Now is the Time: Scalable and Cloud-supported Audio Conferencing using End-to-End Homomorphic Encryption

David Hasselquist
Linköping University, Sweden
Sectra Communications, Sweden

Niklas Johansson
Linköping University, Sweden
Sectra Communications, Sweden

Niklas Carlsson
Linköping University, Sweden

ABSTRACT

Homomorphic encryption (HE) allows computations on encrypted data, leaking neither the input nor the computational output. While the method has historically been infeasible to use in practice, due to recent advancements, HE has started to be applied in real-world applications. Motivated by the possibility of outsourcing heavy computations to the cloud and still maintaining end-to-end security, in this paper, we use HE to design a basic audio conferencing application and demonstrate that our design approach (including some advanced features) is both practical and scalable. First, by homomorphically mixing encrypted audio in an untrusted, honest-but-curious server, we demonstrate the practical use of HE in audio communication. Second, by using multiplication operations, we go beyond the purely additive audio mixing and implement advanced example features capable of handling server-side mute and breakout rooms without the cloud server being able to extract sensitive user-specific metadata. Whereas the encryption and decryption times are shown to be magnitudes slower than generic AES encryption and roughly ten times slower than Signal's AES implementation, our solution approach is scalable and achieves end-to-end encryption while keeping performance well within the bounds of practical use. Third, besides studying the performance aspects, we also objectively evaluate the perceived audio quality, demonstrating that this approach also achieves excellent audio quality. Finally, our comprehensive evaluation and empirical findings provide valuable insights into the tradeoffs between HE schemes, their security configurations, and audio parameters. Combined, our results demonstrate that audio mixing using HE (including advanced features) now can be made both practical and scalable.

CCS CONCEPTS

• **Security and privacy**; • **Computing methodologies** → **Distributed computing methodologies**; • **Networks** → **Cloud computing**;

KEYWORDS

Audio Conferencing, Privacy, End-to-End Encryption, Homomorphic Encryption, Secure Computation, Cloud Computing

CCSW '23, November 26, 2023, Copenhagen, Denmark

ACM ISBN 979-8-4007-0259-4/23/11.
https://doi.org/10.1145/3605763.3625245

ACM Reference Format:
David Hasselquist, Niklas Johansson, and Niklas Carlsson. 2023. Now is the Time: Scalable and Cloud-supported Audio Conferencing using End-to-End Homomorphic Encryption. In *Proceedings of the 2023 Cloud Computing Security Workshop (CCSW '23), November 26, 2023, Copenhagen, Denmark.* ACM, New York, NY, USA, 13 pages. https://doi.org/10.1145/3605763.3625245

1 INTRODUCTION

Secure online communication is essential for many businesses and people. To protect against potential eavesdroppers, the data must therefore typically be carefully encrypted. However, different encryption-based solutions and the encryptions they use come with their own overheads and limitations.

Today, most multi-party audio systems use one of two encryption approaches: (1) *end-to-middle* encryption with a server in the middle decrypting, mixing, and re-encrypting the mixed audio streams, or (2) *end-to-end* encryption with clients performing audio mixing after obtaining encrypted streams of each other participating client. A major downside of the first approach is that it requires users to trust the service provider (who would need to decrypt the data on middle servers before mixing the audio). While this is not the case with the second approach (since the server does not have access to the decryption keys), a heavy load must be put on the end devices. For example, in the case of a group conversation with n users, each user is required to maintain O(n) end-to-end streams, where each stream must be encrypted, decrypted, and mixed.

In contrast, we instead consider a third approach: Cloud-supported audio mixing using homomorphic encryption (HE). Here, clients encrypt recorded audio data and send their encrypted streams to a middle server (e.g., in the cloud) that performs the audio mixing on the encrypted streams before sending the modified-but-still-encrypted streams to the participating clients. The use of HE ensures that the middle servers can apply the necessary numeric calculations needed for proper audio mixing directly on the ciphertexts, without them ever having access to any decryption keys or the decrypted audio streams themselves. Similar to the first approach, this allows the solution to scale more easily, as the load on the clients is independent of the number of clients, and similar to the second approach, the solution ensures that servers cannot observe the streams themselves.

While performance limitations [48] associated with HE long have limited its practical use, recent advancements in the field have come to change this. For example, fully HE is now implemented and used in Microsoft's Edge browser to compare users encrypted login information to known leaks [16, 17, 37]. HE has also been used in the contexts of face recognition [23], privacy-preserving blockchains [68], medicine [7, 36, 65], and various machine learning problems [12, 13, 15, 20, 22, 35, 65]. Supported by our performance

results, we demonstrate and argue that now also is the time to start applying HE to secure end-to-end group communication.

In this paper, we present the design and evaluation of a secure, end-to-end, audio conferencing application implemented using HE, with our comprehensive evaluation providing the first practical demonstration that the audio mixing using HE is both practical and scalable using current state-of-the-art HE. Instead of placing the audio mixing and the mass encryption/decryption of streams on the client side, we use HE to mix audio on an untrusted server before forwarding the mixed audio to each user. We assume an honest-but-curious threat model where the untrusted server observes all internal operations and attempts to learn all possible information. Our approach maintains the end-to-end encryption and is scalable, as each client must only encrypt and decrypt (as well as send/receive) a single stream. Our results capture the various tradeoffs between the HE schemes, their security configurations, and the chosen audio parameters.

While we are not the first to suggest the use of HE in the audio communication context (see Section 7), we provide the first comprehensive demonstration and evaluation of such a solution. Prior works have theoretically reduced the circuit depth, ignored the actual audio quality perceived by the end users, and have only considered additive features [56]. In contrast, we present practical implementations that include new advanced features (e.g., server-side mute and breakout rooms) that demonstrate the practicality of using multiplicative HE (needed for some of the implemented features) and the scalability achievable with such implementations. Our comprehensive experiments provide many insights into the best design choices when implementing end-to-end encrypted group communication using HE.

By studying the different components of the communication process both individually and in combination, we share insights into individual performance bottlenecks, and, as reference points, we compare the performance to the AES counterparts used in (1) a generic cryptographic library and (2) the Signal application. Furthermore, through the use of practical Quality of Experience (QoE) metrics, we provide the first insights into how the perceived audio quality that such solutions can offer is impacted as the system is scaled and/or implemented in various ways. Our findings provide valuable insights for service providers wanting to implement a practical and scalable end-to-end encrypted audio conferencing service. We also note that the approach may come to challenge the constraints that many services today put on the maximum number of participants they can support [44, 59, 67, 69]. For example, Microsoft Teams currently offers end-to-end encryption for only one-on-one calls [44], while Signal recently increased their support from 5 to 40 participants [59].

Summary of contributions: (1) The first comprehensive demonstration that audio mixing using HE is both practical and scalable, with our results showing that *now* is the time to start using HE to secure end-to-end communication. (2) New advanced features implemented using HE that enable server-side mute and breakout rooms without the server being able to extract sensitive user-specific metadata. (3) Quantitatively supported insights into how performance bottlenecks and audio quality is impacted by different design and implementation choices.

Outline: Section 2 first presents the background of HE. Section 3 then presents our framework for audio mixing, the supported features, and their parameter selection. The performance of the HE schemes is presented in Section 4, and the end-to-end performance in Section 5. Finally, we discuss QoE and perceived audio quality in Section 6, related works in Section 7, and conclusions in Section 8.

2 HOMOMORPHIC ENCRYPTION

Homomorphic encryption (HE) allows a third party to perform computation over encrypted data without learning the cleartext input or output. The concept was introduced early by Rivest et al. [54] in 1978 but became realized in 2009 when Gentry [26] presented the first feasible fully homomorphic encryption scheme. Since then, several schemes have been proposed, and rapid advancements in efficiency have been made. During the last decade, the computation time has been reduced from minutes to milliseconds [19, 24, 48].

There are currently three leading HE schemes: BFV [10, 25], BGV [11], and CKKS [18]. BFV is similar to BGV in its design, and they both support encryption and computation based on integers. In contrast, CKKS is approximate and supports numerical calculations on float numbers. The security of these schemes is based on the Ring Learning with Errors (RLWE) problem [41], a continuation of the Learning with Error (LWE) problem first formulated by Regev [52].

In 2012, Fan and Vercauteren [25] proposed an extension of Brakerski's [10] scheme, becoming the Brakerski/Fan-Vercauteren scheme (BFV). Brakerski's scheme builds on the LWE problem and reduces the noise growth from quadratic to linear when performing multiplication. Fan and Vercauteren improved the scheme by extending the problem to RLWE, making the relinearization key smaller and the computations faster using modulus switching.

More recently, Cheon et al. [18] presented a HE scheme supporting computations of float numbers, the Cheon-Kim-Kim-Song scheme (CKKS). As opposed to previous schemes, the results obtained after decryption are not exact but approximate due to a rescaling procedure in the scheme.

The hardness of the RLWE problem is based on lattice-based cryptography, currently considered quantum-safe, and cannot (as long as this common conjecture holds) be solved by quantum computers in polynomial time [41, 42]. A homomorphically encrypted ciphertext consists of a vector $v + x$, where v is in the ideal lattice, and x is the error/noise vector. In simplified terms, each ciphertext can be seen as a polynomial of degree n (ring dimension) with added noise. In BFV, we consider the case where the ring dimension corresponds to the number of available slots in the ciphertext and the batch size is defined as the number of filled slots. For example, using a ring dimension of n=2^{12}=2048 and a batch size of b=1280, only 1280 out of the 2048 slots are filled. The remaining slots are unused, but must be kept in the ciphertext to ensure the security properties of the scheme. This means that the ciphertext size remains constant as long as the batch size is less or equal to the ring dimension. With CKKS, the largest batch size is $n/2$. In practice, the unused ciphertext slots can be used for metadata or other types of communication data (e.g., chat messages or screen sharing) without the cost of additional bandwidth. In the audio communication context, the overhead can also be used to add (adaptive) forward error correction [8] and obtain error control in the audio transmission.

