. For group calling, it adopts Signal protocol as a key exchange mechanism and SFrame as a E2EE mechanism. There is a technical paper [25] written by one of the coauthors (Emad Omera) of the original specification [26]. The source code is not available, however, 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. There is a recent whitepaper entitled "Zero-Trust Security for Webex White Paper" [5]. The whitepaper describes the path to their goal called Zero-Trust Security, and suggests to use 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 [4]. The repository maintainer warns that the specification is in progress. As of March 2021, it supports

– AES-GCM with 128 or 256-bit key, with 16-byte tag,
– AES-CM-HMAC with 4 or 8-byte tag.

Jitsi Meet. An open-source video communication application called Jitsi Meet[8] was presented at FOSDEM 2021, a major conference for open source projects[9]. Although 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 [15]. It supports

– AES-CM-HMAC with 4 or 10-byte tag, where 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 [26]. Then, we define the adversary models for our security analysis with reference to them defined by Isobe and Minematsu [14].

Definition 1. (Malicious User) *A malicious user, who is a legitimate user but 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.*

[7] `https://duo.google.com/about/`
[8] `https://meet.jit.si/`
[9] `https://fosdem.org/2021/schedule/`

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, *E2E adversary* is defined in [14], however, we do not explain this definition because this adversary is out of 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 [17]. According to this draft, 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.*

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 to enhance the E2EE systems, however, we do not explain these definitions because these features are out of scope for our security analysis.

3.3 Security Goals of AEAD for E2EE

Dodis *et al.* [7] proposed a new primitive called *encryptment* for the message franking scheme, which enables cryptographically verifiable reporting of malicious content in end-to-end encrypted messaging. In addition, they defined *confidentiality* and *second-ciphertext unforgeability* (SCU) as security goals to ensure the securily level of the encryptment scheme.

Definition 6. (Second-Ciphertext Unforgeability (SCU)) *An adversary $\mathcal{A}$ is given $K \xleftarrow{\$} \mathcal{K}$, which means a randomly chosen key K form 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*

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

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* $\perp$ *denotes a symbol that represents a decryption failure.*

The adversary in SCU game is given with key, hence this is not captured by the standard AEAD security notions of confidentiality and integrity [3,29]. When there exists 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) since 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*. Here, we focus on two types of second-preimage resistance, and define them with references to [23,30] as follows:

Definition 7. (Second-Preimage Resistance) *Let* $\mathcal{A}$ *be an adversary attempting to find any second input which has the same output as any specified input, i.e., for any given message* $M \xleftarrow{\$} \mathcal{M}$*, which means a randomly chosen message* M *form 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* $H : \mathcal{K} \times \mathcal{M} \to \mathcal{Y}$ *be 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*

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

The everywhere second-preimage or eSec resistance, introduced by Rogaway and Shrimpton [30], is called (a slight extension of) a strong form of second-preimage resistance. In this article, we assume the standard hash function (SHA2) as an instantiation of keyed function, say by using IV as a key, since otherwise standard security reduction is not possible (see [30]). For simplicity, we assume 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 on the security of AEAD used by SFrame. Here we view Alg. 1 as an encryption of AEAD for the reason that viewing Alg. 2 as a full-fledged AEAD does not make sense (see below). Then, effectively, the keys are contained by $\mathsf{KeyStore}[\mathsf{KID}]$ and the nonce is CTR, the associated data is a tuple $(\mathsf{S}, \mathsf{KID}, \mathsf{frame\_metadata})$, and the plaintext is M.

In Alg. 1, the variable N is a sum of $salt^{\mathsf{KID}}$ and ctr (Line 8), where the former is essentially a part of key (via HKDF), 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 $(\mathsf{S}, \mathsf{KID}, \mathsf{CTR})$. Since aad contains CTR as well as N, if the internal AEAD is AES-CM-HMAC of Alg. 2, HMAC takes the nonce (CTR) in addition to AD ($\mathsf{frame\_metadata}$) and the ciphertext C. Hence the lack of $N = salt^{\mathsf{KID}} \oplus ctr$ 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 key of AES-CTR.

