Enforcing End-to-end Security for Remote Conference Applications

Yuelin Liu*, Huangxun Chen†, Zhice Yang*
*School of Information Science and Technology, ShanghaiTech University, China
†IoT Thrust, Information Hub, Hong Kong University of Science and Technology (Guangzhou)

***Abstract*—Remote conference applications are increasingly widely used, but currently, their improper data encryption methods, proprietary implementations, and dial-in access cause concerns about privacy breaches. As such, there is a need for trustworthy and secure solutions for these production tools. In this paper, we present *mTunnel*, a transparent software layer in the host system for securing conference applications without sacrificing the key functionalities and convenience. The basic idea of *mTunnel* is to encrypt sensitive data, such as audio, video, text, *etc.*, before it is obtained by untrusted application clients. *mTunnel* leverages the audio and video streaming capabilities of the conference applications to tunnel the encrypted content to the remote end. *mTunnel* involves a software framework to accommodate the media interception and representation through I/O virtualization based on virtual drivers. Moreover, *mTunnel* supports complete E2EE group conversations even in a mixed IP and public switched telephone network (PSTN). We implement *mTunnel* and evaluate it with several commercial products. Results show its feasibility and overhead.**

1. Introduction

Real-time remote conference applications have been an essential communication mode since the outbreak of the pandemic [1]. They stream the user's audio and video to the end at a far distance to reproduce the user's presence through network and cloud technologies. Compared with traditional messaging applications like Telegram and WhatsApp, remote conference applications focus more on facilitating learning, collaboration, and productivity in enterprise situations. For this reason, they feature group video calls, screen sharing, and dial-in access over telecommunication networks. Currently, many home, business [2], and academic activities [3] still and are likely to continue relying on conference applications. Along with this trend, their privacy and security issues are increasingly attracting attention [4], [5], [6].

Data streaming within remote conference applications is not merely a practice of end-to-end data transmission over IP networks. As shown in Fig. 1, video and audio data have to be processed and forwarded by the cloud servers to achieve reliable and good service quality. However, in many cases, the cloud is neither transparent nor trustworthy. To name a few examples, Zoom was blamed for abusing user information for profit [7]. An Amazon employee was found stealing user data from the cloud [8]. These ill purposes also suggest that user data is of high attacking value and faces threats along its way to the remote end.

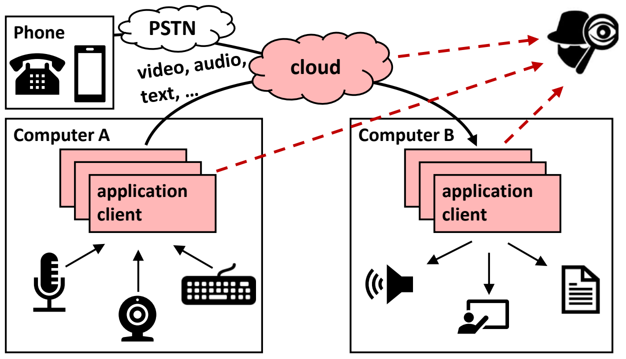


Figure 1: **Data Security Issues in Remote Conference Applications.** User data is vulnerable to privacy breaches due to untrusted clouds, blackbox client software, and unencrypted dial-in connections.

A common way to establish a trustworthy streaming path is to use end-to-end encryption (E2EE). E2EE ensures that no entities other than the participants can get access to the plaintext, which largely avoids the aforementioned threats. Some conference products claim to provide E2EE features [9], [10], but the problem remains. First, the application clients of widely-used commercial products are proprietary. Users have no easy means to audit their E2EE implementation, such as the key distribution process [11], [12]. Secondly, practical choices in conference tools must balance various factors, including security, service quality, availability, institutional subscription, *etc.*, which may lead to diverse choices in different circumstances. Moreover, as an essential feature of enterprise applications, dial-in access from telecommunication networks is currently incompatible with E2EE, as the cloud requires the use of unencrypted plaintext to exchange media information with the public switched telephone network (PSTN).

In this paper, we propose *mTunnel*, a software layer in the host system for securing conference applications without

Corresponding author: Zhice Yang

sacrificing their functionalities and convenience. *mTunnel*'s approach is to encrypt the media streams before they are acquired by the application clients. At the same time, it exposes a consistent media I/O interface that is transparent to existing programs. Its advantages are twofold. First, *mTunnel* encapsulates media data with an additional encryption layer, thereby preventing unauthorized clients and servers from snooping on media streams. Second, regardless of the chosen conference tools, *mTunnel* consistently offers E2EE capabilities upon them without introducing modifications.

To realize *mTunnel*, the first challenge is the encryption method. Raw media data, such as video and audio, are different from ordinary binary data as it contains a significant amount of redundancy [13], and the application client compress them in a lossy manner to reduce network overhead before transmitting. However, since the data is encrypted by *mTunnel* before entering the client, the encryption method has to be able to resist compression loss, but known secure ciphers[1] are all based on lossless data. Our solution is to treat the conference media streams as channels for tunneling the information rather than the information itself. Specifically, *mTunnel* first encrypts the original media stream into ciphertext chunks and then represent them in a new media stream consisting of frames of "barcode". The new stream is then fed to the client to transmit.

The second problem is how to apply the above method in a convenient and generalized manner, preferably without modifying application clients/servers as well as the host operating system. Similar to virtualization, *mTunnel* provides a consistent media input and output (I/O) interface to application clients through making full use of virtual drivers. The media content of the virtual driver is produced by a user-space *mTunnel* service process, which acquires and encrypts/decrypts the actual media content. Deploying *mTunnel* only requires the standard program installation on the host system.

Thirdly, remote conferences likely involve multiple participants, and *mTunnel* has to ensure security and trustworthiness among each and every participant. To achieve this, *mTunnel* is based on Signal, an E2EE protocol for group conversation [16]. *mTunnel* incorporates a module to manage key material and implements the full E2EE protocol over its media stream channels.

To evaluate the above designs, we implement *mTunnel* and test it with mainstream commercial products, including Zoom, Teams, and Tencent Meeting. Our evaluation shows its effectiveness and overhead under different network conditions and system configurations.

Our main contributions are as follows:

• We investigate the security risks in current remote conference solutions and propose a practical software layer in the host system to enforce end-to-end encryption.

• We propose systematic methods to utilize audio, video, and text streams of conference application clients for general-purpose data transmission. We validate their effectiveness across various clients and practical situations, including PSTN dial-in connections.

• We propose a lightweight I/O virtualization framework that utilizes virtual drivers to intercept and process host I/O data, thereby avoiding the need for modifications to the operating system and upper-layer software.

• We develop a full set of security protocols based on Signal to ensure E2EE for group conferences.

1. Encryption-then-compression methods, *e.g.*, chaos-based encryption [14], image block shuffling [15], are computational intensive and not cryptographically secure.

Conference Product	Download†	E2EE	OpenSource	PSTN
Ding Talk [17]	600+*	○	×	◧
Tencent meeting [18]	300+*	○	×	◧
BlueJeans [19]	1+	○	×	◧
Zoom [20]	500+	◐	×	◧
Microsoft Teams [21]	100+	◐	×	◧
Webex Meetings [22]	50+	◐	×	◧
Jitsi Meet [23]	10+	◐	✓	◧
Element [24]	1+	●	✓	□
- E2EE Tool				
mTunnel Layer	/	●	✓	■

○: Point-to-point encryption
◐: End-to-end encryption but not set by default
●: End-to-end encryption and set by default
□: No PSTN dial-in support
◧: PSTN dial-in support in P2PE mode
■: PSTN dial-in support in E2EE mode
†The majority of download data (in millions) is sourced from Google Play. For those marked with *, they have been obtained from the financial report of relevant companies, as they are not available on Google Play.

TABLE 1: **Remote Conference Applications Summary.**

2. Secure Communication Schemes in Remote Conference Applications

This section analyzes the security mechanisms employed by remote conference applications in the market. As shown in Table 1, we categorize them into three types: point-to-point encryption (P2PE), proprietary end-to-end encryption (P-E2EE), and open-source end-to-end encryption (O-E2EE).

Point-to-point encryption (P2PE) is similar to common TLS-based web access. It encrypts the data between the cloud server and application clients. However, from the server's perspective, the user's data streams remain unencrypted. Thus, a malicious insider or an external adversary who has compromised the server would still be able to intercept communications. P2PE allows companies to exploit user privacy [25], [26] and facilitates government censorship. For these reasons, P2PE is generally considered untrustworthy.

End-to-end encryption (E2EE), on the other hand, encrypts data between application clients. The clients authenticate each other, and the cryptographic keys are generated and kept confidential locally, ensuring that even the cloud servers do not have access to the encrypted content. Now then, many conference applications offer the E2EE option, but it is generally not enabled (for free). For example, Zoom employs P2PE as the default mode, and users have to manually enable E2EE. Teams only offers E2EE for Office

365 subscribers. However, for these proprietary solutions, there is often no easy way of auditing their implementation. In 2020, Zoom came under scrutiny for falsely claiming to offer end-to-end encryption [25]. Moreover, even with correct E2EE implementation, since user identities are still managed and verified in blackbox, the application providers can still intercept user data by employing man-in-the-middle attacks using forged identities. This echoes a survey conducted in 2021, which indicated that around one-third of respondents expressed a lack of trust in proprietary E2EE (P-E2EE) implementations [27].

In comparison to P-E2EE, open-source E2EE (O-E2EE) solutions allow users to audit E2EE implementation. However, they do not stand out in terms of functionality and service quality beyond the O-E2EE feature. This is one of the reasons why O-E2EE solutions only capture a small market share as shown in Table 1. Both P-E2EE and O-E2EE solutions offer the on-premise deployment, *i.e.*, in local IT infrastructure, to address concerns related to cloud processing. However, on-premise approach incurs higher deployment costs and does not completely mitigate threats from external adversaries.

In Table 1, another feature that both P-E2EE and O-E2EE fail to support is dial-in access. In many meeting scenarios, participants might need to join the conference using PSTN conference phones or cellular phones (for stability). The primary reason E2EE fails to work is that conference servers have to access the plaintext of the media to facilitate data exchange between IP and PSTN networks.

3. Threat Model and Design Goals

Our threat model includes two primary roles: the victims, who refer to the participants of the remote conference and want to keep their conversation content (*e.g.*, video/audio) confidential, and the adversary, who intends to obtain one or multiple victims' data transmitted through the conference applications. The conference application consists of two components: the application client, which is installed on the victim's host device, and the infrastructure, which includes the involved data transmission network and data processing servers . We make no assumptions about the implementation and deployment of the conference application infrastructure, *i.e.*, it can be cloud or on-premise.

We assume that participants have obtained information for verifying each other's identity, *i.e.*, identity and public key pairs, through other trusted channels before launching the conference.

We assume that the adversary is a network attacker, meaning it cannot physically capture/record the victims' screen or audio. We assume that the adversary has full control over the application infrastructure, meaning that the adversary can access, replay, and generate arbitrary on-path data and key information. Such attackers are most likely to originate from within the cloud service provider, but could also potentially come from external advanced attackers.

