Security Analysis of End-to-End Encryption for Zoom Meetings*

Takanori Isobe[1,2,3] and Ryoma Ito[2]

[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.

Abstract. In the wake of the global COVID-19 pandemic, video conference systems have become essential for not only business purposes, but also private, academic, and educational uses. Among the various systems, Zoom is the most widely deployed video conference system. In October 2020, Zoom Video Communications rolled out their end-to-end encryption (E2EE) to protect conversations in a meeting from even insiders, namely, the service provider Zoom. In this study, we conduct thorough security evaluations of the E2EE of Zoom (version 2.3.1) by analyzing their cryptographic protocols. We discover several attacks more powerful than those expected by Zoom according to their whitepaper. Specifically, if insiders collude with meeting participants, they can impersonate *any Zoom user* in target meetings, whereas Zoom indicates that they can impersonate only the current meeting participants. Besides, even without relying on malicious participants, insiders can impersonate any Zoom user in target meetings though they cannot decrypt meeting streams. In addition, we demonstrate several impersonation attacks by meeting participants or insiders colluding with meeting participants. Although these attacks may be beyond the scope of the security claims made by Zoom or may be already mentioned in the whitepaper, we reveal the details of the attack procedures and their feasibility in the real-world setting and propose effective countermeasures in this paper. Our findings are not an immediate threat to the E2EE of Zoom; however, we believe that these security evaluations are of value for deeply understanding the security of E2EE of Zoom.

Keywords: Zoom, End-to-End Encryption, Impersonation attacks

1 Introduction

Video conference systems are being increasingly used for a variety of purposes – for business meetings and functioning, private communications, educational

* The part of this paper was presented at the 26th Australasian Conference on Information Security and Privacy (ACISP 2021).

purposes, and so on – since the Covid-19 pandemic has severely limited the practicality of physical meetings. Hence, security measures such as end-to-end encryption (E2EE) have become essential. In this study, the E2EE of Zoom, which is one of the most used software for video communication worldwide today, is thoroughly examined for potential security gaps.

1.1 Background

E2EE. E2EE is a secure communication scheme for messaging applications and video conference systems in which only the people who are communicating can send and read the messages. That is, nobody except each participant, not even the service provider, has access to the encryption keys that are used to encrypt the contents. After Edward Snowden's revelations regarding surveillance programs, the E2EE receives much attentions as a technology to protect a user privacy from the mass interception and surveillance of communications carried out by governmental organizations such as the NSA (National Security Agency) of the US government.

Signal Protocol is the widely used E2EE protocol. The core of Signal Protocol has been adopted by WhatsApp, Facebook Messenger, and Google Duo. A novel technology called the "ratcheting" key update structure enables advanced security properties such as perfect forward secrecy and so-called the post-compromise security [6]. Since Signal Protocol is an open-source application and its source code for Android and iOS are available on Github [21], its security has been thoroughly studied by the cryptographic community.

iMessage, which is a widely deployed messaging application of Apple, supports an original E2EE protocol in which a message that is compressed by gzip is encrypted by a sender's secret key and distributed with a digital signature for guaranteeing the integrity to the recipient. Unfortunately, the initial iMessage had several security flaws as pointed out in November 2015 [9]. These vulnerabilities originated from the misuse of cryptographic primitives. Apple fixed these problems and released the new version in March 2016. LINE, which is a widely used message application in East Asia, is also based on an original E2EE protocol for efficient software performance. The previous version of the E2EE schemes of LINE was called Letter Sealing; its message integrity was broken by exploiting the vulnerabilities of cryptographic primitives and protocols [13]. In response to this, in October 2019, LINE released the new version of Letter Sealing to address these security issues.

Thus, analysis of the E2EE protocol is crucial for enhancing the security of E2EE as E2EE technologies are not mature enough and their security is not well understood yet despite of their wide use in the real world.

Zoom. Zoom is, at present, the most widely deployed video conference system in the world. The number of daily active users in the world was about 300 million in April 2020. It is currently a key platform for business and online education worldwide.

Zoom Video Communications first announced their plan to support E2EE in May 2020 to protect conversations in the meeting; they published the technical details of encryption schemes published as a whitepaper [14]. In October 2020, Zoom rolled out phase 1 out of 4 of their E2EE project, and made E2EE available globally for paid and free Zoom users for 30 days as a technical preview.

E2EE of Zoom is based on AES-GCM [2], HKDF algorithm [16], Diffie–Hellman over Curve25519 [3], and EdDSA over Ed25519 [4] as an authenticated encryption, key derivation, key exchange, and signature schemes, respectively. To launch an E2EE session for a Zoom meeting, a meeting leader generates a meeting key and securely distributes it to other participants via a bulletin board by key exchange, key derivation function, and signature schemes. Thereafter, each meeting stream is encrypted by AES-GCM with the shared meeting key.

In the whitepaper [14], Zoom claims the security goals of confidentiality, integrity, and abused prevention against insiders, outsiders, and meeting participants, where insiders are the service providers, namely, Zoom, and outsiders are the legitimate users of Zoom but not participants in the target meeting.

1.2 Our Contribution

In this study, we conduct thorough security evaluations of the E2EE of Zoom (version 2.3.1), and consider several attacks. For comparison, we consider an unavoidable attack in which the meeting participants colluding with malicious insiders send a meeting key to insiders. In this case, insiders can break the confidentiality of contents in the meeting. We will explore attacks beyond this unavoidable attack by exploiting the vulnerabilities of the E2EE protocol and underlying primitives. Specifically, we propose the following impersonation attacks and their countermeasures.

Impersonation Based on No Entity Authentication. First, we discuss impersonation attacks by malicious meeting participants without colluding with insiders. This attack exploits the fact that there is no entity authentication of a meeting stream in a group meeting. Specifically, the stream data sent from any meeting participant are encrypted by AES-GCM with the same meeting key. Although this vulnerability is pointed out in the whitepaper [14], we reveal the details of the attack procedures, their feasibility, and the impacts on real-world applications. Besides, we discuss a simple countermeasure, which is also mentioned in the whitepaper [14].

Impersonation of any Zoom User. We show that insiders without colluding with participants can impersonate any legitimate Zoom user, even an uninvited user, for the target meeting. This attack exploits the fact that insiders have free access to bulletin boards and they can issue a meeting ID and UUID, which functions as the nonce of binding information to identify users. Using these facts, insiders can reuse the binding information of Zoom users, which is posted on bulletin boards in previous meetings, for target meetings. Note that in this

attack, insiders cannot decrypt the meeting stream as the meeting key of the target meeting is unknown. However, it can have adverse effects; for example, in the case of a negotiation, the fact that an influential person is attending can impose a silent pressure on others. Thus, this attack makes sense in the practical case.

Furthermore, if colluding with participants, insiders can get the meeting key. Then, they fully impersonate any legitimate user, i.e., they can actively attend the target meeting as a target user. This is obviously beyond the unavoidable attack in which insiders can only passively eavesdrop the meeting streams. In addition, we show that it can be easily fixed by adding time information to binding information.

Impersonation of Another User on a Shared Device. Finally, we show impersonation attacks in the case where multiple users share a device for Zoom meetings by colluding with insiders. In this attack, a malicious user can obtain the device key of another user who utilizes the same device for Zoom meetings. This attack exploits the fact that the key for encrypting the device key is stored in the Zoom server. In the E2EE setting, insiders cannot be trusted; nevertheless, insiders hold these keys.

