Friedrich-Alexander-Universität Erlangen-Nürnberg

Master Thesis

Similarity Estimation of Audio Data in the Encrypted Domain

submitted by

Srivatsav Chenna

submitted

August 24, 2022

Supervisor / Advisor

Prof. Dr. Nils Peters
Dr. Albrecht Petzoldt
Dr. Mikhail Korotiaev

Reviewers

Prof. Dr. Nils Peters

AUDIO LABS

International Audio Laboratories Erlangen
A joint institution of the
Friedrich-Alexander-Universität Erlangen-Nürnberg (FAU) and
the Fraunhofer-Institut für Integrierte Schaltungen IIS.

Fraunhofer IIS

Erklärung

Ich versichere, dass ich die Arbeit ohne fremde Hilfe und ohne Benutzung anderer als der angegebenen Quellen angefertigt habe und dass die Arbeit in gleicher oder ähnlicher Form noch keiner anderen Prüfungsbehörde vorgelegen hat und von dieser als Teil einer Prüfungsleistung angenommen wurde. Alle Ausführungen, die wörtlich oder sinngemäß übernommen wurden, sind als solche gekennzeichnet.

Erlangen, August 24, 2022

Srivatsav Chenna

Acknowledgements

First, I would like to express my gratitude to Prof. Dr. Nils Peters for his supervision and support throughout my thesis and also during my research internship at the Audio labs. I'm very grateful for the effort and time spent to help me improve my research and bringing out the best in me. I really appreciate his understanding to my interests and helping me build my inter-disciplinary research around it. I have truly enjoyed working with him and would like to thank him for creating a welcoming environment. I would like to thank my advisors Dr. Mikhail Korotiaev and Dr. Albrecht Petzoldt for bringing their expert knowledge and helping me with understanding the research topic. I have gained a lot from our interactions and really appreciate your insightful feedback and support. I was always excited for our meetings and would like to thank both my supervisor and advisors for creating a space where I could share my ideas and express myself.

I would like to thank my family and friends for their constant support at every step in my masters journey. I'm very grateful to have my wonderful parents for giving me the opportunity to pursue my studies and believing in me. Thank you for always have my back.

Finally, I would like to thank my girlfriend and best friend Deeksha. She has been my support system and motivates me to be my best. Thank you for inspiring me and making me feel at home anywhere I am.

Abstract

Data is everywhere, more people have access to the internet than ever before. As a result, the amount of audio data being collected over the internet for communication, telemedicine, machine learning, speech recognition, and many more applications is growing exponentially. With increase in connected audio devices, security and privacy aspects of distributed audio data is becoming a growing concern. While traditional encryption secures data at rest and in transit, it does not secure the data while in use. In recent years, research on privacy preserving techniques like homomorphic encryption, differential privacy, zero knowledge proofs are under way to allow data protection while in transit. This thesis looks into the feasibility of using Fully Homomorphic Encryption (FHE) to preserve privacy of audio data. FHE enables certain mathematical operations to be performed on encrypted data without the need for decryption. This allows to preserve privacy of the data while operating over cloud server or untrusted parties. Combining FHE schemes with operations like audio correlation, more efficient and secure distributed audio applications can be developed. In this thesis, we perform similarity estimation for different audio data (audio fingerprint and audio signal data) in FHE domain. To compute audio data in the homomorphic domain, unique techniques and algorithms are proposed which could be used in larger audio applications. We conduct experiments to evaluate and analyse the performance of different FHE schemes based on accuracy, speed and memory requirements. This thesis provides a discussion into the application of different FHE schemes for audio data.

Contents

Chapter 1

Introduction

The advancement in connected technology is leading to a world with more and more connected IoT devices. With the introduction of 5G communication technology, the IoT infrastructure is expected to grow with increased speed, capacity and decreased latency [27]. It is now more convenient to outsource complex computing to the cloud which also reduces cost for infrastructure. However, as the technology becomes more accessible, it raises serious questions on the privacy and data integrity as pointed out in [8] and [5]. With raising improvements in speech recognition and language processing technologies, everyday appliances like speakers, washing machines, refrigerators and other smart home appliances collect enormous amount of speech data from the consumers and process this data over the cloud using machine learning algorithms to improve the quality of service. This is leading to a large amount of audio data being processed and stored in the cloud. The privacy concerns during data exchange and storage have been largely researched upon and address the problems by using cryptography techniques like Rivest, Shamir Adleman (RSA) [6]. One of the more challenging problems to solve with respect to privacy in cloud based applications is securing the data while in use. Majority of audio IoT devices used in home settings record the audio from the consumers and process this data over the cloud, using the current infrastructure the consumer confidentiality is breached while processing on speech data. Some of the privacy preserving techniques being researched upon are differential privacy where limited information is leaked while data is in use and the sensitive information is kept secure. Zero knowledge proof is another privacy preserving technique in which the secret information of one party can be proved by another party without exchange of any information between the two parties [24]. In this thesis, to overcome the privacy problem we propose making use of fully homomorphic encryption (FHE) to secure audio data while in use.

Homomorphic encryption enables processing on encrypted data i.e. operations can be performed on encrypted data without the need for decryption. There has been a increase in research interest in homomorphic encryption over the recent years with the security concerns over consumer

data. In [24], the Big Data UN Global Working Group suggests homomorphic encryption as one of the emerging privacy-preserving techniques. The privacy and confidentiality issues can be solved by integrating homomorphic encryption techniques with the current IoT infrastructure [40]. There is also a need for a more reliable security infrastructure as the current security infrastructure is expected to be obsolete with the advancement in quantum cryptography [13]. However, homomorphic encryption comes with its own challenges [3], it is significantly more complicated to implement and slower for wide range practical applications. This thesis looks into the feasibility of using homomorphic encryption in the audio processing context. We experiment with operations on audio fingerprint data and signal data in the homomorphic encrypted domain. According to the audio data we chose three different FHE schemes and propose homomorphic techniques that can be applied to larger audio applications.

1.1 Thesis Organization

This thesis is organised as follows:

In Chapter 2, we give the background on homomorphic encryption along with the theory required for this thesis and explain the different FHE schemes used with respect to their properties. We discuss the current state of the art in the field of FHE and audio. And cover the different audio data used in our implementations for similarity estimation. Chapter 3 covers the different algorithms and techniques used to estimate similarity score using the integer based schemes BFV and BGV. The experiments and different parameters used to evaluate the schemes are described along with detailed analysis of the results from the experiments. In Chapter 4 the algorithms related to correlation estimation using CKKS scheme are covered. An analysis of the algorithms used, based on accuracy and performance is discussed. In the end, the experiment setup and results are presented and discussed. Lastly, in Chapter 5 summarises the thesis and an outlook for the future work is provided.

Chapter 2

Theoretical Foundations and State of the Art

In this Chapter we layout the theoretical foundations and basic primitives of the fully homomorphic encryption in Section 2.1. We introduce the different homomorphic encryption schemes used in this thesis and give a brief description of the tools used for FHE implementations in Section 2.2. Finally, we discuss the current state of the art of FHE in the audio context in Section 2.3 and cover the different audio data used in this thesis in Section 2.5.

2.1 Homomorphic Encryption

Homomorphic encryption allows mathematical operations to be performed on encrypted data. The main property of homomorphic encryption is, the same output should be obtained from operating on the initial plaintext as from decrypting the operated ciphertext. The history of homomorphic encryption dates back to 1978 when Rivest, Adleman, and Dertouzos first introduced the idea in [34]. However the first fully homomorphic encryption scheme was proposed by Gentry in 2009 [20]. In this paper Gentry first constructs a somewhat homomorphic schemes and modifies it to a fully homomorphic scheme with the introduction of bootstrapping technique. This was considered a breakthrough in the cryptography field and many more FHE schemes were proposed in the coming years. All the schemes that were proposed over the years are divided into 4 generations. The first generation schemes were based on ideal lattices and only operated over integers [20] [44]. The second generation schemes were based on stronger cryptographic hardness assumptions and had faster computing time than the previous schemes. The third generation schemes had different approach to noise management and were based on weaker cryptopgraphic hardness assumptions. The fourth generation scheme operates over fixed point values and

performs approximate evaluations [14]. Homomorphic encryption schemes are categorised into 3 main forms: somewhat, partially and fully homomorphic. Somewhat homomorphic schemes support both addition and multiplication operations but have a fixed amount of noise that can be added to the ciphertext, this allows only limited number of operations allowed on the ciphertext after which decryption is not possible. Partially homomorphic schemes allow only either addition or multiplication operations. Finally, fully homomorphic encryption schemes allow both addition and multiplication operations with no limit to the number of operations. In this thesis we work with Brakerski-Fan-Vercauteren (BFV) [19] and Brakerski-Gentry-Vaikuntanathan (BGV) [11] FHE schemes for integer based applications, and Cheon-Kim-Kim-Song (CKKS) [14] FHE scheme for fixed-point based applications.

2.1.1 Lattice Based Cryptography

A lattice can basically be thought of as any regularly spaced grid of points stretching out to infinity. There are problems based on lattices that are extremely hard to solve. In [2] Ajtai proposed the first lattice based cryptographic construction whose security was based on the hardness assumption of solving lattice problems. Using lattices we can define problems that are easy to construct but hard to crack. The advantages of lattice problems is there are no efficient algorithms that can solve the problems (classical or quantum) in better than exponential time [21]. The security of FHE schemes discussed in this thesis are based on the hardness of solving learning with error (LWE) and ring-learning with error (RLWE) lattice problems [33]. The FHE schemes are constructed in such a way that they are hard to crack if some particular information about the problem construction is unknown, but simple to solve if the problem construction is known. The RLWE problem is a extended version of the LWE problem to polynomial rings. As a consequence we operate in polynomial rings and the homomorphic operations are over polynomials.

2.1.2 Noise Management

In FHE schemes, noise is added to the ciphertext to conceal the plaintext. Figure 2.1 show how the noise increases after each homomorphic operation. After an homomorphic multiplication operation the noise also increases multiplicative and eventually the noise is too large and accurate decryption of the ciphertext is not possible. To get around this problem, Gentry in [20] proposed the bootstrapping technique for FHE schemes. In this technique the noise of the ciphertext is reset by first decrypting the ciphertext in a separate encryption system and encrypting it again to get a fresh noise level. Figure 2.2 gives an illustration of this process. Bootstrapping can be a time consuming processes and many different techniques have been proposed to improve this step of the FHE [22] [18] [17]. There are also other techniques used for managing noise like

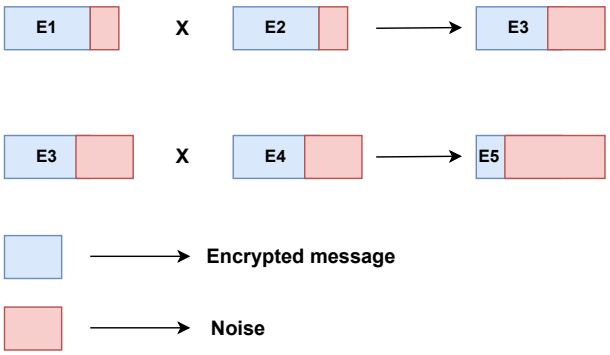


Figure 2.1: Illustration of the noise growth in homomorphic encryption.

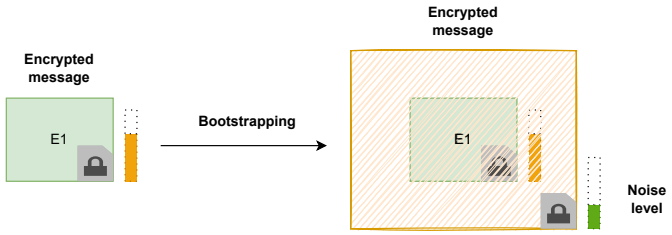


Figure 2.2: Example of the bootstrapping technique to reduced the noise in homomorphic encryption.

relinearisation and modulus switching techniques which are used based on the FHE scheme. We will introduce these techniques in the later sections while discussing the different FHE schemes.

2.1.3 Basic Primitives

In this section we will introduce the notations and definitions that are common in the all FHE schemes. The structure of FHE is similar to the traditional encryption schemes with additional algorithms for encryption parameters and homomorphic operations. The choice of the encryption parameters directly impact the performance, complexity and security of the FHE schemes. Therefore, it is important to choose optimal parameters according to the requirements of the implementation. In [4] the FHE standardization consortium provide the basic primitives of homomorphic encryption with a goal to standardise them in the future.

$ParamGen(\lambda) \rightarrow Params$: The parameter generation algorithm takes as input the security parameter λ, which is a number used to define the security level of the FHE scheme, and returns a set of encryption parameters. Values for λ can be 128 and 256.

$KeyGen(Params) \rightarrow (Pk, Sk, Ek)$: The Key generation algorithm takes as input the encryption parameters and returns a set of keys. In addition to secret keys (Sk) and public keys (Pk) used in traditional encryption, FHE schemes have evaluation keys (Ek) which are used to perform

homomorphic operations on the ciphertext.

$\underline{Encrypt(Pk, M) \rightarrow C :}$ The encryption algorithm takes input as Pk and message M and returns ciphertext C as the output.

$\underline{Decrypt(Sk, C) \rightarrow M :}$ The decryption algorithm takes input as Sk and C and returns message M as the output.

$\underline{EvalAdd(Params, Ek, C1, C2) \rightarrow C3 :}$ The $EvalAdd$ algorithm takes input as $Params$, Ek and two valid ciphertexts $C1$ of message $M1$ and $C2$ of message $M2$, and returns ciphertext $C3$ which is $(M1 + M2)$.

$\underline{EvalMult(Params, Ek, C1, C2) \rightarrow C3 :}$ The $EvalMult$ algorithm takes input as $Params$, Ek and two valid ciphertexts $C1$ of message $M1$ and $C2$ of message $M2$, and returns ciphertext $C3$ which is $(M1 * M2)$.

2.2 Homomorphic Encryption Schemes

In this section we discuss the different FHE schemes, give definitions to scheme-specific encryption parameters and the type of data the encryption schemes support. In this thesis, all the FHE schemes were implemented using the Simple Encrypted Arithmetic Library (SEAL) developed by Microsoft therefore we define the FHE schemes according to the SEAL manual [30].

2.2.1 Brakerski/Fan-Vercauteren Scheme (BFV)

The Brakerski/Fan-Vercauteren scheme, (also referred to as Fan-Vercauteren (FV) scheme) is a second generation FHE scheme [19], constructed based on Brakerski's fully homomorphic scheme with Learning With Errors (LWE) problem [9] ported to the ring-LWE setting. Compared with its predecessors the BFV scheme introduced optimal relinearisation and bootstrapping methods to achieve faster computations. Relinearisation method is used to reduce noise while keeping the ciphertext at the same size. The BFV scheme supports arithmetic circuits evaluation, therefore the inputs are integers. It also supports Single Instruction/Multiple Data (SIMD) or batching operations where multiple input messages can be encrypted in to a single ciphertext, this allows parallel computations and reduces memory and complexity. This form of packed ciphertext encryption is based on techniques proposed in [10] and [41]. The batching process does not allow access to elements of the encrypted ciphertext. All homomorphic operations apply to the packed data as a whole. There are small differences between the Textbook-BFV implementation in [19] and the BFV implementation in SEAL [30]. The SEAL implementation of BFV improved upon the noise consumption and computation time for addition and multiplication operations. Additionally, operations like adding and multiplying multiple ciphertexts simultaneously, adding

M_1	$1 + 2x + 1x^2$
M_2	$1 + 1x + 1x^2$

Table 2.1: Example of plaintext messages in polynomial form for $(n, t) = (3, 4)$

and multiplying ciphertext with a plaintext and negate operations are possible using SEAL BFV implementation.