A slightly more formal analysis is given below. Alg. 1 combined with AES-CM-HMAC can be interpreted as an encryption routine 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), and $\widetilde{\mathsf{Enc}}_K$ denotes the plain counter mode encryption with a pseudorandom offset to nonce (*i.e.*, $salt^{\mathsf{KID}}$, which is derived via HKDF), and $\widetilde{\mathsf{MAC}}_{K'}$ denotes the HMAC with a certain bijective input encoding. This means that Alg. 1 is exactly reduced to the encryption-then-MAC generic composition (assuming HKDF as a PRF) whose security is proved when $\widetilde{\mathsf{Enc}}$ is IND-CPA secure and $\widetilde{\mathsf{MAC}}$ is a PRF [18, 24]. Proving the latter claim is trivial. Hence Alg. 1 is secure under the standard assumptions that AES is a pseudorandom permutation and HMAC is a PRF. We remark that Alg. 2 itself is not a generically secure (*i.e.*, when nonce N and AD aad are independently chosen) AEAD as it ignores N in the computation of tag. This issue was raised at the discussion in CFRG[10] and our analysis provides an answer.

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

4.2 Impersonation against AES-CM-HMAC with Short Tags

While the AEAD security of Alg. 1 is sound, it does not necessarily mean the full E2EE security. In this section we point out that there is a 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.

Hereafter, we simplify the model and stick to the standard AEAD notation, namely the input is (N, A, M) for nonce N, associated data A, plaintext M and the output is (C, T) for ciphertext C and tag T. Also we consider the case that the signature is computed for each tag for simplicity. The notational discrepancies from Alg. 1 and Alg. 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 $\mathsf{Sig} = \mathsf{Sign}(K_{\mathrm{sig}}, T)$ 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:

$$\begin{aligned} C &\leftarrow \text{AES-CTR}(K_e^{\mathsf{KID}}, N, M) \\ T &\leftarrow \mathsf{truncate}(\text{HMAC-SHA256}(K_a^{\mathsf{KID}}, (N, A, C)), \tau), \end{aligned} \tag{8}$$

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 which we call a target user. This U_M is able to 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), and 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 so that it is likely to be used by U_T (these information are public and N is a counter so 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 $T^* = T'$, $C^* \neq C'$, then she replaces C' in that frame with C^*. Since the signature Sig is computed over the tag T' which is not changed after the replacement, this manipulated frame will pass the verification. Fig. 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. (8), 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.

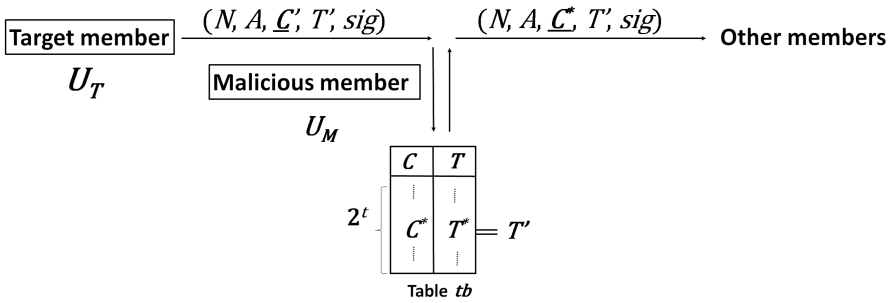

Fig. 2: Impersonation against AES-CM-HMAC with short tags. In the offline phase, a malicious group member U_M stores a set of (M, C, T) into the table tb. In the online phase, U_M intercepts a target frame $(N', A', C', T', \mathsf{Sig})$ sent by the target user U_T, searches a tuple (M^*, C^*, T^*)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

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, replaces C' with C^* in the target frame, and sends $(N', A', C^*, T', \mathsf{Sig})$ to other group members.

The manipulated frame including (C^*, T') successfully pass the signature verification by other group members due to a tag collision, *i.e.*, 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. The above 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 needs to intercept a legitimate message. It implies the adversary may collude with an intermediate server, or E2EE adversary, which is the central operating server. The practicality of this is beyond the scope of this article, however we remark that preventing 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 tuple $(N, A, C, T, \mathsf{Sig})$ that was previously sent without corruption; here (N', A') is chosen so that 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 we guess whether it is detected as replay depends on the actual system, so we keep it open. The cost of detecting a reply of randomized algorithm is generally high since 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 case $\tau = 32$ (i.e. 4-byte tags) if U_M prepares 2^{32} precomutation tables in the offline, the success probability is almost one. Thus, this forgery attack is practically feasible with a high success probability for the 4-byte tag. Besides, in this attack, the adversary fully controls the decryption result (M^*) of the manipulated frame except 32 bits which are used for generating 2^{32} different tags in the offline phase.