Further Security Evaluations. We discuss some security issues of E2EE in Zoom. The first one is the use of the authenticated encryption mode, GCM. It is well known that if the same nonce is used, it is easy to recover an authentication key from only ciphertexts [8, 15, 18]. The whitepaper [14] describes that nonces are generated by counters. However, the client application is made by Zoom. Under the E2EE assumption, Zoom can inject a trapdoor such that the same nonce is used in some points to the software. To avoid such suspicions, we recommend using a more secure authenticated encryption scheme that has the nonce-misuse resistance in E2EE, or the client application should be public as an open source software so that third parties can audit it. Besides the security issues, we discuss the denial of service to target users by insiders.

1.3 Uncovered Results and Limitation

Table 1 summarizes our results. Zoom deems some attacks, including in-meeting impersonation attacks in which a malicious but otherwise authorized meeting participant colluding with a malicious server can masquerade as another authorized meeting participant, as out of scope.

Since some of our impersonation attacks involve colluding with insiders, these may be beyond the security claims of Zoom. However, we uncover several attacks more powerful than that expected by Zoom.

- If insiders collude with meeting participants, they can impersonate *any Zoom user* in target meetings, while the whitepaper [14] claims that they can impersonate only current meeting participants.

Table 1. Summary of our results: *impersonation*, *tampering*, and *denial of service* attacks. Each type of attack is classified into two types: *active* and *passive* attacks. In an active-type attack, an adversary can not only join the target meeting, but also properly send and receive the meeting streams. In a passive-type attack, an adversary can perform the attack, but cannot properly send and receive the meeting streams. The adversary and victim models consist of an *insider*, *outsider*, *meeting leader*, and *meeting participant*, which are denoted as I, O, L, and P, respectively. We use "c.w." as an abbreviation of "colluding with".

Attack	Type	Adversary	Victim	Reference
Impersonation	Active	L/P	L/P	Sec. 4.1
Impersonation	Passive	I	L/P/O	Sec. 5.1
Impersonation	Active	I c.w. L	L/P/O	Sec. 5.1
Impersonation	Active	O c.w. I, L	L/P/O	Sec. 5.2
Impersonation	Active	O c.w. L	L/P/O	Sec. 5.2
Impersonation	Active	O c.w. I	O	Sec. 6.1
Tampering	Passive	I	L/P	Sec. 7.1
Denial of service	Passive	I	P	Sec. 8.1

– Even without relying on malicious participants, insiders can impersonate any Zoom user for target meetings though they cannot decrypt the meeting stream.

Our results are based on the whitepaper [14] and we have analyzed only the cryptographic protocol of E2EE (version 2.3.1). In order to demonstrate the feasibility of the proposed attacks, we should have implemented and tested the proposed attacks, however, this paper only presents the theoretical evaluations of E2EE for Zoom because the source code of E2EE for Zoom is not available. Therefore, we discussed with Zoom to confirm the feasibility of the proposed attacks (refer to Section 1.4 for detail).

Our findings are not an immediate threat to E2EE for Zoom. However, our results show that there is room of improvement in the E2EE for Zoom as a cryptographic scheme. We believe that these security evaluations are of value for understanding well and enhancing the security of E2EE for Zoom.

1.4 Responsible Disclosure

In November 2020, we informed Zoom of our findings in this paper via the vulnerability disclosure platform of Hacker One [12]. They acknowledged our impersonation attacks and other attacks while they already recognized some attacks as discussed before. They told us that they have a plan to address these issues in the future version or clearly state these as limitations of the current version of their E2EE in the whitepaper. In each section, we describe the details of their responses and results of the discussion with Zoom.

1.5 Related Works

We summarize the part of previously discovered vulnerabilities in the Zoom application, which were published by the Zoom security team on their website[1].
CVE-2018-15715 (publication date: November 30, 2018). Zoom clients on Windows (before version 4.1.34460.1105), Mac OS (before version 4.1.34475.1105), and Linux (before version 2.5.146186.1130) are vulnerable to unauthorized message processing. A remote unauthenticated attacker can spoof UDP messages from a meeting attendee or Zoom server to invoke functionality in the target client. This allows the attacker to remove attendees from meetings, spoof messages from users, or hijack shared screens.
CVE-2019-13450 (publication date: July 9, 2019). In the Zoom MacOS Client prior to version 4.4.5 and RingCentral MacOS client prior to version 4.4.5, remote attackers can force a user to join a video call with the video camera active. This occurs because any website can interact with the Zoom web server on localhost port 19421.
CVE-2019-13567 (publication date: July 12, 2019). The Zoom Client before 4.4.52595.0425 on macOS allows remote code execution, a different vulnerability than CVE-2019-13450. If the ZoomOpener daemon (aka the hidden webserver) is running, but the Zoom Client is not installed or can't be opened, an attacker can remotely execute code with a maliciously crafted launch URL.

On the other hand, our study has focused on the E2EE mechanism described in the whitepaper version 2.3.1 [14], which was published on November 3, 2020. That is, it is clear that all the above vulnerabilities are unrelated to our discovered vulnerabilities, because they are related to the vulnerabilities in the Zoom application that does not implement the targeted E2EE mechanism.

According to vulnerability reports published by the Zoom security team, the following is the only vulnerability reported since November 3, 2020.
CVE-2021-28133 (publication date: March 26, 2021). In all Windows Zoom Client versions, Linux Zoom Client versions prior to 5.5.4 on Ubuntu, and All Linux Client versions on other supported distributions, an attacker can affect these clients' share screen functionality when sharing individual application windows, in which screen contents of applications which are not explicitly shared by the screen-sharing users may be seen by other meeting participants for a brief moment if the "sharer" is minimizing, maximizing, or closing another window.

This vulnerability is also unrelated to our discovered vulnerabilities; thus, this is the first time we have reported the vulnerabilities in the E2EE mechanism for Zoom.

1.6 Organization of this Paper

The rest of the paper is organized as follows. In Section 2, we define adversary models and security goals of E2EE for Zoom. In Section 3, we briefly describe the E2EE specifications for Zoom meetings. In Section 4, we introduce impersonating attacks based on no entity authentication. In Section 5, we explain how

[1] https://zoom.us/trust/security/security-bulletin

a malicious insider or a malicious outsider can impersonate any Zoom user, including users who are not invited to the target meeting. In Section 6, we describe how a malicious outsider can impersonate another user on a shared device. In Sections 7 and 8, we evaluate the security against tampering and denial of service attacks. Sections 4-8 also present the feasibilities and countermeasures for these attacks. Finally, Section 9 concludes the paper.

2 Adversary Models and Security Goals

This section explains the adversary models and security goals of the E2EE for Zoom meetings. Although our definitions are primarily based on the whitepaper [14], we also consider the security models described in some other papers [13, 19].

2.1 Adversary Models

In the whitepaper [14], the designers defined *insiders*, *outsiders*, and *meeting participants* as the adversary models. With reference to the adversary models reported by Isobe and Minematsu [13], we redefine these models for our security analysis:

Definition 1. (Insiders) *Insiders develop and maintain Zoom's server infrastructure and its cloud providers. A malicious insider can intercept, read, and modify any meeting streams sent over the network, and has full access to Zoom's server infrastructure.*

Definition 2. (Outsiders) *Outsiders are legitimate users of Zoom meetings but not part of Zoom's trusted infrastructure and do not have access to non-public meeting access control information. A malicious outsider may monitor, intercept, and modify network traffic and may attempt to break one of the security goals in other E2EE sessions by maliciously manipulating the protocol.*

Definition 3. (Meeting Participants) *Meeting participants can access a meeting, because they know the ID and password of the meeting or exercise other qualifying credentials. A malicious meeting participant attempts to break one of the security goals by deviating from the protocol.*

According to the whitepaper [14], there exists a *meeting leader* among the meeting participants, and he/she has higher authority than other meeting participants as follows:

Definition 4. (Meeting Leader) *A meeting leader has the responsibility of generating the shared meeting key, authorizing new meeting participants, removing unwanted participants from the meeting, and distributing keys. A malicious meeting leader attempts to break one of the security goals by deviating from the protocol.*

As described in the paper reported by Isobe and Minematsu [13], a malicious outsider, a malicious meeting participant, and a malicious meeting leader can collude with a malicious insider, or a malicious insider himself/herself can be a malicious meeting participant or a malicious meeting leader.