2.2.2 Brakerski-Gentry-Vaikuntanathan Scheme (BGV)

Brakerski, Gentry and Vaikuntanathan proposed a new approach for constructing a leveled FHE scheme to further improve performance and security of lattice-based FHE schemes [11]. Both BFV and BGV are lattice-based and their security is based on learning with errors (LWE) hardness assumption, they operate in the same plaintext space but differ in their handling of noise management, homomorphic multiplication operation and message encoding. Additionally, the BGV scheme also works in the ring-learning with errors (RLWE) instance and achieves better performance in this setup. The BGV scheme supports arithmetic circuits evaluation, SIMD or batching operation similar to the BFV scheme. The key difference of the BGV scheme is the noise is managed in the scheme by reducing the ciphertext modulus after each multiplication in order to keep the noise level constant, this processes is called modulus switching and was first introduced by Brakerski and Vaikuntanathan [12]. With use of the modulus switching technique the BGV scheme supports faster multiplications. The SEAL BGV implementation is constructed based on HElib open source library [25]. In [28] the authors make a comparative analysis between the BFV and the BGV scheme. They observe in the current practical implementations BFV scheme is faster than the BGV scheme but insist BGV still has lower theoretical complexity than BFV.

2.2.3 Parameters for integer based FHE schemes

The BFV and BGV schemes are constructed based on the same idea, therefore they share many common features. In integer based FHE schemes we perform polynomial evaluations. Operating with polynomials provide a good trade-off between security and efficiency. The plaintext space (P) and ciphertext (C) space are distinct polynomial rings given by $P = R_t = Z_t/(x^n + 1)$ and $C = R_t \times R_q$ respectively, where $R_q = Z_q/(x^n + 1)$, $n \in Z$ is the ring dimension and $t \in Z$ and $q \in Z$ are the plaintext and ciphertext coefficients respectively. In practice the ciphertext space C is larger than the plaintext space P to allow multiple valid ciphertexts in C for a single plaintext message M in P, this implies q is greater than t. Table 2.1 shows examples of plaintexts in polynomial form for $n = 3$ and $t = 4$.

polynomial modulus	*ciphertext modulus*
1024	27
2048	54
4096	109
8192	218
16384	438
32798	881

Table 2.2: Examples of *polynomial modulus* value and the corresponding maximum possible *ciphertext modulus* value

The number of arithmetic operations that can be performed on encrypted data is depends on the encryption parameters in the BFV and BGV schemes. Since pseudo random noise is added to conceal the plaintext, the noise in the ciphertext is increased every time an operation is performed with the encrypted data. Every ciphertext has a fixed amount of noise called noise budget which is measure in bits, If the noise budget of a ciphertext reaches zero, decryption of the ciphertext is incorrect. The addition operation in general consumes less noise budget compared to multiplication as the noise is additive. Similarly, multiplication produce significantly more noise since the noise is multiplicative and as the number of multiplications increases the noise also increases exponentially.

Polynomial evaluations are performed in BFV scheme using modular arithmetic. The plaintext and ciphertext space affect the performance and security level of the scheme, therefore they should be defined precisely according to the requirement of the implementation. The *polynomial modulus*, *plaintext modulus* and *ciphertext modulus* are encryption parameters of the BFV scheme which collectively define the plaintext space, ciphertext space and the noise budget available for the homomorphic operations. The encryption parameters are defined below.

polynomial modulus: This parameter must be a positive power of 2 integer. Larger *polynomial modulus* enables more complicated encryption computations at the same time making the ciphertext size larger consequently making all operations slower. Recommended values are 1024, 2048, 4096, 8192, 16384, 32768 , but it is also possible to go beyond this range.

plaintext modulus: This parameter can be any positive integer. It determines the size of the plaintext and consumption of noise budget in encrypted multiplications. Therefore, smaller the *plaintext modulus*, lower the the noise consumption and better the performance. In homomorphic encryption we use modular addition and multiplication, the modulus value is determined by the *plaintext modulus*.

ciphertext modulus: This parameter is the product of prime numbers each up to 60 bits in size. It determines the size of the ciphertext. Larger *ciphertext modulus* increases the noise budget and allows for more operations to be performed on the encrypted data. The maximum value for

ciphertext modulus is determined by the *polynomial modulus*. The *ciphertext modulus* should be large enough to accommodate the noise from all arithmetic operations for your application. Table 2.2 shows some examples for *polynomial modulus* value and the corresponding maximum possible *ciphertext modulus* value.

slot size: While using the SIMD or batching technique, the input is encoded into a plaintext matrix. The number of elements encoded into the plaintext matrix is called slot size. The slot size is determined by the *polynomial modulus* value. For example, if the polynomial modulus value is 8192, the input is encoded into a 2×4096 matrix.

2.2.4 Cheon-Kim-Kim-Son Scheme (CKKS)

Cheon, Kim, Kim and Son constructed an approximate homomorphic encryption scheme that operates over fixed-point values [14]. Its is a forth generation FHE scheme that is based on the ring learning with errors (RLWE) hardness assumption. CKKS schemes uses a *rescaling* procedure to reduce the noise and error during approximate homomorphic operations. They propose a new SIMD or batching technique to allow parallel computations and preserve the precision of the plaintext. The SIMD technique allows us to encrypt multiple plaintext values in a single ciphertext, therefore the plaintext is encoded in form of a matrix similar to the integer based schemes. In CKKS the noise growth is linear and not exponential unlike the previous generation FHE schemes. The CKKS scheme also allows operations over vectors of complex values. The outputs from the CKKS schemes are approximate values with predetermined precision.

CKKS also works with polynomials, the message space is a complex number field $C^n/2$, where n is the *polynomial modulus*, which is a power of 2. The plaintext space is defined by polynomials in the cyclotomic ring $Z[X]/(X^n + 1)$ and the ciphertext space is defined by $(Z_q[X]/(X^n + 1))^2$ where q is the *ciphertext modulus*. The message is first encoded into the plaintext space and the plaintext is encrypted into the ciphertext space. After the homomorphic operations, the encrypted message is decrypted to the plaintext space and decoded to the message space. The encryption parameters *polynomial modulus* and *ciphertext modulus* have similar properties as discussed in Section 2.2.3.

The *scale* parameter is unique to the CKKS scheme. The *scale* determines the precision of the output. The multiplication operation causes the *scale* in the ciphertext to grow and if the *scale* of the ciphertext grows close to the *ciphertext modulus* value, accurate decryption will not be possible. To address this issue in the CKKS scheme, *rescaling* operation is implemented. The *rescaling* operation is similar to *modulus switching* (See 2.2.2) operation used in the BGV scheme. The scale of the ciphertext is reset by reducing the size of the ciphertext. For managing noise further, relinearisation operation (See 2.2.1) used the BFV scheme is used.

Name	Developers	FHE Libraries
SEAL	Microsoft	BFV, BGV, CKKS
PALISADE	Consortium of DARPA	BFV, BGV, CKKS, TFHE
HElib	IBM	BFV, BGV

Table 2.3: Popular open source libraries and the FHE schemes implemented.

2.2.5 Other Homomorphic Encryption Schemes

There are other popular homomorphic encryption schemes like Gentry-Sahai-Waters (GSW) [22] scheme, it is a lattice-based scheme that operates over integers. For noise management the authors propose a new conceptually simpler approximate eigenvector method. Torus FHE (TFHE) scheme [16] is a FHE C++ library and the implementation is based on the ring variant of the GSW scheme and it operates over Boolean circuits. The FHEW scheme [17] implements a faster FHE schemes in which the bootstrapping process is optimised to be less than a second. There are some popular open source libraries that implement the FHE schemes like Microsoft SEAL, PALISADE, HElib and TFHE. Table 2.3 shows the list of libraries and the FHE schemes they support.

2.3 Current Work on Homomorphic Encryption in Audio Domain

In recent years with the heightened interest in homomorphic encryption and its privacy-preserving properties, audio researchers are exploring FHE integration to solve audio security problems. Shortell and Shokoufandeh proposed a secure signal processing in FHE domain [38]. They implemented FHE scheme based on the GSW [22] scheme and made modification to the FHE scheme to operate over real values, reduce errors in results and increase the practicality of FHE scheme. The scheme was tested by experimenting on filtering images in the encrypted domain. They observed that the encrypted processing introduced small error in the image which slightly decreased the quality of the image. Although the implementation and testing was conducted over image signal data, the FHE implementation could be extended to support audio signal processing.

Machine learning is another area where FHE could provide solutions. Privacy is a huge issue with models that are trained using user sensitive data. In [47], Zuber, Carpov and Sirdey implemented a FHE system based on combination BFV [19] and TFHE [16] schemes to hide the data in a speaker recognition neural-network model. They propose operations to compute the distance of the speakers and present two methods to find minimum distance. Their implementation showed accurate results and their FHE system can be used for larger machine learning applications.

Compression and decompression are increasingly being outsourced over the internet. To protect sensitive data being shared to untrusted parties, a homomorphic FLAC decompression method is proposed in [42]. The authors use the BGV FHE scheme for their implementation. They regularizes dynamic controls in FLAC decoding with static controls to achieve the decompression. They only encrypt the FLAC payload to keep the basic audio and compression information readable and make use of the SIMD technique to increase efficiency. The implementation can also be applied to decompress similar audio standards. However the processing time for the decompression is slow and needs further improvements to be practical.

A secure end-to-end voice over IP (VoIP) system using homomorphic encryption was proposed by Rohloff, Cousins and Sumorok in [35]. To support homomorphic mixing of the encrypted VoIP data, the authors propose VoIP encoding scheme to reduce the circuit depth that is scalable to existing VoIP infrastructure. They use the LTV homomorphic encryption scheme [31] for their implementation. The data is encrypted at the user end and sent to the VoIP server for homomorphic processing. The encrypted data is returned to the end user where it is decrypted. Since mobile communication is time sensitive the end-to-end homomorphic system introduces delays in communication which makes it impractical at its current state.

Encrypted audio watermarking in cloud servers using homomorphic domain was proposed in [29]. A system was proposed in which encrypted audio and encrypted watermark data is sent to the cloud server, the could server would perform homomorphic operations and send back the encrypted watermarked audio. The implementation was constructed using the CKKS encryption scheme and making use of SIMD technique to increase efficiency. The similarity between homomorphically computed audio watermark was compared with the original watermark to check if the audio has been tampered or not. There was no analysis presented with regards to the quality of the encrypted watermarked audio. They tested their implementation by computing encrypted audio watermark for different sizes of audio and evaluated the security and processing time.

A speech homomorphic encryption scheme was constructed in [37] to secure speech storing in public computing. They proposed an encryption techniques to reduce data expansion in ciphertext during encryption. The speech signal is represented in form of a speech matrix and to encrypt the speech matrix, a matrix encryption method is used. The proposed homomorphic scheme supports only addition operation. The scheme has low computational time and is resistant to attacks. A similar speech homomorphic encryption scheme is proposed in [46]. In this paper, Qiu-Yu and Yu-Jiao divide the audio into sound and silent parts by comparing the audio energy. Then they make use of BGV scheme to encrypt sound information and Paillier homomorphic encryption algorithm to encrypt silent information. The two ciphertext are combined to construct the scheme. The system has good efficiency and security analysis was also performed by testing their system against attacks.

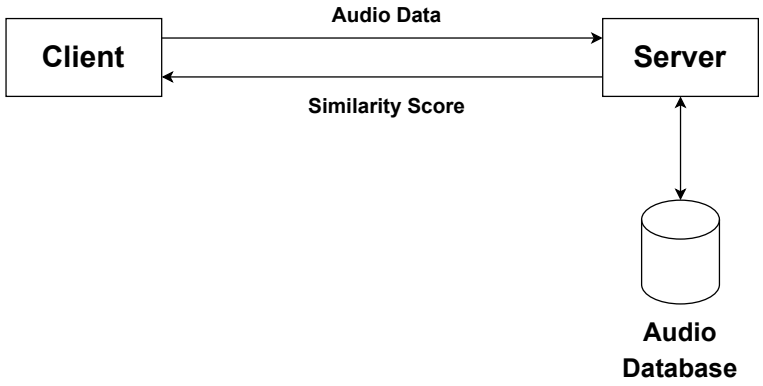


Figure 2.3: Overview of the client-server application scenario for similarity estimation.

Homomorphic encryption has a wide range of applications in the health care system. In the area of telemedicine to preserve privacy of patient data like voice calls, voice notes and video recording, a homomorphic approach is proposed in [26]. A homomorphic searching system to search sensitive medical data of the patients from the cloud is proposed using the BGV scheme. They use speech recognition models for the searching algorithm and tested their system by experimenting the search algorithm with different size audio files. They present their results based on the encryption time, searching time and decryption time.

Additionally, implementations of fast Fourier transform (FFT) have also been implemented using the FHE schemes [39]. From all the research over the years we can observe over time the computing time for the homomorphic operations is becoming more efficient and practical.

2.4 Application of Similarity Estimation in the Encrypted Domain

In this section we discuss the application scenario covered in this thesis and describe the techniques, algorithms and experiments performed to achieve the application constraints in the later sections. We give a simple example of a client-server model, in this scenario the client needs to compare sensitive audio data with the data from a audio database. The server takes the clients audio data as the input and performs the comparison operation with the database and returns the similarity score. We require this process to complete without revealing any information about the clients audio data to the server. An overview of the application scenario is shown in Figure 2.3.

In this thesis we solve this problem by using fully homomorphic encryption, we propose two methods for estimating similarity score for two different types of audio data using three different homomorphic schemes. We propose similarity estimation for audio fingerprint data using integer based homomorphic encryption schemes BFV and BGV, and correlation of audio signal data

using fixed-point homomorphic encryption scheme CKKS. The application scenario presented above is only an example for the use case of the methods proposed in this thesis, the algorithms proposed can be applied to preserve privacy in various settings like audio data from smart speakers and voice assistants, music data identification, training audio-based machine learning models, media applications and so on.

2.5 Audio Data

In this thesis we evaluate the homomorphic encryption on two different types of audio data: audio fingerprint data and audio signal data. The commonly used FHE schemes BFV and BGV are integer based schemes and to take advantage of this in the audio processing domain we chose audio fingerprint. The CKKS scheme operated over fixed-point values which is suitable for audio signal processing. We perform the function of matching audio tracks in the FHE domain for BFV and BGV. We use XNOR operation to match the fingerprint data. For CKKS we find the correlation of the audio signals. For the integer based schemes we get accurate results for our implementation and for CKKS implementation we obtain approximate results. In this thesis we investigated the different encryption schemes and how they can be utilised for different audio applications. To this end we propose different algorithms and techniques that can be used in larger audio applications which we discuss in the later sections.