Through the application client, we assume that the adversary cannot compromise the victim's operating system

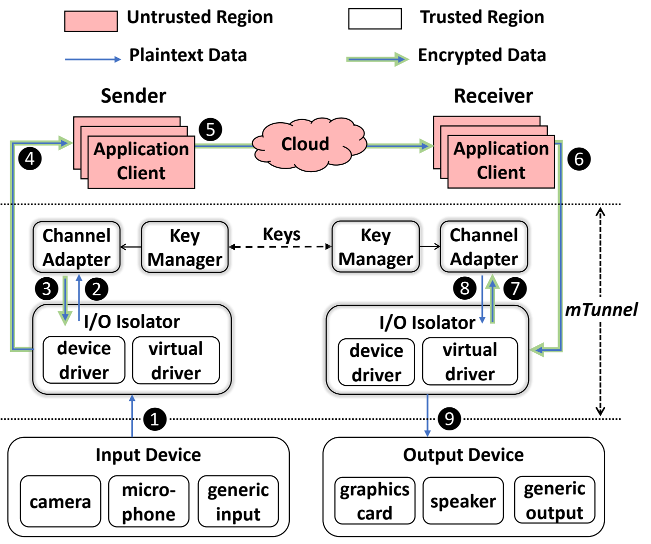


Figure 2: ***mTunnel* Architecture**

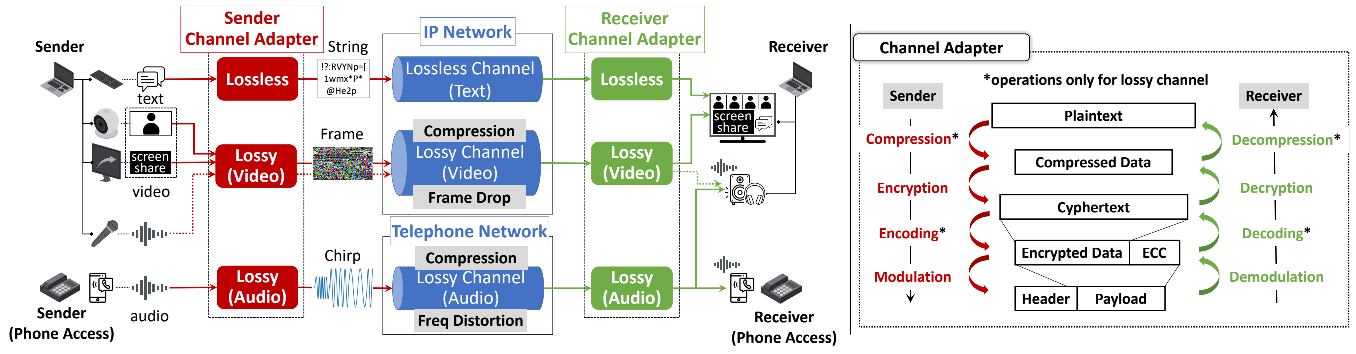
and can only access I/O resources specified by the victim. The latter is feasible since the victim can use effective flow control [28], permission management tools [29], open-source application client, or isolate the client in a sandbox.

Our design goal is to secure the victim's video, audio, and text information obtained and transmitted by the conference application client. In order to cover a wide user base, we do not consider rendering a new conference application. Instead, we want to leverage existing conference applications, including P2PE and P-E2EE implementations, no matter whether they are trustworthy or not. Additionally, we aim to preserve the functionalities of the original conference applications, such as allowing dial-in access, while avoiding modifying the application and the host operating system.

However, we do not protect against deny of service (DoS) attacks, as they do not pose privacy leakage issues. Also, we do not protect against privacy breaches related to the identity information of the victim's conference application account. When necessary, the victim can use fake identities and proxies when registering and using the service. We also do not protect against timing channels resulting from the message exchanges during the conference communication.

4. Design Overview

We present *mTunnel* in the following sections. As shown in Fig. 2, *mTunnel* is a software layer in the host system, transparent to application clients. It encrypts data streams prior to their delivery to the application clients, and decrypts them before their presentation to the user. In other words, we consider the original media channel to be a tunnel for transporting the actual content encapsulated by additional encryption operations. The design of *mTunnel* comprises three key components: Channel Adapter, I/O Isolator, and Key Manager.



(a) Channel Adapter Utilizes the Data Streams of Remote Conference Applications to Convey Information (b) Data Framing in Channel Adapter

Figure 3: **Overview of Channel Adapter**.

Channel Adapter (§5) aims to facilitate E2EE data transmission with general media streams. In particular, it conveys information over the media channels of existing conference applications, such as video and audio streams. Each channel has specific properties. Moreover, due to compression and transcoding, most media channels are inherently lossy. To enable reliable and secure transmission over lossy media channels, the channel adapter incorporates dedicated preprocessing, encryption, and modulation designs.

I/O Isolator (§6) is a lightweight middle layer that isolates untrusted application clients from accessing raw I/O resources. It utilizes virtual drivers to delegate the channel adapter to apply protection operations on media streams and abstract consistent I/O interfaces for upper-layer programs. I/O isolator is a universal solution that can work with all remote conference applications without requiring any client and server modifications.

Key Manager (§7) is designed to implement the E2EE protocol stack. When launching the conference, the key manager verifies the identities of the participants and generates initial pair-wise session keys. During the conversation, it advances the session keys used by the channel adapter.

The workflow of *mTunnel* is represented by the numbered arrows in Fig. 2. For ease of description, we consider a scenario with two participants, sender and receiver. ❶ On the sender's side, once the raw data is captured by input devices like cameras, microphones, and keyboards, it is first intercepted by the I/O isolator. ❷ The raw data is then passed to the channel adapter to undergo additional protection operations, including compression, encryption, encoding, and modulation. ❸ The channel adapter subsequently returns the encrypted data back to the I/O isolator. ❹ The I/O isolator then uses virtual drivers to present the processed data to the application clients as if the data is from actual device drivers. ❺ Then, the application clients process and stream the processed data to the remote receiver.

On the receiver's side, the procedures are reversed. ❻ The application clients post the encrypted content, *e.g.*, video, audio, text, *etc.*, to the operating system for presentation. ❼ The I/O isolator intercepts the streams and feeds them to the channel adapter for demodulation, decoding, decryption, and decompression. ❽❾ Afterwards, the recovered data is delivered to appropriate output devices, enabling users on the receiver end to consume the original content.

5. Design: Channel Adapter

As depicted in Fig. 3(a), the remote conference scenario covers a diverse range of data modalities, comprising text messages produced by keystrokes, video streams from cameras and screen sharing, and audio signals captured by microphones. Remote conference applications, *e.g.*, Zoom, serve as intermediaries to transmit data of multiple modalities from the sender to the receiver.

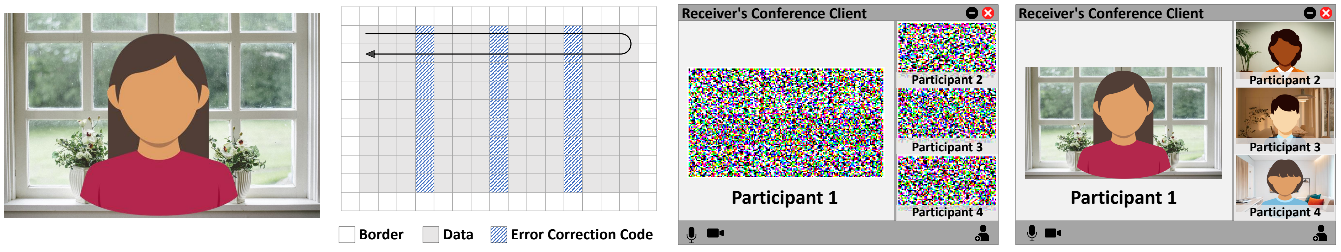
The transmission channels exposed by these data modalities can be classified into two primary categories: lossless channels and lossy channels. Text messages exemplify a lossless channel, as the data is transmitted reliably without undergoing any lossy modifications. As encrypting and decrypting messages on a lossless channel is similar to local operations, we omit its description in the main body (§A).

In contrast, video and audio streams are lossy channels. In order to ensure cryptographic security, we need to use ciphers on lossless input/output. However, there is no simple solution for achieving secure and reliable data transmission over such lossy channels. The reason is not only due to media compression, but also because of the complexity of network transmissions. Media streaming applications dynamically adjust compression rate, *i.e.*, the media channel capacity, based on network conditions [30], Additionally, these channels are subject to various unpredictable and inherent noises and distortions [31] caused by packet loss, subtle media gateway processing, *etc.*

As shown in Fig. 3(b), the Channel Adapter essentially applies an additional encapsulation layer over the encrypted content to enhance the resilience against lossy operations. We present the detailed Channel Adapter designs for video (§5.1) and audio (§5.2) streams in the following two subsections, respectively.

5.1. Lossy Channel Adapter for Video

Our approach is to compress and then encrypt the most essential information to be transmitted, and then project it



(a) Original Video Frame (b) 2D Barcode Structure (c) Client UI (d) Recovered and Recomposited Client UI

Figure 4: **Illustration of Video Channel and Re-composition**

into a space with sufficient redundancy to withstand the impact of channel loss. To achieve this, we employ a 2D barcode similar to QR codes, since they face similar communication challenges, but as our scenario is non-mobile, our code uses colors and smaller blocks to represent bits, leading to a higher coding density.

Sender Operations: As shown in Fig. 3(b), once the camera/screen frame intercepted by the I/O Isolator is received, the lossy channel adapter will first perform lossy compression on the frames, including downsizing and intra-frame compression to expand the redundancy for subsequent encoding. It is worth noting that inter-frame compression is intentionally avoided, *i.e.*, all frames are key frames. This is because a drop of a key frame in inter-frame compression can result in a set of subsequent frames becoming undecodable. Therefore, we only adopt intra-frame compression to ensure the scheme is not impacted by frame dropping caused by network congestion or proactive network flow control.

After compression, the lossy channel adapter encrypts the compressed frame data with key streams from the Key Manager. It appends an error correction coding (ECC) suffix for error resistance. We use a 4-byte Reed–Solomon code [32] (RS code) for every 100 bytes of data as ECC. Subsequently, the channel adapter employs a colored 2D barcode to convey the encoded data. As illustrated in Fig. 4(b), the code area consists of square blocks, with each block filled with one of 8 different colors indicated by the encoded data, allowing each block to represent 3 bits. The border of the code area is kept white to facilitate code locating for the receiver.

Receiver Operations: As depicted in Fig. 4(c) and Fig. 4(d), the I/O Isolator keeps monitoring the display frame buffer of the User Interface (UI) of the application client and reports the content to the channel adapter. The adapter uses edge detection to locate every 2D code in the client UI based on the large gradient of the white border. Note that the actual resolution of the received 2D code might vary with different client layouts, so the border is further utilized to locate each color block by calculating the resizing ratio. Subsequently, the channel adapter performs error correction, decryption, and decompression to recover the original transmitted frame and delivers it to the I/O Isolator for recomposition and display over the client UI.

Adaptation to Different Client Layouts: Depending on user preferences, the receiver's client may exhibit different layouts to arrange participants' video views. A typical layout is shown in Fig. 4(d). The speaker's camera view or the shared screen content occupies the majority of the UI space, while the thumbnail views of other participants are displayed on the sidebar. The different view sizes lead to different capacities of the corresponding video channels. A larger size allows for a larger code block size (more robust) and more code blocks (more information per code). By default, the sender chooses conservative configurations, *i.e.*, 4×4-pixel blocks for the speaker view and 8×8-pixel blocks for the thumbnail views. Currently, the size of the code area is fixed at 1280×720 (720p) pixels, leading to a capacity of 4.9k bytes[2] in one thumbnail view frame.

5.2. Lossy Channel Adapter for Audio

Conference applications handle video and audio streams in a similarly lossy way, but two key differences lead to the specific audio adapter designs. Firstly, the audio stream is a broadcast channel where the voices of all participants are mixed together in the time domain. Although it can be multiplexed in the frequency domain, our practice suggested that conference products limit the spectrum of their audio streams to the audible range[3] to reduce the network overhead, which inherently limits the number of concurrent participants. Secondly, an important feature of enterprise conference applications is the support for PSTN dial-in access, which, however, relies on transparent transcoding on the application cloud and hence is incompatible with current E2EE solutions. §5.2.1 and §5.2.2 describe our designs to address the two differences.

5.2.1. Piggybacking Audio Streams in Video Channel.

To support simultaneous audio conversation among multiple participants, as shown by the dashed line in Fig. 3(a), our method is to piggyback the audio stream with the video stream in the colored barcode. As the conference client displays thumbnail videos of different participants (Fig. 4(c)), the audio streams of multiple speakers can be separated accordingly[4]. The encoding and decoding proce-

2. (1280-64)×(720-32)×3/(8×8)/8=4902, (minus the border).

3. Zoom's high fidelity music mode can almost fully utilize the audible range by providing an audio bandwidth of up to 24 kHz.