2.2 Security Goals

In the whitepaper [14], the designers defined *confidentiality*, *integrity*, and *abuse prevention* as the security goals. With reference to the security goals of E2EE reported by Isobe and Minematsu [13], we redefine these goals, excluding abuse prevention, for our security analysis:

Definition 5. (Confidentiality) *If only legitimate meeting participants can view the decrypted meeting streams, then it ensures the confidentiality that the meeting stream is kept secret from all but those who are authorized to view it.*

Definition 6. (Integrity) *If a meeting stream is received and successfully verified as message authentication, then it ensures the data integrity that the meeting stream has not been altered by unauthorized or unknown means.*

In our security analysis, we focus on *authenticity* rather than abuse prevention, and this term is defined with reference to the handbook written by Manezes et al. [19] as follows:

Definition 7. (Authenticity) *If a meeting stream is received and successfully verified as entity authentication, then it ensures the authenticity that the meeting stream was indeed sent by a particular meeting participants.*

3 E2EE Specifications for Zoom Meetings

The E2EE specifications for Zoom meetings is written in the whitepaper published by Zoom [14]. This section describes the system components, cryptographic algorithms, protocol flow, and local key security, which is the focus of our security analysis.

3.1 System Components

This subsection describes the signaling channel and bulletin board among the system components.

The signaling channel is used to distribute encrypted messages between meeting participants. Meeting participants route control messages on TLS tunnels over TCP, through the multimedia routers, which are a part of the Zoom infrastructure. TLS is terminated at the Zoom servers.

Each meeting has its own bulletin board that is accessible to the meeting participants. Meeting participants can post cryptographic messages to the bulletin board, which is implemented over the signaling channel.

The Zoom server controls the signaling channel and the bulletin board, and therefore, it can tamper with the cryptographic messages posted on the bulletin board.

3.2 Cryptographic Algorithms

The E2EE for Zoom meetings adopts the following cryptographic algorithms and uses the signing scheme and authenticated public-key encryption scheme:

- All meeting streams are encrypted with AES-GCM [2].
- Key derivation uses the HKDF algorithm [16].
- Diffie–Hellman (DH) over Curve25519 is used for key exchange [3].
- EdDSA over Ed25519 is used for signing [4].

The signing scheme consists of the key generation algorithm $\mathsf{Sign.KeyGen}$, the signing algorithm $\mathsf{Sign.Sign}$, and the verification algorithm $\mathsf{Sign.Verify}$. $\mathsf{Sign.KeyGen}$ generates a keypair (vk, sk), where vk and sk denote a verification and signing key, respectively. $\mathsf{Sign.Sign}$ takes a context string $\mathsf{Context}$ and a message M as the inputs and outputs a signature Sig over $\mathsf{SHA256}(\mathsf{Context}) \parallel \mathsf{SHA256}(M)$. $\mathsf{Sign.Verify}$ takes a signature Sig a context string $\mathsf{Context}$ and a message M as the inputs, and outputs True upon verification success and False upon failure.

The authenticated public-key encryption scheme consists of the key generation algorithm $\mathsf{Box.KeyGen}$, the encryption algorithm $\mathsf{Box.Enc}$, and the decryption algorithm $\mathsf{Box.Dec}$. $\mathsf{Box.KeyGen}$ generates a keypair $(pk_{\mathsf{Box}}, sk_{\mathsf{Box}})$, where pk_{Box} and sk_{Box} denote a public key and a secret key, respectively. $\mathsf{Box.Enc}$ takes the sender's secret key $sk^{\mathsf{S}}_{\mathsf{Box}}$, receiver's public key $pk^{\mathsf{R}}_{\mathsf{Box}}$, a context string $\mathsf{Context}_{\mathsf{KDF}}$ and $\mathsf{Context}_{\mathsf{cipher}}$, metadata Meta, and a message M as the inputs, and outputs a ciphertext C as follows:

1. Generate a 192-bit random string $\mathsf{RandomNonce}$.
2. Compute $K' \leftarrow \mathsf{DHKE}(pk^{\mathsf{R}}_{\mathsf{Box}}, sk^{\mathsf{S}}_{\mathsf{Box}})$, which is the DH key exchange.
3. Compute $K \leftarrow \mathsf{HKDF}(K', \mathsf{Context}_{\mathsf{KDF}})$, using an empty HKDF salt.
4. Compute $D \leftarrow \mathsf{SHA256}(\mathsf{Context}_{\mathsf{cipher}}) \parallel \mathsf{SHA256}(\mathsf{Meta})$.
5. Encrypt the plaintext M with XChaCha20/Poly-1305 taken the symmetric key K, the associated data D, and the nonce $\mathsf{RandomNonce}$ as the inputs, and return the ciphertext C'.
6. Output $C \leftarrow (C', \mathsf{RandomNonce})$.

$\mathsf{Box.Dec}$ takes the receiver's secret key $sk^{\mathsf{R}}_{\mathsf{Box}}$, sender's public key $pk^{\mathsf{S}}_{\mathsf{Box}}$, a context string $\mathsf{Context}_{\mathsf{KDF}}$ and $\mathsf{Context}_{\mathsf{cipher}}$, metadata Meta, and a ciphertext C as the inputs, and outputs a message M or error as follows:

1. Parse C as $(C', \mathsf{RandomNonce})$.
2. Compute $K' \leftarrow \mathsf{DHKE}(pk^{\mathsf{R}}_{\mathsf{Box}}, sk^{\mathsf{S}}_{\mathsf{Box}})$, which is the DH key exchange.
3. Compute $K \leftarrow \mathsf{HKDF}(K', \mathsf{Context}_{\mathsf{KDF}})$, using an empty HKDF salt.
4. Compute $D \leftarrow \mathsf{SHA256}(\mathsf{Context}_{\mathsf{cipher}}) \parallel \mathsf{SHA256}(\mathsf{Meta})$.
5. Encrypt the ciphertext C' with XChaCha20/Poly-1305 taken the symmetric key K, the associated data D, and the nonce $\mathsf{RandomNonce}$ as the inputs, and return the plaintext M.
6. If decryption fails, then output error. Otherwise, output M.

When Zoom user i upgrades their Zoom application to the first version that supports E2EE, they generate a long-term signature key pair (IVK_i, ISK_i) with $\mathsf{Sign.KeyGen}$, where IVK_i and ISK_i denote a verification key and a signing key for user i, respectively. Subsequently, they post IVK_i to the Zoom server, and store ISK_i on their device. They continue to use the long-term signing key pair unless they reinstall the OS or applications and destroy the disk.

3.3 Join/Leave Protocol Flow

The protocol to establish an E2EE session for Zoom meetings consists of four phases: *participant key generation*, *leader join*, *participant join* (*leader*), and *participant join* (*non-leader*) *phases*. After the E2EE session is established, the meeting leader/participants encrypt all meeting streams with AES-GCM using the meeting key MK shared during the participant join (leader/non-leader) phase as an input. We call this phase the *encryption phase*.