2.5.1 Audio Fingerprint

Audio fingerprinting is a unique identification method for audio files. It is a condensed digital summary, deterministically generated from an audio signal. Fingerprints represent the most relevant acoustic components of an audio file. Manipulation of the audio is reflected in the audio fingerprint, this can be used to identify audio samples. The fingerprints are obtained by algorithms that extract time and frequency differences or audio features in spectrograms. Chromaprint is an audio fingerprinting library that was developed by AcoustID [1]. Chromaprint generates audio fingerprint by transforming frequencies into musical notes called *chroma features* [7]. These features go through filtering and normalisation processes to extract an audio fingerprint that is robust to audio compression, gain and pitch changes. There are other popular audio fingerprinting tools developed by Shazam, Phillips and Intrasonics that are used for music identification and audio forensics applications. Audio fingerprints obtained by Chromaprint is one of the audio data that is used in our experiments. The fingerprints are represented in a list of 32-bit integers which are converted to binary data for the purpose of similarity estimation in our application.

2.5.2 Audio Correlation

The correlation operation is used to find the similarity between two audio signals. Correlation is a commonly performed operation for sound localisation, speaker recognition, source separation and many more audio applications. Since we use the SIMD techniques in our similarity estimation algorithm, we represent our plaintext in form of matrix. We can find the correlation for audio data encoded in matrix form using the Equation 2.1. We compute the floating point time series values of the audio signal using the librosa library [32]. Since we have to operate over floating point values, to find the correlation between audio signals we use the CKKS FHE scheme. We encode the first 4000 time series values from the audio signals as the plaintext for our similarity estimation implementation.

$$r = \frac{\mathbf{a} \cdot \mathbf{b}}{|a||b|} \tag{2.1}$$

Chapter 3

Audio Fingerprint Matching in the Encrypted Domain

The most accurate way to measure the similarity of the raw audio fingerprint generated from Chromaprint is to convert the integer values of the two fingerprints that need to be compared into binary 0's and 1's and perform bit-wise XNOR operation. Considering the integer data type of the audio fingerprint, the BFV and BGV homomorphic encryption schemes described in Section 2.2.1 and 2.2.2 are ideal for our implementation. By choosing the optimal encryption parameters we can achieve efficient and accurate results for estimating the similarity score for audio fingerprint data. Since both BFV and BGV schemes are integer based schemes, they have the same encryption parameters but differ in noise management process (See Section 2.2.3), thus making the algorithms of both schemes similar.

In this Chapter, in Section 3.1 and 3.2 we introduce the different algorithms and techniques used for audio fingerprint matching using integer based homomorphic encryption schemes. Lastly, in Section 3.3 we describe the experiments and present our results.

3.1 Similarity Estimation for Fixed Length Audio Tracks

In this section we describe the algorithm for computing the similarity score for fixed length audio track (we used 30 second audio tracks). This section provides the base algorithm for similarity estimation which is further improved on according to additional requirements in the next section.

We describe the algorithms with an example scenario that is similar to our experiments which we cover in the later sections. We consider an encrypted audio data base with audio fingerprint data of Y different audio tracks. Table 3.1 gives the algorithm for similarity estimation of audio fingerprint for fixed length audio tracks. In Section 2.2.1, we discussed the different encryption

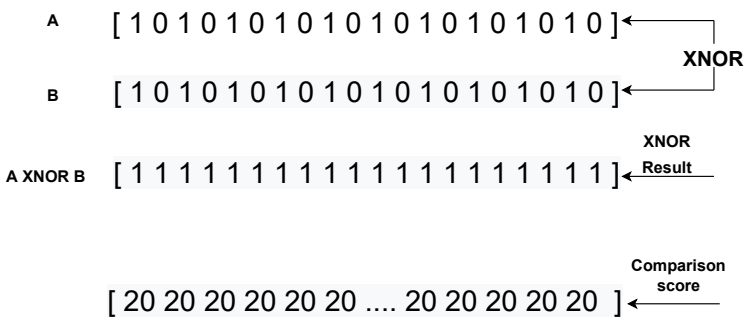


Figure 3.1: Example of bit-wise XNOR operation between matrix A and B.

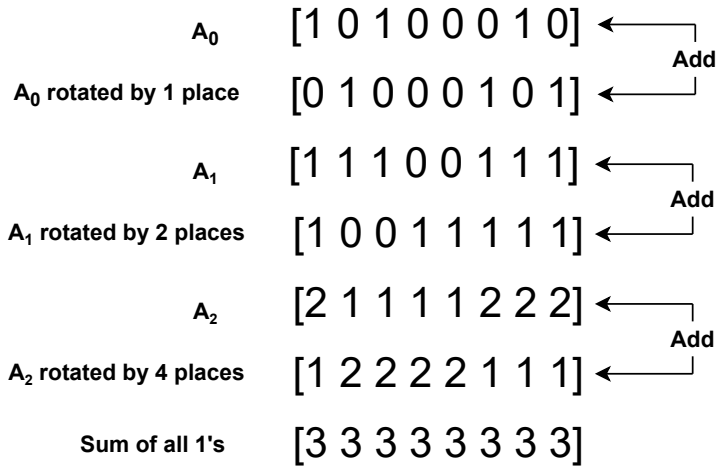


Figure 3.2: Example for the summing the 1's in a matrix.

parameters and their impact on the performance, in Table 3.1 we define the encryption parameter values used for the similarity estimation for fixed length audio tracks implementation. The *polynomial modulus* defines how many plaintext messages can be encrypted in a single plaintext matrix, here the *polynomial modulus* value is 8192 i.e. 8192 bits from the audio fingerprint data are encoded in single plaintext matrix. The 8192 slots of the matrix are encoded in form of a 2×4096 plaintext matrix. The *ciphertext modulus* or noise budget is 218 bits which is the maximum amount of noise available for a *polynomial modulus* value of 8192 (See Figure 2.2). This indicates the noise growth from the addition and multiplication operations cannot exceed 218 bits. The *plaintext modulus* value is 786433 meaning all the homomorphic operations are computed using 786433 as the modulus. This value is computed by SEAL library [36] function according to the *polynomial modulus* value. The output is $1 \times Y$ matrix where each element of the matrix is a similarity score an audio track with the audio tracks from the database.

$$I0 = 1 - (E0 + E1) - (2 \times E0 \times E1) \tag{3.1}$$

Since our input fingerprint data is 0's and 1's we perform the bit-wise XNOR operation between two input matrices using Equation 3.1 where $E0$ and $E1$ are the input matrix and $I0$ is the XNOR result. Figure 3.1 show an example of the bit-wise XNOR operation, here each element of the matrix A and B is encrypted into a ciphertext and because of the SIMD or batching property of the encryption scheme, we are able to encode multiple ciphertexts into a single plaintext matrix and perform homomorphic operations on them in parallel. After the XNOR operation, all the 1's in the matrix are added up using repeated cyclic rotations and additions as shown in Algorithm 1. In Algorithm 1 the number of *for* loop iterations needed is $\log_2(N)$, where N is the size of the matrix. Rotations were incremented in logarithmic order to reduce the number of iterations to compute the addition, thus saving noise budget and improving performance. An example of this process is shown in Figure 3.2, for a matrix of length 8, 3 addition and rotation operations are used to compute the sum of all the 1's. Finally, the output from algorithm in Table 3.1 is the similarity score in a $1 \times Y$ matrix, where Y is the number of audio tracks in the audio database. Each element in the output matrix is a similarity score of input audio track fingerprint $E1$ with each file in the audio database. To read the output matrix you need to know how the audio files are arranged in the database. In this setup, we estimate similarity score for fixed length audio track, however a more robust estimation would be computing the estimation score between arbitrary length audio tracks i.e. computing estimation scores for 5 second fragments for a audio tack with 30 second audio tracks.

Algorithm 1 An algorithm for summing the 1's

Input: X ▷ Matrix after bit-wise XNOR
Output: $Y \leftarrow X$
$i \leftarrow 0$
$N \leftarrow 0$
for $i \leftarrow 0$ **to** $i \leftarrow \log_2(N)$ **do** ▷ N is number of elements in input matrix X
    $N \leftarrow 2^i$
    $Y \leftarrow$ rotate rows N times
    $Y \leftarrow Y + X$
    $X \leftarrow Y$
end for

3.2 Similarity Estimation for Arbitrary Length Audio Tracks

In the previous section we proposed an algorithm to estimate similarity score among audio tracks of the same size, in this section we propose further improvements on Algorithm 3.1 to be able to compute similarity score for audio tracks of any length. For example, 5, 6 or 10 second audio tracks can be compared with each other. The maximum length of the audio tracks for describing the algorithms is kept at 30 seconds in this thesis. However this can be varied by adjusting the encryption parameters.

Algorithm: Similarity estimation for fixed length audio tracks.

Encryption parameters:
1. Polynomial modulus - 8192
2. Ciphertext modulus - 218
3. Plaintext modulus - 786433
4. Input martix size - 2×4096 matrix

Inputs: Encrypted audio database ($E1$) and Encrypted audio track to be compared ($E0$).

Outputs: Encrypted estimation score ($S0$) represented by $1 \times Y$ matrix, where Y is the number of audio tracks in the database.

Procedure:
1. Initialise the encryption parameters and plaintexts required.
2. Initialise $E1$ as the first encrypted audio file from the database.
3. Compute bit-wise XNOR using $I0 = 1 - (E0 + E1) - (2 * E0 * E1)$.
4. Add all the 1's from matrix $I0$ to get the estimation score.
5. Store the estimation score to $S0$ and rotate the $S0$ matrix by one place.
6. Change $E1$ to the next encrypted audio file from the database and repeat 2-5 till all the files from the data base have been compared.
7. Output the estimation score matrix $S0$.

Table 3.1: Iterative algorithm for calculating the estimation score for audio fingerprints of fixed length audio tracks (30 seconds).

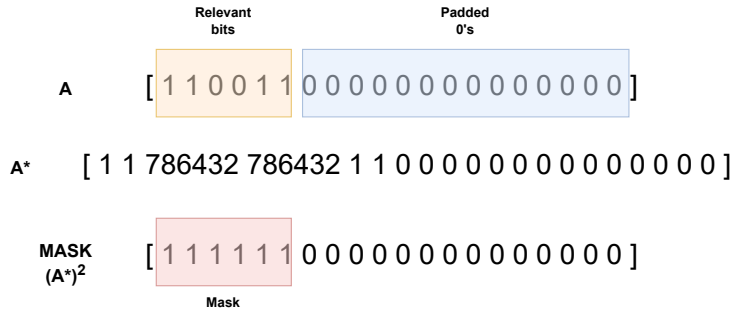


Figure 3.3: Example of creating a *mask* for a given matrix A. A* is the modified matrix where the 0's from the relevant bits of A are modified. The modified matrix A* is then squared to obtain the mask for A.

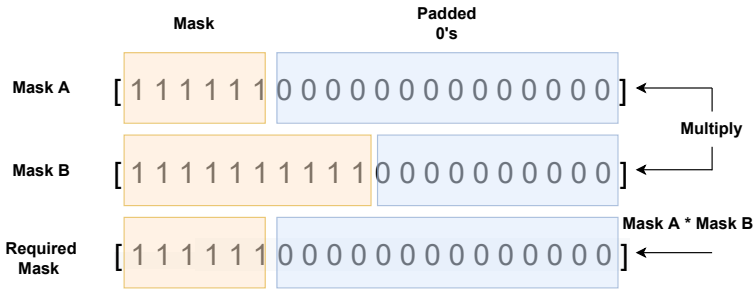


Figure 3.4: Example for computing the shorter *mask* from 2 different *masks* A and B.

3.2.1 Masking

To estimate similarity score for arbitrary length audio tracks we require the knowledge of length of the audio track. Since in the homomorphic domain we have no information on the inputs we need different techniques and additional operations to achieve estimation of arbitrary length tracks. In this thesis we propose a novel method to calculate an indicator for the length of the audio track by modifying the input audio fingerprint using the concepts of modular arithmetic. For example, in modular arithmetic if 8 is the modulus value and we need to compute (2+7 mod 8), the result we obtain from this problem is 1. The input wraps around 8 back to 1, we exploit this property to compute a *mask* for the input which is an indicator of the length of the audio fingerprint. The masking technique is only possible with FHE schemes operative over polynomials where the input value wraps around the *plaintext modulus* values (See 2.2.3).

$$(modulus - 1)^2 = 1 \tag{3.2}$$

Figure 3.3 we give an example on how we can modify the input and make use of modular arithmetic to create a *mask*. Making use of Equation 3.2 we can encode the 0's as $(modulus - 1)$ and square the matrix make all the relevant bits as 1's which will give us the indicator of the

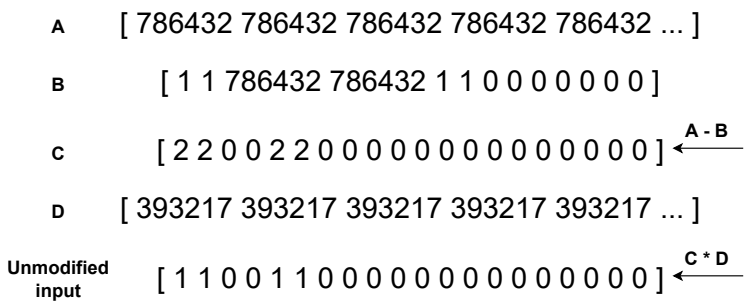


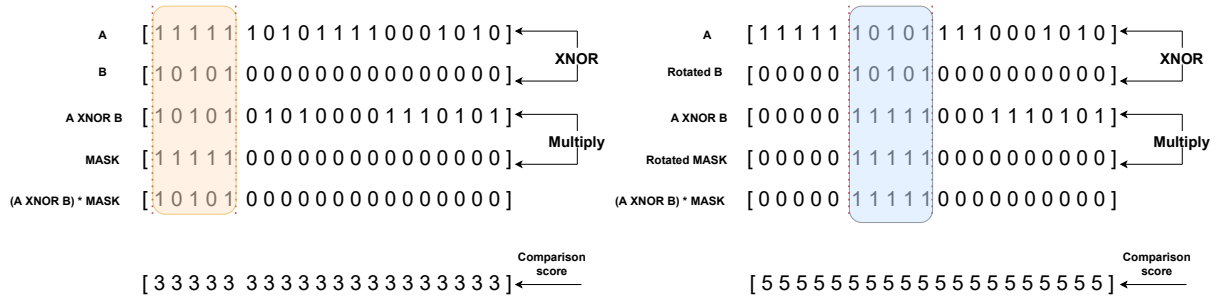
Figure 3.5: Example for getting back the unmodified input. In this example the modulus values is 786433 and 393217 is the multiplicative inverse of 2.

length of the relevant bits. As an example we take the *plaintext modulus* to be 786433 (See Table 3.1), therefore we encode all the 0's from the relevant bit of the audio fingerprint to 786432 and further square this matrix to obtain the mask. Consider the example shown in Figure 3.3, 1^2 is one and 786432 is also 1 (From Equation 3.2) converting all the relevant bits from matrix A to 1 and essentially making an indicator for the length of the matrix. We want to find the estimation score of small fragments of the audio tracks with larger audio tracks, for this we need the smaller mask between the two audio fingerprints. This can be calculated by first computing the mask for both the fingerprints and multiplying the masks with each other as shown in Figure 3.4, this works as the irrelevant bits of the fingerprint are padded with zeros.