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

Even in the case of 8- and 10-byte tag, if U_M prepares 2^{56} tables, which is feasible by the nation-level adversary, the success probability is non-negligible, 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 impersonation attack as 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 above observation by showing SCU security of AES-CM-HMAC with long tags. According to Alg. 2, let D and D^* be (N, A, C) and (N^*, A^*, C^*), respectively (see Line 3 in Tag.Generation procedure). Note that N is included in A (aad) as partial information (see Lines 7-10 in Alg. 1). For simplicity, the tag generation by HMAC is abstracted as follows:

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

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 HMAC algorithm (see [34] 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*

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

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

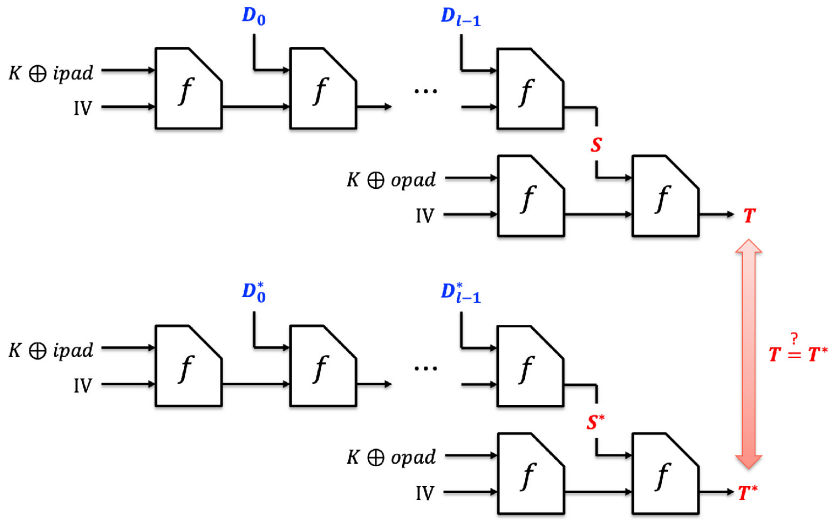

Fig. 3: SCU scenario against AES-CM-HMAC with long tags for E2EE. In this scenario, given a transcript of encryption query (N, A, M, C, T) derived on K, the adversary $\mathcal{A}$ is required to find a successful forgery (N^*, A^*, C^*, T^*) on K such that $T^* = T$ and $D^* \neq D$, *i.e.*, $(N^*,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

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 the tag T is the output of HMAC taking K and $D = (N, A, C)$ and thus the plaintext M is not needed in the attack. Fig. 3 illustrates this scenario, where IV denotes the initial hash value, $D = D_0 \| \ldots \| D_{l-1}$, $D^* = D_0^* \| \ldots \| D_{l-1}^*$, $S = H\big((K \oplus ipad) \| D\big)$, and $S^* = H\big((K \oplus ipad) \| D^*\big)$. Each D_i and D_i^* denotes 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 $H(K \oplus ipad \| D) = H(K \oplus ipad \| D^*)$, hence a second preimage against the target input $K \oplus ipad \| D$ is obtained. For Case 2, when $S \neq S^*$ and $T = T^*$, it means the adversary finds a second preimage against the target (2-block, thus 1024-bit) input $K \oplus opad \| S$. Both cases are covered by the eSec security of H, hence we have

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

which concludes the proof. □

Theorem 1 tells that the SCU security of AES-CM-HMAC with long tags depends on the security of underlying hash function. According to the Internet draft [26], SFrame adopts SHA256 as the hash function used in AES-CM-HMAC.