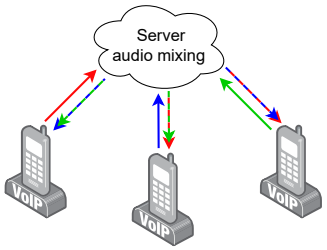


Figure 1: Overview of three VoIP clients and audio mixing on cloud-based middle server.

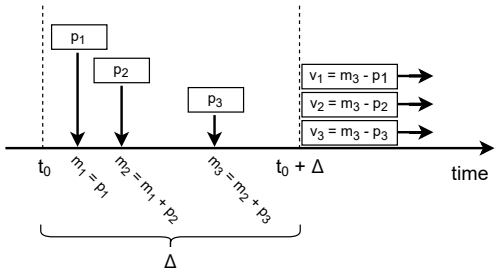
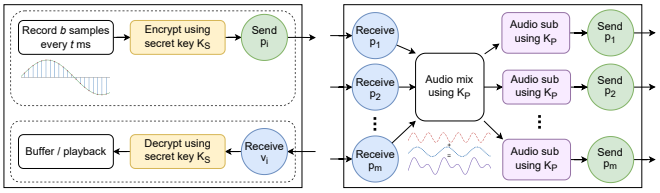


Figure 2: Example iteration of continuous server mixing with three clients and batch time Δ.



(a) Example client i (b) Server

Figure 3: Overview of the execution flow for a batch iteration with m clients.

Furthermore, with the Single Instruction Multiple Data (SIMD) paradigm [62], each slot in a ciphertext can be homomorphically calculated at the cost of one instruction. For example, to add two ciphertexts $c^{(1)}$ and $c^{(2)}$, each with a degree of n, a single homomorphic instruction is used to calculate $c_i^{(3)} = c_i^{(1)} + c_i^{(2)}$ for all i $(0 \leq i < n)$ of the resulting ciphertext $c^{(3)}$, where $c_i^{(1)}$, $c_i^{(2)}$, and $c_i^{(3)}$ represent the individual values of the three ciphertexts.

However, with HE, the noise grows with the number of ciphertext operations, with multiplications being several magnitudes more expensive than additions. To ensure correctness when decrypting the ciphertext, the noise must be kept below a certain threshold. In fully HE, the ciphertext can be refreshed using bootstrapping and a relinearization process. This effectively reduces the ciphertext size and noise but is considered very slow in practice [26].

3 AUDIO MIXING FRAMEWORK

Audio signals consist of waves that can be combined by adding them together according to the superposition principle. This process is usually referred to as audio mixing. When capturing analog signal and converting to digital, pulse code modulation (PCM) is usually preferred. Here, the continuous signal is sampled at a predefined *sampling rate* and the sample value is converted into a digital value using a predefined number of bits, called the bit depth. The resolution of the discretized signal increase with the bit depth and sampling rate. Common standard sampling rates in teleconferencing include narrowband (8 kHz), wideband (16 kHz), super-wideband (32 kHz), and fullband (48 kHz).

3.1 High-level framework overview

To evaluate HE in the audio context, we design and implement a client-server VoIP application. Figure 1 shows a conceptual overview of our framework with three participating clients and an untrusted cloud server. Here, each client uses one upstream and one downstream to send and receive audio, respectively. As the audio streams are mixed at the server, each downstream contains the mixed audio of the other participants. However, to avoid delayed playback of a client's audio, each client's audio is subtracted from the mixed audio before being sent to the client. Therefore, in a call with m participants, the mixed audio sent over the network from the server to a client contains the audio of $m - 1$ clients.

In VoIP, continuous audio needs to be sampled and batched together before being sent over the network. Using a sample rate of r, each client records their audio and creates groups of batch time Δ. We define this as one iteration, resulting in an audio batch size $b = \Delta \times r$. Given a ring dimension of n, the maximum batch size b is n for BFV, and $n/2$ for CKKS. The maximum batch time is bounded by $\Delta \leq n/r$ for BFV, and $\Delta \leq (n/2)/r$ for CKKS. In a call configured to use Δ=40 ms and super-wideband (r=32 kHz), a client encrypts and sends b=1280 audio samples every 40 ms to the server for mixing. Simultaneously, each client receives 1280 mixed audio samples every 40 ms from the server to be decrypted and played. The underlying BFV and CKKS implementation use 64 bits to internally represent encrypted values. Therefore, once decrypted, we can perform audio normalization on the client side to avoid audio clipping (which can occur when the digital samples bounded by their bit depth have been summed).

At the server, the audio is continuously mixed as it arrives. At every iteration of Δ, the audio of each user is subtracted from the mixed audio before being sent to the user. Figure 2 shows an example of a server mixing with three clients. Here, the server receives packet p_i from client i, and the audio samples are continuously mixed as arrived (except for the first packet in that iteration). At the end of the iteration, the server calculates v_i by subtracting the resulting mixed audio with p_i and sends v_i to client i. In the example, all packets are received within one iteration. However, due to jitter and bandwidth variations, packets may be received in a different time batch. If a server receives several packets in the same iteration from the same user, only the first packet is mixed, and the remaining packets are buffered for mixing in the next iteration. In case no audio is received from a client within a time iteration, no audio is subtracted, and the full mixed audio is sent to that client.

Figure 3 summarizes the execution flow for every iteration for a client and a server. On the client side (Figure 3(a)), the audio is

captured and received simultaneously. The sending process captures the audio using PCM and then encrypts using the secret key K_S, while the receiving process handles the decryption and audio playback. For scalability at the server (Figure 3(b)), packets are received in parallel before being merged for mixing. Given that the two audio samples have been recorded using the same sample rate, the audio mix is accomplished by combining their respective digital samples. When played back, the resulting audio wave contains the other two audio waves, resulting in an audio mix. After the mixing, the client's audio is subtracted and sent individually to each client. Furthermore, each audio sample sent over the network includes a timestamp to measure the end-to-end delay, and discard packets delayed by more than 150 ms. Clients and servers operate asynchronously and can join and leave the call continuously.

In our proof-of-concept implementation, all clients share a pre-distributed secret key K_S for encryption and decryption. We consider the pre-distribution of these keys out of scope for this paper. With this scheme, the server does not have access to the secret key, but instead uses the public key K_P to perform ciphertext calculations. This ensures that an honest-but-curious middle server would not be able to decrypt and extract sensitive information.

3.2 Implemented example features

In our framework, we implement three example features and use the smallest ring dimension required to support each feature given a security level. The *basic* feature models the requirements for a minimal, scalable audio conference application. The *server-side mute (SSM)* and *breakout-room privacy (BRP)* features then build upon this basic feature by adding additional functionalities. These two advanced features (SSM and BRP) are implemented using homomorphic multiplications. In addition to the features implemented and demonstrated here, we note that additional advanced server-side features can easily be implemented to improve the service further. We next present our three features in more detail.

Basic: We have designed the basic feature to support conference calls of at least 1000 participants and at least 8 participants speaking simultaneously. Unless all speakers are close to screaming, this limit is usually much higher. Our implementation allows users to continuously join and leave the call, speak and listen simultaneously, and the received audio does not contain any (delayed) playback of the user's own voice (as we remove this after the mixing). Here, each user sends an audio sample every batch time Δ to the server for mixing, regardless of speaking or being silent. This provides speaker anonymity, where the server cannot distinguish between a speaker and a silent user. While this comes at the cost of some additional bandwidth, for some services, this may be a small price to pay for additional privacy protection. Furthermore, we note that the bandwidth needed per client is upper bounded by one upstream and one downstream, and that the server's bandwidth scales linearly with the number of participants. Finally, we consider the communication service to be practical if the end-to-end latency is less than 150 ms, a threshold considered acceptable in VoIP communication [47]. The basic feature requires no ciphertext multiplication.

Server-side mute (SSM): As each client continuously sends audio packets to the server to preserve speaker anonymity, the basic feature does not allow a conference administrator to control the contents or selection of audio packets to mix. To allow an administrator to mute each participant individually without the server knowing which participants are muted, our implementation of the SSM feature therefore uses a preconfigured homomorphically encrypted ciphertext of dimension n for each user. These ciphertexts contain either the value 1 or 0 in all its slots, and can be continuously updated by an administrator throughout the call. Upon receiving an encrypted audio sample from a user, the sample is multiplied with the preconfigured ciphertext, resulting in a new ciphertext of either silence (only values 0) or the original encrypted audio sample. The result of the ciphertext–ciphertext multiplication is then mixed with the results from other users before the user's audio is subtracted from the mixed ciphertext and sent to the user. The SSM feature requires each ciphertext to be multiplied once.

Breakout-room privacy (BRP): A common feature in many conferencing applications is the breakout room, where users are divided into groups, and each group can communicate in parallel. However, the information of whom the user is communicating with can be privacy sensitive and should be preserved if possible. With the BRP feature, users can communicate separately in groups without the server knowing the room to which each user belongs. This further improves user privacy and reduces the metadata leakage and the trust needed to be placed on the server. The BRP feature is achieved using the following steps.

