POSTER: Participant and Channel Privacy in End-to-End Encrypted VoIP Teleconferencing

Samuel P. Laney*, Justin A. Blanco†, Travis Mayberry‡ and Daniel S. Roche§
United States Naval Academy
*Email: sam.laney18@gmail.com
†Email: blanco@usna.edu
‡Email: mayberry@usna.edu
§Email: roche@usna.edu

I. Introduction

Voice over Internet Protocol (VoIP) chat services such as Discord allow groups to easily connect to one another with voice teleconferencing. However, most VoIP traffic is either unencrypted or encrypted only in transit, allowing for a compromised or overly-curious server to analyze and listen to any traffic sent through it. Schemes currently exist that improve security by implementing end-to-end encryption, preventing the server from interpreting the contents of a message, but these schemes still reveal conversation *metadata* such as who is in a channel and who is talking at any given time.

This paper describes our work on a teleconferencing application with a conversation model similar to Discord that not only features full end-to-end encryption, but also hides conversation metadata. Obscuring this metadata is important because in many cases the contents of a given conversation matter less than the fact that a conversation occurred between two or more parties. Hackers, cyberstalkers, and leakers do not need to know the contents of communications in order to paint a clear picture of one's associations, interests, and habits; knowing who one connects with, how long they talk for, and how frequently, is often more than enough.

We developed our scheme with the following objectives:

- Full end-to-end encryption
- Multiple channels on a given server
- Multiple users per channel
- Protection against malicious users
- Metadata hiding through:
  - Channel population hiding
  - Obscuring who is speaking at any given time

Current secure alternatives for teleconferencing, such as Signal, utilize a peer to peer infrastructure. This inherently limits the number of concurrent users (originally to 8, now 40), requiring the client to mix and perform all computations for the protocol [1]. Our use case expands the single call model to multiple channels, like Discord, which are mixed at the server. This allows for increased scalability as the computation burden remains constant for users. Rohloff et al. were successful in implementing a centrally mixed, end-to-end encrypted voice conferencing platform using homomorphic operations, but their scheme did not obscure channel membership [2].

II. The Protocol

The primary novel feature of our scheme is that the central server does not know which user is in each channel. This prevents the server from tracking who talks to one another, when they do so, and how often they communicate in a given channel. Our threat model presumes an honest-but-curious server, and malicious users that do not collude with the server. The protocols utilize the Microsoft SEAL homomorphic encryption library[3]. Our system consists primarily of four protocols to perform initial setup, to specify each user's channel membership, to send and receive streaming audio, and to identify attempted malfeasance.

A. Setup

Assume a teleconferencing server with n users amongst m channels. A common key pair is used by the users and is agreed upon out of band. The server will utilize the public key in order to perform homomorphic operations, but it does not possess the secret key.

B. Pick Channel

Each user$_i$, where i runs from 1 to n, decides the channel [1, 2, … , *m*] they want to communicate in. User$_i$ then creates a length-*m* vector initialized to 0, except for the channel they are in, which holds a 1. A user in channel 2 on a server with 5 channels, for example, would have the membership vector [0, 1, 0, 0, 0]. Each user then encrypts their membership vector, resulting in a vector of ciphertexts $\mathbf{h_i}^T = [c_{i1}, c_{i2}, c_{i3}, ..., c_{im}]$. Although all but one of these values is 0 when decrypted, the CKKS encryption scheme used by SEAL is randomized, so the ciphertext objects at each index of the vector differ. Each user sends their membership vector to the server where it is stored. Therefore, the server has n membership vectors, each containing m ciphertexts. This protocol requires several expensive encryptions, so it will run only when a user joins a server.

C. Send Audio

Assuming 16kHz mono-audio, user$_i$ generates 4,000 Pulse Code Modulation (PCM) frames (samples) per quarter-second. The user batches these values and encrypts them into the vector $\mathbf{v_i}$, then sends this ciphertext to the server. The server then

multiplies this by the user's channel vector stored on the server *(the following operations are homomorphic, meaning they are performed in the encrypted space):*

$$\mathbf{B_i} = \mathbf{v_i} \otimes \mathbf{h_i} \tag{1}$$

$\mathbf{B_i}$: user$_i$'s contribution to all channels on the server

Note that since each user is only in one channel, the audio samples will be multiplied by the encryption of 1 once, and every other column in $\mathbf{B_i}$ remains as 4,000 encryptions of 0. The server performs this operation for each user on the server concurrently and adds all n matrices together:

$$\mathbf{D} = \sum_{i=1}^{n} \mathbf{B_i} = [\mathbf{d_1}, \mathbf{d_2}, ..., \mathbf{d_m}] \tag{2}$$

$\mathbf{D}$: $4,000 \times m$ matrix containing the encrypted contents of each channel for one epoch

This produces the matrix $[\mathbf{d_1}, \mathbf{d_2}, ..., \mathbf{d_m}]$ where $\mathbf{d_1}$ is a length 4,000 vector of the encrypted sum of all audio signals in channel 1, $\mathbf{d_2}$ is channel 2, etc. Finally, the server computes the correct sum to send to each user based on what channel they are in and subtracts their own audio to eliminate echo:

$$\mathbf{a_i} = \mathbf{D h_i} - \mathbf{v_i} \tag{3}$$

With this final matrix-vector product, $\mathbf{a_i}$ ends up being an encryption of just the channel that user$_i$ is in since the other channels' audio is multiplied by an encryption of 0. User$_i$ decrypts $\mathbf{a_i}$ using the secret key and has a quarter-second series of audio samples from their channel, while the server cannot determine which channel the user is communicating in.