Participant Key Generation Phase. When any participant i joins the meeting $\mathsf{meetingID}$ on their device $\mathsf{deviceID}$, they perform the following procedures:

1. Generate a new keypair $(pk_i, sk_i) \leftarrow \mathsf{Box.KeyGen}()$ for the DH key exchange.
2. Query the insider for the server-generated $\mathsf{meetingUUID}$ for the meeting. No participant has any control over the $\mathsf{meetingUUID}$.
3. Compute $\mathsf{Binding}_i \leftarrow (\mathsf{meetingID} \,\|\, \mathsf{meetingUUID} \,\|\, i \,\|\, \mathsf{deviceID} \,\|\, IVK_i \,\|\, pk_i)$.
4. Define $\mathsf{Context} \leftarrow$ "Zoombase-1-ClientOnly-Sig-EncryptionKeyAnnouncement".
5. Compute $\mathsf{Sig}_i \leftarrow \mathsf{Sign.Sign}(ISK_i, \mathsf{Context}, \mathsf{Binding}_i)$.
6. Store sk_i for the duration of the meeting.
7. Post Sig_i to the bulletin board, so that all participants can see it.

Leader Join Phase. When any leader joins the meeting $\mathsf{meetingID}$, they perform the following procedures:

1. Fetch $\mathsf{meetingUUID}$ from the insider.
2. Generate a 32-byte seed MK using a secure random number generator
3. Get the full list of meeting participants I from the insider.
4. Perform the "Participant Join (Leader)" phase for each participant $i \in I$.

Participant Join (Leader) Phase. When a leader ℓ and a participant i join the meeting $\mathsf{meetingID}$ on $\mathsf{deviceID}$, the leader performs the following procedures:

1. Fetches IVK_i from the insider.
2. Fetches Sig_i and pk_i from the bulletin board in the meeting.
3. Computes $\mathsf{Binding}_i \leftarrow (\mathsf{meetingID} \,\|\, \mathsf{meetingUUID} \,\|\, i \,\|\, \mathsf{deviceID} \,\|\, IVK_i \,\|\, pk_i)$.
4. Defines $\mathsf{Context}_{\mathsf{sign}} \leftarrow$ "Zoombase-1-ClientOnly-Sig-EncryptionKeyAnnouncement".
5. Verifies the signature: $\mathsf{Sign.Verify}(IVK_i, \mathsf{Sig}_i, \mathsf{Context}_{\mathsf{sign}}, \mathsf{Binding}_i)$.
6. If verification fails, it is aborted.

7. Computes $\mathsf{Meta} \leftarrow (\mathsf{meetingID} \,\|\, \mathsf{meetingUUID} \,\|\, \ell \,\|\, i)$.
8. Defines $\mathsf{Context}_{\mathsf{KDF}} \leftarrow$ "Zoombase-1-ClientOnly-KDF-KeyMeetingSeed".
9. Defines $\mathsf{Context}_{\mathsf{cipher}} \leftarrow$ "Zoombase-1-ClientOnly-Sig-EncryptionKeyMeetingSeed".
10. Computes $C_i \leftarrow \mathsf{Box.Enc}(sk_\ell, pk_i, \mathsf{Context}_{\mathsf{KDF}}, \mathsf{Context}_{\mathsf{cipher}}, \mathsf{Meta}, \mathsf{MK})$.
11. Posts (i, C_i) to the bulletin board.

Participant Join (Non-Leader) Phase. When any participant i joins the meeting meetingID on deviceID, they perform the following procedures:

1. Fetch IVK_ℓ for the leader ℓ and the meetingUUID from the insider.
2. Fetch Sig_ℓ, pk_ℓ, and (i, C_i) from the bulletin board in the meeting.
3. Compute $\mathsf{Binding}_\ell \leftarrow (\mathsf{meetingID} \,\|\, \mathsf{meetingUUID} \,\|\, \ell \,\|\, \mathsf{deviceID} \,\|\, IVK_\ell \,\|\, pk_\ell)$.
4. Define $\mathsf{Context}_{\mathsf{sign}} \leftarrow$ "Zoombase-1-ClientOnly-Sig-EncryptionKeyAnnouncement".
5. Verify the signature: $\mathsf{Sign.Verify}(IVK_\ell, \mathsf{Sig}_\ell, \mathsf{Context}_{\mathsf{sign}}, \mathsf{Binding}_\ell)$.
6. If verification fails, it is aborted.
7. Compute $\mathsf{Meta} \leftarrow (\mathsf{meetingID} \,\|\, \mathsf{meetingUUID} \,\|\, \ell \,\|\, i)$.
8. Define $\mathsf{Context}_{\mathsf{KDF}} \leftarrow$ "Zoombase-1-ClientOnly-KDF-KeyMeetingSeed".
9. Define $\mathsf{Context}_{\mathsf{cipher}} \leftarrow$ "Zoombase-1-ClientOnly-Sig-EncryptionKeyMeetingSeed".
10. Compute $\mathsf{MK} \leftarrow \mathsf{Box.Dec}(sk_i, pk_\ell, \mathsf{Context}_{\mathsf{KDF}}, \mathsf{Context}_{\mathsf{cipher}}, \mathsf{Meta}, C_i)$.

3.4 Local Key Security

To protect a long-term signing key stored on the device, each Zoom user encrypts the long-term signing key with the committing AEAD scheme CtE1 [10] as follows:

1. Generates a 32-byte random string KWK, which is called the key-wrapping key, and requests the server to store it persistently associated with the user.
2. Defines Context $\leftarrow$ "Zoombase-1-ClientOnly-KDF-SecretStore".
3. Computes $C \leftarrow$ CtE1-Enc(K=KWK, H=Context, M=ISK_i), where H is the associated data parameter for the underlying AEAD, and stores it in the system keychain.

If two Zoom users share a device, the local key security can prevent a malicious user from exploiting another user's long-term signing key.

4 Impersonation Based on No Entity Authentication

This section describes how a malicious meeting leader/participant who possesses the shared meeting key can impersonate other legitimate meeting participants. This exploits the following vulnerability during the encryption phase.

Vulnerability 1 (No Entity Authentication) *Even if a meeting stream is received from a particular meeting participant, the authenticity of the meeting stream is not ensured because there is no entity authentication.*

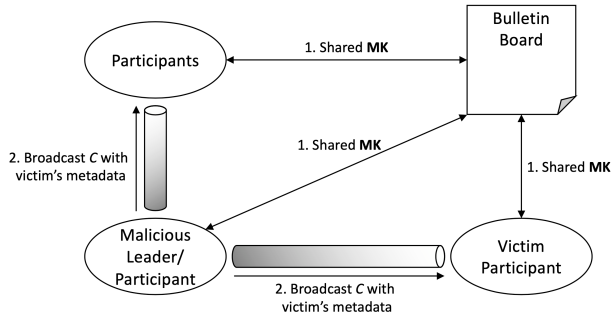

Fig. 1. Impersonation based on Vulnerability 1.

In the encryption phase, all meeting participants broadcast the meeting streams encrypted with AES-GCM. Although AES-GCM ensures the confidentiality and integrity of the meeting streams, it does not ensure the authenticity because of the lack of the entity authentication. In fact, Section 3.12 of the whitepaper [14] states that properly signing all meeting streams is a challenge from the perspective of performance and repudiation, i.e., it is clear that there is no entity authentication in the encryption phase. In this section, we show a practical attack scenario and provide its feasibilities and countermeasure.

4.1 Impersonation Based on Vulnerability 1

By exploiting Vulnerability 1, a malicious meeting leader/participant impersonates any legitimate meeting participant (victim) in the following scenario (see also Figure 1):

1. A malicious meeting leader/participant joins the meeting as a legitimate meeting leader/participant and derives the shared meeting key MK during the participant join (leader/non-leader) phase.
2. They encrypt meeting streams M with MK and broadcasts the encrypted meeting streams C with the victim's metadata, e.g., sender ID, to all meeting participants via Zoom infrastructure.

