Security Analysis of SFrame

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

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

1 Introduction

1.1 Background

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 applied to an application for communication 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 a significant attention 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 or Webex. 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 [GGK+16, IM18, RMS18, CGCD+20].

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 [OUGM20]. 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 a major proprietary software to an open-source application.

1.2 Our Contribution

We looked into the original specification [OUGM20], and found an issue regarding the use of authenticated encryption and signature algorithm. The specification [OUGM20] defines two AEAD algorithms, 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 user 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.

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 [OUGM20], 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.

1.3 Responsible Disclosure

In March 2021, we reported our results in this paper to the SFrame designers via email and video conference. They acknowledged that our attacks are feasible under the existence of a malicious user, quickly decided to remove the signature mechanism [Ema21b] and extend tag calculation to cover nonces [Ema21a], and updated the specification in the Internet draft on March 29, 2021 [OUGM21]. They have a plan to review the SFrame specification and support signature mechanism again in the future.

1.4 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 Internet draft [OUGM20]. However, it does not specify the key exchange protocol between the parties and the choice is left to the implementors. In practice Signal protocol [Ope17], Olm protocol [Mat16], or Message Layer Security (MLS) protocol [BBM+20] 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.

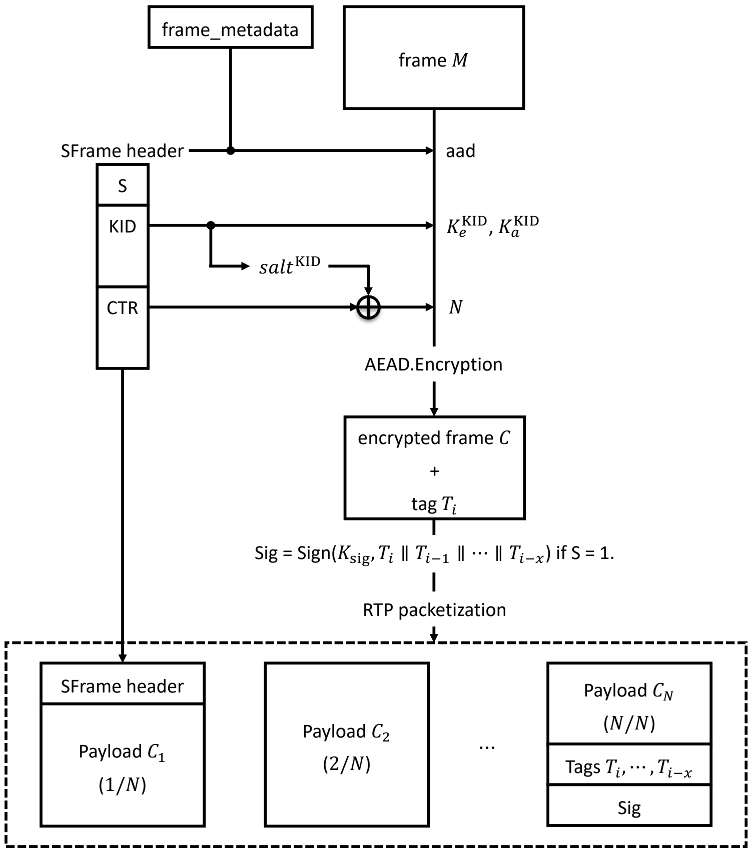
Cryptographic Protocol. Suppose there is a group of users, G. All users in G first perform a predetermined key exchange protocol, such as Signal protocol, Olm protocol, or MLS protocol, and share multiple group keys $K_{\mathsf{base}}^{\mathsf{KID}}$ associated with the key ID, which is called *base key* in the original draft [OUGM20]. 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) [McG08, Rog02] used for frame encryption, e.g., AES-GCM and AES-CTR in combination with HMAC-SHA256. The authentication tag may be truncated.
– An optional signature algorithm, e.g., EdDSA over Ed25519 and ECDSA over P-521.