To compute the mask we modified the inputs. We need to now get back the input with only 0's and 1's to be able to find the similarity score. Therefore, to continue forward with computing the bit-wise XNOR, we need to undo the modifications done on the input matrix, this can be done by performing a subtraction and a multiplication operation as shown in Algorithm 2 and Figure 3.5. A modulus value of 786433 is assumed again to explain the process of obtaining the unmodified input. To obtain back the 0's from the input matrix we first subtract input matrix with a 786432 matrix where all the elements of the matrix are 786432, however since the homomorphic operation applies to all the elements of the matrix even the 1's are subtracted by 786432, so we need another step to get back the 1's. After the subtraction, the 1's are converted to 2 (Because of modular arithmetic). We now need to divide the whole matrix by 2 to get back the unmodified input, however BFV and BGv schemes do not support division operation. To get around this we use the multiplicative inverse of 2. For our example the multiplicative inverse of 2 in 786433 modulus domain is equal to 393217, therefore multiplying a 393217 matrix gives us the unmodified input matrix. Figure 3.5 shows an example for this technique.

Algorithm 2 An algorithm for obtaining the mask

Input: X ▷ Audio fingerprint matrix
$i \leftarrow 0$
$M \leftarrow 0$ ▷ Mask matrix
for $i \leftarrow 0$ **to** $i \leftarrow N$ **do** ▷ N is the length of audio fingerprint matrix
  if $X[i]$ is 0 and $X[i]$ not padded 0 **then**
    $X[i] \leftarrow Y$ ▷ Y value is (modulus -1)
  end if
end for
$M \leftarrow X^2$ ▷ Mask of X
$X \leftarrow X - Y$
$X \leftarrow X \times 1/2$ ▷ Get back unmodified audio fingerprint matrix



(a) Step 1 (b) Step 2

Figure 3.6: An example for the computing similarity score for arbitrary length audio tracks using the siding window technique. Figure 3.6a shows the first iteration of bit-wise XNOR between matrix A and B and Figure 3.6b shows the second iteration of bit-wise XNOR after rotating matrix B.

3.2.2 Sliding Window

After obtaining the mask and getting back the unmodified input, we propose a sliding window technique to estimate the similarity score of the audio track fragment with the audio track. For example, consider we get 500 bits of fingerprint data from the first 5 seconds of a 30 second audio track and 3000 bits of fingerprint data from the whole 30 second audio track. The first 500 bits of the 3000 bits fingerprint data will be equal to the 500 bits fingerprint from the 5 second audio fragments. Figure 3.6 shows an example the proposed sliding window process, we use rotation operations to compare the audio fragment with the complete audio track. We use a fixed stride to determine the number of rotations. For example if the stride is 300 bits, 300 places are rotated from the matrix after each iteration to obtain the similarity score. If the total length of the fingerprint data matrix is around 5000 bits, a total of 18 iterations are required to compute the similarity score. This is because 300×18 gives us 5400 which would cover all the bit in a 5000 bit matrix.

Algorithm: Similarity estimation for arbitrary length audio tracks.

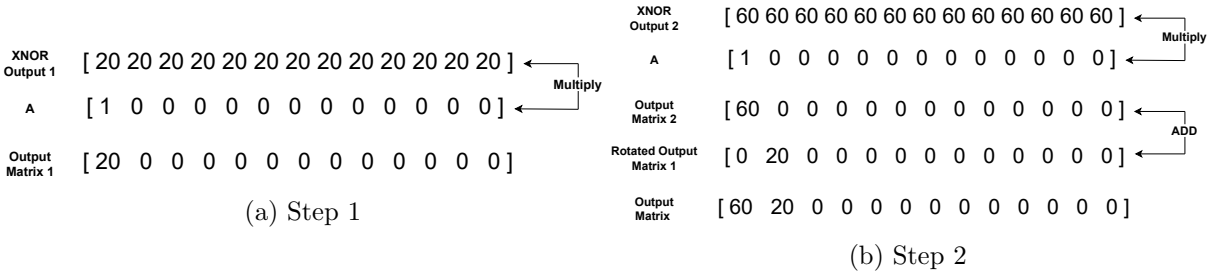
Encryption parameters: 1. Polynomial modulus - 16384
2. Ciphertext modulus - 438
3. Plaintext modulus - 786433
4. Input martix size - 2×8192 matrix

Inputs: Modified encrypted audio database ($E1$) and Encrypted audio track to be compared ($E0$).

Outputs: Encrypted estimation score ($S0$) represented by $1 \times Y$ matrix, where Y depends on the number of audio tracks in the database and stride used for sliding window technique

Procedure: 1. Initialise the encryption parameters and plaintexts.
2. Initialise $E1$ as the first encrypted audio file from the database.
3. Compute the mask for $E1$ and $E0$.
4. Convert back $E1$ and $E0$ to unmodified matrix.
5. Compute bit-wise XNOR using $I0 = 1 - (E0 + E1) - (2 * E0 * E1)$.
6. Multiply $I0$ with the mask
7. Rotate $E0$ by Z places and repeat steps 5-7 for Z/N iterations. Where Z is the stride and N is the length of the input matrix
8. Add all the 1's from matrix $I0$ to get the estimation score.
9. Store the estimation score to $S0$ and rotate the $S0$ matrix by one place.
10. Change $E1$ to the next encrypted audio file from the database and repeat 2-5 till all the files from the data base have been compared.
11. Output the estimation score matrix $S0$.

Table 3.2: Iterative algorithm for calculating the estimation score for audio fingerprints of arbitrary length audio tracks (30 seconds).



(a) Step 1

(b) Step 2

Figure 3.7: An example for the computing the bit-wise XNOR outputs in a single matrix. Figure 3.7a shows the first iteration of storing the XNOR result in the output matrix and Figure 3.7b shows the second iteration for storing the next XNOR output.

Table 3.2 shows the iterative algorithm to compute similarity score for arbitrary length audio tracks. We use different encryption parameters compared to parameters used for computing similarity score for fixed length audio tracks (See Table 3.1). To accommodate for the additional operations from masking and sliding window techniques we require a higher noise budget, therefore the *polynomial modulus* is increased to 16384 giving us a maximum noise budget of 438 bits (See Table 2.2). The input matrix is encoded into a 2×8192 matrix, 16384 plaintext values can be encoded in a single plaintext matrix (See Section 2.2.3). Most of the steps from the algorithm are identical to the algorithm from Table 3.1 with additional steps for generating mask for the inputs and an iterative process to compute similarity score for different fragments using sliding window technique. To get all the bit-wise XNOR results in a single output matrix, we we use a multiplication and an addition operation. We rotate the output matrix after computing each bit-wise XNOR output and add the output matrix with bit-wise XNOR result as shown in Figure 3.7. The length of the output matrix depends on the number of audio tracks in the audio database and the stride used in the sliding window technique. If we have 20 audio tracks and we use 18 iterations for the sliding window technique, the output would be a 1×360 matrix where 18 bit-wise XNOR operations give 18 similarity scores for comparison of each encrypted file from the data base with the encrypted input file. In order to understand the output similarity score, we need the knowledge of the arrangement of the encrypted database. If we performed 18 iteration of sliding window technique, the first 18 similarity score are the similarity score of the input audio fingerprint with the first audio fingerprint data from the database and next 18 score would be the similarity scores with the second fingerprint from the database and so on.

3.3 Experiments and Results

In the previous Sections 3.1 and 3.2 we described the algorithms to evaluate similarity score for audio fingerprint data using integer based homomorphic encryption schemes. We discussed different techniques and algorithms with examples explaining their workings. In this section, we

No.	Audio tracks	Size (in Bits)	No.	Audio tracks	Size (in Bits)
1	blues 0	6928	11	jazz 0	6824
2	blues 1	6487	12	jazz 1	6744
3	classical 0	6965	13	metal 0	7035
4	classical 1	6642	14	metal 1	7060
5	country 0	6626	15	pop 0	6645
6	country 1	6667	16	pop 1	6972
7	disco 0	7054	17	reggae 0	7002
8	disco 1	6996	18	reggae 1	6506
9	hiphop 0	7033	19	rock 0	7029
10	hiphop 1	7072	20	rock 1	6790

Table 3.3: Example for dataset audio tracks and their audio fingerprint sizes.

first present the datasets used for evaluation and give an overview of the setup for the experiments. We later describe the experiments performed in this thesis for the integer based homomorphic encryption schemes and lastly, we present and discuss our results.

3.3.1 Datasets

The dataset for evaluation was taken from GTZAN Genre Collection dataset [43]. The data set contains 1000 tracks with 10 different genre and each genre having 100 tracks. The tracks are all 22050 Hz monophonic 16-bit audio files in *.wav* format. The dataset was used in over 100 papers, the 30 second clips are small with a lot of features making it ideal for research evaluation. In this thesis, 30 tracks were used for the experiments. The tracks included 2 tracks from each genre along with ten 5 second clips of the 30 second audio tracks. In Table 3.3 we see an example of different audio files from the dataset and their corresponding audio fingerprint sizes.

3.3.2 Experiments for BFV and BGV

In Section 2.4, we give the application scenario used in this thesis, in this section we describe the experiments for computing similarity score for audio fingerprint data based on the application scenario. For both BFV and BGV schemes we use the procedure from Table 3.2 to construct our experiments. In Section 2.2.3, we discussed the different encryption parameters and their impact on the performance, in Table 3.4 we define the encryption parameter values used for the implementation. In Section 3.1 we cover the encryption parameters and their effect on the inputs with respect to our implementation.

Encryption Parameters	BFV	BGV
polynomial modulus	16384	16384
ciphertext modulus	250	390
plaintext modulus	786433	786433

Table 3.4: Encryption parameters used for BFV and BGV experiments.

3.3.2.1 Encrypted Audio Database

First an encrypted audio database is created, the audio fingerprint of all the audio tracks from the dataset (See Table 3.3) are computed in python programming language using the Chromaprint library [1]. In Section 3.2 we modified the input fingerprint data to support arbitrary length similarity estimation. Similarly, after we modify our fingerprint data for the encryption parameters shown in Table 3.4. All the 0's are encoded as 786432 to compute the mask (See Section 3.2.1), this is now our audio fingerprint input data. The modified fingerprint from each of the audio tracks is stored into separate text files. These text files are encrypted using the encryption parameters and stored in a database which we refer to as the encrypted audio database. The schematics of this process is show in Figure 3.8.

3.3.2.2 Inputs and Output

Now we move to the inputs and output of our experiments. The maximum fingerprint data has around 7000 values, a *polynomial modulus* value of 8192 is sufficient to encode our fingerprint data into the input matrix. (*polynomial modulus* determines the number of input values that can be packed into a plaintext matrix). However we require a higher noise budget for the homomorphic operation. Therefore, a *polynomial modulus* values of 16384 is used to get a noise budget of 438 bits (See Table 2.2). This allows us to perform more encrypted operations on the data. For our implementation we do not make use of all the 438 bits of noise budget, therefore the noise budget is reduced to 250. This reduces the size of the ciphertext and gives up better performance time. The inputs to our experiments are the encrypted audio data base and an encrypted audio fingerprint file whose similarity score has to be computed. In Section 3.2 we describe the sliding window technique which uses a stride to compare throughout the fingerprint data. For our implementation we use a stride of 300 bits. Since our audio fingerprint is around 7000 bits long, we use 18 iterations to fully compare through all the bits. We compare over 20 different audio tracks which gives us a output encrypted matrix with 360 different similarity scores (20×18).

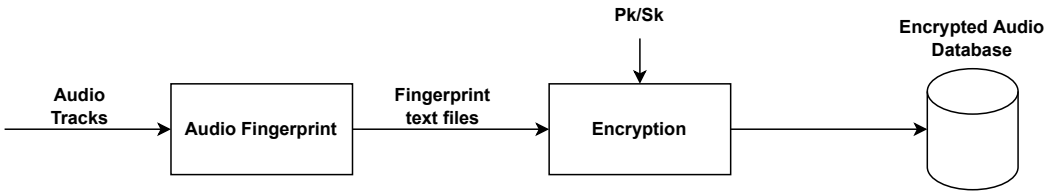


Figure 3.8: Schematics for creating the encrypted audio database

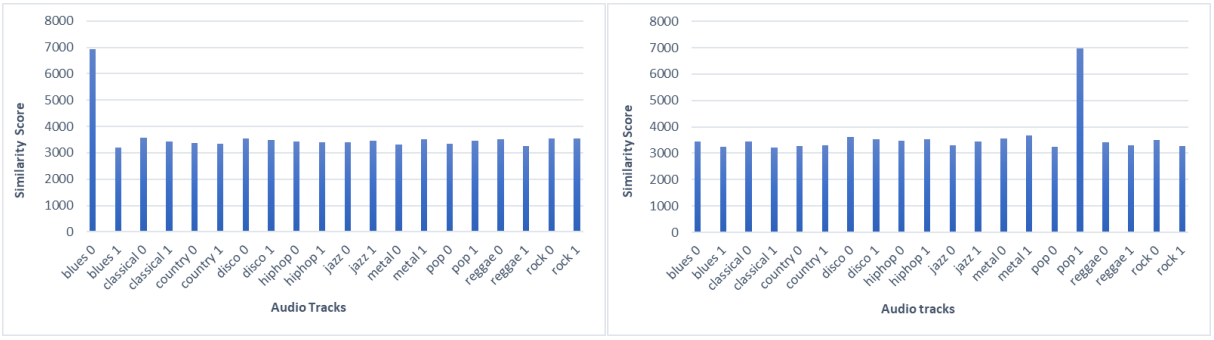
3.3.2.3 Similarity Estimation

The algorithm for the procedure to compute the similarity score is shown in Table 3.2 is implemented in C++ programming language using the SEAL library [36]. The bit-wise XNOR operation is performed linearly throughout the audio database. If *blues 1* is the input file whose similarity score has to be determined, first *blues 1* is compared with *blues 0*, then with *blues 1*, after that *classical 0* and so on according to the index shown in Table 3.3. This help us identify the output according to the index of the output matrix. This also implies we need the knowledge of how the encrypted database is arranged in order to analyse the output. From the description of the experiment we can observe that the program has no knowledge of the data being operated on, thus preserving privacy.

3.3.3 Results and Discussion

In the following, we discuss the results from the BFV and BGV experiments.All the experiments were implemented using Windows 10 operating system and Intel i5 2.30GHz processor with 8GB RAM. We decrypt the output matrix to get readable plaintext results. If the encryption parameters are not correct for the implementation, decrypting accurately is not possible. If the noise budget used is not sufficient we get incorrect output. The optimised noise budget was chosen after multiple runs according to the expected output. Each output score is compared to the size of the fingerprint according to the index, if the value from the output matches the size of the fingerprint from Table 3.3, the fingerprints and the audio signals are identical to each other.

In Figure 3.9 the results for the BFV experiments are shown. In Figure 3.9a the estimation scores for the *blues 0* files is shown, we can observe a peak in the similarity score of *blues 0* when compared with itself. The value of the peak is 6928, which is the same as the fingerprint size of the *blues 0* file as seen in Table 3.3, therefore accurately matching fingerprint from the encrypted audio database. We can observe similar results for matching *pop 1* fingerprint data with the database in Figure 3.9b. From the figures we can observe that the similarity score for the non matching files is around 3500, this is because we are performing bit-wise XNOR operation over 0's and 1's and 50% of the time the values would match which is evident from the



(a) Similarity scores for *blues 0*.

(b) Similarity scores for *pop 1*.

Figure 3.9: Similarity scores for the BFV experiments.