Since the meeting stream M is encrypted with meeting key MK shared among all meeting participants, they can decrypt it and successfully verify it as message authentication. In addition, the attached metadata makes non-victim meeting participants unaware that the encrypted meeting stream C was broadcast by the malicious meeting leader/participant. The victim should be aware of this fact but cannot formally refute it because of the lack of the entity authentication. Therefore, this reveals that the E2EE for Zoom meetings does not ensure the authenticity of the meeting streams against a malicious meeting leader/participant.

4.2 Discussion

This subsection discusses feasibilities and a countermeasure against the impersonation attack described in the previous subsection.

Feasibility. To impersonate any legitimate meeting participant (victim), a malicious meeting leader/participant must prepare the victim's meeting streams in advance. This is feasible by collecting the meeting streams from the meetings the victim previously joined and editing them. The impersonating based on Vulnerability 1 has the following feasibilities:

- If the victim is not broadcasting a meeting stream, then other meeting participants properly receive a meeting stream prepared in advance by the malicious meeting leader/participant. This causes the victim to lose the trust of other meeting participants, depending on the content of the broadcast meeting stream.
- If the victim is broadcasting a meeting stream, then his meeting stream conflicts with a meeting stream prepared in advance by a malicious meeting leader/participant. This causes interference in the victim's communication and prevents other meeting participants from properly receiving the content of their meeting stream.

Countermeasure. To prevent the impersonation based on Vulnerability 1, all meeting streams should be properly signed as entity authentication. As mentioned earlier, the whitepaper [14] states this countermeasure as a challenge from the perspective of performance and repudiation, and therefore, it will be an important task in the future.

Note that other E2EE schemes, such as WhatsApp [23], Facebook Messenger [5], and Google Duo [20], also have the same limitation in their current-deployed version. On the other hand, SFrame [7], which is an end-to-end media encryption mechanism, has an optional feature to sign all media stream by the sender's signature key.

4.3 Response from Zoom

Zoom also recognized this type of impersonating attacks as discussed in the whitepaper [14]. Due to performance and repudiability concerns, they are currently not ready to implement the countermeasure. However they told us that they will be open to re-evaluating it in the future.

5 Impersonation of any Zoom User

This section presents how a malicious insider or a malicious outsider can impersonate even any legitimate Zoom user who is uninvited to the target meeting. This exploits the following vulnerabilities in addition to Vulnerability 1 described in Section 4.

Vulnerability 2 (Free Access to the Bulletin Broad) *Insiders and meeting participants have free access to the bulletin board. Particularly, insiders are free to collect and tamper with all values, including the signatures and public keys generated by individual participants, posted on the bulletin board.*

This vulnerability is based on the description of the bulletin board in Section 3.1. During the participant join (leader/non-leader) phase, the encrypted meeting key and the public key and signature pairs for all meeting participants are posted on the bulletin board. Hence, this vulnerability allows the insiders and all meeting participants to collect them, and further allows the insiders to tamper with them.

Vulnerability 3 (Same Binding as in the Previous Meeting) *If the meeting IDs, which are* $\mathsf{meetingID}$ *and* $\mathsf{meetingUUID}$*, generated by the insiders and the public key generated by the meeting participant are reused, then the metadata* $\mathsf{Binding}$ *of the meeting participant has the same value. Since the signing key pair of the meeting participant is utilized for a long-term period, the same signature* Sig *is always generated from the same metadata* $\mathsf{Binding}$*.*

The metadata $\mathsf{Binding}_i$ of the meeting participant i is computed as described in Section 3.3 (see Step 3 during the participant join (leader) phase). Meeting participants reuse i, $\mathsf{deviceID}$, and IVK_i as fixed values in all meetings excluding special cases, e.g., after the Zoom application is reinstalled. Hence, if you get the tuple $(\mathsf{meetingID}, \mathsf{meetingUUID}, pk_i)$ used in the previous meeting, then you can compute $\mathsf{Binding}_i$ used in the previous meeting. Only the insiders are involved in generating both $\mathsf{meetingID}$ and $\mathsf{meetingUUID}$, i.e., only malicious insiders can exploit Vulnerability 3.

Vulnerability 4 (Leader-generated Meeting Key) *Only the meeting leader is involved in generating a 256-bit shared meeting key.*

This vulnerability implies that a malicious meeting leader may intentionally reuse the meeting key MK used in the previous meeting.

5.1 Impersonation Based on Vulnerabilities 1-3

By exploiting Vulnerabilities 2 and 3, a malicious insider can impersonate even any legitimate Zoom user A uninvited to the target meeting in the following scenario (see also Figure 2):

1. A malicious insider stores Sig_A and pk_A posted on the bulletin board in the previous meeting.
2. They reuse $\mathsf{meetingID}$ and $\mathsf{meetingUUID}$ used in the previous meeting.
3. They post Sig_A and pk_A to the bulletin board in the new meeting.

During the participant join (leader) phase, a meeting leader can compute $\mathsf{Binding}_\mathsf{A}$ used in the previous meeting from the same $\mathsf{meetingID}$, $\mathsf{meetingUUID}$, and pk_A. Since Sig_A is the value derived from signing $\mathsf{Binding}_\mathsf{A}$ with ISK_A, the meeting leader can successfully verify Sig_A with IVK_A. Therefore, this reveals that a malicious insider can impersonate any legitimate Zoom user A without being noticed

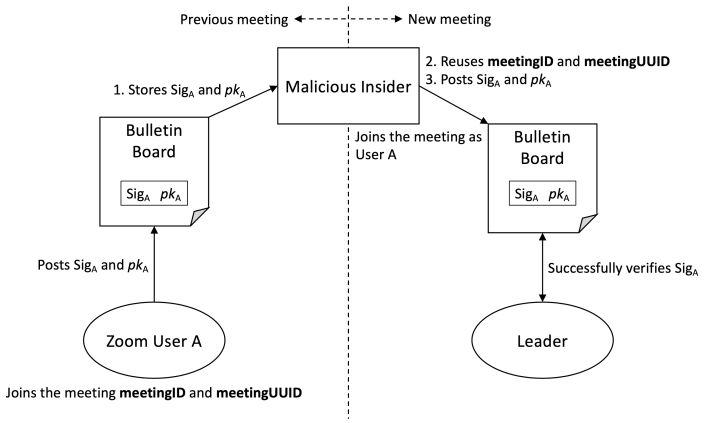

Fig. 2. Impersonation based on Vulnerabilities 2 and 3.

by him. The malicious insider cannot derived MK in the meeting because they do not know sk_{A} corresponding to pk_{A}, i.e., they can join the meeting but cannot decrypt the meeting streams.

Now, we suppose that the malicious insider colludes with the malicious meeting leader. In this scenario, if the malicious insider obtains the shared meeting key MK from the malicious meeting leader, then the malicious insider can completely impersonate legitimate Zoom user A. Hence, the malicious insider will be able to not only join the meeting as Zoom user A, but also properly broadcast and receive the meeting streams by exploiting Vulnerability 1. Given that non-legitimate meeting participants cannot completely impersonate by simply obtaining the shared meeting key MK, such a scenario is not trivial.

5.2 Impersonation Based on Vulnerabilities 1-4

By exploiting Vulnerabilities 1-4, a malicious outsider can impersonates even any legitimate Zoom user B uninvited to the target meeting in the following scenario (see also Figure 3):

1. A malicious meeting leader stores $\mathsf{Sig}_{\mathsf{B}}$ and pk_{B} posted on the bulletin board in the previous meeting and provides them to a malicious outsider.
2. A malicious insider reuses meetingID and meetingUUID used in the previous meeting.
3. If the malicious outsider joined the previous meeting, then the malicious meeting leader reuses MK used in the previous meeting, i.e., the malicious outsider knows the reused MK, otherwise provides it to the malicious outsider.
4. The malicious outsider posts $\mathsf{Sig}_{\mathsf{B}}$ and pk_{B} to the bulletin board in the new meeting.