4. The physical camera can be disabled to switch to the "audio-only" mode. In this case, the virtual camera will generate blank video frames, causing the receiver to automatically display black (blank) thumbnails.

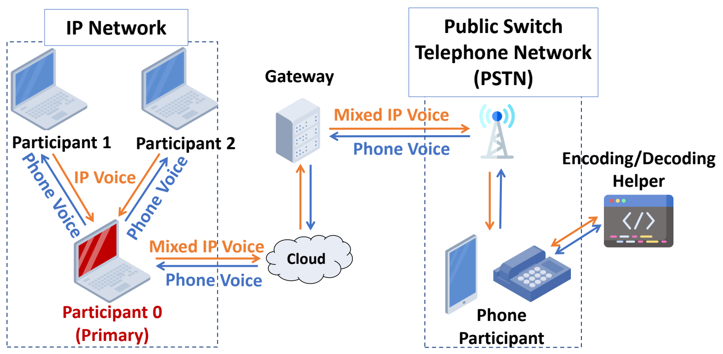


Figure 5: **Securing Audio Streams for Dial-in Participant**. Participants in IP network vote a primary participant for en/decrypting audio data to/from the PSTN participant.

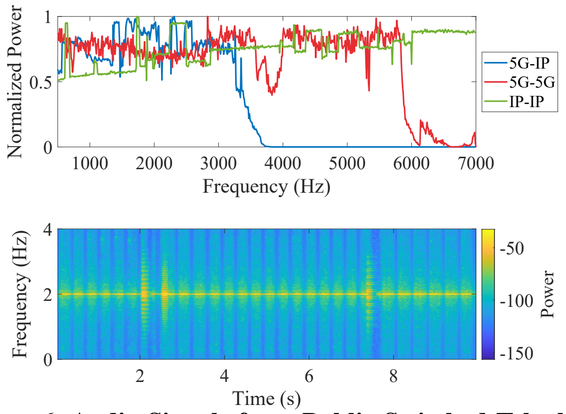
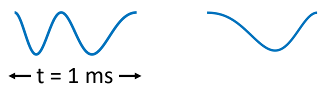


Figure 6: **Audio Signals from Public Switched Telephone Network (PSTN).** The upper subfigure shows the frequency response by sending a tone sweeping from 500 Hz to 7000 Hz; The lower subfigure is the spectrogram when sending a 2000 kHz tone, showing frequency distortions occurs randomly (at 2 s, 2.5 s, and 7.5 s).

$$chirp(t) \coloneqq \cos(2\pi \int_{t_0}^{t} \frac{f_1 - f_0}{t_1 - t_0}\tau + f_0 d\tau)$$



(a) Formula (b) '1' waveform (c) '0' waveform

Figure 7: **Audio Channel Modulation.** Bits are represented in chirp signals to withstand noise.

dures are similar to the video case. The audio stream is compressed at a constant bit rate (CBR) of 1600 bps with Codec2, a modern speech compression algorithm designed for low-bandwidth usage with a bit rate range of 450 to 3200 bps [33], [34]. By default, 168 bits in each barcode frame are reserved for audio content.

Bubbling Up Current Speakers: The prerequisite of the above method is that the conference clients are displaying the thumbnail of the speaker, and this is determined by the layout of the client. The Gallery View (where thumbnails of participants are arranged in a square array) allows for conversation among all participants (*e.g.*, 7×7). However, in scenarios involving content sharing, the client will switch to the Speaker View (Fig. 4(d)) to enlarge the display area of the shared content, such as the pinned main speaker, desktop, slides, *etc.* This layout only displays thumbnails in the sidebar (typically 3 to 4 participants). To enable multi-participant conversations in this layout, we utilize the thumbnail view switching feature.

That is, when a client generates voice in the actual audio channel, other conference clients will automatically switch that client's thumbnail view to the top of the sidebar to enhance interactivity. Based on this feature, when the sender of *mTunnel* detects that its user is speaking, it will intentionally generate a 5000 Hz signaling tone in the actual audio channel to stimulate its user's thumbnail to be shown in the sidebar of all participants. The signaling frequency is chosen to avoid interfering with PSTN audio signals, see §5.2.2. With this scheme, even when the holding space of the thumbnail views is limited by the layout, the video/audio streams of the current speakers can be prioritized, displayed, and played to the audience.

5.2.2. Securing PSTN Dial-in Audio Streams. Current conference applications utilize well-established Voice over Internet Protocol (VoIP) gateways to accommodate PSTN dial-in participants. As shown in Fig. 5, the application cloud gathers audio streams from the IP network, synthesizes them into a single audio track, and then broadcasts it to PSTN telephone participants through the dedicated gateway. The audio streams generated by the PSTN participants undergo the reverse process to reach the IP participants.

In this sense, the audio channel is the only established relation between IP and PSTN participants. To maintain the confidentiality of audio streams and ensure compatibility with the PSTN infrastructure throughout the process, we employ an approach different from the pure IP case: the original audio stream to/from the PSTN participant is compressed and encrypted, and then delivered using the audio channel, rather than being loaded in the video barcode.

Audio Channel Modulation: As shown in Fig. 6 (5G-to-IP), audio signals transmitted between PSTN and IP networks exhibit a narrow bandwidth, while IP-to-IP and 5G-to-5G calls show a wider bandwidth (higher quality). This is likely because VoIP gateways intentionally limit the bandwidth of the audio signals passing through to ensure maximum compatibility, as legacy PSTN uses a cutoff frequency of 3400 Hz [35].

Another issue in Fig. 6 is the notable presence of subtle frequency-domain noise. The audio signal may undergo acceleration or deceleration over a short period, potentially resulting from network delay variations during application cloud or VoIP gateway transcoding. To leverage these signals for information transmission, we employ a specialized modulation scheme: the chirp signal [36]. A chirp signal (Fig.7(a)) dynamically varies its frequency over time, with the key advantage that its autocorrelation energy remains robust in the presence of such frequency noise.

Specifically, as shown in Fig. 7(b)(c), we use a waveform with its frequency increases from f_0=800 Hz to f_1=1200 Hz to represent bit '0' and another waveform decreasing from f_0=2200 Hz to f_1=1800 Hz to indicate bit '1'. When demodulating, the bit is determined according to the power of the two frequency ranges. When a bit is decoded, we apply correlation on the chirp signal to detect the precise location of the bit. This ReSync(hronization) procedure helps to resist sample offsets due to prominent frequency noise.

By default, the chirp signal for 1 bit lasts for 1 ms, corresponding to a raw data rate of 1000 bps. The audio channel is framed every 40 ms. Each frame carries a 28-bit media payload (700 bps), a 4-bit ECC, and 8-bit key material. Additionally, every 25 frames are accompanied by a 10-bit header for synchronization and a 3-bit CRC. The original audio stream is compressed to 700 bps using Codec2, and then loaded into the audio frames.

Secure Voice Relay between IP and PSTN: Unlike ordinary use cases, as the application cloud is no longer able to decrypt and correctly synthesize the encrypted audio streams, we introduce a primary participant to take this role, (Fig. 5). Specifically, before a conference begins, one of the IP participants, *e.g.*, the host, is nominated as the primary participant. It exchanges audio streams between the phone participant and other IP participants:

$\Rightarrow$ IP to PSTN: The primary participant receives, decrypts, and mixes all the audio streams from the IP video channel, it re-encrypts the result with the session key (§7) of the phone participant. Then, it modulates and transmits the encrypted audio into the IP audio channel. The application cloud will forward the audio stream to the phone participant through the PSTN gateway.

$\Leftarrow$ PSTN to IP: When the phone participant transmits the modulated audio over PSTN, the application cloud distributes it to all the IP participants through the IP audio channel. However, only the primary participant is able to decrypt it. Upon finishing decryption, the primary participant re-encrypts and distributes it among IP participants through the IP video channel.

To emulate audio processing on the phone, we currently connect all audio signals from the conference speakerphone and the smartphone out to an external computer, *i.e.*, the En-/Decoding Helper. We believe that this can also be achieved through privileged software or firmware upgrades.

It is worth noting that the above relay method only allows a single PSTN phone participant to join the secure conversation. We will discuss how to scale it to support more PSTN participants in §11. Moreover, due to the limited data rate of the audio channel, the PSTN participant is not able to maintain pairwise encryption with every participant. It has to rely on the E2EE connection to the primary participant to reach other attendees. We will revisit this issue in §7 and §9.

6. Design: I/O Isolator

The purpose of the I/O isolator is twofold: to isolate untrusted applications from accessing unprotected I/O data and to present the channel adapter as a standard and consistent I/O interface to applications. Depending on the desired level of isolation, the I/O isolator can be implemented in various ways, such as through virtual machines or isolated physical hosts. In this section, we propose a lightweight isolation method using virtual drivers.

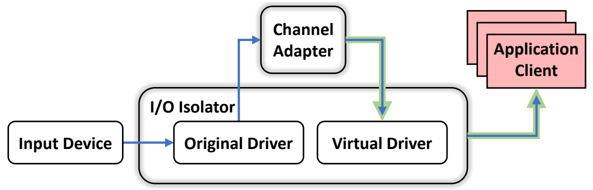


Figure 8: **Data Flow in Sender's I/O Isolator.** The I/O Isolator first obtains the raw data from the original device driver, and then streams it to the channel adapter for encryption. Subsequently, the encrypted content represented in the correct format is returned back to the virtual driver to feed to the applications. As a result, the applications are isolated from the sensitive raw input data.

Its operating concept is shown in Fig. 8. On the sender side, the I/O isolator intercepts the data stream from the original device driver. The intercepted data is then forwarded to the channel adapter for encryption, and subsequently directed to the virtual driver to feed to the applications. The receiver's I/O isolator operates reversely. Throughout the life cycle of the data streams, the application clients and the cloud are only able to access the encrypted copy and hence ensure confidentiality. Considering the adoption of Windows in office scenarios, we elaborate our implementation of various virtual devices on Windows platform[5].

Virtual Camera: The virtual camera is implemented with IMFVirtualCamera API, offered by the Microsoft Media Foundation (MMF) platform [38]. MMF allows us to create a user-space software program that applications can detect and use as if it were an actual camera device [39]. It retrieves barcode frames from the channel adapter, and simulates the video stream generated by a camera by using the MMF media source component to produce a video stream with specific encoding, frame rate, and resolution profiles. The application clients are compelled to use the virtual camera by restricting their access permissions to other camera instances.

Virtual Display Buffer: The content in the display buffer, *i.e.*, framebuffer, will be output to the monitor by the graphics card. We maintain an additional layer of display buffer to allow users to see the decrypted content. The I/O isolator initially hides the original client UI in a background virtual window space and continuously captures its content. The captured content is then sent to the channel adapter for demodulation and decryption. Subsequently, the I/O isolator composites an identical client UI with overlaid decrypted videos onto the corresponding areas. Technically, we utilize ffmpeg gdigrab [40] to capture the display buffer and Tkinter [41] to render the new UI.