In a call with m breakout rooms, each participant has a preconfigured vector R containing m different ciphertexts, with each ciphertext having a degree of n. One of the ciphertexts, say R_j, where $0 \leq j < m$, contains the value 1 in each of its ciphertext slots, while the other $m-1$ ciphertexts all contain values 0. Internally, the vector R can be viewed as an encrypted matrix with m rows and n columns, with one of the rows containing values 1 while other rows values 0. Here, rows represent different ciphertexts and columns different slots within the ciphertexts. This preconfigured vector R can, for example, be configured by an administrator or encrypted and sent by the user when joining the call. By modifying R, a user can also be switched between breakout rooms.

Upon receiving an encrypted audio sample at the server, the sample is multiplied with each of the ciphertexts in R. Similar to the SSM feature, multiplication corresponds to muting a participant in every other channel except the one in which the user participates. This produces a new vector that now contains the encrypted audio samples on one of its ciphertexts, while other ciphertexts contain values 0. Next, each vector is added with other vectors produced by other users, creating an audio mix for each breakout room. At this point, we have a vector of m ciphertexts, with each ciphertext containing the mixed audio for a corresponding breakout room. The server has now homomorphically mixed the audio without knowing in which breakout room a user is a member.

Next, we perform another ciphertext multiplication to filter the audio and only send the audio of the room where the user participates. Again, we multiply with each of the ciphertexts in the preconfigured vector R for the user, resulting in m ciphertexts where $m-1$ of those contain no audio. This ensures that the user only receives the audio from the breakout room in which the user is a member. The server then sums the m ciphertexts together, subtracts the user's incoming audio to avoid delayed playback, and sends the resulting audio to the user.

Table 1: Example implementations, their configuration, encrypted sizes, and supported features.

HE scheme	Security level	Ring dimension	Encrypted (compressed) size	Mult. depth	Example features		
					Basic	SSM	BRP
BFV	128	2^{11}	33 (31) KB	0	✓	✗	✗
		2^{12}	131 (88) KB	1	✓	✓	✗
		2^{13}	524 (432) KB	2	✓	✓	✓
	256	2^{12}	66 (64) KB	0	✓	✗	✗
		2^{13}	262 (195) KB	1	✓	✓	✗
		2^{14}	1.05 (0.89) MB	2	✓	✓	✓
CKKS	128	2^{12}	66 (66) KB	0	✓	✗	✗
		2^{13}	262 (205) KB	1	✓	✓	✗
		2^{13}	393 (277) KB	2	✓	✓	✓
	256	2^{13}	131 (131) KB	0	✓	✗	✗
		2^{14}	524 (408) KB	1	✓	✓	✗
		2^{14}	787 (556) KB	2	✓	✓	✓

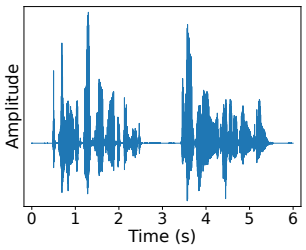


Figure 4: Audio example from the PESQ/POLQA framework.

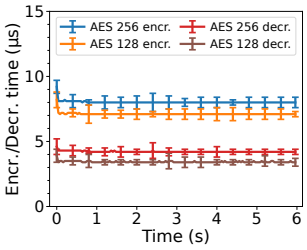


Figure 5: AES encryption and decryption performance over time.

Naturally, using a polynomial degree of n, the application supports up to n rooms ($m \leq n$), with each ciphertext slot holding and mixing the audio per breakout room. As the server cannot distinguish between active breakout rooms with participants, one can also use additional breakout rooms to avoid revealing the number of active rooms, although this comes with significant performance overheads. In total, the BRP feature is achieved using $2 \times m$ multiplications, but each ciphertext is only multiplied twice, resulting in a multiplication depth of 2 (by writing the computation in a tree-like structure, one can multiply k ciphertexts with $\log(k)$ depth). In practice, several variations of the BRP feature can be implemented; e.g., by multiplying with all 1, we can broadcast a user's audio in every breakout room. The BRP feature also supports the SSM feature, as the first multiplication can be used to mute a participant by setting all the ciphertext values to 0.

3.3 Parameter selection and supported features

Choosing the appropriate cryptographic parameters in HE is a non-trivial task. If not done carefully, the encryption scheme can become too weak or unnecessarily slow. Therefore, we model and evaluate various HE settings in the context of audio conferencing and present parameters for practical use. Table 1 shows the configuration parameters used in our framework, the encrypted and compressed ciphertext sizes, and the features supported by our example implementation. We focus on the two classes of HE schemes (BFV and CKKS) and study the use of 128- and 256-bit security.

Standard ring dimension sizes (polynomial degree) of current HE schemes are n^k, where k is an integer $10 \leq k \leq 15$ [3]. Using a larger dimension allows for more computation but significantly increases the computation time of encryption, decryption, and ciphertext operations. The chosen HE parameters must be such that the noise is kept below a certain threshold to ensure correct decryption. For example, to support the basic feature, the noise threshold must allow 1000 ciphertext additions. We find the smallest possible ring dimension to be $n = 2^{11}$ for BFV and $n = 2^{12}$ for CKKS.

Using the same ring dimension, a higher security level results in a smaller noise budget (i.e., fewer allowed HE operations) and (counter-intuitively) smaller ciphertext size. This is because the security level is dependent on the relative size of the noise compared to the ciphertext size, where a smaller ciphertext makes the noise relatively larger. However, since a smaller noise budget results in fewer allowed ciphertext operations that can be performed, we must compensate by increasing the ring dimension, which increases the ciphertext size. Increasing the ring dimension also provides higher security as the complexity and dimensionality of the underlying scheme increase. Therefore, the security level is affected by the combination of the chosen ring dimension and the number of allowed ciphertext operations (often measured using a multiplicative depth, i.e., the maximum number of consecutive multiplication operations allowed on a single ciphertext).

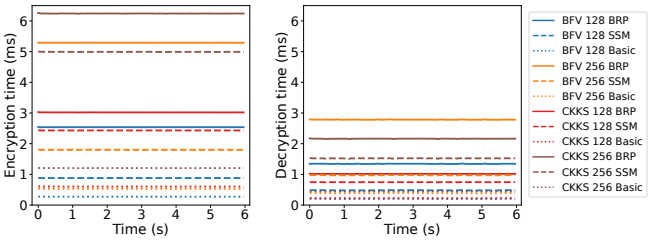
Due to the tradeoffs between security level, ring dimension, ciphertext size, and multiplicative depth, the scheme parameters must be carefully chosen. In general, it is desired to minimize the parameters such that the application features are still supported. In our framework, we use a multiplicative depth of 0, 1, or 2, and the smallest ring dimension required to support the Basic, SSM, and BRP features, respectively. Furthermore, in the underlying scheme for BFV, we select the plaintext modulus to 20 bits. For CKKS, we use a scaling factor of 2^{30}, a first and last prime size of 60 bits, and intermediate prime sizes in the modulus chain of 30 bits.

4 EVALUATION OF HOMOMORPHIC ENCRYPTION SCHEMES

For our evaluation, we run performance tests on a single workstation using Ubuntu 22.04 with an AMD Ryzen 7 PRO 5850U 1.9 GHz CPU and 32 GB RAM. Our implementation uses the Pyfhel [31] and SEAL library [43] for homomorphic encryption, conforming to the homomorphic encryption standard [3]. We also use standardized audio files from the PESQ/POLQA framework [6, 55]. Figure 4 shows the amplitude plot of one such example file sampled using super-wideband (32 kHz). Here, the audio is 6 seconds long and contains two spoken phrases with silence between the phrases. The 2-second silence aims to test the system's performance and the quality of its output with silent sound input.

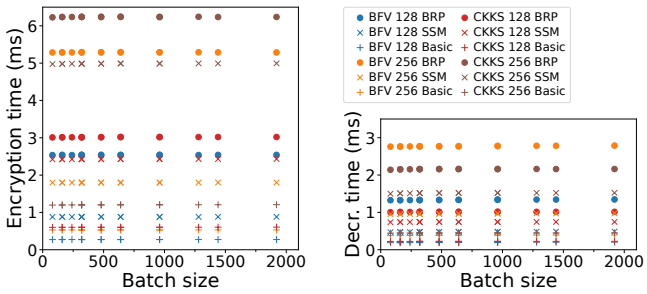
4.1 Performance over time

To test the performance of HE operations over time, we first divide the audio file into time batches of 40 ms. Then, using ten different generated keypairs, we perform the homomorphic operation independently 100 times for each pair. For an audio file of 6 seconds, this gives us 150,000 measurements. For a comparison baseline, Figure 5 first shows the performance of AES encryption and decryption over time. The measurements clearly show that the encryption and decryption time is independent of the audio contents. During the silence period, the measured time is indistinguishable from other



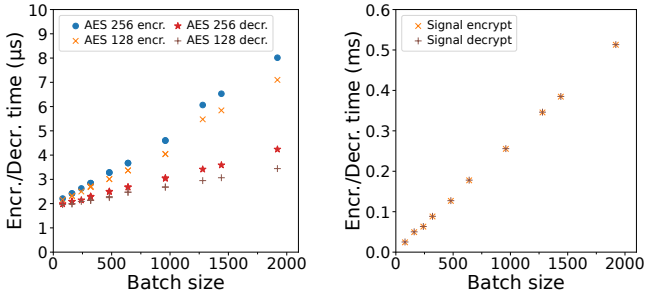
(a) Encryption (b) Decryption

Figure 6: Homomorphic performance over time.



(a) Encryption (b) Decryption

Figure 7: Homomorphic performance per batch size.