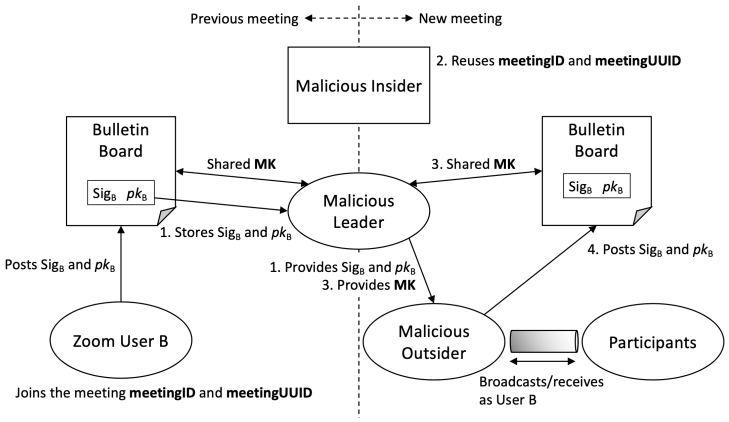

Fig. 3. Impersonation based on Vulnerabilities 1-4. Note that the malicious meeting leader and the malicious outsider join the previous and new meetings.

For the above scenario to be reality, the malicious outsider has to collude with the malicious insider and malicious meeting leader. This scenario is similar to the one discussed in Section 5.1, but this scenario supposes that the malicious meeting leader wants the malicious outsider to impersonate Zoom user B. In addition, by exploiting Vulnerabilities 1 and 4, she can not only join the meeting, but also properly broadcast and receive meeting streams.

5.3 Impersonation Based on Vulnerabilities 1-4 without Colluding with a Malicious Insider

We explain how a malicious outsider can impersonate even any legitimate Zoom User B uninvited to the target meeting without colluding with a malicious insider in the following scenario:

1. A malicious outsider stores Sig_B and pk_B posted on the bulletin board in the previous meeting.
2. A malicious meeting leader uses meetingID as the personal meeting ID.
3. The malicious outsider collects the meetingUUID generated by a malicious insider with the meetingUUID used in the previous meeting.
4. If the malicious outsider joined the previous meeting, then the malicious meeting leader reuses MK used in the previous meeting, i.e., the malicious outsider knows the reused MK, otherwise provides it to the malicious outsider.
5. The malicious outsider posts Sig_B and pk_B to the bulletin board in the new meeting.

To realize the above scenario, the malicious outsider only needs to collude with the malicious meeting leader. To generate meetingID, a meeting leader can choose to automatically generate it with the help of the insiders or use a fixed value as

the personal meeting ID. If a malicious meeting leader generates meetingID as a personal meeting ID, then the meeting participants use the same meetingID as the previous meeting. In addition, if meetingUUID is generated according to RFC 4122 [17] (although we do not know if this is actually correct because the generation process of a MeetingUUID is not disclosed in the whitepaper), the meetingUUID is identical to the previous meetingUUID in 2^{61} trials by executing a birthday attack. Based on these procedures, the malicious outsider can probabilistically use the same meetingID and meetingUUID as in the previous meeting without colluding with a malicious insider.

5.4 Discussion

This subsection discusses feasibilities and two countermeasures against the impersonation attacks described in the previous subsections.

Feasibility. It can be effective in some cases to show other meeting participants that a specific individual is just joining a meeting without broadcasting anything. For example, suppose you want to make some negotiations proceed smoothly but the negotiating partner has joined the meeting with a malicious insider who impersonates an influential person, as described in Section 5.1. You may feel that the partner imposes silent pressure, and the negotiation may not proceed as desired (rather, we think that the negotiations proceed at the partner's pace). Since a malicious insider can easily perform such impersonation, we suppose a scenario in which meeting participants request the malicious insider to impersonate a specific individual. Therefore, the impersonation attacks described in the previous subsections is feasible.

We further discuss a feasibility against the impersonation attack described in Section 5.3. Zoom Video Communications announced on its blog that more than 300 million daily meeting participants join Zoom meetings as of April 2020 [24]. Assuming that all meetings have only two participants, only $2^{35.67}$ meetings will be held worldwide in a year. Therefore, there is not much feasibility of executing a birthday attack with 2^{61} trials to make meetingUUID coincide with the previous meetingUUID. Even if the meetingUUID coincides with the previous meetingUUID, how a malicious outsider posts Sig_B and pk_B to the bulletin board implemented on the signaling channel is an open problem. We suggest that a malicious meeting leader posts Sig_B and pk_B on behalf of a malicious outsider as one solution, but we cannot confirm the feasibility of this attack. In summary, although there is not much feasibility of impersonating even any legitimate Zoom User B uninvited to the target meeting without colluding with a malicious insider, the protocol must include countermeasures in the event of such an impersonation.

Countermeasure. To prevent the impersonating described in the previous subsections, we propose the following countermeasures against Vulnerability 3:

1. Add time information time, e.g., the date and time when the meeting starts, to the metadata $\mathsf{Binding}_i$ as follows:

$$\mathsf{Binding}_i \leftarrow (\mathsf{meetingID} \parallel \mathsf{meetingUUID} \parallel i \parallel \mathsf{deviceID} \parallel IVK_i \parallel pk_i \parallel \mathsf{time}).$$

2. Add a procedure to verify the time information when verifying the signature.

If an adversary attempts to exploit Vulnerability 3, then the time information must be the same as that in the previous meeting. In addition, even if the adversary uses the same time information as in the previous meeting, by detecting the time information mismatch when verifying the signature, the adversary can be prevented from exploiting Vulnerability 3.

5.5 Response from Zoom

Zoom acknowledged these limitations for user identification in the current version. They told us that they will clearly state it as a limitation of the currently-deployed end-to-end encryption (Phase 1), and update the protocol to prevent it before the next phases are deployed (Phases 2 and 3).

To be more specific, in the currently deployed version, there are no cryptographic mechanisms preventing anyone from changing their display name to whatever they please. They will address this issue before Phase 2 is deployed. For completeness, note that in some cases an account admin can instruct the Zoom server to prevent display name changes for its members, but this server-enforced feature is not meant to protect against Zoom insiders.

6 Impersonation of Another User on a Shared Device

This section presents how a malicious insider or a malicious outsider can impersonate another legitimate Zoom user who shares their device with the malicious outsider. This exploits the following vulnerability in storing a long-term signing key.

Vulnerability 5 (Long-term Signing Key) *The long-term signing key ISK of the meeting participant is encrypted using a server-synchronized key-wrapping key* KWK*. The ciphertext is stored on the participant's device, which may be a computer shared by two users, and* KWK *is stored on the Zoom server.*

This vulnerability is based on the description of the local key security (Section 3.4). The description suggests that a malicious outsider can obtain the encrypted long-term signing key stored on the shared device. In addition, the key-wrapping key KWK for encryption is stored on the Zoom server. Hence, the long-term signing key of the victim may be decrypted by colluding with a malicious insider and a malicious outsider.