5. Linux-based implementations can leverage loopback drivers [37].

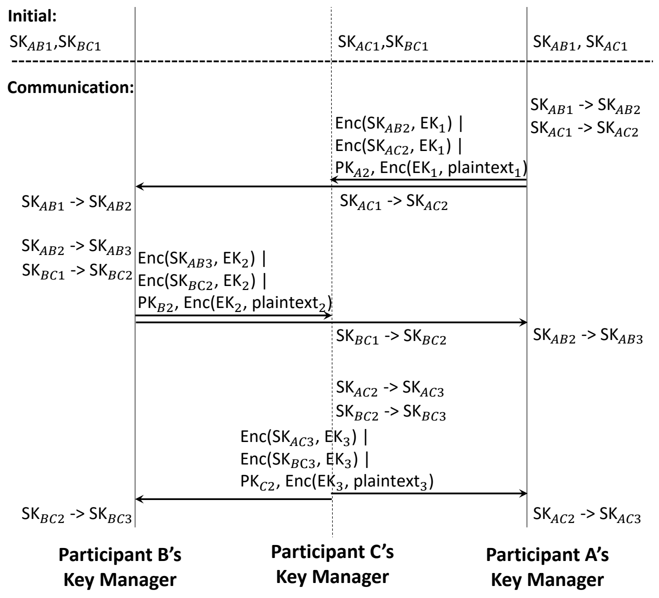

Initial:
SK_AB1,SK_BC1
SK_AC1,SK_BC1
SK_AB1, SK_AC1
Communication:
SK_AB1 -> SK_AB2
SK_AC1 -> SK_AC2
Enc(SK_AB2, EK1) | Enc(SK_AC2, EK1) | PK_A2, Enc(EK1, plaintext1)
SK_AB1 -> SK_AB2
SK_AC1 -> SK_AC2
SK_AB2 -> SK_AB3
SK_BC1 -> SK_BC2
Enc(SK_AB3, EK2) | Enc(SK_BC2, EK2) | PK_B2, Enc(EK2, plaintext2)
SK_BC1 -> SK_BC2
SK_AB2 -> SK_AB3
SK_AC2 -> SK_AC3
SK_BC2 -> SK_BC3
Enc(SK_AC3, EK3) | Enc(SK_BC3, EK3) | PK_C2, Enc(EK3, plaintext3)
SK_BC2 -> SK_BC3
SK_AC2 -> SK_AC3
Participant B's Key Manager
Participant C's Key Manager
Participant A's Key Manager

Figure 9: **Key Manager Operation for Group Conversation.** Key managers update session keys to encrypt the group conversation.

The display buffer also serves as an input device during screen sharing in the sender. To secure the shared screen/application window, the I/O isolator initially obtains the Process ID (PID) of the application that the user intends to share. It then captures the display buffer associated with that PID and sends it to the channel adapter for encryption. Following this, it allocates a new display buffer to contain the generated 2D barcode. Application clients are then forced to access and share the barcode buffer instead of other display buffers.

Virtual Microphone and Speaker: Unlike the virtual camera driver, Windows does not offer a user space API for virtual audio devices. We resort to Kernel Streaming (KS) [42] based on the Microsoft Windows Driver Model (WDM) audio components [43]. Specifically, we adopt Virtual Audio Cable (VAC) [44] as our virtual audio driver. The I/O isolator captures, encrypts, and re-feeds the audio stream to the application client in a process similar to that of a virtual camera.

7. Design: Key Manager

Key manager consists of two main functionalities: session key establishment and key updating. Our implementation is built on the framework of the Signal protocol [45], [16]. In the following, we briefly introduce the Signal's implementation and highlight *mTunnel*'s modifications.

Key Establishment: Before launching the conference, all participants have already shared their initial public ratchet keys and identity keys through a Trust on First Use (TOFU)-like scheme [46] or a trusted channel, *e.g.*, email, SMS messages. The ratchet key is used for computing session keys and is valid only for a particular conference. The identity key is a long-term public key associated with the identity of the participant.

Upon joining in the conference, the participants' key managers start to establish pairwise session keys. Consider participant A for example. It uses its initial private ratchet key SK_{A1}, participant B's initial public ratchet key PK_{B1}, and both parties' identity keys to compute an initial session key SK_{AB1} with ECDH [47] and HKDF [48] algorithms. Similarly, other pairwise session keys can be generated.

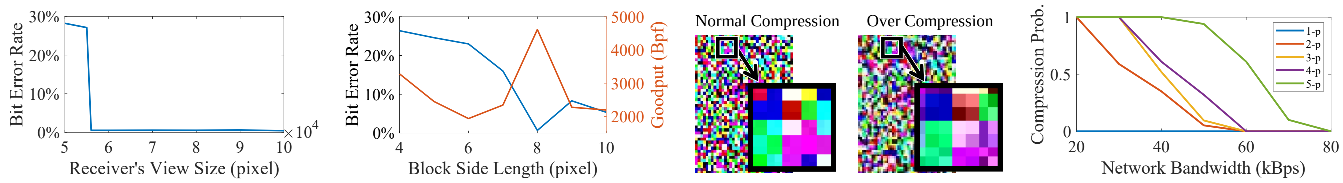
Key Updating for Group Conversation: The pairwise session keys are not suitable for group conversations because each key pair is unique, hence the sender has to duplicate and encrypt the same content for the same number of times as the pairs, which lacks scalability for large video conference. Therefore, the key manager generates a random number as an ephemeral key [49], *i.e.*, EK, to encrypt the media streams, and the channel adapter uses each and every pairwise session key to encrypt the ephemeral key and prepend them to the encrypted media streams. In this way, the media content is only transmitted once and only the participant with the correct session key can obtain EK to decrypt the media content.

As shown in Fig. 9, the Signal protocol uses the Double Ratchet algorithm [50] to update session keys to enhance security. A simplified example is: participant A communicates with participants B and C. It obtains two pairwise session keys (SK_{AB1} and SK_{AC1}) in the previous phase. For the subsequent message, participant A generates an ephemeral key (EK_1) to encrypt its media data with AES-128-GCM. Then, it generates a new ratchet key pair (SK_{A2} and PK_{A2}) and uses SK_{A2} alone with other participants' prior public ratchet keys (PK_{B1} and PK_{C1}) to update the session keys (SK_{AB2} and SK_{AC2}). Subsequently, it uses the new session keys to encrypt EK_1 and append them alone with PK_{A2} to the message. When participant B receives A's message. It retrieves PK_{A2} and combines it with SK_{B1} to calculate the new session key SK_{AB2} for decryption. When participant B needs to update SK_{AB2}, it generates a new ratchet key pair (SK_{B2} and PK_{B2}) and utilizes SK_{B2} and PK_{A2} to compute the next session key SK_{AB3}.

Key Manager of PSTN Participant: If one of the participants is a phone participant, its key manager only performs key establishment and key updating with the primary host through the audio channel. The phone participant relies on the dedicated primary participant to receive and broadcast messages from/to the IP participants.

Key Material Transmission: The key material used for E2EE, *e.g.*, EK, PK, MAC, *etc.*, is carried through the video and audio channels[6]. We update the ratchet keys every 10 seconds, and the content exceeding the capacity of a single frame is fragmented and transmitted with multiple frames. We utilize an additional ECC layer to protect these information against burst errors.

6. Depending on the media compression ratio and the number of ECC bits, the available capacity of the video and audio channel for key material is about 15 kbps and 98 bps, respectively.



(a) BER v.s. View Size (b) BER, Goodput v.s. Block Size (c) Over Compression Situation (d) Over Compression Probability

Figure 10: **Factors Affecting Video Tunneling Capacity**.

8. *mTunnel* Performance

In this section, we conduct experiments to evaluate non-security-related performance of *mTunnel*.

8.1. Evaluation Methodology

All experiments are conducted on PCs with Intel i5-8500HQ and 8 GiB RAM. The operating system is Windows 11. The display resolution of its desktop is 1920×1080. A HIKVISION DS-E12 webcam utilized for the experiments is set to a resolution of 1280×720 pixels. Both the sender and receiver operate in 10 fps video frame rate to ensure reliability. The sampling rate of the audio interface card is 48,000 Hz. We set the microphone to be mono channel for all experiments. The experiments primarily focus on Zoom as a representative remote conference application, but the feasibility of *mTunnel* is also tested on other applications, including Teams and Tencent Meeting.

For Zoom, the Speaker View layout and default settings are used, with the exception of disabling the noise reduction function during voice transmission. The Zoom meeting host is a Zoom One Pro user to allow PSTN participants to join. All other participants are free users. We evaluate group conferences with 10 IP participants. Unless otherwise stated, we involve three IP participants in evaluation tests. Two participants are within the same local area networks (LAN) and the third is connected to the public network. We did not notice special optimization for LAN connections in Zoom.

In the following, we will first evaluate the performance of tunneling two primary multimedia modalities, *i.e.*, video and audio streams, in remote conference applications. We will examine the factors that affect tunneling capacity, as well as the quality degradation and introduced latency caused by the adoption of *mTunnel*. We also will assess the feasibility of supporting group conversations. Then, we will examine the system overhead imposed by *mTunnel* on computing and memory resources. The adopted evaluation metrics are as follows:

• Bit Error Rate (BER) indicates the number of bit errors over the number of received bits.

• Goodput (bps or Bpf) refers to the number of correctly received bits in a unit of time. Actual goodput is determined by not only the BER but also the adopted ECC scheme. More suitable ECC schemes lead to a higher goodput at a certain BER. We use the channel capacity [51] to estimate the theoretical goodput independent of ECC. It is calculated from BER by considering the channel as a binary symmetric channel. We measure the goodput of the audio channel in bits per second (bps), and the video channel in Bytes per frame (Bpf).

• Structural Similarity Index Measure (SSIM) quantifies the similarity between two images. Its value ranges from 0 to 1, where 1 means a perfect similarity between the images, while 0 indicates no similarity between them.

• Latency (ms): Since network delay is less relevant to our designs, we primarily focus on latency induced by the *mTunnel* processing stack, *e.g.*, encoding, encryption, *etc.*

• CPU Usage (%) and Memory footprint (KiB): CPU usage refers to the percentage of CPU power utilized by *mTunnel* at a particular time, while memory footprint represents the amount of occupied memory during its execution.

8.2. Video Tunneling Performance

8.2.1. Impact of Barcode Parameters. When integrating *mTunnel* with conference applications, two primary factors affect the BER and goodput of the video channel:

• **Receiver's View Size**: The capacity of the video channel is determined by the size of the barcode area. The size of the received barcode is determined by the display resolution of the receiver and the client layout. A larger view window allows smaller color blocks to be used to composite the barcode, achieving higher capacity. By default, we use the Speaker View layout in the conference client. Its thumbnail view offers around 61k pixels. In the following, we modify the thumbnail view size by adjusting the client window size to investigate the impact.

In this experiment, the sender adopts the default setting: a 1280×720 pixels encoding area with 8×8 pixel color blocks. As depicted in Fig. 10(a), *mTunnel* demonstrates a commendable performance with low BER across a range of view sizes, spanning from approximately 61k pixels to 100k pixels. However, when the view size becomes excessively small, such as below 55k pixels in Fig. 10(a), the BER increases significantly. This can be attributed to the video compression algorithms of the client. It is worth noting that in normal daily usage, the view size is unlikely to be reduced to such an extent.

• **Sender's Color Block Size**: Given a fixed view size of 61k of the receiver, the capacity of the video channel is determined by the size of the encoding/decoding color block. As illustrated in Fig. 10(b), a larger block size indicates greater resilience against decoding errors, resulting in a lower BER. However, this comes at the expense of lower throughput/goodput. Conversely, a smaller block size increases data encoding density but comes with a higher BER (as it takes more bits as redundancy to correct the error bits). It is worth noting that the block size of 8×8 pixels, *i.e.*, a prominent peak point in Fig. 10(b), achieves an exceptionally low BER and high goodput. This is because this particular block size aligns with the operational unit of compression algorithms, *e.g.*, JPEG-like compression typically divides images into blocks of 8×8 pixels. This configuration leads to almost error-free frames in practice. Occasional error frames are due to network fluctuations (§8.2.2).

8.2.2. Impact of Network Conditions. In practical scenarios, network conditions can be subject to dynamic changes. Existing remote conference applications employ different strategies to handle network degradation. We classify them into two types according to their behaviors:

Frame Drop: Applications such as Tencent Meeting and Ding Talk drop video frames to cope with the decreased network bandwidth.

Frame Compression: Besides frame drop, applications including Zoom and Microsoft Teams further employ high compression ratios (uncompressed size over compressed size) to counter poor network conditions.

mTunnel is compatible with the *Frame Drop* strategy, as it does not involve intra-frame encoding/decoding operations. Dropped frames only lead to dropped frames in the video channel. However, *mTunnel* may potentially be affected by the *Frame Compression* strategy. The reason is shown in Fig. 10(c). Due to the high compression ratio, the color blocks tend to interfere with neighboring blocks and cause BER in decoding. We term this phenomenon as *over compression*. We describe our experiments on different products:

- **Case Study on Zoom**: When the available bandwidth is relatively low, Zoom dynamically adjusts the data rate, attempting to slightly increase it until packet loss is detected. Subsequently, Zoom reduces the data rate once again. This behavior exhibits similarities to TCP congestion control. However, unlike TCP, Zoom does not retransmit lost frames. Consequently, this downward adjustment of the bit rate often results in *over compression*, even if the actual bandwidth could support clearer frame transmission.