(a) Generic AES (b) Signal's AES 128 implementation

Figure 8: AES performance per batch size.

periods with actual sound. This is an important property of audio communication and should not be taken for granted. For example, previous works [4, 9, 27, 28, 66] have shown that various contents can be extracted, despite the traffic being encrypted. By ensuring that the operations and encrypted sizes are constant regardless of the audio contents, many of these attacks may be prevented.

Figure 6 shows the corresponding homomorphic encryption and decryption time for the various configurations. Here, *BFV-128-Basic* denotes the HE scheme BFV using 128-bit security, supporting the basic feature (and thus a multiplicative depth of 0). In general, we see that the encryption and decryption times are in order of milliseconds. The chosen security level highly influences the performance of the different configurations and features supported. Using BFV with the basic feature (*BFV-128-Basic* and *BFV-256-Basic*), the encryption time is consistently below 1 ms. The encryption and decryption times increase with a higher security level and a larger multiplicative depth (more features). We also observe CKKS being slower than BFV, except for the decryption of the configuration supporting the BRP feature (*CKKS-128-BRP* and *CKKS-256-BRP*). This is because, for these two configurations, CKKS achieves a smaller encrypted ciphertext size than the BFV counterparts.

Compared to AES, we observe 2-to-3 orders of magnitude longer encryption times but with small standard deviation (omitted from the figures as the standard deviations are well within 0.1 ms). For example, when comparing AES encryption to BFV using 128-bit security, BFV is roughly 38, 124, and 357 times slower for the three feature variants, respectively. For CKKS, we observe slightly higher numbers: 85, 342, and 424 times slower for the three features. For decryption, both the homomorphic implementations and AES are faster than their encryption counterparts, with the homomorphic operations being of similar magnitudes slower than AES as for the corresponding encryption cases.

For encryption (Figure 6(a)), the large gap between *CKKS-128-BRP* and *CKKS-256-SSM* can be explained by the underlying ring dimension. The three configurations with encryption times of 5 ms or above (*CKKS-256-SSM*, *BFV-256-BRP*, and *CKKS-256-BRP*) all use a dimension of 2^{14}, while the other configurations use a dimension of 2^{13} or less. This again highlights that a larger ring dimension comes with a significant performance penalty, but allows more ciphertext operations and the possibility for more advanced features.

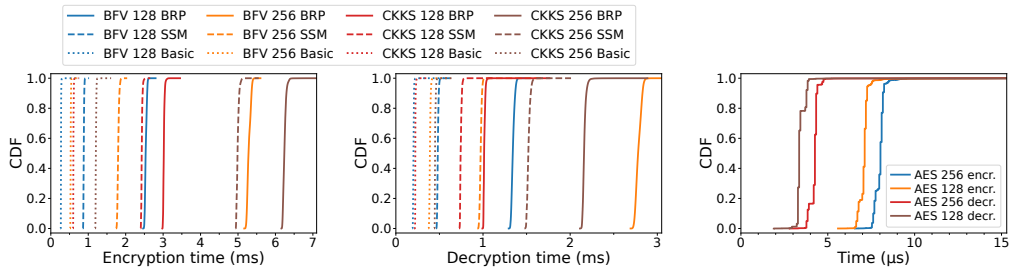
Finally, we observe similar trends for the HE ciphertext operations, with the performance being independent of the audio contents. Figures for homomorphic addition, subtraction, and multiplication are included in Appendix A.

4.2 Effects of batch time and sample rate

To measure the impact of the selected audio sample rate r and batch time Δ, we run experiments using batch times $\Delta \in \{10, 20, 30, 40\}$ ms together with the sample rates $r \in \{8, 16, 32, 48\}$ kHz. Figure 7 shows a scatter plot for encryption and decryption time based on the batch size. As the batch size is defined as the number of audio samples in a ciphertext (i.e., a combination of both Δ and r), we include 16 measurements for each configuration. We note that several combinations of Δ and r can result in the same batch size (e.g., Δ=40 ms, r=16 kHz, and Δ=20 ms, r=32 kHz both result in a batch size of 640 samples). The high overlap (barely visible in Figure 7) of configurations with the same batch size again highlights that the encryption and decryption times are independent of the audio contents. The performance of the homomorphic encryption and decryption over various batch sizes (Figure 7) are consistent with the performance over time (Figure 6). Here, we again observe the general trend that operation times increase with higher security levels and the number of features supported.

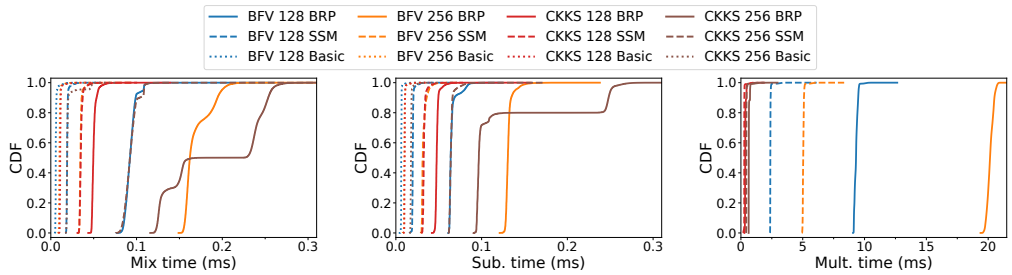
The straight horizontal line for each configuration shows that the performance is independent of the selected batch size. To maximize the performance, it is therefore desired to select the largest batch time Δ and sample rate r possible, conditioned on the constraint that the resulting batch size must be less than the ring dimension for BFV, or less than half of the ring dimension for CKKS (i.e., the maximum number of slots per ciphertext for the two schemes).

The above behavior differs from that observed for AES. Figure 8 shows the performance of AES encryption and decryption for (a) a generic AES library in C++ [21] and (b) AES 128 as used in the



(a) Homomorphic encrypt (b) Homomorphic decrypt (c) AES encrypt/decrypt

Figure 9: Client-side CDFs for homomorphic operations and AES.



(a) HE mixing (addition) (b) HE subtraction (c) HE multiplication

Figure 10: Server-side CDFs for HE ciphertext operations.

Signal Calling Service [60] for end-to-end group calls [59]. Here, we see that the encryption and decryption times of these AES implementations scale linearly with the batch size, with a larger batch size resulting in relatively larger processing times. This is because, for AES, more data results in an increased number of blocks that have to be encrypted and decrypted. In contrast, a single block is used for HE schemes, and we get the same performance regardless of how many slots are filled in the ciphertext (i.e., batch size).

In contrast to the generic AES library (Figure 8(a)), for which the decryption operations are slightly faster than the encryption counterparts, for the Signal application (Figure 8(b)), we observe similar performance for encryption and decryption. We also note significant differences in the relative encryption and description speeds of the implementations, with Signal being much slower than the generic library (due to implementation differences) but substantially faster than the homomorphic encryption and decryption (Figure 7). Yet, and perhaps most importantly, all these times (including with our out-of-the-box, non-optimized HE implementation) are smaller than both the typical end-to-end round-trip time between communicating parties and the batch granularity itself.

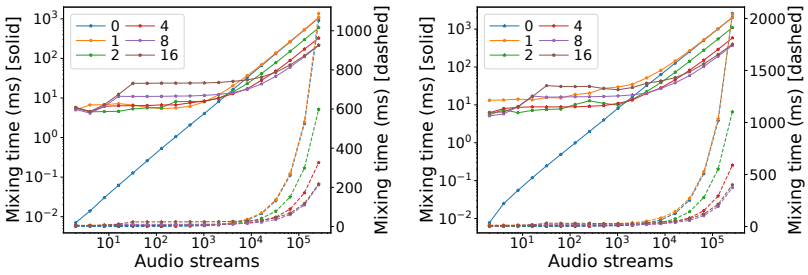
Finally, for HE ciphertext operations, we observe similar trends where the performance is independent of the batch size. The scatter plots for addition, subtraction, and multiplication per batch size are shown in Appendix B.

4.3 Tail performance analysis

In real-time communication systems, performance outliers can cause the system to lag behind. In audio conferencing, packets may instead be skipped if delayed for too long. To illustrate the fraction of packets that belong to this tail, Figure 9 shows the cumulative distribution functions (CDFs) for operations on the client-side (i.e., encryption and decryption), and Figure 10 shows the CDFs for server-side HE ciphertext operations.

Encryption and decryption: For homomorphic encryption and decryption (Figures 9(a) and 9(b)), in addition to observing performance ordered based on security level and supported features, and decryption (again) being faster than encryption, we clearly see a highly concentrated distribution. Here, the performance variations are low and the tail of higher-delay packets (if any) is minimal with no packet delayed by more than 1 ms. Compared to AES (Figure 9(c)), we again observe (1) low performance variations, (2) sessions with higher security levels resulting in longer processing times, and (3) homomorphic encryption and decryption being roughly 30-400 times slower than their AES counterparts.

Mixing and subtraction: For audio mixing and subtraction (Figures 10(a) and 10(b)), we observe a similar trend where a higher security level and more features lead to longer processing times, and that these ciphertext operations typically take less than 0.3 ms. Here, the configurations have very similar performance (e.g., *BFV-256-SSM* and *CKKS-128-SSM*, or *BFV-128-BRP* and *CKKS-256-SSM*), despite being based on two fundamentally different HE schemes. This is due to the underlying ring dimension and the requirements to support these configurations resulting in similar ciphertext sizes. This suggests that at those speeds (< 0.3 ms), homomorphic addition and subtraction of ciphertext are more dependent on ciphertext sizes and the system capability, rather than the security level or HE scheme. Furthermore, for *CKKS-256-BRP*, there is a clear plateau where some packets take longer to process. These are due to performance hiccups and saturations, causing every other packet (or every fifth packet, as in the case of subtraction) to take roughly double the time to process. The fluctuation of processing is further illustrated in the performance over time plots in Appendix A,



(a) BFV scheme (b) CKKS scheme

Figure 11: Parallel audio mixing time when delegated to n extra processes. Mixing times shown on both logarithmic (solid) and linear (dashed) scale.