Specifically, [OUGM20] 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.



frame_metadata
frame M
SFrame header
aad
S
KID
$K_e^{\mathrm{KID}}, K_a^{\mathrm{KID}}$
$salt^{\mathrm{KID}}$
CTR
N
AEAD.Encryption
encrypted frame C + tag T_i
Sig = Sign($K_{\mathrm{sig}}, T_i \parallel T_{i-1} \parallel \cdots \parallel T_{i-x}$) 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.

Fig. 1 and Alg. 1 show the media frame encryption flow in an E2EE session for SFrame using the above ciphersuite. When AES-GCM is adopted, AEAD.ENCRYPTION in Alg. 1 is executed according to NIST SP 800-38D [Dwo07]. Before performing by the AEAD encryption procedure by AES-GCM, HKDF [KE10] is used to generate the secret key K_e^{KID} and the salt $salt^{\mathsf{KID}}$ for encrypting/decrypting media frames as follows:

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

where KeyLen and NonceLen are the length (byte) of a secret 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, AEAD.ENCRYPTION in Alg. 1 is executed according to Alg. 2. Before performing

Algorithm 1 Media frame encryption scheme

```
Require: S, KID, CTR, frame_metadata, $M$: frame
Ensure: $C$: 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:         $C, T = \mathrm{AEAD.E{\scriptsize NCRYPTION}}(K_e^{\mathsf{KID}}, N, \mathsf{aad}, M)$
13:     else
14:         $C, T = \mathrm{AEAD.E{\scriptsize NCRYPTION}}(K_e^{\mathsf{KID}}, K_a^{\mathsf{KID}}, N, \mathsf{aad}, M)$
15:     end if
16: end procedure
```

AES-CM-HMAC, HKDF [KE10] 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 K_e^{KID}, K_a^{KID}, and $salt^{\mathsf{KID}}$, 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 user. 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 is computed over a list of authentication tags with a fixed size, $(T_i, T_{i-1}, \ldots, T_{i-x})$, as follows:

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

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.