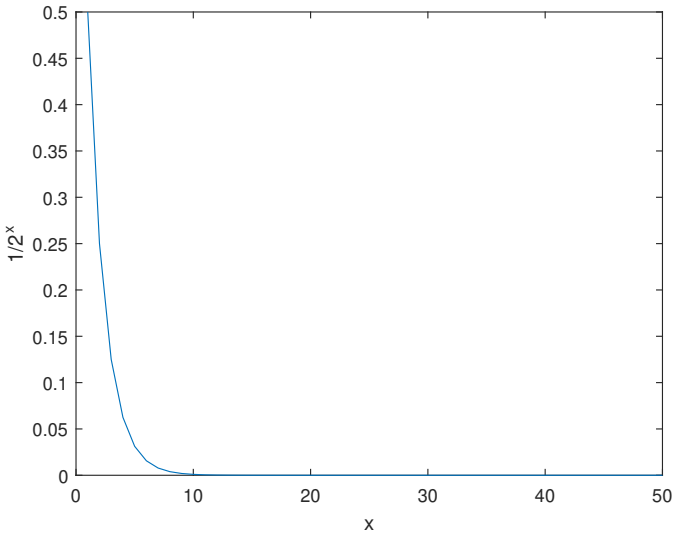
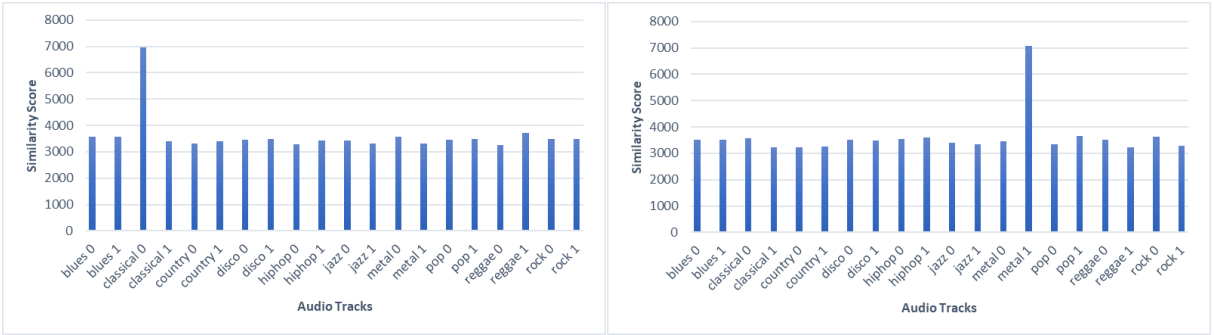


Figure 3.10: Probability odds for fingerprint data matching. The figure shows probabilities for fingerprint data from two different audio tracks to match by random chance. x is the number of bits.



(a) Similarity scores for *classical 0*.

(b) Similarity scores for *metal 1*.

Figure 3.11: Similarity scores for the BGV experiments.

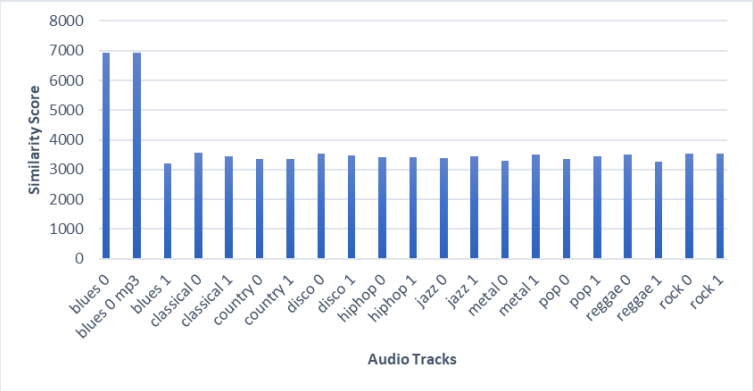


Figure 3.12: Similarity score for .wav and .mp3 formats for the same *blues 0* audio file.

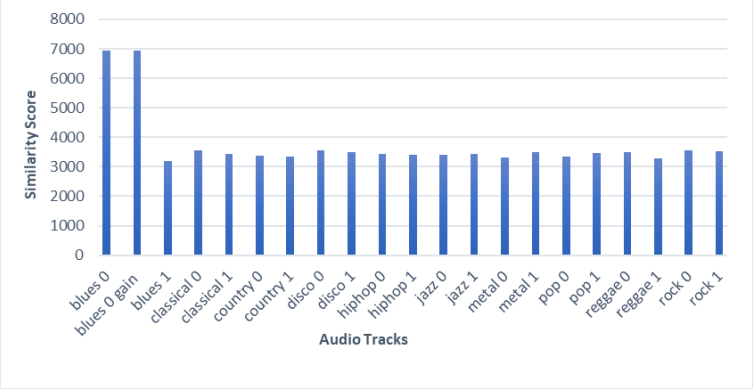
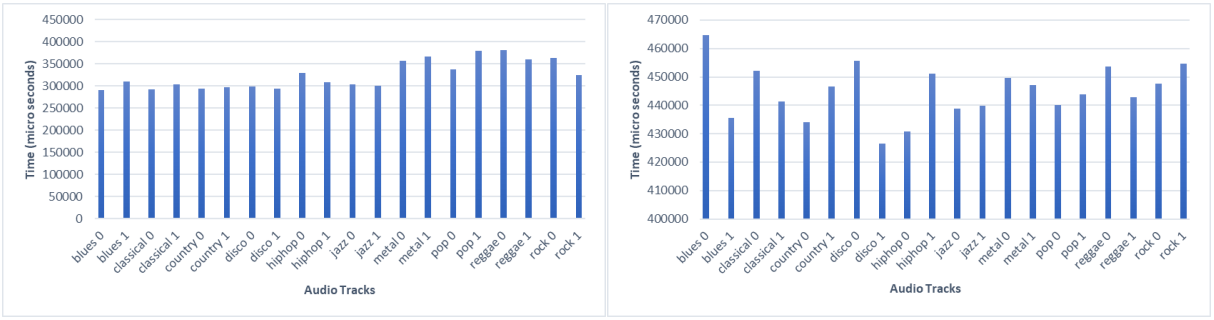


Figure 3.13: Similarity score for *blues 0* file with modified gain of -10dB.

results. Figure 3.10 show the probability of the fingerprint data matching for different audio signals. The figure shows the probability graph for up to 100 bits of the fingerprint data to match by random chance. From the Figure 3.10 we can see that the probability for 10 fingerprint bits to match in random is 0.097 %, in our case the probability of 7000 bits to match by random chance is negligible. In Figure 3.11 we have the similarity score for *classical 0* and *metal 1* files which were computed using the BGV scheme, we observe similar peaks where there is a match in the fingerprint data. Since both BFV and BGV are integer based schemes and have similar algorithms and experiments setups, we get accurate similarity scores for both schemes. One of the properties of the Choromaprint audio fingerprint is that the fingerprint does not change with change in format or changes in gains made to the audio file. To test this, we computed similarity scores for *blues 0* mp3 file and *blues 0* file with -10dB gain modification. The results are shown in Figure 3.12 and 3.13. From this we can observe the similarity scores are identical for *blues 0* .wav file and *blues 0* .mp3 file. Analogously, We get identical similarity scores when comparing *blues 0* and *blues 0* file with -10dB gain modification.

Now we will look at the performance of the BFV and BGV schemes. Figure 3.14 shows the



(a) Processing time *blues 0* in BFV.

(b) Processing time *blues 0* in BGV.

Figure 3.14: Comparison of processing time for BFV and BGV.

processing time for the comparison of each fingerprint file in BFV (Figure 3.14a) and BGV (Figure 3.14b) schemes. The timing values were measure using C++ high resolution clock function. We measured the timing values using 2 clocks, The first clock measured the time taken to estimate each similarity score excluding the encryption and decryption times. The second clock measured the time taken for estimating the similarity scores for one fingerprint with all the fingerprint from the database, this included the encryption and decryption times. The average time it takes to compute similarity score for one fingerprint with another fingerprint from the database in BFV scheme is 324657 μs and for BGV scheme is 449332 μs. The average time taken to compute similarity score for one fingerprint with the whole database is 113.299995 seconds for BFV scheme and 160.942683 seconds for BGV scheme (All the processing times were computed by averaging over the timing values obtained after 10 runs). We can observe from the results that on average the BFV scheme performs significantly better compared to BGV in our experiment setting. The BFV scheme is on average 30% more efficient than the BGV scheme. This is because the BFV scheme uses relinearisation (See Section 2.2.1) and the BGV scheme uses modulus switching (See section 2.2.2) to manage the noise in the ciphertext. In the BFV scheme the noise is reduced while keeping the ciphertext size the same, this make the processing time for all the multiplications the same. In BGV the noise is reduced by decreasing the ciphertext size after each multiplication, this makes the multiplications with reduced ciphertext size faster i.e. the processing time decreases as the number of multiplications increases however at the same time the noise budget also decreases at a higher rate.

Chapter 4

Audio Correlation in the Encrypted Domain

In chapter 3, we implemented similarity estimation algorithm for audio fingerprint data using BFV and BGV integer based schemes. However, many audio applications require processing of audio signal data which is usually floating-point values. In this chapter we explore the feasibility of the CKKS FHE scheme discussed in Section 2.2.4 for computing audio signal correlation. The CKKS scheme operates over fixed-point values, which makes it suitable for our audio correlation application. In Section 4.1 we introduce the different algorithms and techniques used for our implementation. In Section 4.2 we present the algorithm for computing approximate correlation in CKKS using the techniques covered in Section 4.1. Finally, we describe our experiments and discuss our results in Section 4.3.

4.1 Algorithms for Inverse and Square Root in CKKS Scheme

In this section we will discuss the techniques and algorithms used for our correlation in CKKS implementation. The Equation 4.1 is used for computing the correlation between our input vectors. In Equation 4.1 the numerator $\sum_{i=1}^{n} a_i b_i$ can be computed homomorphically using the multiplication operation and the summation can be computed using Algorithm 1. To compute magnitude of the vectors $|a|$ and $|b|$ we require a square root operation. Finally, to compute the correlation we need to find the inverse of the denominator $|a||b|$. CKKS scheme supports only addition and multiplication operation, in order to compute the square root and inverse we require a different approach. In [15] the authors propose an approximate inverse and square root operation using multiplications and additions. We employ the approximate inverse and square root algorithms proposed in [15] to compute the correlation using CKKS FHE schemes.

In the following subsections we will cover the details on the approximate inverse and square root algorithms.

$$r = \frac{\sum_{i=1}^{n} a_i b_i}{|a||b|} \tag{4.1}$$

4.1.1 Inverse in CKKS Scheme

We need an inverse operation to compute the Equation 4.1, we use a approximate inverse operation for this. The algorithm for computing the approximate inverse for real number values is based on the Goldschmidt's division algorithm [23]. Algorithm 3 **Inv(x;d)** shows the iterative method used to compute the approximate inverse for a given value x. For $x \in (0, 2)$ and a positive integer d, the error in the output value depends on the number of iterations d. As the number of iterations increases the approximate result from **Inv(x;d)** gets close to $1/x$ (error reduces) but is always smaller than $1/x$. Figure 4.1 give an example representation of the inverse operation in CKKS scheme. The input and output in CKKS scheme is in form of a matrix. The input for Algorithm 3 should be scaled to the range $0 > x > 2$, additional operations might be required like dividing the input by a large number to fit the range.

Algorithm 3 Inv(x;d) (Cheon, Jung Hee, Dongwoo, Lee and Lee, 2019 [15])

Input: $0 < x < 2, d \in N$
Output: an approximate value for 1/x
$a_0 \leftarrow 2 - x$
$b_0 \leftarrow 1 - x$
for $n \leftarrow 0$ **to** $d - 1$ **do**
    $b_{n+1} \leftarrow b_n^2$
    $a_{n+1} \leftarrow a_n \cdot (1 + b_{n+1})$
end for
Return: a_d

4.1.2 Square Root in CKKS

To compute the correlation using Equation 4.1 we need a square root operation. To compute the magnitude of $|a|$ and $|b|$ as shown in Equation 4.2, we use an approximate square root operation. The algorithm for computing the approximate square root for real number values is based on the Wilkes's iterative method [45]. Algorithm 4 **Sqrt(x;d)** shows the iterative method used to compute the approximate square root for a given value x. For $x \in (0, 1)$ and a positive integer d, similar to the **Inv(x;d)** operation the error in the output value depends on the number of iterations d. As the number of iterations increases the approximate result from **Sqrt(x;d)** gets close to $\sqrt{x}$ but is always smaller than $\sqrt{x}$. Figure 4.2 give an example representation of the

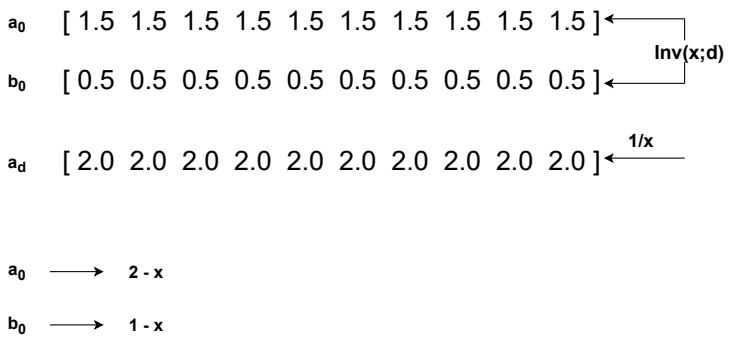


Figure 4.1: Example for finding the inverse in CKKS scheme based on Algorithm 3. x = 0.5 in this example.

the square root operation in CKKS scheme. The input for Algorithm 4 should be scaled to the range $0 > x > 1$, both the algorithms **Sqrt(x;d)** and **Inv(x;d)** need additional operations to fit the input in range.

$$|a| = \sqrt{\sum_{i=1}^{n} a_i^2} \tag{4.2}$$

Algorithm 4 Sqrt(x;d) (Cheon, Jung Hee, Dongwoo, Lee and Lee, 2019 [15])

Input: $0 \leq x \leq 1, d \in N$
Output: an approximate value for $\sqrt{x}$
$a_0 \leftarrow x$
$b_0 \leftarrow x - 1$
for $n \leftarrow 0$ **to** $d - 1$ **do**
  $a_{n+1} \leftarrow a_n(1 - \frac{b_n}{2})$
  $b_{n+1} \leftarrow b_n^2(\frac{b_n - 3}{4})$
end for
Return: a_d

4.2 Approximate Correlation in CKKS Scheme

Making use of the algorithms covered in Section 4.1.1 and 4.1.2 we build an implementation for finding the correlation between two audio signals in CKKS scheme. We give a description of the algorithm for finding the correlation between two audio tracks in Table 4.1. The inputs needed to find the correlation are time series values of the audio tracks. The encryption parameters for CKKS scheme are similar to the BFV and BGV schemes. Therefore, the definitions covered in Section 2.2.3 holds true for all three schemes. The CKKS scheme does have the *plaintext modulus* values and has an additional parameter called *scale* (See Section 2.2.4) which is used

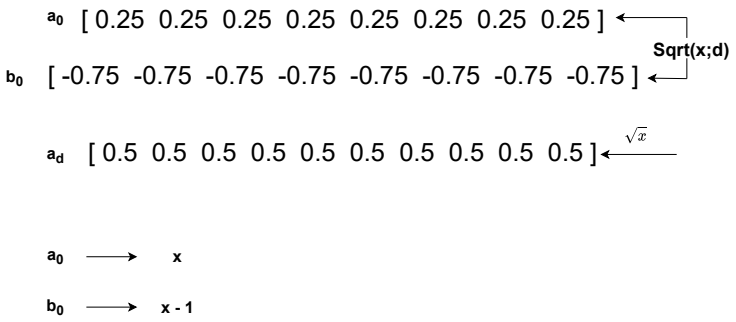


Figure 4.2: Example for finding the square root in CKKS scheme based on Algorithm 4. x = 0.25 in this example.

for precision and to *rescale* the ciphertext size. In Table 4.1 we give the encryption parameter values. In the algorithm the *polynomial modulus* value is 32768, this indicates the size of the input matrix. 32768 time series values from the audio track can be encoded into a single plaintext matrix and homomorphic operations can performed on the elements of the matrix simultaneously. The *polynomial modulus* value of 32768 defines a predetermined maximum *ciphertext modulus* value of 881 bits (See Table 2.2). This is also the noise budget, every homomorphic operation consumes some amount of noise budget. When the noise budget value reduces to 0, decryption will be false and we do not obtain correct results. The *scale* parameter determines the amount of noise introduced to the ciphertext. This allows to set predetermined precision for the outputs of CKKS operations. For example, if the *scale* parameter is set to 40, it allows about 10 bits of precision before and after the decimal point in the results.