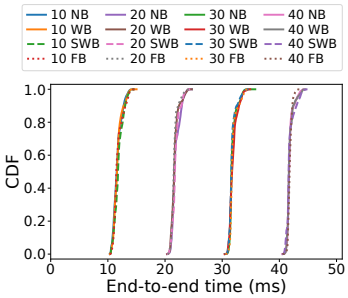


Figure 12: End-to-end performance with different configs.

where it is shown that the fluctuations occur regardless of the audio contents. Finally, we note that removing part of an audio from the mixed audio, as in the case of subtraction, is slightly faster than mixing that audio for the first time. While the differences are minimal (typically less than 0.05 ms), it can be explained by the memory management and the fact that processing the same encrypted audio for the second time is slightly faster due to it being cached.

Multiplication: The general trend thus far for encryption, mixing, subtraction, and decryption has shown the CKKS scheme to be slower than the BFV scheme. However, this is not the case when looking at the ciphertext multiplication time (Figure 10(c)). For CKKS, while the multiplication times are still larger than addition and subtraction, they clearly outperform the BFV counterparts with processing times of typically less than 1 ms. Again, for both CKKS and BFV, we observe longer processing times when using implementations with a higher security level and more supported features (and thus also the ciphertext size), with processing times of BFV ranging from nearly 2.5 ms to roughly 20 ms.

Parallelization: Even though two ciphertexts can be processed in parallel, this typically comes with large overheads. Figure 11 shows the mixing times with different number of parallel server processes for the *BFV-128-Basic* and *CKKS-128-Basic* configurations, respectively. Due to the low mixing time (< 0.1 ms) and relatively large ciphertext sizes (33-66 KB), it is clear that the interprocess communication results in significant overhead and that it may be more beneficial to perform operations in a single process instead of delegating to several parallel processes. For BFV (Figure 11(a)), we see that parallel mixing only benefits a call with more than 1300 participants. For CKKS (Figure 11(b)), this number is slightly lower due to the larger ciphertext mixing time. Therefore, to best scale audio conferencing, it is important to find a good tradeoff between the number of participants and parallel processes.

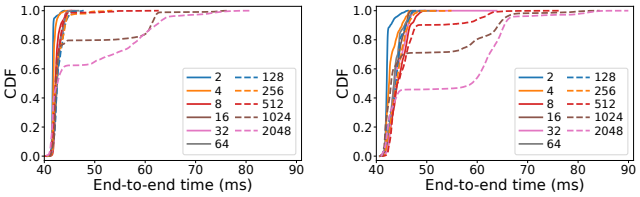
5 END-TO-END PERFORMANCE

5.1 Effects of batch time, sample rate, and number of participants

We next study the end-to-end performance using a client-server architecture. To minimize the effects of network variability, we run tests on a local network with low latency (< 3 ms). We also deploy and evaluate our framework on a research-based cloud service (WARA-Ops) using different machines for each client and server, and perform clock synchronization using NTP [45] to get accurate end-to-end delay measurements. While we note higher fluctuations due to shared resources in the cloud environment, we observe similar results as running our tests locally. To allow for reproducibility and ease of comparison, we report the local tests with low latency and performance variations. Using the *BFV-128-Basic* configuration, we measure the end-to-end delay in a call with three participants. Two participants are actively speaking by looping the PESQ/POLQA standardized files audio [6, 55], while the third is listening and measuring the total delay using timestamps in packet metadata. As clients operate asynchronously, we consider the delay to be the earliest time of either all the mixed ciphertexts, or the beginning of the mix iteration at the server.

Figure 12 shows the CDF of the end-to-end delay for various batch times and sample rates narrowband (NB), wideband (WB), super-wideband (SWB), and fullband (FB). In contrast to before, where neither the batch time nor the sample rate affected the encryption and decryption times, here the end-to-end delay reduces with smaller batch times, while the sample rate does not have any impact. However, it should also be noted that choosing a smaller batch time comes at the cost of more processing and bandwidth usage at both the server and the clients. For example, a batch time of 20 ms results in twice as much processing and bandwidth compared to with a batch time of 40 ms, as clients encrypt, sends, and decrypt audio batches every batch time. To reduce the computational processing and the bandwidth consumed, it may therefore be desired to use a larger batch time in the homomorphic context than the otherwise typically used batch times of 20 ms [57]. We also note that the bandwidth consumed per user is independent of the number of call participants, and while larger ciphertexts come with big overheads, the bandwidths they consume are still well within the bounds of realistic modern networks. Furthermore, we see that the end-to-end delay is consistently slightly larger than the chosen batch time. This can be explained by the server mixing process, where the server waits up to batch time to mix packets. As previously shown, the processing times of encryption, mixing, and decryption are relatively low (less than 1 ms together for this configuration). Therefore, in this case, the major contribution to the end-to-end delay is the waiting time for mixing at the server.

Using the basic configuration, we also measure the performance as the system scales. Figure 13 shows the end-to-end time with different number of participants using the BFV and CKKS schemes.



(a) BFV scheme (b) CKKS scheme

Figure 13: End-to-end performance with different number of call participants. Note the horizontal axis start value.

Here, we observe a clear trend with increasing end-to-end delay as the number of participants increases. This is expected as the server processing increases with the number of participants. As the audio subtraction occurs after the batch iteration (as previously shown in Figure 2), the time consumed by this process is directly added to the end-to-end delay. For 1024 and 2048 participants, we clearly see some packets being delayed to roughly 60 ms and outliers being delayed up to roughly 80ms (double the batch time). This occurs when many packets arrive to the server near the end of the batch iteration, resulting in some packets not being processed in time in the current iteration, and instead are delayed to the next batch.

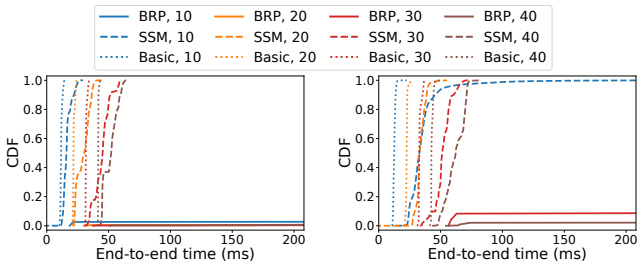
We note that while only a small subset of all participants are actively speaking, the server still continuously receives audio from all participants. For example, in a call with 2048 participants, the server mixes 2048 audio batches every iteration of batch time. This is to preserve speaker anonymity, i.e., the server cannot identify active speakers. In practice, a variation can be done, where only a subset of the participants continuously send audio, providing speaker anonymity among a subset of users. This substantially reduces the computational performance required at the server, allowing the server to support many more participants with minimal additional computational overhead.

5.2 Effects of configurations and features

To capture the end-to-end impact of the different processing requirements associated with different configurations, we next study the performance using different configurations and batch times. Figures 14 and 15 show the end-to-end delay under various configurations when using BFV and CKKS, respectively, in a call with 128 participants. For the basic feature, the end-to-end delay is slightly larger than the selected batch time. Furthermore, due to increased processing requirements, 256-bit security generally results in a slightly larger end-to-end delay than 128-bit security. However, the delay remains well within the acceptable threshold of 150 ms [47].

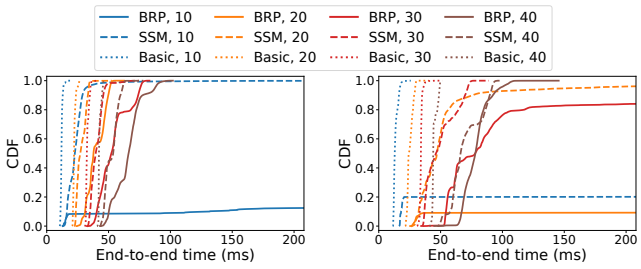
For the SSM feature, the use of BFV-128, BFV-256, and CKKS-128 results in an increased delay of 10-20 ms due to the extra processing required for ciphertext multiplications. For CKKS-256, we see that the selection of a too low batch time can make the system unstable. When using a batch time of 10 ms, the processing queue grows faster than the server can handle. The system is stable for batch times of 20 ms and larger, although we observe a higher increase in end-to-end delay compared to BFV or CKKS-128.

For the BRP feature, we again see instability and big variations in the end-to-end delays. Due to the many multiplications required



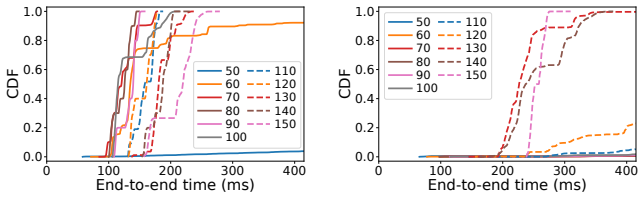
(a) BFV 128-bit security (b) BFV 256-bit security

Figure 14: BFV performance using different configs.



(a) CKKS 128-bit security (b) CKKS 256-bit security

Figure 15: CKKS performance with different configs.



(a) BFV 128-bit security (b) BFV 256-bit security

Figure 16: BFV performance with BRP feature using different batch times.

to implement the feature, BFV with the BRP feature completely overwhelms the server with a batch time of 40 ms or less. For CKKS-128, this is observed only for a batch time of 10 ms, while for CKKS-256, the unstableness occurs with a batch time of 30 ms or less. Therefore, the lowest usable batch time with CKKS and the BRP feature is 20 ms for 128-bit security, and 40 ms for 256-bit security. With CKKS, using a larger batch time than 40 ms only results in a larger end-to-end delay.