Algorithm 2 AEAD encryption by AES-CM-HMAC

```
Require: $K_a^{\mathsf{KID}}$, $\mathsf{aad}$: additional associated data, $C$: encrypted frame
Ensure: $T$: truncated authentication tag
1: procedure TAG.GENERATION($K_a^{\mathsf{KID}}$, $\mathsf{aad}$, $C$)
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

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

2.2 Available Implementations

We list some implementations of SFrame that are publicly available. Some of them do not strictly follow the original specification [OUGM20] 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 coauthors (Sergio Garcia Murillo) of the Internet draft publicly available [Ser20]. 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[5] 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 [Oma20] written by one of the coauthors (Emad Omera) of the original specification [OUGM20]. 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.

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

Cisco Webex. Webex[6] is a major video meeting application developed by Cisco. There is a recent whitepaper entitled "Zero-Trust Security for Webex White Paper". 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 [Cis20]. 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. Recently, an open-source video communication application called Jitsi Meet[7] was presented at FOSDEM 2021, a major conference for open source projects[8]. 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 [Jit20]. It supports

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

3 Security Goals

In February 2021, the Internet draft entitled "Definition of End-to-end Encryption" was released [KBK+21]. According to this draft, the fundamental features for E2EE require *authenticity*, *confidentiality*, and *integrity*, which are defined as follows:

Definition 1. (Authenticity) *A system provides message authenticity if the recipient is certain who sent the message and the sender is certain who received it.*

Definition 2. (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 3. (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 analysis.

[6] `https://www.webex.com`
[7] `https://meet.jit.si/`
[8] `https://fosdem.org/2021/schedule/`

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 $\mathsf{KeyStore[KID]}$ 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 $\mathit{salt}^{\mathsf{KID}}$ 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 $N = \mathit{salt}^{\mathsf{KID}} \oplus \mathit{ctr}$ 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 key of AES-CTR. Therefore we think Alg. 1 combined with AES-CM-HMAC is provably secure. In other words Alg. 2 itself is not a generically secure (*i.e.*, nonce N and AD aad are independently chosen) AEAD as it ignores N in the computation of tag.

When combined with AES-GCM, as it uses CTR for encryption, it is secure as well.

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 user 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:

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

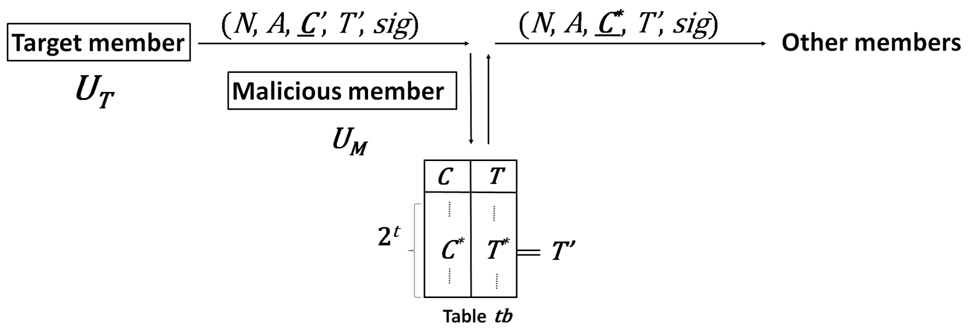


Fig. 2: Impersonation against AES-CM-HMAC with Short Tags.

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 malicious member U_M is able to mount a forgery attack (impersonation) by manipulating a frame sent by U_T. Specifically, this forgery attack consists of offline and online phases. In the online 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 store 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 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. 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 $C = \text{AES-CTR}(K_e^{\mathsf{KID}}, N, M)$ and then obtains a tag $T = \text{HMAC-SHA256}(K_a^{\mathsf{KID}}, (N, A, C))$, 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, 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 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.

The security requirement discussed here is equivalent to the property called *Second-ciphertext unforgeability* (SCU) introduced by Dodis et al. at CRYPTO 2018 [DGRW18]. For a randomly chosen K and a transcript of encryption query (N, A, M, C, T) derived on K, SCU represents the hardness of finding a successful forgery (N', A', C', T) (on K) such that $(N', A', C') \neq (N, A, C)$ with the knowledge of K and (N, A, M, C, T). An extension of AE called Encryptment [DGRW18] is SCU-secure.

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 $\tau = 32$ (the tag is 4 bytes), 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 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 phrase.

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 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 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.

4.4 Considerations on Authentication Key Recovery