To understand the impact of this adaptive strategy, we conduct a test by launching a conference and restricting the download bandwidth with NetLimiter [52] for one participant while others continue video streaming. We observe that low bandwidth might lead to disconnection in Zoom, when the connection is valid, we measure the BER of the received valid frames, their average BER is approximately 21% when *over compression* happens, implying that this situation will corrupt tunneling data.

The rate of the participant is further adjusted to analyze the probability of *over compression*. Fig. 10(d) shows that a bandwidth of 80k Bytes per second (Bps) is sufficient to support a group conferencing of five participants. Therefore, under normal network conditions, the strategy will not cause problems for using *mTunnel*.

- **Case Study on Microsoft Teams**: Unlike Zoom, Teams does not significantly reduce the data rate when packet loss occurs. Consequently, as long as the available bandwidth satisfies the transmission requirements, the frames can be transmitted clearly without substantial degradation. Based on our experiments, Teams only requires a network download bandwidth of approximately 34 kBps per participant to achieve clear thumbnail views for them. When the bandwidth falls below this threshold, Teams initiates *over compression*, resulting in an average bit error rate of approximately 7.5%.

8.2.3. Video Quality Performance. Fig. 11(a) shows the screenshot of the Zoom client on the receiver's side before recovery. Fig. 11(b) shows the re-composited client UI with recovered video content[7]. The text content in screen sharing can be clearly recognized even when the font size is 16 pt. Fig. 11(c) compares the received thumbnail frames with and without *mTunnel*. The quality of the video is slightly degraded due to compression.

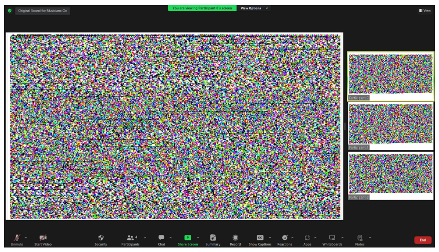
To quantitatively compare the quality loss of *mTunnel*, we utilize SSIM scores to quantify the similarity between the original frame at the sender side and the received frame at the receiver side. As depicted in Fig. 12(a) after bare-metal Zoom transmission, the received frame exhibits an SSIM score of 0.96. For *mTunnel*, we examine the SSIM scores across different compression ratios of JPEG. The default thumbnail frame (61k pixels) consumes 183 kB (in 24-bit RGB color space) without compression. In order to accommodate this frame within one barcode (4.9 kB capacity) of *mTunnel*, a compression ratio of 38:1 is required. As shown in Fig. 12(a), when choosing a compression ratio of 38:1, the bare-metal and tunneling thumbnail views are very close in their SSIM scores. This implies that *mTunnel* can preserve the video quality. The re-composited client UIs of Tencent Meeting and Microsoft Teams are shown in Appendix.B. They differ slightly in UI layouts, but present similar quality performance when using the same encoding configuration.

8.2.4. Introduced Video Latency. *mTunnel* introduces additional latency to video streams due to processing. As illustrated in Fig. 12(b), the modulation procedure is the major factor. Additionally, the total delay is related to the payload size, *i.e.*, the compressed video frame size, which directly affects the video quality. To deliver high-quality video streams, we utilize the full capacity of 4.9k Bytes per code frame as the default setting, corresponding to a total latency of approximately 13.5 ms. The piggybacked audio content is subject to more latency. It will be analyzed later in §8.3.4.

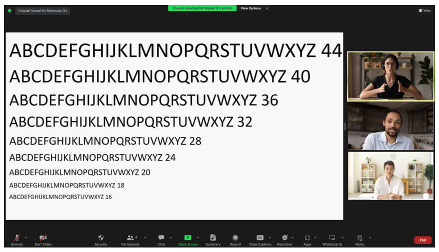
8.3. Audio Tunneling Performance

Voice tunneling in *mTunnel* comprises two parts: transmission through the lossy video channel and transmission

7. We feed licensed photos from public database [53] into I/O isolators as three participants' camera streams for illustration.

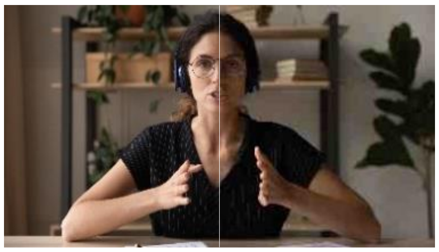


(a) Screenshot of Client UI (Zoom)



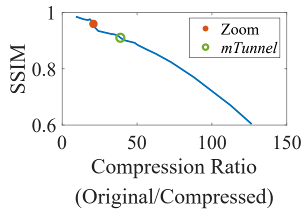
ABCDEFGHIJKLMNOPQRSTUVWXYZ 44
ABCDEFGHIJKLMNOPQRSTUVWXYZ 40
ABCDEFGHIJKLMNOPQRSTUVWXYZ 36
ABCDEFGHIJKLMNOPQRSTUVWXYZ 32
ABCDEFGHIJKLMNOPQRSTUVWXYZ 28
ABCDEFGHIJKLMNOPQRSTUVWXYZ 24
ABCDEFGHIJKLMNOPQRSTUVWXYZ 20
ABCDEFGHIJKLMNOPQRSTUVWXYZ 18
ABCDEFGHIJKLMNOPQRSTUVWXYZ 16

(b) Screenshot of Recomposited Client UI (Zoom)

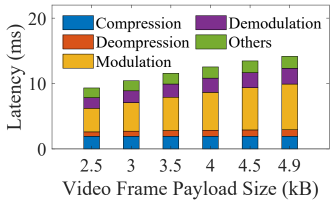


(c) Thumbnail View w/o (L) with (R) Tunneling

Figure 11: **Video Tunneling Results**.

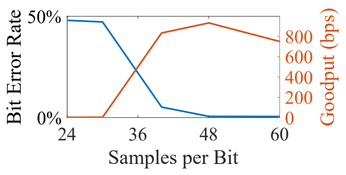


(a) SSIM v.s. Compression Ratio

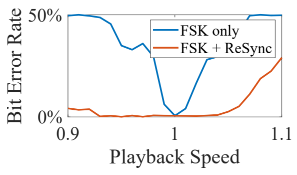


(b) Video Tunneling Latency

Figure 12: **Video Tunneling Quality and Latency**.



(a) BER, Goodput of Audio Tunnel



(b) Performance of ReSync

Figure 13: **Factors Affecting Audio Tunneling Capacity**.

through the lossy voice channel. Since the behaviors of the former are largely similar to video frame tunneling, this section specifically focuses on the latter, the transmission over the voice channel of PSTN networks.

As illustrated in Fig. 5, we developed a proof-of-concept prototype of the encoding/decoding helper in a PC to process the audio signals to/from the speakerphone connected to PSTN. The primary participant hosts a conference, and the PSTN participant joins the conference by dialing in.

8.3.1. Impact of Audio Modulation. The duration of the chirp waveform used to represent bits is the major factor affecting the tunnel capacity. A longer waveform for one bit leads to a lower BER but also a lower modulation density. Since we use a fixed 48 kHz sampling rate, we measure the length of the waveform in terms of the number of samples representing one bit.

In our evaluation, we keep the audio frame length and header length fixed and only vary the non-header part. Using more bits or time to represent 1 bit means fewer errors and a smaller bit rate. As shown in Fig. 13(a), using 48 samples for 1 bit achieves the maximum goodput. This configuration leads to a goodput of approximately 934 bps, which can meet the requirement of relatively reliably streaming data in this channel (§5.2.2).

8.3.2. Impact of Network Conditions. We note that the network bandwidth required for transmitting audio is small. However, in cases of fluctuating network conditions, some conference applications may occasionally experience acceleration or deceleration playback. We analyze the effectiveness of our method in mitigating such frequency distortions.

To emulate this situation, we manually control the playback speed of the incoming audio from the PSTN network to observe the robustness of the demodulation algorithms. As for comparison, we also evaluate the performance of pure FSK without the help of ReSync. The results, shown in Fig. 13(b), indicate that the decoding process is barely affected when the speedup/down ratio is below 5%. The ReSync algorithm performs better in the case of slowdowns because the slowdown does not result in the loss of information. The errors observed in the ReSync algorithm mainly stem from the failure of header locating, rather than distortions introduced by the playback speed variation. Overall, the ReSync algorithm improves the goodput of the audio transmission over pure FSK, as it helps to recover from frequency errors.

8.3.3. Audio Quality Performance. We utilize the Mean Opinion Score (MOS) [54] to evaluate the subjective audio quality. The MOS value ranges from 1 to 5, with higher values indicating better audio quality. The detailed scoring protocol and results are presented in Appendix.C. The MOS of the audio over the PSTN tunnel is 2.66, suggesting that the transmitted voice quality is relatively fair. Additionally, we gauge the Listening-Effort Opinion Score (MOS$_{LE}$) to measure the cognitive effort needed to comprehend the sentences' meanings. A MOS$_{LE}$ value of 3.01 indicates a moderate level of effort required.

To further assess whether the conveyed voice information is sufficient, we sample 100 speeches from the Common Voice Dataset [55], compress and transmit them over the PSTN audio tunnel. The received voice is transcribed using the OpenAI speech-to-text API [56]. Results show that the word error rate (WER) is approximately 20.3%. In contrast, the WER of the original audio is 9.0%. Although this indicates that the audio tunneling, especially its compression procedure, introduces a certain level of error in terms of the voice meaning, it is worth noting that (when we validate the word errors) humans can still understand the overall message with the help of contextual cues. As a rapidly advancing field, with further progress in speech understanding technology, we can anticipate lower error rates.

Type	Latency (ms)
Video Frame Buffering	100
Video Channel Processing	13.5
Audio Frame Buffering	90
Audio Channel Processing	31
Network Transmission	84
End to End (IP-IP)[8]	197.5
End to End (PSTN-IP)[9]	377.5

TABLE 2: **Audio Tunneling Latency.**

8.3.4. Introduced Audio Latency. Audio streams are tunneled in two ways: IP participants piggyback it in the video channel; PSTN participants transmit it through the audio channel. Both types of channels, besides network transmission, introduce buffering and processing-induced latency. We measure and list them in Table 2. The buffering latency is due to the fact that both types of channels have to send data periodically (with intervals of 100 ms and 40 ms, respectively). From an end-to-end perspective, different users might experience different latencies. The audio latency among IP participants is mainly determined by the frame interval of the video channel and the network latency; the latency between PSTN and IP participants contains extra latency from the PSTN audio channel hop.

According to the previous study [57], the perceptible impact of latency becomes noticeable at around 300 ms, so the introduced latency in IP participants remains relatively low and falls within the range. When the network conditions are good, the latency of PSTN participants also falls within this range. Further, our implementation has not undergone much optimization, so there is room for reducing the processing latency as well.

8.4. Group Conversation Performance

To support multiple participants in one conference and in the Speaker View layout, we adopt the bubbling up scheme. We evaluate its feasibility in this subsection.

This test involves ten IP participants, with participant 0 selected as the receiver. We slightly resize the receiver's view window to accommodate four thumbnail views in its sidebar, corresponding to viewing three other participants, with the top thumbnail view representing the receiver itself. Among the remaining nine participants, speaking slots are randomly assigned to each of them. Each speaking slot lasts for 2 seconds and is triggered at the second boundary of the system time on their host PCs. These participants keep sending ID messages in the video channel and playing the signaling sound in the speaking slot.

When a participant "speaks", its thumbnail will be switched to the top of the sidebar of the receiver. The receiver logs thumbnail views and the received ID messages. The results are shown in Fig. 14. It can be observed that almost all messages are successfully received except for the one at the second 18. The results imply that the bubbling up scheme works well and can support a large number of participants to talk in the remote conference.