To find the lowest batch time that supports the BFV scheme with the BRP feature, we repeat the experiments using batch times up to 150 ms. Figure 16 shows the results from these experiments. For 128-bit security (Figure 16(a)), we observe the overall lowest end-to-end delay using a batch time of 80 ms, i.e., below the acceptable threshold of 150 ms. For 256-bit security (Figure 16(b)), while a 130 ms batch time achieves the lowest end-to-end delay, the delay is above 200 ms. Therefore, we consider BFV with 256-bit security unsuitable for practical use with the BRP feature. Instead, the BRP feature with 256-bit security is restricted to the CKKS scheme.

6 AUDIO QUALITY

VoIP performance is often evaluated using Quality of Service (QoS) [64] and Quality of Experience (QoE) [33] metrics. Having studied

Table 2: MOS rating* based on HE scheme, sample rate, and simultaneously active speakers.

Sample rate		8 kHz					16 kHz					32 kHz					48 kHz				
Speakers		2	3	4	5	6	2	3	4	5	6	2	3	4	5	6	2	3	4	5	6
PESQ	BFV	4.00	4.10	3.90	3.90	3.80	4.54	4.54	4.49	4.49	4.54	N/A									
	CKKS	4.00	4.10	3.90	3.90	3.80	4.54	4.54	4.49	4.49	4.54										
POLQA	BFV	4.20	4.03	3.95	3.87	3.85	4.32	4.30	4.35	4.38	4.41	4.75									
	CKKS	4.20	4.32	4.01	3.85	3.85	4.31	4.29	4.35	4.38	4.41										

*Poor: 1 - 2, Fair: 2 - 3, Good: 3 - 4, Excellent: 4 - 5

the QoS by measuring the end-to-end delay, we next study the QoE impact. For this analysis, we qualitatively sanity checked the audio quality ourselves and then rely on well-established QoE metrics to provide quantitative comparisons of the relative perceived audio quality of the mixed audio when using different implementations.

To quantitatively and objectively study the QoE, we use the two objective algorithms PESQ [55] and POLQA [6], recommended by the International Telecommunication Union (ITU) and widely used in research [5, 14, 30, 58]. PESQ is standardized as *ITU-T Rec. P.862*, and POLQA is a newer version standardized as *ITU-T Rec. P.863*. They are both developed to, as closely as possible, resemble the Mean Opinion Score (MOS) from subjective testing, giving a MOS value between 1 and 5. PESQ supports the evaluation of audio sampled using narrowband and wideband, while POLQA extends this and, in addition, supports super-wideband and fullband. The main difference between the PESQ and POLQA algorithms is that POLQA supports additional sample rates and recognizes "time-warpings" in modern codecs (which PESQ would evaluate as errors). In general, POLQA performs better than PESQ in audio quality measurements due to its new alignment methods in combination with its new advanced perceptual model [6]. Both PESQ and POLQA use intrusive quality assessment with a reference signal to assess the sound quality [61]. The output of the quality assessment is a MOS value between 1 and 5, where 5 represents excellent quality. We note that a reference signal compared to itself will not always produce a perfect score of 5 as the scoring process assumes the reference signal has a balanced timbre without noise or reverberation, and judges any deviations from this as a degradation [32]. This is to better approximate the results when compared to subjective testing.

Table 2 shows the MOS values using PESQ and POLQA for different HE schemes, sample rates, and the number of simultaneously active speakers in a call. Here, we report the MOS values for up to 6 active speakers, as more simultaneous speakers often make the audio unintelligible. (The number of listeners does not directly impact the mixed audio quality as their ciphertexts only include zeros.) Even though CKKS gives an approximate value when decrypted, we observe little-to-no quality difference between the two schemes. For PESQ, a sample rate of 8 kHz and 16 kHz results in an average score of 3.94 and 4.52, respectively. Comparing with the reference signal itself (which had a highest MOS value of 4.5 and 4.64), this corresponds to a quality degradation of 12.4% and 2.6%, respectively. For POLQA, we observe an average score of 4.01 and 4.35 for a sample rate of 8 kHz and 16 kHz, resulting in a quality degradation of 15.7% and 8.4%, respectively. For 32 kHz and 48 kHz, regardless of the scheme or number of simultaneously active speakers, we can achieve a score of 4.75. In this case, the maximum score is 4.75, i.e., we receive no quality degradation. This shows that as the sample rate increases, we see a clear increase in audio quality.

Furthermore, we note that the number of simultaneously active speakers primarily impacts the lower sample rates, where we observe a clear quality degradation with more speakers only when using a sample rate of 8 kHz. This is because of the limited number of samples in narrowband audio, causing a more significant disturbance in the audio when mixing. In contrast, audio with higher sample rates has more samples, which results in the disturbance being distributed more among multiple samples. Therefore, the quality remains relatively stable for other sample rates with an increasing number of simultaneously active speakers. Overall, the audio quality results show that if sampling with 16 kHz or greater, homomorphic encryption can be used in an audio conferencing context with excellent quality.

7 RELATED WORK

Audio privacy: Previous works have considered audio communication using additive HE. Rohloff et al. [56] modified the voice chat application Mumble [46] and its audio server to include end-to-end HE. Using additive HE, they reduce the circuit depth of the mixing operations, demonstrating the first practical use of HE in an audio context. Kamal et al. [34] study the use of HE from a hardware perspective to mix encrypted audio. Using hardware-assisted FPGA, they boost the performance of homomorphic operations. In contrast to our work, these works take a more theoretical approach and study only the use of additive HE. In this paper, we instead take a more experimental approach and compare HE schemes to identify various bottlenecks. Using multiplication operations, we also present advanced features and study the perceived audio quality.

Other works have focused on the metadata leakage of audio communication [1, 38–40]. For example, Ahmad et al. [1] present Addra, a scalable system for hiding audio metadata over fully untrusted networks. While these works also improve the privacy of online communication, they focus on the metadata leakage and do not use HE to protect the audio contents.

Audio quality and mixing: Hu and Loizou [30] evaluate different objective quality measurement methods for forecasting noisy speech enhanced by noise suppression algorithms, finding PESQ to yield the most accurate results compared to subjective tests. Beerends et al. [5] study the performance of POLQA and compare the MOS values to the corresponding subjective data, finding that POLQA can predict the score with high accuracy, highlighting its benefits for assessing speech quality. Other works have focused more on the aspects of audio mixing and the resulting perceived quality. Chandra et al. [14] study different audio mixing algorithms for multi-party conferencing and propose an algorithm based on the IIR filter. They present objective methods for evaluating algorithm performance and measure the audio quality using PESQ, distortion

and power loss measures, and the output signal-to-noise ratio (SNR). Sethi et al. [58] propose an audio mixing algorithm for VoIP conferencing applications using voice enhancement features such as noise reduction, level control, and voice activation detection. They evaluate the new algorithm using PESQ and perceived audio level (PLL), showing quality benefits compared to previous algorithms. The proposed algorithms by Chandra et al. [14] and Sethi et al. [58] are relatively complex in a homomorphic context. Therefore, in this work, we present and use a simpler mixing approach to reduce the homomorphic complexity.

VoIP performance: Ngamwongwattana [49] study the effects of packetization in VoIP and show that while small packets are generally more desirable to minimize the end-to-end-delay, they increase the network load and can cause queueing delays in case of congestion. On the other hand, as larger packets increase the wait time needed to capture the audio, the author concludes that the best strategy to minimize end-to-end delay is to use adaptive packetization and change packet sizes depending on the network load. In this paper, while the goal is to minimize the end-to-end delay, the packet sizes cannot be easily adapted due to the ciphertext sizes being dependent on the homomorphic parameters and their underlying security properties. Ngamwongwattana and Thompson [50] present Sync & Sense, a tool used for measuring VoIP end-to-end delay. They show that high accuracy can be achieved without clock synchronization. While such a tool can be useful in our work, clients and servers instead operate asynchronously to avoid additional overhead. When synchronization is needed to measure the end-to-end delay, we use the more traditional approach with NTP servers [45]. Tsetse et al. [63] study the performance of VoIP in 802.11ac networks under different network conditions and security configurations, observing an increase in latency and decrease in MOS values when adding security. While our study is similar to theirs, in this work, we focus on the performance aspects and MOS values when using homomorphic encryption.

HE performance: Others focus on the performance of HE [2, 29, 53]. Al Badawi et al. [2] study the performance of Residue Number System (RNS) in the BFV scheme, focusing on the Halevi-Polyakov-Shoup (HPS) variant in SEAL [43] and Bajard-Eynard-Hasan-Zucca (BEHZ) in Palisade [51]. They show significant improvements in homomorphic multiplication time compared to previous works, and that multi-threaded CPU only provides small performance increases, which corresponds well to the insights provided in this paper (e.g., as previously shown in Figure 11). Reis et al. [53] study the use of near-memory processing (NMP) and computing-in-memory (CiM) paradigms to improve the HE performance in the BFV scheme. They show performance improvements up to 9.1 times compared to regular CPU implementations. While related, these works focus on the HE performance of other aspects. In this work, we instead focus on the VoIP and audio communication context.