In the correlation algorithm from Table 4.1 the input audio track time series value matrix is labeled $E0$ and $E1$. The correlation is computed using Equation 4.1 and the procedure can be split into 2 parts. First, the numerator part of the equation is computed. The $E0$ and $E1$ encrypted matrix are homomorphically multiplied and all the elements of the multiplication result matrix are summed up. The summation is performed using the Algorithm 1 described in Section 3.1. The second part, the denominator part of Equation 4.1 is computed. The square root algorithm from Algorithm 4 is used to find the magnitude of $E0$ and $E1$ using the Equation 4.2 and the magnitudes are multiplied together. The inverse of $|E0||E1|$ is computed next using Algorithm 3. Finally, the correlation is computed by multiplying the results from multiplying $a \cdot b$ and inverse of $|E0||E1|$. The output is in form of a matrix as shown in Figure 4.3.

4.3 Experiments and Results

In the previous sections we covered the different algorithms used for calculating the correlation in CKKS scheme. Since we cannot perform inverse or square root operations directly in homomorphic

Algorithm: Correlation of two audio tracks in CKKS scheme.

Encryption parameters: 1. Polynomial modulus - 32768

2. Ciphertext modulus - 881

2. Scale - 40

4. Input martix size - 2×16384 matrix

Inputs: Time series values of Audio track 1 ($E0$) and Audio track 2 ($E1$)

Outputs: Correlation score of $E0$ and $E1$

Procedure: 1. Initialise the encryption parameters and plaintexts required.

2. Multiply $E0$ and $E1$ and sum all the elements of the matrix and store the result in $S0$ (Numerator from Equation 4.1).

3. Compute the magnitude of $E0$ and $E1$ using the approximate square root operation from Algorithm 4.

4. Multiply the magnitudes $|E0|$ and $|E1|$ (Denominator from Equation 4.1).

5. Compute the approximate inverse for $|E0||E1|$ using the Algorithm 3 and store the result in $S0$

6. Multiply $S0$ and $S1$ and store the result in $R0$.

7. Output the correlation score $R0$.

Table 4.1: Iterative algorithm for calculating the correlation score of audio tracks.

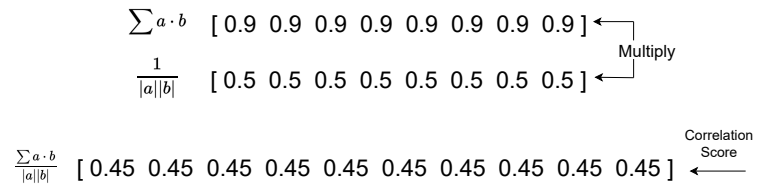


Figure 4.3: Example for finding the correlation in CKKS scheme.

domain, we looked into approximate inverse and square root algorithms that use multiplication and addition operations. In this sections we will present the datasets used for experiments and analyse the performance and accuracy of inverse and square root operations individually. Later, we describe the experiments for finding the approximate correlation between audio signals using CKKS scheme and lastly, we present and discuss our results.

4.3.1 Datasets

We use the same dataset that was used for evaluating the integer based schemes in Chapter 3. The GTZAN Genre Collection dataset [43] contains 1000 audio tracks of which we use 20 audio tracks from 10 different genres. Table 3.3 from Section 3.3.1 show the different genres of audio tracks used in our experiments. We use first 8000 time series values of the audio signals to compute our correlation. Further details of the audio data will be explained while describing the experiments for our correlation process in the later sections.

4.3.2 Approximate Inverse Operation Analysis

The results from the the inverse operation are approximate values. In this section we analyse the error and performance of the approximate inverse operation performed in the homomorphic domain using CKKS scheme. We implement the approximate inverse algorithm from Algorithm 3 using the CKKS scheme. As described in Section 4.1.1 the accuracy of the inverse operation depends on the number of iterations of the algorithm. To get an idea of the choice for the number of iterations, we analysed the performance by running the algorithm for up to 6 iterations as shown in Figure 4.4. We experiment with 10 values from 0 to 1 with a 0.1 increment after each value (0.1, 0.2, 0.3 and so on till 1). In Figure 4.4 the reference values are the accurate 1/x results computed in plaintext and the measured values are the approximate 1/x results obtained from computing inverse homomorphically using the CKKS scheme. We can see from the figures as the number of iterations increases the measured approximate values are converging closer to the inverse values. The Figures 4.4a, 4.4b and 4.4c show values from 1, 2 and 3 iterations in which the measured values have a noticeable deviation from the reference values for $x < 0.5$. Figure 4.5 shows the root mean square error (RMSE) values and processing time for different iterations of the inverse algorithm. These RMSE values were computed using Equation 4.3, where e is measured values, r is reference values and $N = 10$. The processing time were measured using the high resolution clock function in C++. We can observe in Figure 4.5 the RMSE value converges close to 0 from 4 iterations and after. However, from the computation time graph from Figure 4.5 we see as expected the processing time increases with the number of iterations. Figure 4.6 shows the error graph where the errors are computed by subtracting the reference values from the measured values. We observe irrespective of the number of iterations the error in measured

results are significantly smaller for values $x > 0.5$. This can be used to scale the input x to be $x > 0.5$ and perform the inverse operation with only 1 iteration and improve performance. For our implementation we use 4 iterations to compute the approximate inverse algorithm as we achieve reasonable accuracy and performance. The average time taken after 10 runs of the inverse algorithm for 4 iterations is 9.22668 seconds.

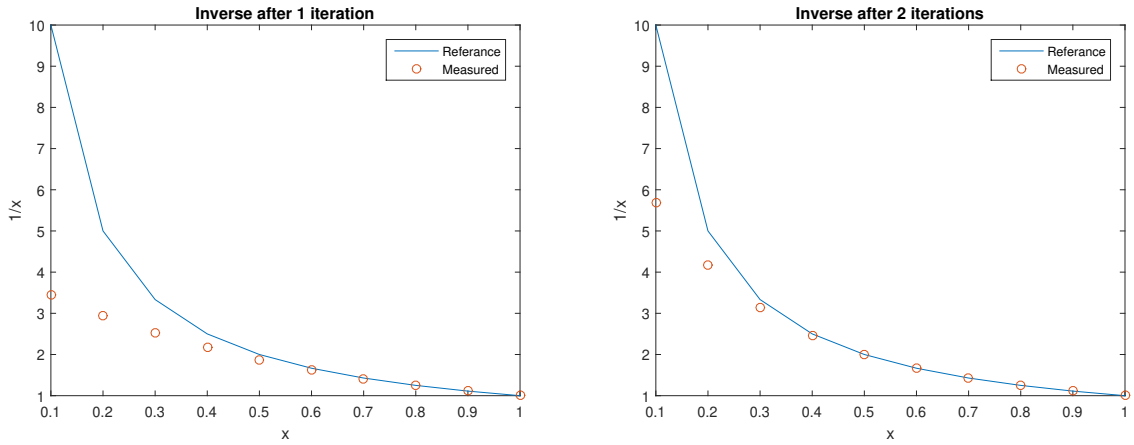
$$RMSE = \sqrt{\frac{\sum_i^N (e_i - r_i)^2}{N}} \tag{4.3}$$

4.3.3 Approximate Square Root Operation Analysis

In this section we analyse the square root operation to find the ideal iteration choice for our correlation using CKKS implementation. We implement our square root operations using Algorithm 4. The performance and accuracy of the square root result depends on the number iterations on the algorithm. In Figure 4.7 the accuracy of the square root operations after each iterations is illustrated. We experiment with 10 values (0, 0.04, 0.09, 0.16, 0.25, 0.36, 0.49, 0.64, 0.81, 0.96) for which we find the approximate square root. In Figure 4.7 the reference values are the accurate $\sqrt{x}$ results computed in plaintext and the measured values are the approximate $\sqrt{x}$ results obtained from computing square root homomorphically using the CKKS scheme. We can observe as the number of iterations increases the measured values coincide withe the reference values. Figures 4.7a and 4.7b show results for 1 and 2 iterations, the measured values have a significant deviation from the reference values. Figures 4.7c and 4.7d show results for 3 and 4 iterations, the measured values come closer to reference values for $x > 0.5$. Figures 4.7e and 4.7f show results for 5 and 6 iterations, the error in measured values in these figures is very small even for $x < 0.5$ values and the improvement in performance from 5 iterations to 6 iterations is not significant. Figure 4.8 shows the root mean square error (RMSE) and performance graph over multiple iterations. From this we observe that we can obtain good accuracy and performance trade-off for 5 iterations. For our implementation we use 5 iterations to compute the approximate square root algorithm as we achieve reasonable accuracy and performance trade-off. The average processing time computed after 10 runs for 5 iterations of the square root algorithm is 11.810 seconds.

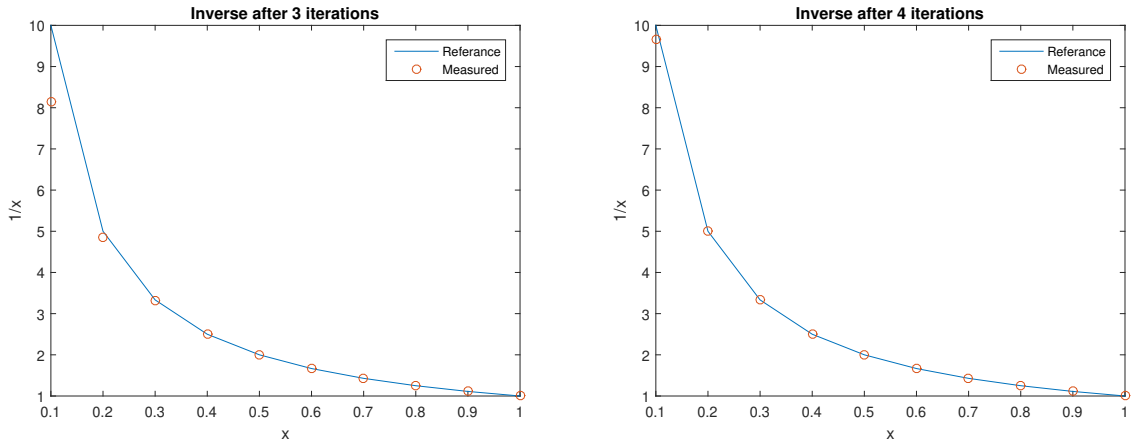
4.3.4 Experiments for Correlation Estimation

In Section 4.1 we discussed the algorithms for approximate inverse and square root operations and in Section 4.3.2 and 4.3.3 we analysed the approximation algorithms and found the optimal iterations for our correlation implementation. In this section, we describe the experiments for approximate correlation of audio signal data using CKKS scheme. Figure 4.9 gives the overview



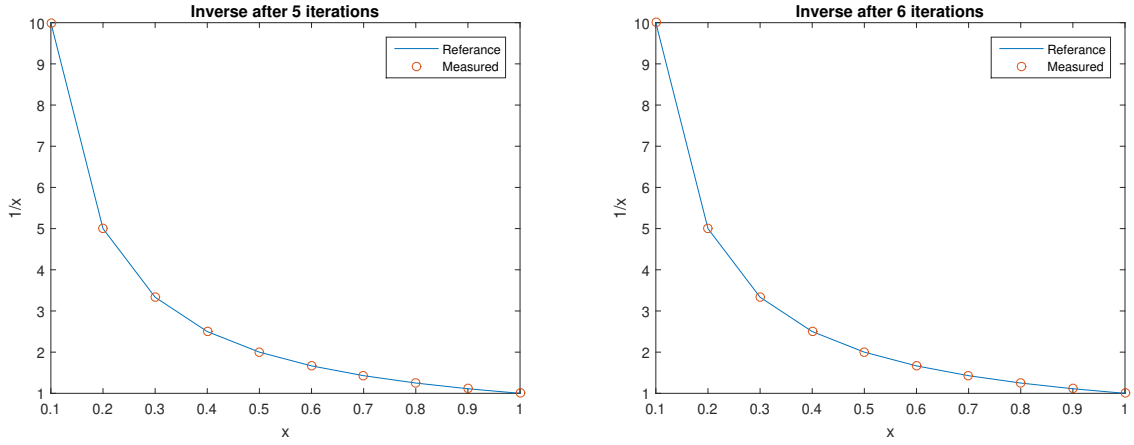
(a)

(b)



(c)

(d)



(e)

(f)

Figure 4.4: Approximate inverse values after each iteration. Reference values are 1/x calculated in plaintext without error, Measured values are 1/x calculated using CKKS scheme.

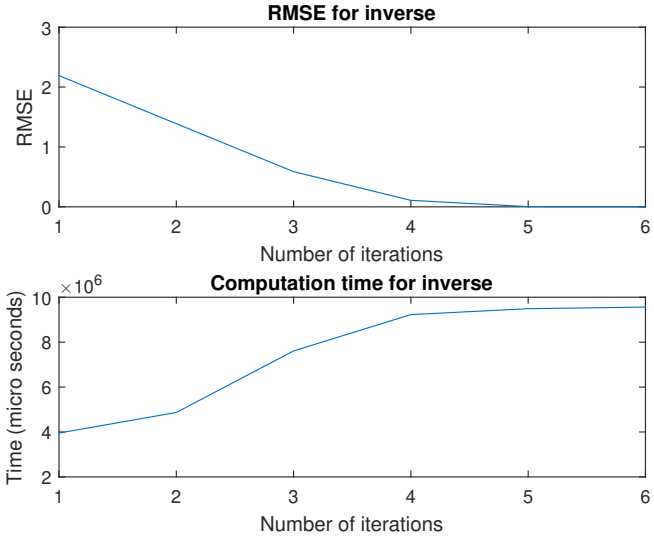


Figure 4.5: Inverse RMSE values and processing time for 6 iterations.