8. The receiver does not require video buffering, leading to 100 (Video Send Buffering) + 13.5 (Video Processing) + 84 (Network) = 197.5 ms.

9. 40 ms is the minimum buffer unit of Codec2. We use 5×10 ms buffers to capture a 40 ms frame, leading to 90 ms latency in total. However, when the primary participant relays audio frames to video frames, there is no need to wait for the audio buffering. Hence, the latency is slightly reduced to around 65 ms: 65 (Average Audio Buffering) + 31 (Audio Processing) + 84 (Network) + 197.5 (IP-IP) = 377.5 ms.

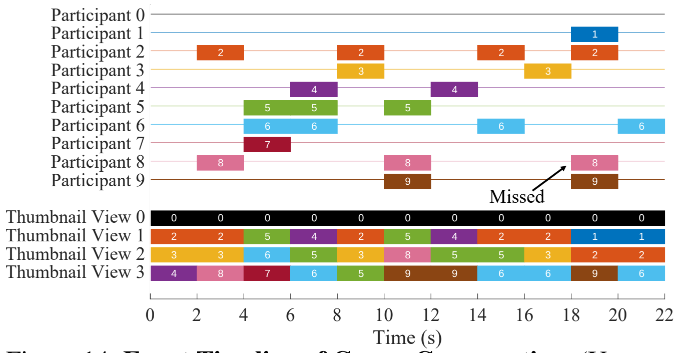


Figure 14: **Event Timeline of Group Conversation**. (Upper half) Each colored timeline indicates the behavior of the corresponding participant. The bar periods indicate speaking events, while the line periods indicate silence. (Lower half) The bars signify the participants whose thumbnail views are observed by the receiver.

The message from participant 8 is lost due to exceeding the capacity of the sidebar. However, it is worth mentioning that conference applications also only support a limited number of participants to speak simultaneously. For example, the number of Zoom is three [58]. When the fourth participant begins to talk, it will be automatically muted. So, the capability of *mTunnel* is consistent with these commercial solutions in terms of simultaneous speakers.

Additionally, we observe that it takes an average of 312 ms for Zoom to switch the thumbnail view after the signaling sound is sent out. This may result in the loss of some heading audio signals, but in practical use, we found that it is not a significant issue, especially if the participant speaks continuously.

Path	CPU Usage (%)	Memory Footprint (KiB)
Video Sender	2.98	48,928
Video Receiver	0.73	22,722
Audio Sender	0.36	36.3
Audio Receiver	0.45	68.5

TABLE 3: **Overhead of *mTunnel* Software Layer**

8.5. System Overhead

We assess the system overhead of *mTunnel* from two perspectives: CPU usage and memory footprint. Specifically, we employ Process Explorer [59] to monitor the *mTunnel* process. Table 3 shows the overhead when two participants are involved, both utilizing the thumbnail view to communicate. Each "Path" includes the I/O isolator and the channel adapter corresponding to the media stream. The values demonstrate that the overhead of *mTunnel* is generally minimal. Typically, conference clients can support more participants, where the sending path overhead remains constant, while the receiving path overhead increases proportionally with the number of participants displayed. It is worth noting that even in the densest gallery view, the number of displayed participants is limited, *e.g.*, within 7×7. Participants exceeding the display capacity will not be shown by the client and hence do not introduce *mTunnel* overhead.

9. Security Analysis

mTunnel takes full control of the encryption process and isolates the original media streams from the conference clients. As long as the host system is not compromised, the conversation is expected to be secure. In the following, we discuss the security properties of *mTunnel* and several non-trivial attacks against the protection.

Forward Secrecy and Post-compromise Security: *mTunnel* employs the Double Ratchet algorithm [50] from Signal to update the session keys. The algorithm uses the Diffie-Hellman protocol to periodically generate new session key pairs. This can ensure forward secrecy, *i.e.*, the adversary cannot compromise the past conversations even if the participant's long-term private keys are compromised. The algorithm also allows *mTunnel* to secure the conversation even if its secret has been compromised (Post-compromise Security) [60]. However, exploiting these compromised keys allows adversaries to launch stronger attacks in conjunction with other types of threats [45], *e.g.*, impersonation.

A related issue is that if a participant is unable to transmit messages, *e.g.*, due to falsely disabled video streaming, adversary's in-path blockage, *etc.*, then other participants could no longer update their symmetric ratchet key pairs, which leads to a downgradation of the security level.

Message Consistency: Message consistency assures that all the participants receive the same copy of the message. *mTunnel* enforces message consistency through pairwise E2EE. However, due to the bandwidth limitation, the phone participant has to rely on its primary participant to reach IP participants, thereby the primary participant can tamper with the messages sent by/to the phone participant. To ensure the consistency, the phone participant should choose a trusted participant as its primary host or alternatively set up its own permanent conference client in its home network as the primary participant.

Group Member Changes: When a new participant enters the meeting, ze should not be able to decrypt previous messages. *mTunnel* uses pairwise keys, so the new participant has to establish new session keys with other participants and cannot access previous sessions. Similarly, when a participant leaves the meeting, ze should not be able to decrypt later messages. *mTunnel* will stop enclosing the ephemeral key encrypted by the session key of the offline participant. As current conference clients do not actively notify which participants have gone offline, future *mTunnel* implementation needs to actively watchdog this information.

Replay Attack: The adversary can record the network traffic of a past conversation, and resend some of the messages to fool the receiver. *mTunnel* can withstand replay attacks, as it utilizes the Double Ratchet to frequently update the session keys. A message with an expired session key will not be accepted. *mTunnel* further includes timestamps in video frames to ensure that the messages are only valid for a certain period of time.

Backdoor Attack: The adversary is a malicious insider who can insert backdoor code into the conference clients. This backdoor can fetch private data through the client and send it to the remote adversary. *mTunnel* isolates the client process from the raw media data and only allows access to the ciphertext.

Man-in-the-middle Attack: The adversary is either a malicious insider or an adversary who compromises the application cloud. It has an on-path capability to eavesdrop, modify, and inject traffic to/from conference participants. The adversary might initiate a connection with two participants. Then, it can decrypt the messages from both sides and relay them to each other. *mTunnel* can prevent such attacks as long as the participants can properly verify public keys at the initial stage.

Zoom Bombing Attack: The adversary can join the meeting without an invitation to eavesdrop on the conversation or deliver harmful messages. Usually, Zoom Bombing is due to improper settings or on-purpose leakage of the password of the conference application. *mTunnel* can secure the conversation content if the adversary's key and identity are not within the authorized set. However, *mTunnel* cannot forbid the adversary to derive video and audio to the normal channels of the conference clients. Note that while *mTunnel* can block these contents, they can still interfere with *mTunnel* messages, *e.g.*, the bubbling up signals in the audio channel, and lead to DoS. To completely mitigate zoom bombing, the participants should utilize the features of conference clients, *e.g.*, the waiting room.

10. Related Work

Several works propose schemes to transmit data over the GSM voice channel based on the variant of FSK [61], [62]. Some studies employ barcodes and QR codes to realize data transmission over the visual channel in the air interface [63], [64]. These efforts provide a general basis for our communication design, but they are not dedicated to addressing specific security issues. *mTunnel*, in particular, focuses on the security concerns within remote conferencing applications or more generally, media streaming applications. Due to the sensitivity of media data, there have been many relevant efforts in this domain.

Securing Audio Streams: Analog speech encryption is one way to secure voice signals. It relies on time-domain or frequency-domain scramblers [65], [66], and pseudo-noise sequences [67]. The encryption has a small overhead at the cost of low-level security. Hou *et al.*apply the Hermes Algorithm to transmit certificates over the VoIP audio channel to verify the identity of the caller [68]. Similarly, Reaves *et al.*execute a TLS-inspired authentication protocol over the telephone network [69]. However, their protocol could only transmit data in a low bit rate instead of supporting real-time media communication like *mTunnel*. Castiglione *et al.*add an encryption layer and a security control layer in 3G-324M architecture to secure end-to-end communication over the CSD of 3G networks [70]. The two layers encrypt the digital signal before modulation. Nevertheless, their design only supports 3G videotelephony and requires modification of the 3G-324M protocol. In [71], Peeters *et al.*design a distance bounding-inspired protocol to protect the global telephone network from redirection attacks by modeling round trip time, but it fails to resist other attacks. The problems addressed by these works are orthogonal to ours.

Securing Video Streams: As far as we know, there is no existing work on E2EE over video channels. Most discussions on protecting video streams are based on conventional network channels. Some work contributes to related aspects. Encryption-then-compression algorithms help the transmission over untrusted channels. Some works [72], [15] try to encrypt images by dividing them into blocks and performing block scrambling-based encryption. The block-based encryption does not require further modulation and is robust to JPEG compression. However, their security is not guaranteed. The adversary can still infer information from the blocks. Other Encryption-then-compression algorithms [73], [74] like compressed sensing require cooperation between encryption and compression, which is computationally intensive and also lacks security proof.

Securing Lossless Streams: A number of works implement software [75] or browser extensions [76] to automatically substitute encrypted text for user input. Other literature proposes new approaches to protect user data stored in untrusted cloud [77], [78]. Our approach to protecting lossless channels is similar to theirs.

11. Limitations

As we describe in §5.2.2, currently, *mTunnel* only supports one phone participant. This is because the available bandwidth of the PSTN audio channel is very limited. A method to increase the number of PSTN participants is to use time-division multiplexing (TDM), a scheme similar to walkie-talkies. Data from PSTN to IP is transmitted in a time-division manner, and data from IP to PSTN is shared using the same key. Another more secure method is to associate each PSTN participant with an independent PSTN connection, *i.e.*, create a new conference for each PSTN participant, and then interconnect them to IP participants through relaying data between the conference instances.

12. Conclusion

In this paper, we emphasize the potential privacy risks in remote conference applications and propose *mTunnel* that utilizes multiple modalities of information channels, such as video and audio streams, as encrypted channels for data transmission. *mTunnel* is designed in a way that does not require any modifications to the user's host system and is compatible with almost all commercial conference products. Furthermore, the design of *mTunnel* can be applied to secure other media streaming scenarios, such as chat apps, video surveillance cameras, and edge IoT sensors.

Acknowledgments

We thank anonymous reviewers for their valuable comments. We thank anonymous shepherd. This work is supported by ShanghaiTech.

References

[1] M. Sako, "From remote work to working from anywhere," *Communications of the ACM*, vol. 64, no. 4, pp. 20–22, 2021.

[2] "Webex meeting customers," https://www.webex.com/customers.html., 2023.

[3] "Zoom guide for yale students," https://academiccontinuity.yale.edu/students/how-guides/zoom-guide-yale-students, 2023.

[4] C. Ling, U. Balcı, J. Blackburn, and G. Stringhini, "A first look at zoombombing," in *2021 IEEE Symposium on Security and Privacy (SP)*. IEEE, 2021, pp. 1452–1467.

[5] T. Reisinger, I. Wagner, and E. A. Boiten, "Security and privacy in unified communication," *ACM Computing Surveys (CSUR)*, vol. 55, no. 3, pp. 1–36, 2022.

[6] M. Zha, "Hazard integrated: Understanding security risks in app extensions to team chat systems," in *2015 IEEE 35th International Conference on Distributed Computing Systems*. IEEE, 2015, pp. 537–546.

[7] "Zoom's privacy and security," https://foundation.mozilla.org/en/privacynotincluded/zoom/, 2022.

[8] "Ex-Amazon employee convicted of hacking Capital One and stealing data of over 100 million people," https://www.insider.com/ex-amazon-worker-convicted-of-hacking-capital-one-and-stealing-data-2022-6, 2022.

[9] "The Road to End-to-End Encryption," https://www.rsaconference.com/library/Presentation/USA/2021/the-road-to-endtoend-encryption, 2022.

[10] "TeamViewer Security Statement ," https://static.teamviewer.com/resources/2017/07/TeamViewer-Security-Statement-en.pdf, 2022.