8 CONCLUSIONS

In this paper, we have presented the design of a secure, scalable audio conferencing application using end-to-end homomorphic encryption. Using an untrusted honest-but-curious server, we demonstrate the practical use of mixing encrypted audio and the feasibility of offloading heavy computations from resource-constrained end devices to more powerful cloud servers. Our results show the tradeoffs between homomorphic encryption schemes, their security configurations, and the chosen audio parameters. For example, we show that while the audio contents, sample rate, and batch time do not affect the homomorphic operations, a lower batch time results in a lower end-to-end delay at the cost of additional overhead. As the ciphertext sizes are fixed based on the encryption scheme parameters, the best strategy is shown to be selecting a batch size as large as possible, and using the remaining slots for other data, like forward error correction.

Using homomorphic multiplications, we have demonstrated the use of the server-side-mute and breakout rooms features, where the middle server cannot extract sensitive user-specific metadata, further increasing users' privacy. While using the BFV scheme is most beneficial for encryption, decryption, and additive mixing, we show the performance benefits of using the CKKS scheme with ciphertext multiplications. By demonstrating the tradeoffs, we show how different schemes and configurations may be used to achieve the lowest possible end-to-end delay while supporting the desired application and features.

Finally, while homomorphic encryption results in longer encryption/decryption times compared to regular encryption, we show that the performance is well within the bounds considered acceptable in voice applications, and that it comes with privacy and scalability benefits. Besides studying QoS and performance aspects, we have also objectively evaluated the QoE by analyzing the perceived audio quality using PESQ and POLQA, showing that excellent audio quality can be achieved using a sample rate of 16 kHz or greater. While production implementations would include more features and functionalities than our basic implementation, our results clearly demonstrate that the approach is both practical and scalable. The insights we provide also help service providers design future communication applications and better understand the tradeoffs of selecting homomorphic encryption schemes and parameters, ultimately achieving the goal of practical and scalable end-to-end encrypted audio conferencing.

ACKNOWLEDGMENTS

The authors are thankful to Ebba Lindström, Herman Nordin, and Christian Vestlund for their early help with this work. This work was partially supported by the Wallenberg AI, Autonomous Systems and Software Program (WASP) funded by the Knut and Alice Wallenberg Foundation. Experimentation performed in this work was partially supported by the WASP WARA-Ops Research Arena.

REFERENCES

[1] Ishtiyaque Ahmad, Yuntian Yang, Divyakant Agrawal, Amr El Abbadi, and Trinabh Gupta. 2021. Addra: Metadata-private voice communication over fully untrusted infrastructure. In *Proc. USENIX OSDI.*

[2] Ahmad Al Badawi, Yuriy Polyakov, Khin Mi Mi Aung, Bharadwaj Veeravalli, and Kurt Rohloff. 2019. Implementation and performance evaluation of RNS variants of the BFV homomorphic encryption scheme. *IEEE Trans. on Emerging Topics in Computing* (2019).

[3] Martin Albrecht, Melissa Chase, Hao Chen, et al. 2018. *Homomorphic Encryption Security Standard.* Technical Report. HomomorphicEncryption.org.

[4] J.S. Atkinson, M. Rio, J.E. Mitchell, and G. Matich. 2014. Your WiFi is leaking: Ignoring encryption, using histograms to remotely detect Skype traffic. In *Proc. IEEE Military Communications Conference (MILCOM).*

[5] John G Beerends. 2019. Subjective and objective assessment of full bandwidth speech quality. *IEEE/ACM Trans. Audio, Speech, and Language Processing* (2019).

[6] John G Beerends, Christian Schmidmer, Jens Berger, et al. 2013. Perceptual Objective Listening Quality Assessment (POLQA), The Third Generation ITU-T Standard for End-to-End Speech Quality Measurement Part I—Temporal Alignment. *Journal of the Audio Engineering Society* (2013).

[7] Marcelo Blatt, Alexander Gusev, Yuriy Polyakov, and Shafi Goldwasser. 2020. Secure large-scale genome-wide association studies using homomorphic encryption. In *Proc. National Academy of Sciences.*

[8] Jean-Chrysostome Bolot, Sacha Fosse-Parisis, and Don Towsley. 1999. Adaptive FEC-based error control for Internet telephony. In *Proc. IEEE INFOCOM.*

[9] Dario Bonfiglio, Marco Mellia, Michela Meo, Dario Rossi, and Paolo Tofanelli. 2007. Revealing Skype traffic: When randomness plays with you. In *Proc. ACM SIGCOMM.*

[10] Zvika Brakerski. 2012. Fully homomorphic encryption without modulus switching from classical GapSVP. In *Proc. CRYPTO.*

[11] Zvika Brakerski, Craig Gentry, and Vinod Vaikuntanathan. 2014. (Leveled) Fully Homomorphic Encryption without Bootstrapping. *ACM TOCT* (2014).

[12] Sergiu Carpov, Nicolas Gama, Mariya Georgieva, and Juan Ramon Troncoso-Pastoriza. 2020. Privacy-preserving semi-parallel logistic regression training with fully homomorphic encryption. In *Proc. iDASH Privacy and Security Workshop.*

[13] Hervé Chabanne, Amaury de Wargny, Jonathan Milgram, Constance Morel, and Emmanuel Prouff. 2017. Privacy-Preserving Classification on Deep Neural Network.

[14] Sekhar P Chandra, Kumar Mani Senthil, and Manikya Prasad P Bala. 2009. Audio mixer for multi-party conferencing in VoIP. In *Proc. IEEE IMSAA.*

[15] Hao Chen, Wei Dai, Miran Kim, and Yongsoo Song. 2019. Efficient multi-key homomorphic encryption with packed ciphertexts with application to oblivious neural network inference. In *Proc. ACM CCS.*

[16] Hao Chen, Zhicong Huang, Kim Laine, and Peter Rindal. 2018. Labeled PSI from fully homomorphic encryption with malicious security. In *Proc. ACM CCS.*

[17] Hao Chen, Kim Laine, and Peter Rindal. 2017. Fast private set intersection from homomorphic encryption. In *Proc. ACM CCS.*

[18] Jung Hee Cheon, Andrey Kim, Miran Kim, and Yongsoo Song. 2017. Homomorphic encryption for arithmetic of approximate numbers. In *Proc. ASIACRYPT.*

[19] Ilaria Chillotti, Nicolas Gama, Mariya Georgieva, and Malika Izabachene. 2016. Faster fully homomorphic encryption: Bootstrapping in less than 0.1 seconds. In *Proc. ASIACRYPT.*

[20] Ilaria Chillotti, Marc Joye, and Pascal Paillier. 2021. Programmable bootstrapping enables efficient homomorphic inference of deep neural networks. In *Proc. Symposium on Cyber Security, Cryptology, and Machine Learning (CSCML).*

[21] Crypto++. 2023. Crypto++ Library 8.8. https://cryptopp.com/.

[22] Roshan Dathathri, Blagovesta Kostova, Olli Saarikivi, Wei Dai, Kim Laine, and Madan Musuvathi. 2020. EVA: An encrypted vector arithmetic language and compiler for efficient homomorphic computation. In *Proc. ACM Programming Language Design and Implementation (PLDI).*

[23] Pawel Drozdowski, Nicolas Buchmann, Christian Rathgeb, Marian Margraf, and Christoph Busch. 2019. On the application of homomorphic encryption to face identification. In *Proc. Biometrics Special Interest Group (BIOSIG).*

[24] Léo Ducas and Daniele Micciancio. 2015. FHEW: Bootstrapping homomorphic encryption in less than a second. In *Proc. EUROCRYPT.*

[25] Junfeng Fan and Frederik Vercauteren. 2012. Somewhat practical fully homomorphic encryption.

[26] Craig Gentry. 2009. Fully homomorphic encryption using ideal lattices. In *Proc. ACM Symposium on Theory of Computing (STOC).*

[27] David Hasselquist, Martin Lindblom, and Niklas Carlsson. 2022. Lightweight Fingerprint Attack and Encrypted Traffic Analysis on News Articles. In *Proc. IFIP Networking.*

[28] David Hasselquist, Christian Vestlund, Niklas Johansson, and Niklas Carlsson. 2022. Twitch Chat Fingerprinting. In *Proc. IFIP Network Traffic Measurement and Analysis Conference (TMA).*

[29] David Hasselquist, Jacob Wahlman, and Niklas Carlsson. 2023. PET-Exchange: A Privacy Enhanced Trading Exchange using Homomorphic Encryption. In *Proc. Privacy, Security, and Trust (PST).*

[30] Yi Hu and Philipos Loizou. 2008. Evaluation of Objective Quality Measures for Speech Enhancement. *IEEE Trans. Audio, Speech, and Language Processing* (2008).

[31] Alberto Ibarrondo and Alexander Viand. 2021. Pyfhel: Python for homomorphic encryption libraries. In *Proc. ACM CCS Workshop on Encrypted Computing & Applied Homomorphic Cryptography (WAHC).*

[32] International Telecommunication Union. 2019. *Application guide for Recommendation ITU-T P.863.* Technical Report. ITU.

[33] Sofiene Jelassi, Gerardo Rubino, Hugh Melvin, Habib Youssef, and Guy Pujolle. 2012. Quality of experience of VoIP service: A survey of assessment approaches and open issues. *IEEE Communications Surveys & Tutorials* (2012).

[34] Ahmed Kamal, Hisham Dahshan, and Ashraf Diaa. 2018. Hardware Assisted Homomorphic Encryption in a Real Time VOIP Conference Application. In *Proc. International Conference on Computer Engineering and Systems (ICCES).*

[35] Andrey Kim, Yongsoo Song, Miran Kim, Keewoo Lee, and Jung Hee Cheon. 2018. Logistic regression model training based on the approximate homomorphic encryption. In *Proc. iDASH Privacy and Security Workshop.*

[36] Miran Kim, Arif Ozgun Harmanci, Jean-Philippe Bossuat, et al. 2021. Ultrafast homomorphic encryption models enable secure outsourcing of genotype imputation. *Cell systems* (2021).