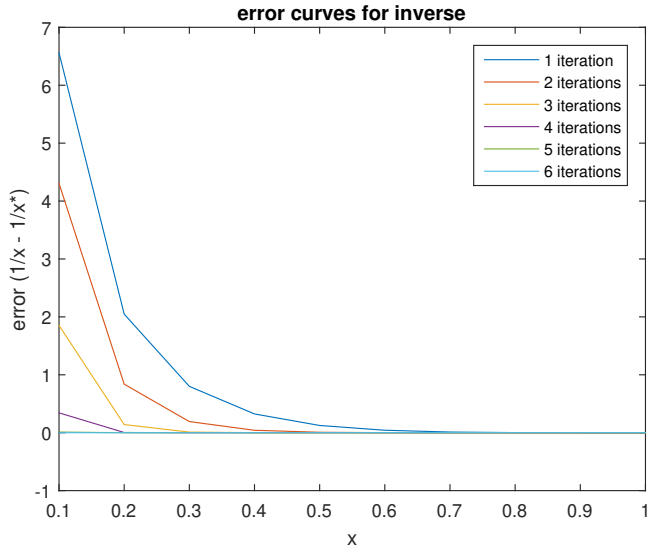
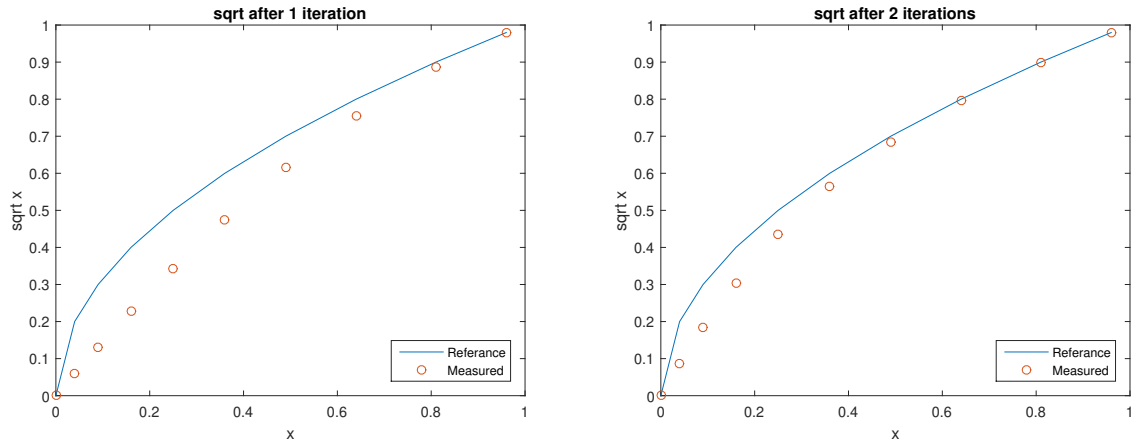
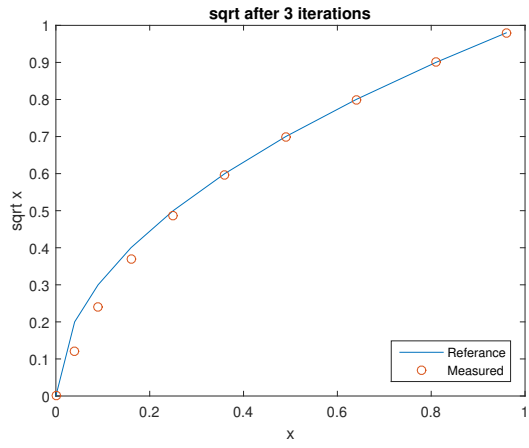


Figure 4.6: Error curve over 6 iterations. 1/x is calculated in plaintext and 1/x* is the approximate value calculated using CKKS scheme.

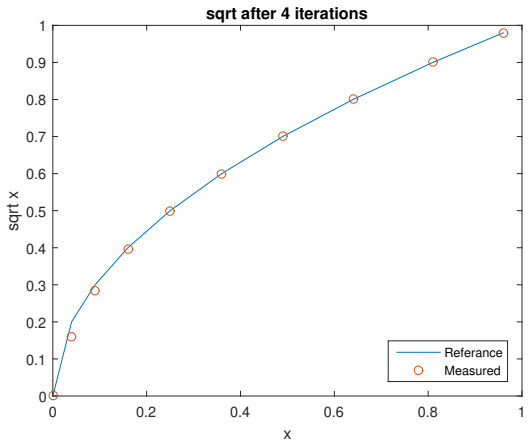


(a)

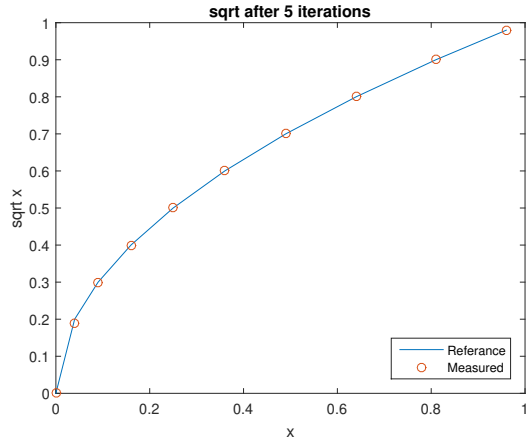
(b)



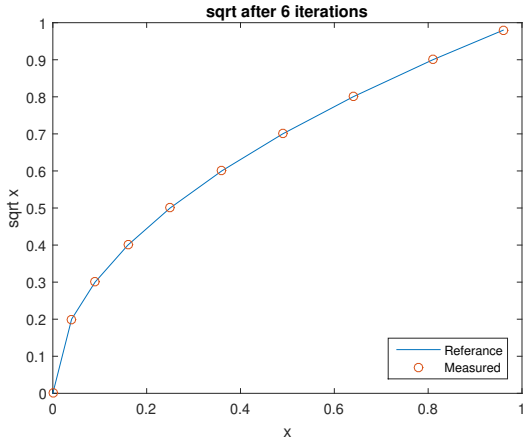
(c)



(d)



(e)



(f)

Figure 4.7: Approximate square root values after each iteration. Reference values are $\sqrt{x}$ calculated in plaintext without error, Measured values are $\sqrt{x}$ calculated using CKKS scheme.

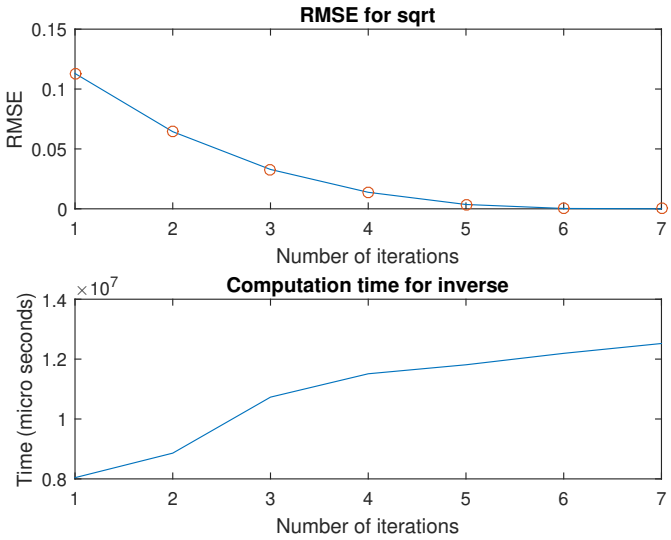


Figure 4.8: Square root RMSE values and processing time for 7 iterations.

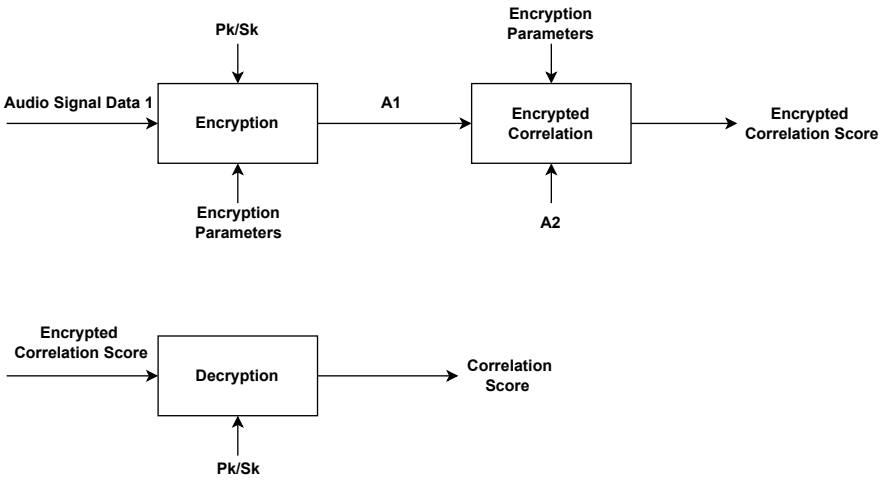


Figure 4.9: Schematics for computing the correlation score using CKKS. A1 and A2 are the audio signal data who's correlation is computed.

of the correlation experiments. The audio signal data for the audio tracks is encrypted, the encrypted correlation algorithm takes as input two encrypted audio signal data and outputs the correlations between them. This can be viewed as a client needing correlation between two audio tracks, the audio track time series values are encrypted input to the server. The server performs the encrypted correlation between the audio tracks and returns the correlation score.

Encryption Parameters	CKKS
polynomial modulus	32768
ciphertext modulus	881
scale	40

Table 4.2: Encryption parameters used for CKKS experiments.

4.3.4.1 Inputs and Outputs

We build our experiments according to the algorithms shown in Table 4.1. The inputs for are experiments are audio tracks from the dataset described in Section 4.3.1. The floating point time series values of the audio tracks are computed in python using the librosa library [32]. CKKS works with fixed-point values, therefore we round off our time series values to the 8th decimal place. We used 8000 time series values from the audio tracks as the audio signal data. The audio signal data is encoded into a plaintext matrix and then encrypted using the encryption parameters shown in Table 4.2. The output is the correlation score of the input encrypted audio signal data.

The *polynomial modulus* value is chosen to be 32768, this allows 32768 plaintext values to be encrypted in a single plaintext matrix. We encoded our 8000 time series values of the audio data in into the plaintext matrix. A large *polynomial modulus* is chosen to obtain a larger noise budget (See Table 2.2). The *ciphertext modulus* or noise budget is 881 bits. Since the implementation requires many multiplication and addition operations to support approximate inverse and square root operations a large noise budget is used. The *scale* value used for our experiments is 40, this gives 10 bits precision after the decimal point for our approximate CKKS scheme.

4.3.4.2 Correlation Estimation

The approximate correlation is implemented according to the procedure shown in Table 4.1 and Equation 4.1. The application was implemented using C++ programming language using the SEAL library [36]. Experiments performed for integer based schemes in Section 3.3 used an encrypted audio data base for similarity estimation. For the approximate correlation experiments we compute the correlation between two audio tracks at a time. This is because of the correlation operation has higher number of multiplication operations and requires longer processing time. In our implementation we use 4 iterations to compute the inverse and 5 iterations to compute the square root. We evaluate our results from our experiments based on the accuracy and processing time.

No.	Audio tracks	Correlation (reference)	Correlation (measured)	Error
1	blues 0	1.00000014	1.04977914	0.04977900
2	blues 0 gain	0.99116422	1.00479660	0.01363238
3	blues 0 invert	-1.00000014	-1.04977248	0.04977234
4	blues 1	0.03100886	0.03115996	0.00015109
5	classical 0	0.01207058	0.01223717	0.00016659
6	classical 1	-0.031578181	-0.03173782	0.000159639
7	country 0	0.095074045	0.09552808	0.000454035
8	country 1	0.015664266	0.01574113	0.000076864
9	pop 0	0.12493632	0.12554668	0.00061036
10	pop 1	0.04132448	0.04154284	0.00021836
11	metal 0	-0.01511109	-0.01518831	0.00007722
12	metal 1	0.00809020	0.00812981	0.00003961

Table 4.3: Correlation scores for *blues 0* audio track with 10 audio tracks from the dataset. Reference values calculated in plaintext and measured values are calculated homomorphically using CKKS scheme.

4.3.5 Results and Discussion

In this section we discuss the results for our CKKS correlation estimation experiments. All the experiments were implemented using Windows 10 operation system and Intel i5 2.30GHz processor with 8GB RAM. The results obtained from the CKKS correlation experiments are decrypted and analysed against correlation scores computed over plaintext. The error between the reference values calculated in plaintext and measured values calculated homomorphically is computed. The encryption parameters chosen for the experiments are optimised to enable correct decryption.

Table 4.3 and 4.4 show the correlation scores for *blues 0* and *pop 0* audio tracks with 10 tracks from the dataset. In Table 4.3 we can observe the measured correlation score obtained from calculating homomorphically for *blues 0* with itself is 1.04977914 and the reference correlation score calculated in plaintext is 1.00000014. The reference and measured correlation score are close to each other with an error of 0.04977900. We observe similar results for correlation scores of *pop 0* in Table 4.4. The correlation score for *blues 0* and *pop 0* tracks with phase inverted and gain modified (-10dB) versions of the tracks was also computed to check the correctness of the implementation. All the measured correlation scores from both the tables are similar to the reference scores up to 2 decimal places. In section 4.3.2 and 4.3.3 we analyse the accuracy of the inverse and square root operations implemented in the correlation estimation algorithm. In our implementation we use 4 iterations to compute the inverse and 5 iterations to compute the square root. The operations in CKKS scheme give us approximate results but the precision can be predefined using the *scale* parameter. For our implementation we define the *scale* parameter to be 40 which gives us 10 bits of precision after the decimal point. Since we round our inputs

No.	Audio tracks	Correlation (reference)	Correlation (measured)	Error
1	blues 0	0.12493632	0.12554629	0.00060997
2	blues 1	0.11250327	0.11307004	0.00056677
3	classical 0	0.03027052	0.03042199	0.00015147
4	classical 1	-0.00685206	-0.00692657	0.00007451
5	country 0	0.05619905	0.05648229	0.00028324
6	country 1	-0.02829657	-0.02843877	0.00014220
7	pop 0	1.00000001	1.00503106	0.00503105
8	pop 0 gain	0.99999995	1.01694382	0.06943825
9	pop 0 invert	-1.00000001	-1.00502980	0.00502979
10	pop 1	0.10764407	0.10818547	0.00054140
11	metal 0	-0.02268830	-0.02280257	0.00011427
12	metal 1	0.01874566	0.01884014	0.00009448

Table 4.4: Correlation scores for *pop 0* audio track with 10 audio tracks from the dataset. Reference values calculated in plaintext and measured values are calculated homomorphically using CKKS scheme.

to 8 decimal points (See section 4.3.4.1), the results from CKKS addition and multiplication operations are accurate. To further reduce the error in the correlation results, the number of iterations for the inverse and square root operations can be increased. This comes at the cost of increasing the computational time of the correlation process. Therefore in CKKS there is a trade off between accuracy and computational time. Each correlation operation between two audio tracks in CKKS takes on average 46.78 seconds to compute. The computation time was measured using high resolution clock function in C++ and the encryption and decryption times are not included. The inverse operation for 4 iterations and the square root operations for 5 iterations takes an average processing time of 9.22668 and 11.810 seconds respectively (All the processing times were computed by averaging over the timing values obtained after 10 runs). The processing time of the correlation estimation implementation can be improved by implementing more efficient algorithms for computing inverse and square root in the homomorphic domain.