We secure channel population metadata by ensuring that all users are indistinguishable from one another to the server. Each user has a channel membership vector of length m and sends a fixed-length, encrypted PCM stream at constant interval, regardless of if a user is speaking or not. All user audio is mixed with the same protocol, yielding no participation information to the server other than if a user is connected.

D. Find Cheaters

As noted, our threat model assumes malicious users, such that users should be able to send to and receive from only one channel at any given time. This would prevent a malicious user from interfering with or listening in to multiple channels. An attacker could make the service unusable by configuring one's channel vector to contain a one (or any nonzero value) in multiple indices, thus "joining" multiple channels, after which one could broadcast shouting or interference to every channel.

The server stores each user's channel membership vector, which should contain exactly one encryption of 1 and $m - 1$ encryptions of 0. However, since each vector is encrypted, the server cannot check to make sure that this is the case. Another user could decrypt and check, but doing so would reveal which channel the user is in, which is data we intend to protect. Instead, the server will compute $m + 1$ values for each user$_i$: one value s_i and m values t_{ij}, one for each index j in the membership vector.

$$s_i = c_{i1} + c_{i2} + ... + c_{im} \tag{4}$$

$$t_{ij} = c_{ij} \times (c_{ij} - ENC_{pk}(1)) \tag{5}$$

If user$_i$ is not cheating, then $DEC_{sk}(s_i) = 1$. Further, each value of $DEC_{sk}(t_{ij})$ should equal 0. This is because each c_{ij} should be either 0 or 1, so t_{ij} should either be an encryption of $1 \times (1 - 1) = 0$ or $0 \times (0 - 1) = 0$. At a varied interval, the server will compute each of these values for each user and send them to randomly selected clients to check that each $DEC_{sk}(s_i) = 1$ and every $DEC_{sk}(t_{ij}) = 0$. This ensures that each user has a properly formatted membership vector and is only in one channel.

III. Conclusion

We secure channel population metadata by ensuring that users' audio is mixed identically, regardless of channel. Each user has a channel membership vector of length m and sends an encrypted, fixed-length PCM stream at constant interval, regardless of if a user is actively speaking or silent during the interval. All user audio is mixed with the same protocol that yields no participation information to the server other than whether a user is connected.

In the current state, our implementation is a command-line Python application that allows for multiple "channels," or independent conference calls, facilitated by a single instance of the server program. The application utilizes the Microsoft SEAL homomorphic encryption library. Our local proof of concept has demonstrated that our scheme outpaces the rate at which audio is supplied. Historically, homomorphic encryption has not been feasible for real-time applications due to its slow computation times. Our scheme outsources most of the intensive computation to the central server, while keeping user work to only encrypting and decrypting audio samples. Server computation scales linearly with the number of users, while user computation remains constant, regardless of how many users are connected. These factors make us optimistic about its ability to scale better than a peer-to-peer infrastructure, handling dozens of live users over the network. The coming months are dedicated to investigating the scalability of the scheme in the cloud environment.

Acknowledgment

The authors would like to thank the US Naval Academy Computer Science Department, the Trident Scholar Program, and the Office of Naval Research for sponsoring this student work.

References

[1] Signal Foundation, *Group Calling - Voice or Video with Screen Sharing*

[2] K. Rohloff, D. B. Cousins and D. Sumorok, "Scalable, Practical VoIP Teleconferencing With End-to-End Homomorphic Encryption," in *IEEE Transactions on Information Forensics and Security*, vol. 12, no. 5, pp. 1031-1041, May 2017, doi: 10.1109/TIFS.2016.2639340.

[3] "Microsoft SEAL (release 3.6)." https://github.com/Microsoft/SEAL. (2020).

Participant & Channel Privacy in End-to-End Encrypted VoIP Teleconferencing

Midshipman 1/C Sam Laney, Computer Science Department, US Naval Academy
Assoc. Prof. Dan Roche, Assoc. Prof. Travis Mayberry, Assoc. Prof. Justin Blanco

The Problem

Current teleconferencing applications are either **unencrypted** or **encrypted only in transit**, meaning that a compromised or over-curious server can analyze and listen to **any and all traffic** sent through it. [1]

The Solution

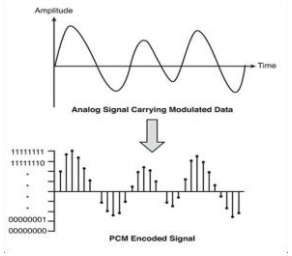
Utilize:

- **End-to-End Encryption**
- **Metadata Protection**
- **Homomorphic Encryption**

To ensure the server does not know *whom is talking to whom and when*

Background

How does a computer represent and mix audio?