Second-Preimage Security of SHA256. Ideally, a n-bit hash function provides a 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.* [16] 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 performs up to 45 rounds (out of 64) with a time complexity of $2^{255.5}$ and a memory complexity of 2^6 words. After that, Andreeva *et al.* [1] presented new generic second-preimage attacks on the basic Merkle-Damgård hash functions. Their best attack allow us to find a second-preimage on the full SHA256 with a time complexity of 2^{173} and a memory complexity of 2^{83}, but this attack is required too long message blocks, e.g., a 2^{118}-block message.

To the best of our knowledge, no study has 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 impersonation attacks described above is a generic attack and the offline attack complexity depends on the tag length. In contrast, if we use AES-GCM, it is easy to mount a similar attack without the offline phase. This is because, the adversary who owns the GCM key and observes a legitimate GCM output of (N, A, C, T) is able to 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 negligible complexity irrespective of the tag length unlike the case of AES-CM-HMAC.

Once the adversary intercepts a 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 lengths of A and C, and multiplications are over $\mathrm{GF}(2^{128})$. $E_K(*)$ denotes the encryption by AES with key K and $L = E_K(0^{128})$, and i_{32} for a non-negative

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}

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 $(N', A', C') \neq (N, A, C)$ as we know K. Say, we first arbitrary choose N' and A', and the fake plaintext block M'_1 to compute C'_1, and finally set C'_2 so 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 member and the forged plaintext is almost arbitrary except 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.

4.5 Considerations on Authentication Key Recovery

The specification [26] appears to implicitly allow 4 and 8-byte tags with AES-GCM. In addition to the attacks described above, it is known that 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 [11] first pointed out this attack, and Mattsson and Westerlund [21] further refined the attack and provided a concrete complexity estimation. According to [21], they point out that the security levels are only 62–67 bits and 70-75 bits for 32-bit and 64-bit tags, respectively, even if we follow NIST requirements on the usage of GCM with short tags, which is shown in 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 there is no restriction regarding L and q, the authenticated key is recovered with data complexity of 2^t as the complexity of the first forgery is dominated. Thus, for 4-byte (= 32-bit) tag length, the authenticated key recovery is feasible

with 2^{32} data complexity. It seems that the specification [26] does not explicitly mention the restrictions of q and L.

Practical effects on SFrame. As far as we checked available implementations of the original [33], Cisco Webex [4], and Jitsi Meet [15], there is no restriction regarding L and q. In this case, for the 4-byte tag, the authenticated key is recovered with data complexity of 2^{32}, which is practically available by a malicious user.

4.6 Recommendations

From the vulnerabilities shown in Sections 4.2 to 4.5, we recommend the followings.

- For AES-CM-HMAC, short tags, especially 4-byte tag, should not be used.
- For AES-GCM, a signature should be computed over a whole frame, not only tags.
- For AES-GCM, the specification should clearly forbid short tags, or refer to NIST requirements on the usage of GCM with short tags.
- As discussed at Section 3, switch to other ciphersuite that works as a secure encryptment scheme, such as HFC [7], with a sufficiently long tag is another option.