The specification [OUGM20] 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 [Fer05] first pointed out this attack, and Mattsson and Westerlund [MW16] further refined the attack and provided a concrete complexity estimation. According to [MW16], 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 it following NIST requirements on the usage of GCM with short tags as shown in Table 1 where 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^{taglen} 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 [OUGM20] does not explicitly mention the restrictions of q and L.

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}

Practical effects on SFrame. As far as we checked available implementations of the original [Ser20] and Cisco [Cis20], Jitsi Meet [Jit20], 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 in by the adversary.

4.5 Recommendations

From these vulnerabilities, our recommendations are as follows.

– 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 4.2, switch to other ciphersuite that works as a secure encryptment scheme, such as HFC [DGRW18], with a sufficiently long tag is another option.

5 Conclusions

This article studies SFrame, a recently proposed end-to-end encryption mechanism built on RTC. It is developed for video/audio group communication applications, and received significant attentions. With investigation on the original specification, we pointed out that there is a 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.

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.

References

BBM+20. Richard Barnes, Benjamin Beurdouche, Jon Millican, Emad Omara, Katriel Cohn-Gordon, and Raphael Robert. The Messaging Layer Security (MLS) Protocol. `https://tools.ietf.org/html/draft-ietf-mls-protocol-10`, October 2020.

CGCD+20. Katriel Cohn-Gordon, Cas Cremers, Benjamin Dowling, Luke Garratt, and Douglas Stebila. A Formal Security Analysis of the Signal Messaging Protocol. *Journal of cryptology*, 33(4):1914–1983, 2020.

Cis20. Cisco Systems. SFrame, 2020. `https://github.com/cisco/sframe`.

DGRW18. Yevgeniy Dodis, Paul Grubbs, Thomas Ristenpart, and Joanne Woodage. Fast message franking: From invisible salamanders to encryptment. In Hovav Shacham and Alexandra Boldyreva, editors, *CRYPTO 2018, Part I*, volume 10991 of *LNCS*, pages 155–186. Springer, Heidelberg, August 2018.

Dwo07. Morris Dworkin. 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.

Ema21a. Emad Omara. Extend Tag Calculation to Cover Nonce #59, 2021. `https://github.com/eomara/sframe/pull/59`.

Ema21b. Emad Omara. Remove Signature #58, 2021. `https://github.com/eomara/sframe/pull/58`.

Fer05. Niels Ferguson. 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`.

GGK$^+$16. Christina Garman, Matthew Green, Gabriel Kaptchuk, Ian Miers, and Michael Rushanan. Dancing on the lip of the volcano: Chosen ciphertext attacks on apple iMessage. In Thorsten Holz and Stefan Savage, editors, *USENIX Security 2016*, pages 655–672. USENIX Association, August 2016.

IM18. Takanori Isobe and Kazuhiko Minematsu. Breaking message integrity of an end-to-end encryption scheme of LINE. In Javier López, Jianying Zhou, and Miguel Soriano, editors, *ESORICS 2018, Part II*, volume 11099 of *LNCS*, pages 249–268. Springer, Heidelberg, September 2018.

Jit20. Jitsi. Jitsi Meet API library, 2020. `https://github.com/jitsi/lib-jitsi-meet/`.

KBK$^+$21. Mallory Knodel, Fred Baker, Olaf Kolkman, Sofia Celi, and Gurshabad Grover. Definition of End-to-end Encryption. `https://datatracker.ietf.org/doc/draft-knodel-e2ee-definition/`, February 2021.

KE10. Hugo Krawczyk and Pasi Eronen. HMAC-based Extract-and-Expand Key Derivation Function (HKDF). *Internet Engineering Task Force - IETF, Request for Comments*, 5869, May 2010.

Mat16. Matrix.org Foundation. Olm: A Cryptographic Ratchet, 2016. `https://gitlab.matrix.org/matrix-org/olm/-/blob/master/docs/olm.md`.

McG08. David A. McGrew. An Interface and Algorithms for Authenticated Encryption. *Internet Engineering Task Force - IETF, Request for Comments*, 5116, January 2008.

MW16. John Mattsson and Magnus Westerlund. Authentication key recovery on galois/counter mode (GCM). In David Pointcheval, Abderrahmane Nitaj, and Tajjeeddine Rachidi, editors, *AFRICACRYPT 16*, volume 9646 of *LNCS*, pages 127–143. Springer, Heidelberg, April 2016.

Oma20. Emad Omara. Google Duo End-to-End Encryption Overview - Technical Paper, 2020. `https://www.gstatic.com/duo/papers/duo_e2ee.pdf`.

Ope17. Open Whisper Systems. Signal Github Repository, 2017. `https://github.com/WhisperSystems/`.

OUGM20. Emad Omara, Justin Uberti, Alexandre Gouaillard, and Sergio Garcia Murillo. Secure Frame (SFrame). `https://tools.ietf.org/html/draft-omara-sframe-01`, November 2020.

OUGM21. Emad Omara, Justin Uberti, Alexandre Gouaillard, and Sergio Garcia Murillo. Secure Frame (SFrame). `https://tools.ietf.org/html/draft-omara-sframe-02`, March 2021.

RMS18. Paul Rösler, Christian Mainka, and Jörg Schwenk. 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)*, pages 415–429. IEEE, 2018.

Rog02. Phillip Rogaway. Authenticated-encryption with associated-data. In Vijayalakshmi Atluri, editor, *ACM CCS 2002*, pages 98–107. ACM Press, November 2002.

Ser20. Sergio Garcia Murillo. SFrame.js, 2020. `https://github.com/medooze/sframe`.