[37] Kristin Lauter, Sreekanth Kannepalli, Kim Laine, et al. 2021. Password Monitor: Safeguarding passwords in Microsoft Edge. https://www.microsoft.com/en-us/research/blog/password-monitor-safeguarding-passwords-in-microsoft-edge/

[38] David Lazar, Yossi Gilad, and Nickolai Zeldovich. 2019. Yodel: strong metadata security for voice calls. In *Proc. ACM Symposium on Operating Systems Principles.*

[39] David Lazar and Nickolai Zeldovich. 2016. Alpenhorn: Bootstrapping secure communication without leaking metadata. In *Proc. USENIX OSDI.*

[40] Stevens Le Blond, David Choffnes, William Caldwell, Peter Druschel, and Nicholas Merritt. 2015. Herd: A scalable, traffic analysis resistant anonymity network for VoIP systems. In *Proc. SIGCOMM.*

[41] Vadim Lyubashevsky, Chris Peikert, and Oded Regev. 2010. On ideal lattices and learning with errors over rings. In *Proc. EUROCRYPT.*

[42] Daniele Micciancio and Oded Regev. 2009. Lattice-based Cryptography. In *Post-Quantum Cryptography.*

[43] Microsoft. 2021. Microsoft SEAL. https://github.com/Microsoft/SEAL.

[44] Microsoft. 2022. Use end-to-end encryption for one-to-one Microsoft Teams calls. https://learn.microsoft.com/en-us/microsoftteams/teams-end-to-end-encryption

[45] David L. Mills. 1991. Internet time synchronization: the network time protocol. *IEEE Trans. on Communications* (1991).

[46] Mumble. 2022. Mumble. https://www.mumble.info/

[47] Songun Na and Seungwha Yoo. 2002. Allowable propagation delay for VoIP calls of acceptable quality. In *Proc. Advanced Internet Services and Applications (AISA).*

[48] Michael Naehrig, Kristin Lauter, and Vinod Vaikuntanathan. 2011. Can homomorphic encryption be practical?. In *Proc. ACM CCSW.*

[49] Boonchai Ngamwongwattana. 2008. Effect of packetization on VoIP performance. In *Proc. Electrical Engineering/Electronics, Computer, Telecommunications and Information Technology.*

[50] Boonchai Ngamwongwattana and Richard Thompson. 2010. Sync & sense: VoIP measurement methodology for assessing one-way delay without clock synchronization. *IEEE Trans. on Instrumentation and Measurement* (2010).

[51] Palisade 2022. Palisade. https://palisade-crypto.org/

[52] Oded Regev. 2009. On lattices, learning with errors, random linear codes, and cryptography. In *Proc. ACM Symposium on Theory of Computing (STOC).*

[53] Dayane Reis, Jonathan Takeshita, Taeho Jung, et al. 2020. Computing-in-memory for performance and energy-efficient homomorphic encryption. *IEEE Trans. on Very Large Scale Integration (VLSI) Systems* (2020).

[54] Ronald L Rivest, Len Adleman, Michael L Dertouzos, et al. 1978. On data banks and privacy homomorphisms. *Foundations of Secure Computation* (1978).

[55] Antony W Rix, John G Beerends, Michael P Hollier, and Andries P Hekstra. 2001. Perceptual evaluation of speech quality (PESQ) - A new method for speech quality assessment of telephone networks and codecs. In *Proc. IEEE ICASSP.*

[56] Kurt Rohloff, David Bruce Cousins, and Daniel Sumorok. 2016. Scalable, practical VoIP teleconferencing with end-to-end homomorphic encryption. *IEEE Trans. on Information Forensics and Security* (2016).

[57] Henning Schulzrinne and Stephen Casner. 2003. *RTP profile for audio and video conferences with minimal control.* Technical Report. RFC 3551.

[58] Sameer Sethi, Prabhjot Kaur, and Swaran Ahuja. 2014. A new weighted audio mixing algorithm for a multipoint processor in a VoIP conferencing system. In *Proc. Advances in Computing, Communications and Informatics (ICACCI).*

[59] Signal. 2021. How to build large-scale end-to-end encrypted group video calls. https://signal.org/blog/how-to-build-encrypted-group-calls/

[60] Signal. 2023. https://github.com/signalapp/Signal-Calling-Service.

[61] O. Slavata and J. Holub. 2014. Evaluation of objective speech transmission quality measurements in packet-based networks. *Computer Standards & Interfaces* (2014).

[62] Nigel P. Smart and Frederik Vercauteren. 2014. Fully homomorphic SIMD operations. *Designs, codes and cryptography* (2014).

[63] Anthony Tsetse, Alexandre Ergatian, Brandon Sershon, and Samuel Tweneboah-Kodua. 2019. VoIP QoS Performance Analysis in 802.11 ac Networks. In *Proc. Computational Intelligence and Communication Networks (CICN).*

[64] Tadeus Uhl. 2004. Quality of service in VoIP communication. *International Journal of Electronics and Communications* (2004).

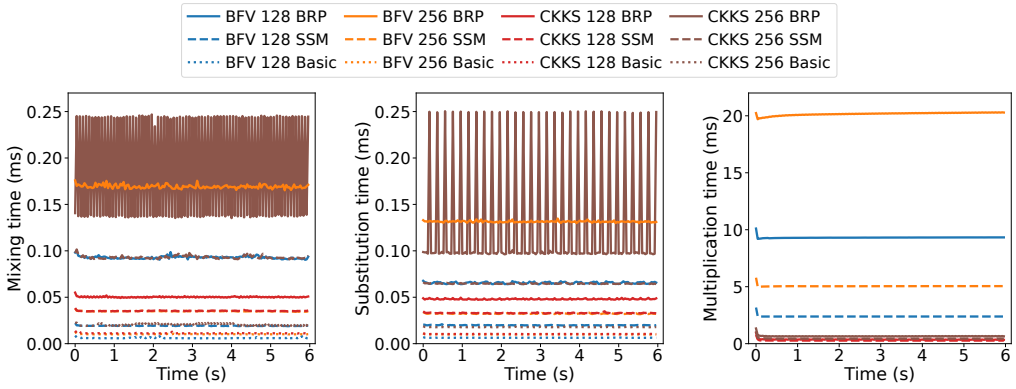
[65] Anamaria Vizitiu, Cosmin Ioan Nită, Andrei Puiu, et al. 2020. Applying deep neural networks over homomorphic encrypted medical data. *Computational and Mathematical Methods in Medicine* (2020).

[66] Xinyuan Wang, Shiping Chen, and Sushil Jajodia. 2005. Tracking anonymous peer-to-peer VoIP calls on the internet. In *Proc. ACM CCS.*

[67] Wire. 2021. Upgrades to Wire Conferencing. https://wire.com/en/blog/upgrades-to-wire-conferencing/

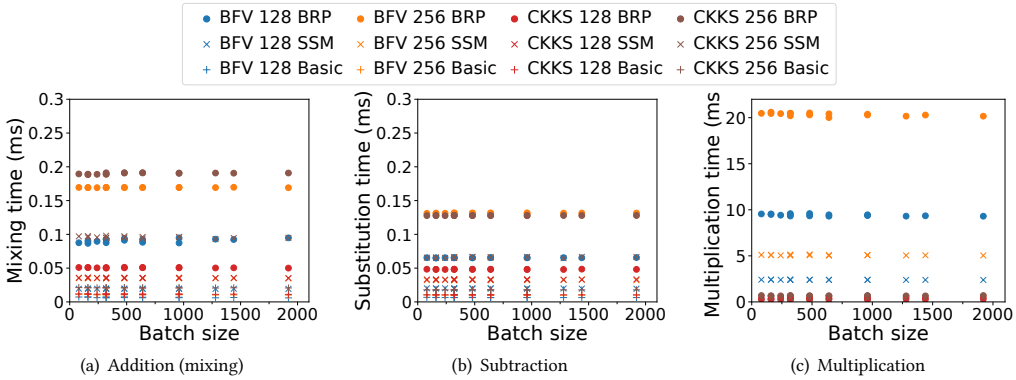
[68] Sharath Yaji, Kajal Bangera, and B Neelima. 2018. Privacy preserving in blockchain based on partial homomorphic encryption system for AI applications. In *Proc. High Performance Computing Workshops (HiPCW).*

[69] Zoom. 2020. Zoom Rolling Out End-to-End Encryption Offering. https://blog.zoom.us/zoom-rolling-out-end-to-end-encryption-offering/



(a) Addition (mixing) (b) Subtraction (c) Multiplication

Figure 17: Homomorphic performance of ciphertext operations over time.



(a) Addition (mixing) (b) Subtraction (c) Multiplication

Figure 18: Homomorphic performance of ciphertext operations per batch size.

A PERFORMANCE OF CIPHERTEXT OPERATIONS OVER TIME

Figure 17 shows the performance of homomorphic ciphertext operations over time for (a) addition, (b) subtraction, and (c) multiplication, respectively. We note that the basic configuration only performs additions and subtractions, no multiplications, and its results are therefore omitted from Figure 17(c). We also note that the periodic behavior of *CKKS-256-BRP* seen in Figures 17(a) and 17(b) are due to machine specific memory management.

B PERFORMANCE OF CIPHERTEXT OPERATIONS PER BATCH SIZE

Figure 18 shows the performance of homomorphic ciphertext operations per batch size for (a) addition, (b) subtraction, and (c) multiplication, respectively.