5 Conclusions

We have shown 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 IETF. SFrame is a young project but going to be adopted by a number of real-world products. Our results show that there is 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 the used AEAD algorithm allows to create a collision on tags with the knowledge of the key. The former applies to the case of AES-CM-HMAC, and 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.* [7]. Moreover, AES-CM-HMAC is, if it is correctly used by the upper layer, a provably secure AEAD because it can be interpreted as a standard encryption-then-MAC generic composition. We notify 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 studied more actively and hope our work help 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 the anonymous reviewers for their insightful comments, and 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. Andreeva, E., Bouillaguet, C., Dunkelman, O., Fouque, P.A., Hoch, J.J., Kelsey, J., Shamir, A., Zimmer, S.: New second-preimage attacks on hash functions. Journal of Cryptology **29**(4), 657–696 (Oct 2016). https://doi.org/10.1007/s00145-015-9206-4
2. 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)
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, Heidelberg (Dec 2000). https://doi.org/10.1007/3-540-44448-3_41
4. Cisco Systems: SFrame (2020), `https://github.com/cisco/sframe`
5. 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`
6. Cohn-Gordon, K., Cremers, C., Dowling, B., Garratt, L., Stebila, D.: A Formal Security Analysis of the Signal Messaging Protocol. Journal of cryptology **33**(4), 1914–1983 (2020)
7. Dodis, Y., Grubbs, P., Ristenpart, T., Woodage, J.: Fast message franking: From invisible salamanders to encryptment. In: Shacham, H., Boldyreva, A. (eds.) CRYPTO 2018, Part I. LNCS, vol. 10991, pp. 155–186. Springer, Heidelberg (Aug 2018). https://doi.org/10.1007/978-3-319-96884-1_6
8. Dworkin, M.: NIST SP 800-38D, Recommendation for Block Cipher Modes of Operation: Galois/Counter Mode (GCM) and GMAC (2007), u.S.Department of Commerce/National Institute of Standards and Technology
9. Emad Omara: Extend Tag Calculation to Cover Nonce #59 (2021), `https://github.com/eomara/sframe/pull/59`
10. Emad Omara: Remove Signature #58 (2021), `https://github.com/eomara/sframe/pull/58`
11. 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`
12. 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 (Aug 2016)
13. Isobe, T., Ito, R.: Security Analysis of End-to-End Encryption for Zoom Meetings. IEEE Access **9**, 90677–90689 (2021)
14. 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, Part II. LNCS, vol. 11099, pp. 249–268. Springer, Heidelberg (Sep 2018). https://doi.org/10.1007/978-3-319-98989-1_13
15. Jitsi: Jitsi Meet API library (2020), `https://github.com/jitsi/lib-jitsi-meet/`

16. 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, Heidelberg (Mar 2012). https://doi.org/10.1007/978-3-642-34047-5_15
17. 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)
18. 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, Heidelberg (Aug 2001). https://doi.org/10.1007/3-540-44647-8_19
19. Krawczyk, H., Eronen, P.: HMAC-based Extract-and-Expand Key Derivation Function (HKDF). Internet Engineering Task Force - IETF, Request for Comments **5869** (May 2010)
20. Matrix.org Foundation.: Olm: A Cryptographic Ratchet (2016), `https://gitlab.matrix.org/matrix-org/olm/-/blob/master/docs/olm.md`
21. Mattsson, J., Westerlund, M.: Authentication key recovery on galois/-counter mode (GCM). In: Pointcheval, D., Nitaj, A., Rachidi, T. (eds.) AFRICACRYPT 16. LNCS, vol. 9646, pp. 127–143. Springer, Heidelberg (Apr 2016). https://doi.org/10.1007/978-3-319-31517-1_7
22. McGrew, D.A.: An Interface and Algorithms for Authenticated Encryption. Internet Engineering Task Force - IETF, Request for Comments **5116** (January 2008)
23. Menezes, A.J., Oorschot, P.C.V., Vanstone, S.A.: Handbook of Applied Cryptography. CRC press (1996)
24. 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, Heidelberg (May 2014). https://doi.org/10.1007/978-3-642-55220-5_15
25. Omara, E.: Google Duo End-to-End Encryption Overview - Technical Paper (2020), `https://www.gstatic.com/duo/papers/duo_e2ee.pdf`
26. Omara, E., Uberti, J., Gouaillard, A., Murillo, S.G.: Secure Frame (SFrame). `https://tools.ietf.org/html/draft-omara-sframe-01` (November 2020)
27. Omara, E., Uberti, J., Gouaillard, A., Murillo, S.G.: Secure Frame (SFrame). `https://tools.ietf.org/html/draft-omara-sframe-02` (March 2021)
28. Open Whisper Systems.: Signal Github Repository (2017), `https://github.com/WhisperSystems/`
29. Rogaway, P.: Authenticated-encryption with associated-data. In: Atluri, V. (ed.) ACM CCS 2002. pp. 98–107. ACM Press (Nov 2002). https://doi.org/10.1145/586110.586125
30. 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, Heidelberg (Feb 2004). https://doi.org/10.1007/978-3-540-25937-4_24
31. 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)
32. Saĺ 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`
33. Sergio Garcia Murillo: SFrame.js (2020), `https://github.com/medooze/sframe`

34. Turner, J.M.: The Keyed-Hash Message Authentication Code (HMAC). Federal Information Processing Standards Publication **198**, 1 (2008)