[11] N. Unger, S. Dechand, J. Bonneau, S. Fahl, H. Perl, I. Goldberg, and M. Smith, "Sok: Secure messaging," in *2015 IEEE Symposium on Security and Privacy*, 2015, pp. 232–249.

[12] T. K. Yadav, D. Gosain, A. Herzberg, D. Zappala, and K. Seamons, "Automatic detection of fake key attacks in secure messaging," in *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security*, 2022, pp. 3019–3032.

[13] R. Hoffman, *Data compression in digital systems*. Springer Science & Business Media, 2012.

[14] X. Chai, X. Fu, Z. Gan, Y. Zhang, Y. Lu, and Y. Chen, "An efficient chaos-based image compression and encryption scheme using block compressive sensing and elementary cellular automata," *Neural Computing and Applications*, vol. 32, pp. 4961–4988, 2020.

[15] T. Chuman, W. Sirichotedumrong, and H. Kiya, "Encryption-then-compression systems using grayscale-based image encryption for jpeg images," *IEEE Transactions on Information Forensics and Security*, vol. 14, no. 6, pp. 1515–1525, 2019.

[16] K. Cohn-Gordon, C. Cremers, B. Dowling, L. Garratt, and D. Stebila, "A formal security analysis of the signal messaging protocol," *Journal of Cryptology*, vol. 33, no. 4, pp. 1914–1983, 2020.

[17] "Ding talk," https://www.dingtalk.com/en.

[18] "Tencent meeting," https://voovmeeting.com/.

[19] “Bluejeans video conferencing,” https://www.bluejeans.com/.

[20] “Zoom,” https://zoom.us/.

[21] “Microsoft teams,” https://www.microsoft.com/en-us/microsoft-teams/group-chat-software.

[22] “Webex meeting,” https://www.webex.com/video-conferencing.

[23] “Jitsi meet,” https://meet.jit.si/.

[24] “Element,” https://element.io/.

[25] “Zoom lied about ’end-to-end encryption’ and gave user data to facebook and google without consent — $85 million settlement,” https://www.techtimes.com/articles/263721/20210804/zoom-lied-about-end-to-end-encryption-and-gave-user-data-to-facebook-and-google-without-consent-85-million-settlement.htm, 2021.

[26] “It’s not just zoom. google meet, microsoft teams, and webex have privacy issues, too.” https://www.consumerreports.org/video-conferencing-services/videoconferencing-privacy-issues-google-microsoft-webex/, 2020.

[27] R. Abu-Salma, E. M. Redmiles, B. Ur, and M. Wei, “Exploring user mental models of End-to-End encrypted communication tools,” in *8th USENIX Workshop on Free and Open Communications on the Internet (FOCI 18)*. Baltimore, MD: USENIX Association, Aug. 2018. [Online]. Available: https://www.usenix.org/conference/foci18/presentation/abu-salma

[28] A. C. Myers and B. Liskov, “A decentralized model for information flow control,” *SIGOPS Oper. Syst. Rev.*, vol. 31, no. 5, p. 129–142, oct 1997. [Online]. Available: https://doi.org/10.1145/269005.266669

[29] Z. C. Schreuders, T. McGill, and C. Payne, “The state of the art of application restrictions and sandboxes: A survey of application-oriented access controls and their shortfalls,” *Computers and Security*, vol. 32, pp. 219–241, 2013. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0167404812001435

[30] “Your video could be awesome,” https://blog.zoom.us/your-video-could-be-awesome/.

[31] R. E. Ziemer, “Channel distortion effects on direct sequence spread spectrum signal demodulation,” in *GLOBECOM ’86 - Global Telecommunications Conference*, vol. 1, Jan. 1986, pp. 171–175.

[32] I. S. Reed and G. Solomon, “Polynomial codes over certain finite fields,” *Journal of the Society for Industrial and Applied Mathematics*, vol. 8, no. 2, pp. 300–304, 1960. [Online]. Available: https://doi.org/10.1137/0108018

[33] “Codec 2,” https://www.rowetel.com/wordpress/?page_id=452, 2017.

[34] “Codec 2 at 450 bit/s,” https://www.rowetel.com/?p=6212, 2018.

[35] L. Kozma-Spytek, P. Tucker, and C. Vogler, “Voice telephony for individuals with hearing loss: The effects of audio bandwidth, bit rate and packet loss,” in *Proceedings of the 21st International ACM SIGACCESS Conference on Computers and Accessibility*, 2019, pp. 3–15.

[36] A. Berni and W. Gregg, “On the utility of chirp modulation for digital signaling,” *IEEE Transactions on Communications*, vol. 21, no. 6, pp. 748–751, 1973.

[37] “loopback,” https://manpages.ubuntu.com/manpages/bionic/man7/oss_audioloop.7.html.

[38] “Microsoft media foundation,” https://learn.microsoft.com/en-us/windows/win32/medfound/microsoft-media-foundation-sdk, 2021.

[39] “Imfvirtualcamera interface,” https://learn.microsoft.com/en-us/windows/win32/api/mfvirtualcamera/nn-mfvirtualcamera-imfvirtualcamera/, 2023.

[40] “ffmpeg documentation - gdigrab,” https://ffmpeg.org/ffmpeg-all.html#gdigrab, 2024.

[41] “tkinter — python interface to tcl/tk,” https://docs.python.org/3/library/tkinter.html, 2024.

[42] “Kernel streaming,” https://learn.microsoft.com/en-us/windows-hardware/drivers/stream/kernel-streaming, 2021.

[43] “Wdm audio drivers overview,” https://learn.microsoft.com/en-us/windows-hardware/drivers/audio/getting-started-with-wdm-audio-drivers, 2022.

[44] “Virtual audio cable (vac),” https://vac.muzychenko.net/en/.

[45] “Signal protocol specifications,” https://signal.org/docs/.

[46] “Signal safety number,” https://signal.org/blog/safety-number-updates/.

[47] A. Langley, M. Hamburg, and S. Turner, “Elliptic Curves for Security,” RFC 7748, Jan. 2016. [Online]. Available: https://www.rfc-editor.org/info/rfc7748

[48] D. H. Krawczyk and P. Eronen, “HMAC-based Extract-and-Expand Key Derivation Function (HKDF),” RFC 5869, May 2010. [Online]. Available: https://www.rfc-editor.org/info/rfc5869

[49] “Private group messaging,” https://signal.org/blog/private-groups/, 2014.

[50] “The double ratchet algorithm,” https://signal.org/docs/specifications/doubleratchet/, 2016.

[51] “Channel capacity,” https://cs.stanford.edu/people/eroberts/courses/soco/projects/1999-00/information-theory/channel_capacity_7.html.

[52] “NetLimiter,” https://www.netlimiter.com/, 2022.

[53] “Dreamstime,” https://www.dreamstime.com/.

[54] “Methods for subjective determination of transmission quality,” https://www.itu.int/rec/T-REC-P.800-199608-I/en.

[55] R. Ardila, M. Branson, K. Davis, M. Henretty, M. Kohler, J. Meyer, R. Morais, L. Saunders, F. M. Tyers, and G. Weber, “Common voice: A massively-multilingual speech corpus,” in *Proceedings of the 12th Conference on Language Resources and Evaluation (LREC 2020)*, 2020, pp. 4211–4215.

[56] “Openai speech to text,” https://platform.openai.com/docs/guides/speech-to-text.

[57] M. Perez, S. Xu, S. Chauhan, A. Tanaka, K. Simpson, H. Abdul-Muhsin, and R. Smith, “Impact of delay on telesurgical performance: study on the robotic simulator dv-trainer,” *International journal of computer assisted radiology and surgery*, vol. 11, no. 4, p. 581—587, April 2016. [Online]. Available: https://doi.org/10.1007/s11548-015-1306-y

[58] “Multiple meeting participants - can they speak at the same time?” https://community.zoom.com/t5/Meetings/multiple-meeting-participants-can-they-speak-at-the-same-time/m-p/116890#M67060, 2023.

[59] “Process explorer,” https://learn.microsoft.com/en-us/sysinternals/downloads/process-explorer, 2023.

[60] K. Cohn-Gordon, C. Cremers, and L. Garratt, “On post-compromise security,” in *2016 IEEE 29th Computer Security Foundations Symposium (CSF)*, 2016, pp. 164–178.

[61] A. Dhananjay, A. Sharma, M. Paik, J. Chen, T. K. Kuppusamy, J. Li, and L. Subramanian, “Hermes: Data transmission over unknown voice channels,” in *Proceedings of the Sixteenth Annual International Conference on Mobile Computing and Networking*, ser. MobiCom ’10. New York, NY, USA: Association for Computing Machinery, 2010, p. 113–124. [Online]. Available: https://doi.org/10.1145/1859995.1860010

[62] M. Perić, P. Milićević, Z. Banjac, and B. M. Todorović, “An experiment with real-time data transmission over global scale mobile voice channel,” in *2015 12th International Conference on Telecommunication in Modern Satellite, Cable and Broadcasting Services (TELSIKS)*, 2015, pp. 239–242.

[63] T. Hao, R. Zhou, and G. Xing, "Cobra: Color barcode streaming for smartphone systems," in *Proceedings of the 10th International Conference on Mobile Systems, Applications, and Services*, ser. MobiSys '12. New York, NY, USA: Association for Computing Machinery, 2012, p. 85–98. [Online]. Available: https://doi.org/10.1145/2307636.2307645

[64] X. Liu, D. Doermann, and H. Li, "Vcode—pervasive data transfer using video barcode," *IEEE Transactions on Multimedia*, vol. 10, no. 3, pp. 361–371, 2008.

[65] N. Jayant, B. McDermott, S. Christensen, and A. Quinn, "A comparison of four methods for analog speech privacy," *IEEE Transactions on Communications*, vol. 29, no. 1, pp. 18–23, 1981.

[66] Y. FulongMa, JunCheng, "Wavelet transform-based analogue speech scrambling scheme," *Electronics Letters*, vol. 32, pp. 719–721(2), April 1996. [Online]. Available: https://digital-library.theiet.org/content/journals/10.1049/el_19960471

[67] V. Kumar, A. Mitra, and S. Prasanna, "On the effectivity of different pseudo-noise and orthogonal sequences for speech encryption from correlation properties," 09 2007.

[68] D. Hou, H. Han, and E. Novak, "Taes: Two-factor authentication with end-to-end security against voip phishing," in *2020 IEEE/ACM Symposium on Edge Computing (SEC)*, 2020, pp. 340–345.

[69] B. Reaves, L. Blue, and P. Traynor, "AuthLoop: End-to-End cryptographic authentication for telephony over voice channels," in *25th USENIX Security Symposium (USENIX Security 16)*. Austin, TX: USENIX Association, Aug. 2016, pp. 963–978. [Online]. Available: https://www.usenix.org/conference/usenixsecurity16/technical-sessions/presentation/reaves

[70] A. Castiglione, G. Cattaneo, G. D. Maio, and F. Petagna, "Secr3t: Secure end-to-end communication over 3g telecommunication networks," in *2011 Fifth International Conference on Innovative Mobile and Internet Services in Ubiquitous Computing*, 2011, pp. 520–526.

[71] C. Peeters, H. Abdullah, N. Scaife, J. Bowers, P. Traynor, B. Reaves, and K. Butler, "Sonar: Detecting ss7 redirection attacks with audio-based distance bounding," in *2018 IEEE Symposium on Security and Privacy (SP)*, 2018, pp. 567–582.

[72] K. Kurihara, S. Shiota, and H. Kiya, "An encryption-then-compression system for jpeg standard," in *2015 Picture Coding Symposium (PCS)*, 2015, pp. 119–123.

[73] J. Zhou, X. Liu, and O. C. Au, "On the design of an efficient encryption-then-compression system," in *2013 IEEE International Conference on Acoustics, Speech and Signal Processing*, 2013, pp. 2872–2876.

[74] X. Chai, Z. Gan, Y. Chen, and Y. Zhang, "A visually secure image encryption scheme based on compressive sensing," *Signal Processing*, vol. 134, pp. 35–51, 2017. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0165168416303309

[75] A. Ozcan, C. Gemicioglu, K. Onarlioglu, M. Weissbacher, C. Mulliner, W. Robertson, and E. Kirda, "Babelcrypt: The universal encryption layer for mobile messaging applications," 01 2015, pp. 355–369.