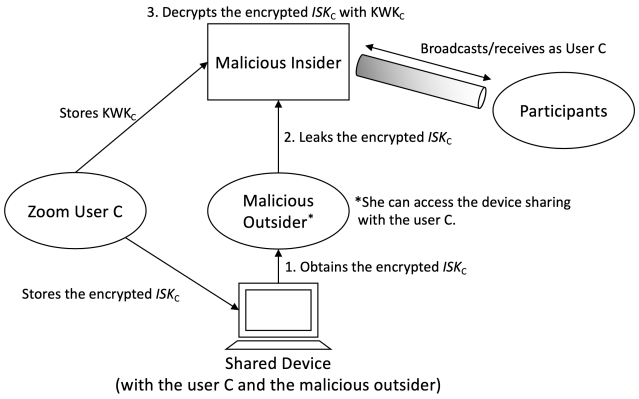

Fig. 4. Impersonation based on Vulnerabilities 1 and 5.

6.1 Impersonation Based on Vulnerabilities 1 and 5

By exploiting Vulnerabilities 1 and 5, a malicious insider or a malicious outsider impersonates another legitimate Zoom user C who shares the device with the malicious outsider in the following scenario (see also Figure 4):

1. A malicious outsider obtains the encrypted long-term signing key for user C.
2. The malicious outsider leaks the ciphertext to the malicious insider or obtains the KWK associated with user C from the malicious insider.
3. One of them decrypts the ciphertext with the KWK.

It is a realistic situation that a malicious outsider leaks the encrypted long-term signing key for user C to a malicious insider and the malicious insider decrypts it. Anyway, if the long-term signing key is obtained by a malicious insider or a malicious outsider, then they can join any meeting and impersonate user C without being noticed by all meeting participants. Based on Vulnerability 1, they can also properly broadcast and receive meeting streams.

6.2 Discussion

In this subsection, we propose the following countermeasures against the impersonation described in the previous subsection:

1. Store KWK on a trusted third party.
2. Use the secret sharing scheme to store KWK.

Since it is difficult to prevent a malicious outsider from obtaining the encrypted long-term signing key for user C, we examine how to store KWK securely. According to the whitepaper [14], Zoom Video Communications outsources the identity management and system auditing to trusted third parties. Similarly, it is realistic to outsource storing KWK to a trusted third party. If the adversary colludes with

the third party or with all insiders that possess a share, the proposed countermeasures cannot prevent the impersonation. However, we believe that adopting the proposed countermeasures can reduce the risk of impersonation.

6.3 Response from Zoom

Zoom explicitly stated that it is incredibly difficult to guarantee protection against any type of breach when an adversary has gained control of a target's device. Therefore, they revealed that the local key security mechanism (Section 3.4) is not meant to protect against collusion between a malicious insider and a malicious outsider with access to the user's device. They plan to update the whitepaper [14] to clarify that this fact is correct.

They also provided an example of many other attack vectors that would allow a malicious outsider to compromise an accessible target's device without colluding with a malicious insider. For example, the malicious outsider might install spyware which waits until the other user logs in and records the key while it is decrypted in memory. We agreed that such an attack is also feasible.

7 Security against Tampering with Meeting Streams

This section evaluates the security against tampering with meeting streams during the encryption phase. Now, the following vulnerability related to AES-GCM is exploited.

Vulnerability 6 (Misuse of Nonce) *All meeting streams are encrypted with AES-GCM. If nonce is misused during the meeting, the existing attack on AES-GCM [8, 15, 18] will be executed and the authentication key will be exposed to third parties.*

Section 3.10 of the whitepaper [14] states that nonces are generated by counters. However, the possibility that a malicious insider could intentionally embed a vulnerability that allows meeting participants to reuse the same nonce should be considered.

7.1 Tampering Based on Vulnerability 6

By exploiting Vulnerability 6, a malicious insider tampers with the encrypted meeting streams in the following nonce-misused scenario:

1. A malicious insider embeds a vulnerability that allows meeting participants to reuse the same nonce.
2. A meeting leader/participant encrypts meeting streams with the reused nonce and broadcasts them to the meeting participants.
3. The malicious insider intercepts the streams sent over the network.
4. The malicious insider derives the authentication key from the streams based on the existing attack on AES-GCM in the nonce-misused setting [8, 15, 18].

The malicious insider cannot obtain the shared meeting key, but they can derive the authentication key in the meeting. Hence, in the above scenario, although there is no tampering with the meaningful meeting stream, the tampered streams can be successfully verified as message authentication.

7.2 Discussion

Even if a malicious insider does not intentionally embed a vulnerability, flaws in the Zoom system may lead to misuse of the nonce. To prevent exploitation of Vulnerability 6, we propose to adopt a misuse-resistant authenticated encryption (MRAE), which was formalized by Rogaway and Shrimpton [22], instead of AES-GCM. Numerous MRAEs are available, for example, the authenticated encryptions selected as finalists in the CAESAR project [1] and the AES-GCM-SIV standardized by the Internet Engineering Task Force [11]. Therefore, we strongly recommend the transition from AES-GCM, which has low misuse resistance, to a MRAE.

7.3 Response from Zoom

In Section 1.3 of the whitepaper [14], Zoom acknowledged that any unknown backdoors and bugs in their client code would compromise the confidentiality of their E2EE system. However, they argued that they have no such known backdoors, and they routinely commission audits by external companies to mitigate this threat - making it a highly unlikely attack vector.

They also provided examples of other attack vectors. For example, the backdoor could target the key generation algorithm or exfiltrate keys through other covert channels. In terms of such attack vectors, they stated that switching GCM with a MRAE would be an ineffective countermeasure; however, we emphasize that a MRAE is useful for enhancing the security of E2EE of Zoom.

8 Security against Denial of Service

This section evaluates the security against denial of service from the participant key generation phase to the participant join (leader) phase. This exploits Vulnerability 2 described in Section 5.

8.1 Denial of Service Based on Vulnerability 2

By exploiting Vulnerability 2, a malicious insider denies service to any legitimate meeting participant D in the following scenario (see also Figure 5):

1. A meeting participant D posts his signature $\mathsf{Sig}_{\mathsf{D}}$ and public key pk_{D} to the bulletin board.
2. A malicious insider replaces pk_{D} with a random value before a meeting leader verifies $\mathsf{Sig}_{\mathsf{D}}$.

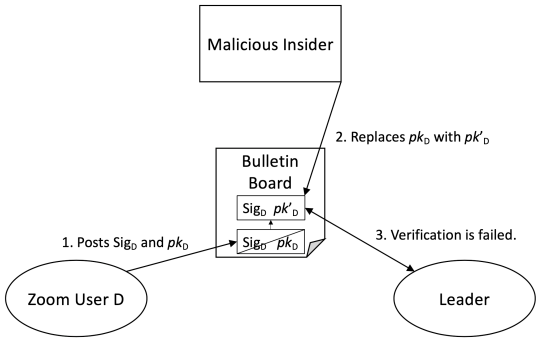

Fig. 5. Denial of service based on Vulnerability 2.

3. A meeting leader verifies Sig_D, but the verification is fails.

As described in Section 5, Vulnerability 2 allows the insider to tamper with any public key. If pk_D is replaced by the malicious insider, the meeting leader always fails to verify Sig_D because he cannot compute the correct metadata $\mathsf{Binding}_\mathsf{D}$ corresponding to Sig_D. Upon checking the bulletin board, the meeting participant D may notice that pk_D has been replaced by the malicious insider. However, if the above scenario is repeated by the malicious insider, meeting participant D will never be able to join the meeting.

8.2 Discussion

This subsection discusses feasibilities and countermeasures against denial of service.

Feasibility. If a malicious insider wants to deny all services in Zoom meetings to any legitimate meeting participant D, the malicious insider only needs to suspend the participant D's Zoom account. On the other hand, the denial of service based on Vulnerability 2 allows the malicious insider to only deny the participant D from joining a particular meeting. Now, we assume the following realistic scenario:

1. A meeting leader invites the participant D to a meeting.
2. Since other meeting participants do not want the participant D to join the meeting, they request insiders to perform the denial of service based on Vulnerability 2 to prevent participant D from joining the meeting.