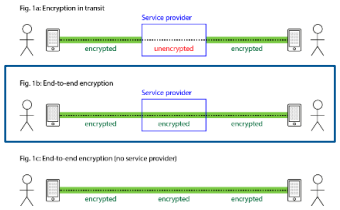
Pulse Code Modulation (PCM) of an analog signal

http://www.electronicshub.org/wp-content/uploads/2013/10/Pulse-Code-Modulation.jpg

- A computer's sound card generates PCM values from a microphone.
- These values can be **added together** to express **multiple parties talking at the same time**

End-to-End Encryption

- End-to-End Encryption is the **unbroken** encryption of a communication from the initiating user to the destination user.
- This ensures that any intermediate servers, routers, or services between the two users cannot inspect the **contents** of their communication.



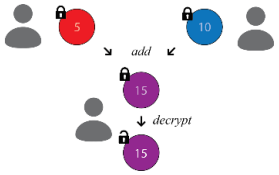
Protocol Encryption Models

Metadata Protection

Our scheme seeks to protect user metadata such as **who is conversing together** in a channel, and **who is speaking** at any given time.

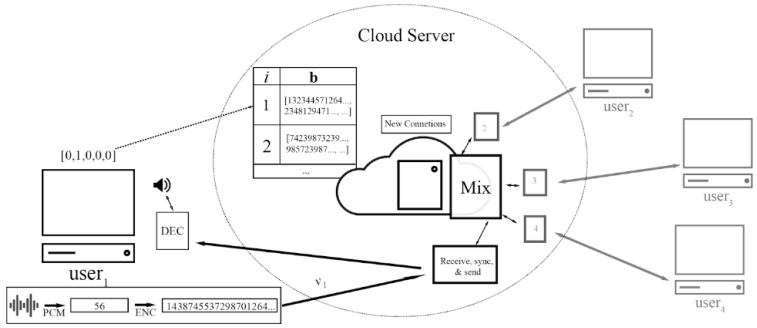
Homomorphic Encryption

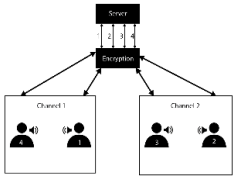
Homomorphic Encryption allows for **mathematical operations** to be performed **on encrypted data**. This allows a central, **untrusted** server to mix, or add, audio signals to **create a teleconference** without revealing any **conversation contents** to the server. Our protocol uses the Microsoft SEAL library. [3]



A simple example of a homomorphic addition

Our Protocol





The mixing scheme obscures channel membership

Pick Channels:

user wants to be in channel 2

$|0,1,0,0|$

$\downarrow ENC_{pk}$

$\mathbf{h} = [774AC2..., 5238A3..., F02342..., A8214F...]$

$\downarrow$

send to central server

Our Pick Channel Protocol

Send Audio:

speaking generates PCM value v

$\downarrow ENC_{pk}(v)$ (bold = encrypted)

send to central server

$\mathbf{b_i} = ENC_{pk}(v)[\mathbf{c_{i1}}, \mathbf{c_{i2}}, ..., \mathbf{c_{im}}]$

$[(v_j)(0), (v_j)(1), (v_j)(0), (v_j)(0)]$

[0, 56, 0, 0]
[0, 35, 0, 0]
[0, 0, 0, 23]
[0, 0, 0, 40]
+
[0, 91, 0, 63]
[0, 56, 0, 0]
−
[0, 35, 0, 63]

$\mathbf{h} = [0, 1, 0, 0]$

$\mathbf{a_i} = 35$

$\downarrow$ *send to user*

$DEC_{sk}(a_j)$ = channel 2 audio = 35

Example epoch

- In our protocol, each user sends audio chunks every 250ms.
- The client encrypts this vector of PCM samples and sends it to the server.
- The server mixes the audio homomorphically and utilizes channel vectors to select and send each user their respective channel audio, without revealing to the server which channel they are in.

Results & Conclusion

- We have successfully implemented our encrypted mixing protocol as a proof of concept.
- Homomorphic encryption is known for being slow, but the scheme outpaces the rate at which audio is supplied.
- We have successfully implemented the multi-channel teleconferencing application in the unencrypted space.
- The next steps for our Trident project are adding encryption to our current application and testing scalability in a cloud environment.

References

[1] K. Rohloff, D. B. Cousins and D. Sumorok, "Scalable, Practical VoIP Teleconferencing With End-to-End Homomorphic Encryption," in IEEE Transactions on Information Forensics and Security, vol. 12, no. 5, pp. 1031-1041, May 2017, doi: 10.1109/TIFS.2016.2639340.

[2] Ahmad Ishtiyaque, et al. "Addra: Metadata-private voice communication over fully untrusted infrastructure," *15th USENIX Symposium on Operating Systems Design and Implementation*, Jul. 2021.

[3] "Microsoft SEAL (release 3.6)." https://github.com/Microsoft/SEAL. (2020).