[76] W. He, D. Akhawe, S. Jain, E. Shi, and D. Song, "Shadowcrypt: Encrypted web applications for everyone," in *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security*, ser. CCS '14. New York, NY, USA: Association for Computing Machinery, 2014, p. 1028–1039. [Online]. Available: https://doi.org/10.1145/2660267.2660326

[77] B. Lau, S. Chung, C. Song, Y. Jang, W. Lee, and A. Boldyreva, "Mimesis aegis: A mimicry privacy Shield–A System's approach to data privacy on public cloud," in *23rd USENIX Security Symposium (USENIX Security 14)*. San Diego, CA: USENIX Association, Aug. 2014, pp. 33–48. [Online]. Available: https://www.usenix.org/conference/usenixsecurity14/technical-sessions/presentation/lau

[78] H. Xiong, X. Zhang, D. Yao, X. Wu, and Y. Wen, "Towards end-to-end secure content storage and delivery with public cloud," in *Proceedings of the Second ACM Conference on Data and Application Security and Privacy*, ser. CODASPY '12. New York, NY, USA: Association for Computing Machinery, 2012, p. 257–266. [Online]. Available: https://doi.org/10.1145/2133601.2133633

[79] S. Josefsson, "The Base16, Base32, and Base64 Data Encodings," RFC 4648, Oct. 2006. [Online]. Available: https://www.rfc-editor.org/info/rfc4648

[80] "Hooks overview," https://learn.microsoft.com/en-us/windows/win32/winmsg/about-hooks, 2022.

[81] "Setwindowshookexa function," https://learn.microsoft.com/en-us/windows/win32/api/winuser/nf-winuser-setwindowshookexa, 2023.

[82] boppreh, "keyboard," https://github.com/boppreh/keyboard, 2023.

Appendix A. Lossless Channel Adapter and I/O Isolator

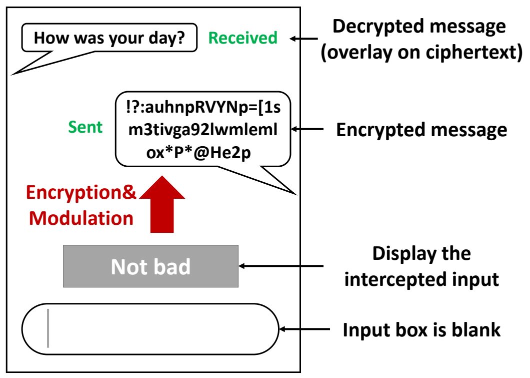
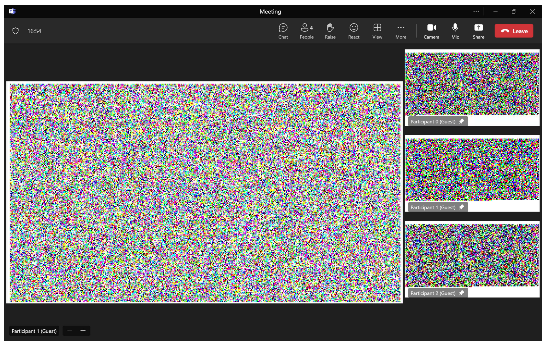

How was your day?
Received
Decrypted message (overlay on ciphertext)
Sent
!?:auhnpRVYNp=[1s m3tivga92lwmleml ox*P*@He2p
Encrypted message
Encryption& Modulation
Not bad
Display the intercepted input
Input box is blank

Figure 15: **Secure Chat Interface *mTunnel*.**

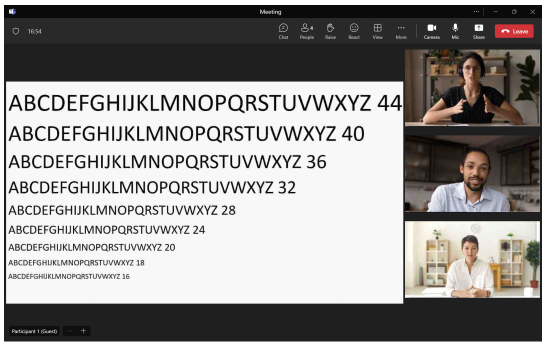
Lossless Channel Adapter: As illustrated in the chat interface displayed in Fig. 15, the lossless channel adapter is responsible for leveraging text in chat input to encapsulate ciphertext. The lossless channel adapter encrypts the input text into ciphertext and then modulates the ciphertext to a set of visible characters through Base64 encoding [79]. It also prepends a header "!?:" to inform the receiver side that an encrypted frame has been received. On the receiver side, after detecting the header, the lossless channel adapter demodulates the received characters into ciphertext and subsequently decrypts the ciphertext to plaintext.

I/O Isolator: On the sender side, the I/O isolator intercepts and suppresses keystrokes to prevent keystroke eavesdropping. On one hand, it displays the plaintext on the screen so that users can view and modify the content. On the other hand, when the user stops typing and presses "Enter" to send a message, it transfers the recorded input to the lossless channel adapter to undergo encryption and modulation.

Windows offers hooks as a special mechanism that permits developers to install a subroutine to monitor keystroke messages and process them before they reach the target window [80]. Using hooks, we can identify a keystroke, suppress it, and play the ciphertext as a combination of other keystrokes instead. Windows provides an API called

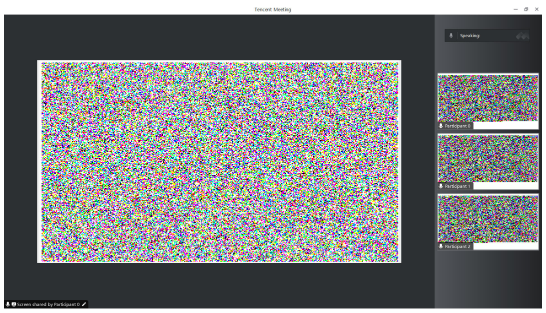


(a) Screenshot of Client UI (Teams)

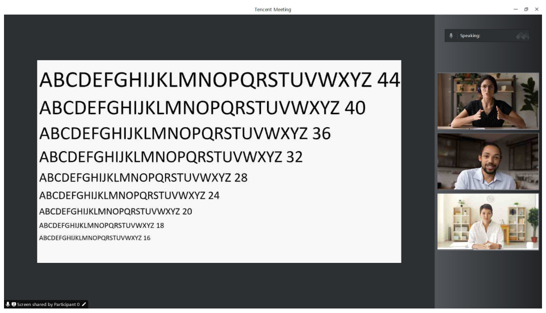


ABCDEFGHIJKLMNOPQRSTUVWXYZ 44
ABCDEFGHIJKLMNOPQRSTUVWXYZ 40
ABCDEFGHIJKLMNOPQRSTUVWXYZ 36
ABCDEFGHIJKLMNOPQRSTUVWXYZ 32
ABCDEFGHIJKLMNOPQRSTUVWXYZ 28
ABCDEFGHIJKLMNOPQRSTUVWXYZ 24
ABCDEFGHIJKLMNOPQRSTUVWXYZ 20
ABCDEFGHIJKLMNOPQRSTUVWXYZ 18
ABCDEFGHIJKLMNOPQRSTUVWXYZ 16

(b) Screenshot of Recomposited Client UI (Teams)



(c) Screenshot of Client UI (Tencent Meeting)



ABCDEFGHIJKLMNOPQRSTUVWXYZ 44
ABCDEFGHIJKLMNOPQRSTUVWXYZ 40
ABCDEFGHIJKLMNOPQRSTUVWXYZ 36
ABCDEFGHIJKLMNOPQRSTUVWXYZ 32
ABCDEFGHIJKLMNOPQRSTUVWXYZ 28
ABCDEFGHIJKLMNOPQRSTUVWXYZ 24
ABCDEFGHIJKLMNOPQRSTUVWXYZ 20
ABCDEFGHIJKLMNOPQRSTUVWXYZ 18
ABCDEFGHIJKLMNOPQRSTUVWXYZ 16

(d) Screenshot of Recomposited Client UI (Tencent Meeting)

Figure 16: **Using *mTunnel* with Microsoft Teams and Tencent Meeting**.

No.	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	Avg.
Gender	M	M	M	M	M	M	M	M	M	M	M	F	F	F	F	F	F	F	F	F	/
MOS	3.6	1.8	2.6	2.8	1.8	2.4	3.0	2.8	3.2	2.2	2.8	3.0	2.0	2.0	1.8	2.4	3.0	4.0	2.8	3.2	2.66
MOS$_{LE}$	4.0	2.0	3.4	2.6	1.8	3.4	2.6	3.2	3.4	2.0	2.2	3.0	2.2	2.4	2.8	3.2	3.2	5.0	4.0	3.8	3.01

TABLE 4: **MOS Values of PSTN-Tunneled Audio (with Codec2@700 bps CBR)**

Score	MOS	MOS$_{LE}$
5	Excellent	Complete relaxation possible; no effort required
4	Good	Attention necessary; no appreciable effort required
3	Fair	Moderate effort required
2	Poor	Considerable effort required
1	Bad	No meaning understood with any feasible effort

TABLE 5: **MOS Opinion Scales**

SetWindowsHookEx[81] to facilitate hook implementation. To implement our text isolator, we use a Python wrapper [82] of this API. By specifying event types and messages, we can use hooks to automatically execute encryption or decryption callback functions.

On the receiver side, the I/O isolator captures the screen and recognizes the encrypted and modulated sequence through optical character recognition (OCR). Then it delegates the lossless channel adapter to decode the received characer. Finally, it displays the decoded plaintext to overlay the ciphertext.

Appendix B. Securing Microsoft Teams and Tencent Meeting

Fig. 16 showcases using *mTunnel* to secure other remote conference applications. It adopts an identical data transmission configuration as the main body evaluation. *mTunnel* is capable of functioning across various remote conference applications after small adaptation to different UI layouts.

Appendix C. MOS Experiment on PSTN-Tunneled Audio

We adhere to the ITU-T MOS guideline [54] to assess the audio quality after the PSTN tunnel. Participants are directed to listen to five groups of sentences, each consisting of two English sentences from [33] encoded using Codec2 700C, *i.e.*, 700 bps CBR. All sentences are approximately 2 to 3 seconds, making it difficult to derive much context-based understanding. The sentences also vary in terms of the talker's gender and intonation, as the codec may respond differently to various voice frequencies.

Table. 5 is the scoring standard presented to participants. Following the playback of each sentence group, participants provide their feedback on the perceived quality of the speech

(MOS), as well as the effort required to understand the sentences (MOS_{LE}). The effort required for understanding is not necessarily determined by the audio quality, as the presence of noise may not hinder the comprehension of words.

Table. 4 shows the average MOS value from each participant. Results indicate that moderate effort is required to understand the sentences.

Appendix D. Meta-Review

The following meta-review was prepared by the program committee for the 2024 IEEE Symposium on Security and Privacy (S&P) as part of the review process as detailed in the call for papers.

D.1. Summary

Voice conferencing is ubiquitous, but platforms vary in their security in practice. This paper proposes mTunnel, a shim that sits between media devices and the conference software. mTunnel encrypts data using a variant of the Signal protocol before it is fed into the conferencing client. The upshot is that the user can independently ensure E2EE regardless of the client's behavior. Along the way, authors show some other neat advantages like enabling E2EE for dial-in clients.

D.2. Scientific Contributions

- Creates a New Tool to Enable Future Science
- Addresses a Long-Known Issue
- Provides a Valuable Step Forward in an Established Field
- Other

D.3. Reasons for Acceptance

1) Authors significantly improve and expand on tunneling encrypted data/media through unpredictable, compressed media channels.
2) This approach provides for defense-in-depth even in E2EE conference platforms that may not be otherwise trustworthy.
3) Support for PSTN bridges required substantial innovation.
4) The contributions are clear, and the tool is well constructed.

D.4. Noteworthy Concerns

This approach faces significant barriers to deployment, including group key distribution challenges.