If the above scenario happened realistically, the malicious insider should find it more feasible to perform the denial of service based on Vulnerability 2 than to suspend the user account because Zoom Video Communications may be criticized by general Zoom users. Therefore, the denial of service based on Vulnerability 2 is feasible.

Countermeasure. From our knowledge based on the whitepaper [14], the Zoom server does not need free access to the bulletin board. Therefore, we propose the following countermeasures against the denial of service:

- A trusted third party should control all the bulletin boards.
- All values in the bulletin board should be encrypted.

The first countermeasure can reduce the risk of the denial of service because there is no advantage for the third party to deny service to a particular user and the malicious insider must collude with the third party. Although the second countermeasure does not need outsourcing the control of the bulletin board to a third party, there is room for consideration of schemes such as encryption and key exchange, which remains as an open problem in the future.

8.3 Response from Zoom

Zoom acknowledged that the possibility of denial of service attacks is unavoidable and common to any centralized system, as they noted in Section 3.7 of the whitepaper [14]. Also, they provided an example of other attack vectors when communication between the parties is mediated by a server, regardless of the cryptographic protocol. For example, a malicious insider can always prevent a specific user from joining a meeting by simply refusing to forward their messages. We agreed that such an attack is also feasible.

9 Conclusion

In this study, we evaluated the security of E2EE for Zoom (version 2.3.1) and revealed several attacks more powerful than that expected by Zoom according to their whitepaper. Specifically, if insiders collude with meeting participants, they can impersonate any Zoom user in target meetings, whereas Zoom indicates that they can impersonate only current meeting participants. Besides, even without relying on malicious participants, insiders can impersonate any Zoom user for target meetings though they cannot decrypt the meeting stream. In addition, we discussed several impersonation attacks conducted by meeting participants or insiders colluding with meeting participants and discussed their feasibility in real-world scenarios. We also discussed effective countermeasures. We hope that our results are useful for enhancing the security of E2EE for Zoom.

Our study has focused on the E2EE mechanism for Zoom; thus, our proposed attacks cannot be applied directly to the E2EE mechanism for other video conferencing systems or messaging applications. This is because the E2EE mechanism for Zoom employs the original cryptographic protocol. From another point of view, we have proposed some countermeasures to mitigate our discovered vulnerabilities in the E2EE mechanism for Zoom. Therefore, we believe that the proposed countermeasures could contribute to the design of new E2EE mechanisms for video conferencing systems or messaging applications.

As described in Section 1, we have provided only theoretical evaluations of E2EE for Zoom, and we have confirmed that the proposed attacks are feasible in practice by discussing with the Zoom security team. However, we consider that we must demonstrate the feasibility of the proposed attacks and show its results. This is left as a future work.

Acknowledgments

The authors are grateful to security team of Zoom Video Communications, Inc. for the fruitful discussion and feedback about our findings. Takanori Isobe is supported by JST, PRESTO Grant Number JPMJPR2031 and SECOM science and technology foundation.

References

[1] CAESAR: Competition for Authenticated Encryption: Security, Applicability, and Robustness. `http://competitions.cr.yp.to/caesar.html`.

[2] 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.

[3] Daniel J. Bernstein. Curve25519: New Diffie-Hellman Speed Records. In Moti Yung, Yevgeniy Dodis, Aggelos Kiayias, and Tal Malkin, editors, *Public Key Cryptography - PKC 2006, 9th International Conference on Theory and Practice of Public-Key Cryptography, New York, NY, USA, April 24-26, 2006, Proceedings*, volume 3958 of *Lecture Notes in Computer Science*, pages 207–228. Springer, 2006.

[4] Daniel J Bernstein, Niels Duif, Tanja Lange, Peter Schwabe, and Bo-Yin Yang. High-speed high-security signatures. *Journal of cryptographic engineering*, 2(2):77–89, 2012.

[5] Katriel Cohn-Gordon, Cas J. F. Cremers, Benjamin Dowling, Luke Garratt, and Douglas Stebila. A formal security analysis of the signal messaging protocol. In *2017 IEEE European Symposium on Security and Privacy, EuroS&P 2017, Paris, France, April 26-28, 2017*, pages 451–466. IEEE, 2017.

[6] Katriel Cohn-Gordon, Cas J. F. Cremers, and Luke Garratt. On post-compromise security. In *IEEE 29th Computer Security Foundations Symposium, CSF 2016, Lisbon, Portugal, June 27 - July 1, 2016*, pages 164–178. IEEE Computer Society, 2016.

[7] Alexandre Gouaillard Sergio Murillo Emad Omara, Justin Uberti. Secure Frame (SFrame), 2020. `https://tools.ietf.org/html/draft-omara-sframe-00/`.

[8] Niels Ferguson. Authentication weaknesses in GCM. Comments on the Choice Between CWC or GCM to NIST, 2005.

[9] 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 *25th USENIX Security Symposium (USENIX Security 16)*, pages 655–672, Austin, TX, 2016. USENIX Association.

[10] Paul Grubbs, Jiahui Lu, and Thomas Ristenpart. Message Franking via Committing Authenticated Encryption. Cryptology ePrint Archive, Report 2017/664, 2017. `http://eprint.iacr.org/2017/664`.

[11] Shay Gueron, Adam Langley, and Yehuda Lindell. AES-GCM-SIV: Nonce Misuse-Resistant Authenticated Encryption. *Internet Engineering Task Force - IETF, Request for Comments*, 8452, April 2019.

[12] HackerOne, 2020. `https://hackerone.com/zoom?type=team`.

[13] 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, *Computer Security - 23rd European Symposium on Research in Computer Security, ESORICS 2018, Barcelona, Spain, September 3-7, 2018, Proceedings, Part II*, volume 11099 of *Lecture Notes in Computer Science*, pages 249–268. Springer, 2018.

[14] Josh Blum and Simon Booth and Oded Gal and Maxwell Krohn and Julia Len and Karan Lyons and Antonio Marcedone and Mike Maxim and Merry Ember Mou and Jack O ' Connor and Miles Steele and Matthew Green and Lea Kissner and Alex Stamos. E2E Encryption for Zoom Meetings – Version 2.3.1, 2020. `https://github.com/zoom/zoom-e2e-whitepaper`.

[15] Antoine Joux. Authentication Failures in NIST Version of GCM. Comments on The Draft GCM Specification to NIST, 2006.

[16] 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.

[17] Paul J. Leach, Michael Mealling, and Rich Salz. A Universally Unique IDentifier (UUID) URN Namespace. *Internet Engineering Task Force - IETF, Request for Comments*, 4122, July 2005.

[18] David A. McGrew and John Viega. The security and performance of the galois/counter mode of operation (full version). Cryptology ePrint Archive, Report 2004/193, 2004. `http://eprint.iacr.org/2004/193`.

[19] Alfred J. Menezes, Paul C. Van Oorschot, and Scott A. Vanstone. *Handbook of Applied Cryptography*. CRC press, 1996.

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

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

[22] Phillip Rogaway and Thomas Shrimpton. A provable-security treatment of the key-wrap problem. In Serge Vaudenay, editor, *EUROCRYPT 2006*, volume 4004 of *LNCS*, pages 373–390, St. Petersburg, Russia, May 28 – June 1, 2006. Springer, Heidelberg, Germany.

[23] WhatsApp. WhatsApp Encryption Overview, 2020. `https://www.whatsapp.com/security/WhatsApp-Security-Whitepaper.pdf`.

[24] Zoom Blog. 90-Day Security Plan Progress Report: April 22, 2020. `https://blog.zoom.us/90-day-security-plan-progress-report-april-22/`.