Chapter 5

Conclusions and Future Work

In this thesis we presented similarity estimation methods for audio fingerprint data and audio signal data in the homomorphic encryption domain. We built a system which can perform the similarity estimation operation on encrypted audio data without leaking any information. To achieve this, three different fully homomorphic schemes were used for similarity estimation implementations. The integer based FHE schemes BFV and BGV were employed to compute similarity score between audio fingerprint data and the fixed-point based CKKS scheme was employed to compute correlation between audio tracks. We conducted experiments to test our FHE implementations and evaluated the schemes based on accuracy and computation time. For testing our systems, we created an encrypted database of different genre audio tracks. We constructed our experiments such that the party performing the fingerprint matching or correlation operations have no information about the audio data. Additionally, we implemented novel masking and sliding window techniques in homomorphic domain to compute fingerprint matching for arbitrary length data in BFV and BGV schemes. We also implemented inverse and square root operations homomorphically to facilitate correlation using CKKS scheme. According to our experiments the integer based schemes showed accurate results. The similarity score between audio fingerprints computed using the BFV and BGV schemes give us the same results as the similarity score computed over plaintext data. Analysing the performance of the BFV and BGV schemes from our experiments, we saw the BFV scheme has around 30% faster processing time compared to that of BGV scheme. This makes the BFV scheme overall a more efficient scheme for computing similarity score for audio fingerprint data. However, BFV and BGV schemes are restricted to only integer operations which can limit wider audio application.

From our correlation estimation experiments, we provide an analysis to the inverse and square root operations based on accuracy and computation time. According to our results we chose optimal inverse and square root algorithms for our implementation with efficient trade-off between accuracy and processing time. The results from our CKKS scheme experiments show

the correlation score for the audio tracks was close to the correlation score computed in plaintext. The correlation score for our implementation are accurate up to 2 decimal places. This precision can be improved further by increasing the accuracy of the inverse and square root algorithms at the cost of longer processing time. For computing the correlation between two audio tracks our implementations takes on average 46.78 seconds. The CKKS based scheme can operate over both integer and fixed-point values making it more practical for audio applications. The homomorphic algorithms and techniques for fingerprint matching and correlation operations proposed in this thesis can be used to construct different homomorphic systems with applications in music identification, speech recognition algorithms, telemedicine and so on.

The future work of the thesis can follow many directions. We implemented our fingerprint estimation using arithmetic circuit FHE schemes. An implementation using Boolean circuit schemes like Torus fully homomorphic scheme that operates directly over binary data could provide better insight into FHE feasibility with respect to different circuit schemes. The CKKS scheme also supports operations on complex numbers, operations like phase unwrapping can be implemented to extend the CKKS correlation implementation proposed in this thesis to perform time delay estimations on audio signals. There was no security analysis performed on our implementations, experiments could be conducted to test the security of our implementations against common attacks. Evaluation of the FHE techniques against other privacy preserving techniques like differential privacy and zero knowledge proof in the audio context would provide an understanding of the state of art in the privacy preserving field. In conclusion, the computational time seen in our homomorphic implementations are still very high and not practical for real world applications. However, homomorphic encryption schemes are getting more efficient and the processing time is reducing on average about 8 seconds every year. With a more online world and advancements in quantum cryptography, homomorphic encryption could be the solution to secure audio data.

Appendix A

Source Code

In this Chapter, the functions created during the writing of this thesis are reproduced. The headers provide the description of the functions, its input/output behavior and the libraries used. We cover our implementation of BFV, BGV and CKKS schemes. Both the python script for modifying the audio data and the C++ programs for implementing the applications are reproduced.

Function: Computing audio fingerprint and formatting it into a text file (BFV/BGV).
Inputs: Audio track
Outputs: Fingerprint text file
Libraries: fpcalc by Chromaprint [1], numpy, subprocess

```
import subprocess
import numpy as np

#compute the fingerprint for the audio track
fpcalc_out = subprocess.check_output('fpcalc -raw -length %i %s'
                                     % (500, "test/blues.00000.wav"))

fpcalc_out = str(fpcalc_out)
fingerprint_index = fpcalc_out.find('FINGERPRINT=') + 12

# convert fingerprint to list of integers
fingerprints = map(int, fpcalc_out[fingerprint_index:-5].split(','))
fingerprints = list(fingerprints)

#convert to binary
```

```
bin_fingerprints = []
for i in fingerprints:
    bin_fingerprints.append(bin(int(i))[2:])

#Format and modify fingerprint
bin_fingerprints = ''.join(bin_fingerprints)
bin_fingerprints = str(bin_fingerprints)
bin_fingerprints = " ".join(bin_fingerprints)
bin_fingerprints = bin_fingerprints.replace("0","786432")

#Write the formatted fingerprint into a text file
with open('blues.00000.txt', 'w') as f:
    f.write(bin_fingerprints)
```

Function: Summing the 1's of a matrix (See Algorithm 1)
Inputs: encrypted_matrix_1 and encrypted_matrix_2
Outputs: encrypted_matrix_2
Libraries: SEAL library [36]

```
for (size_t j = 0; j < 12; j++) {

    //Rotate rows in logarithmic order (2,4,16..)
    int num = pow(2, j);
    evaluator.rotate_rows_inplace(encrypted_matrix_2, num, galois_keys);

    //addition operation
    evaluator.add(encrypted_matrix_1, encrypted_matrix_2, encrypted_matrix_2);

    //storing intermediate sum
    encrypted_matrix_1 = encrypted_matrix_2;
    }
```

Function: Creating mask of a matrix in BFV and BGV(See Algorithm 2)
Inputs: *encrypted_matrix_1*,
sub_matrix ← 786432 (modulus - 1),
mul_matrix ← 393217 (multiplicative inverse of 0.5)

Outputs: *encrypted_matrix_mask*
Libraries: SEAL library [36]

```
//mask
evaluator.square(encrypted_matrix_1, encrypted_matrix_mask);

//relinearisation to reduce ciphertext size
evaluator.relinearize_inplace(encrypted_matrix_1, relin_keys);

//getting back the unmodified input
evaluator.sub_plain_inplace(encrypted_matrix_1, sub_matrix);
evaluator.multiply_plain_inplace(encrypted_matrix_1, mul_matrix);
evaluator.relinearize_inplace(encrypted_matrix_1, relin_keys);
evaluator.relinearize_inplace(encrypted_matrix_mask, relin_keys);
```

Function: Computing audio signal data and formatting it into a text file (CKKS).
Inputs: Audio track
Outputs: time series values text file
Libraries: liberosa [32], numpy

```
import librosa
import numpy as np

file = input("file name") #input audio track

#extract time series values of the audio track
sig, sample_rate  = librosa.load("path/"+ file + ".00000.wav",sr=None)

#roundoff values to 8th decimal place
sig = np.around(sig, 8)

#format data
sig = list(sig)
c = str(sig)
c = ''.join(c)
c = c.replace(",","")
c = c.replace("]","")
```

```
c = c.replace("[","")

#write data in text file
with open('path/'+ file + '.00000.txt', 'w') as f:
    f.write(c)
```

Function: Computing the inverse in CKKS (See Algorithm 3)
Inputs: *encrypted_matrix*_0,
add_matrix ← 1 ,
2_*matrix* ← 2
Outputs: *encrypted_matrix*_2
Libraries: SEAL library [36]

```
//Initial matrices
evaluator.negate_inplace(encrypted_matrix_1);
evaluator.add_plain(encrypted_matrix_0, add_matrix, encrypted_matrix_1);
evaluator.add_plain(encrypted_matrix_0, 2_matrix, encrypted_matrix_2);

for (int i = 0; i < 3; i++){

    // b_n+1 = (b_n)^2
    evaluator.square_inplace(encrypted_matrix_1);

    //Rescale to reduce ciphertext size
    evaluator.relinearize_inplace(encrypted_matrix_1, relin_keys);
    evaluator.rescale_to_next_inplace(encrypted_matrix_1);

    //changing parameters
    add_matrix.scale() = pow(2.0, 40);
    encrypted_matrix_1.scale() = pow(2.0, 40);
    parms_id_type last_parms_id1 = encrypted_matrix_1.parms_id();
    evaluator.mod_switch_to_inplace(add_matrix, last_parms_id1);

    // 1 + b_n+1
    evaluator.add_plain(encrypted_matrix_1, add_matrix, encrypted_matrix_2);
    encrypted_matrix_3.scale() = pow(2.0, 40);
    encrypted_matrix_2.scale() = pow(2.0, 40);
```

```
    parms_id_type last_parms_id2 = encrypted_matrix_3.parms_id();
    evaluator.mod_switch_to_inplace(encrypted_matrix_2, last_parms_id2);

    // a_n (1 + b_n+1)
    evaluator.multiply_inplace(encrypted_matrix_2, encrypted_matrix_3);
    evaluator.relinearize_inplace(encrypted_matrix_2, relin_keys);
    evaluator.rescale_to_next_inplace(encrypted_matrix_2);

    }
```

Function: Computing the square root in CKKS (See Algorithm 4)
Inputs: *encrypted_matrix*_1,
1_*matrix* ← 1 ,
2_*matrix* ← 0.5,
3_*matrix* ← 3,
4_*matrix* ← 4
Outputs: *encrypted_matrix*_3
Libraries: SEAL library [36]

```
//Initial matrix
evaluator.sub_plain(encrypted_matrix_1, 1_matrix, encrypted_matrix_2);

for (int i = 0; i < 5; i++){

    // (b_n)/2
    evaluator.multiply_plain(encrypted_matrix_2, 2_matrix, encrypted_matrix_3);

    //Rescale to reduce ciphertext size
    evaluator.relinearize_inplace(encrypted_matrix_3, relin_keys);
    evaluator.rescale_to_next_inplace(encrypted_matrix_3);

    //changing parameters
    1_matrix.scale() = pow(2.0, 40);
    encrypted_matrix_3.scale() = pow(2.0, 40);
    parms_id_type last_parms_id1 = encrypted_matrix_3.parms_id();
    evaluator.mod_switch_to_inplace(1_matrix, last_parms_id1);
    evaluator.negate_inplace(encrypted_matrix_3);
```

```
//(1 - (b_n)/2)
evaluator.add_plain_inplace(encrypted_matrix_3, 1_matrix);
encrypted_matrix_3.scale() = pow(2.0, 40);
encrypted_matrix_1.scale() = pow(2.0, 40);
parms_id_type last_parms_id2 = encrypted_matrix_3.parms_id();
evaluator.mod_switch_to_inplace(encrypted_matrix_1, last_parms_id2);

// a_n (1 - (b_n)/2)
evaluator.multiply_inplace(encrypted_matrix_3, encrypted_matrix_1);
evaluator.relinearize_inplace(encrypted_matrix_3, relin_keys);
evaluator.rescale_to_next_inplace(encrypted_matrix_3);
encrypted_matrix_1 = encrypted_matrix_3;
3_matrix.scale() = pow(2.0, 40);
encrypted_matrix_2.scale() = pow(2.0, 40);
parms_id_type last_parms_id3 = encrypted_matrix_2.parms_id();
evaluator.mod_switch_to_inplace(3_matrix, last_parms_id3);

// b_n -3
evaluator.sub_plain(encrypted_matrix_2, 3_matrix, encrypted_matrix_4);
encrypted_matrix_4.scale() = pow(2.0, 40);
4_matrix.scale() = pow(2.0, 40);
parms_id_type last_parms_id4 = encrypted_matrix_4.parms_id();
evaluator.mod_switch_to_inplace(4_matrix, last_parms_id4);

// (b_n -3)/4
evaluator.multiply_plain_inplace(encrypted_matrix_4, 4_matrix);

// (b_n)^2
evaluator.square_inplace(encrypted_matrix_2);
evaluator.relinearize_inplace(encrypted_matrix_2, relin_keys);
evaluator.relinearize_inplace(encrypted_matrix_4, relin_keys);
evaluator.rescale_to_next_inplace(encrypted_matrix_2);
evaluator.rescale_to_next_inplace(encrypted_matrix_4);
encrypted_matrix_2.scale() = pow(2.0, 40);
encrypted_matrix_4.scale() = pow(2.0, 40);
parms_id_type last_parms_id5 = encrypted_matrix_2.parms_id();
evaluator.mod_switch_to_inplace(encrypted_matrix_4, last_parms_id5);
```

```
// (b_n)^2 (b_n - 3)/4
evaluator.multiply_inplace(encrypted_matrix_2, encrypted_matrix_4);
evaluator.relinearize_inplace(encrypted_matrix_2, relin_keys);
evaluator.rescale_to_next_inplace(encrypted_matrix_2);
f2pt_matrix.scale() = pow(2.0, 40);
encrypted_matrix_2.scale() = pow(2.0, 40);
parms_id_type last_parms_id6 = encrypted_matrix_2.parms_id();
evaluator.mod_switch_to_inplace(2_matrix, last_parms_id6);

}
```

Appendix B

Software and Hardware

In this chapter, the software tools and hardware used in this thesis is described. All the experiments presented in this thesis were implemented using a PC with Windows 10 operation system and Intel i5 2.30GHz processor with 8GB RAM. The different tools and programming languages employed in this thesis are listed below along with the description of where in the implementation they were used.

No.	IDE	Programming language	Description
1.	Visual Studio Code version 1.70	Python 3.8	Generating audio fingerprint and signal data. Formatting data into a text file. Measuring correlation in plaintext.
2.	Visual Studio version 16.8.2	C++	Implementing similarity estimation experiments using SEAL v3.7 library.
3.	Matlab version R2015a	MATLAB	Computing RMSE results, generating RMSE, timing, probability and error graphs for CKKS experiments.

No.	Libraries	Programming language	Description
1.	SEAL v3.7	C++	BFV, BGV and CKKS implementations.
2.	Chromaprint version 1.5.1	Python 3.8	Generating audio fingerprints.
3.	Librosa version 0.9.2	Python 3.8	Generating time series audio data.

Bibliography

[1] AcoustID, *Chromarprint version 1.5.1.* audio fingerprint library, 2021. `https://acoustid.org/chromaprint`.

[2] M. Ajtai, *Generating hard instances of lattice problems*, in Proceedings of the twenty-eighth annual ACM symposium on Theory of computing, 1996, pp. 99–108.

[3] B. Alaya, L. Laouamer, and N. Msilini, *Homomorphic encryption systems statement: Trends and challenges*, Computer Science Review, 36 (2020), p. 100235.

[4] M. Albrecht, M. Chase, H. Chen, J. Ding, S. Goldwasser, S. Gorbunov, S. Halevi, J. Hoffstein, K. Laine, K. Lauter, S. Lokam, D. Micciancio, D. Moody, T. Morrison, A. Sahai, and V. Vaikuntanathan, *Homomorphic encryption security standard*, tech. rep., HomomorphicEncryption.org, Toronto, Canada, November 2018.

[5] M. S. Aliero, A. M. Ahmad, U. S. Kalgo, and S. A. Aliero, *An overview of internet of things: Understanding the issues and challenges of a more connected world*, International Journal of Computing and Communication Networks, 2 (2020), pp. 1–11.

[6] J. Athena and V. Sumathy, *Survey on public key cryptography scheme for securing data in cloud computing*, Circuits and Systems, 8 (2017), pp. 77–92.

[7] M. A. Bartsch and G. H. Wakefield, *Audio thumbnailing of popular music using chroma-based representations*, IEEE Transactions on multimedia, 7 (2005), pp. 96–104.

[8] S. Basu, A. Bardhan, K. Gupta, P. Saha, M. Pal, M. Bose, K. Basu, S. Chaudhury, and P. Sarkar, *Cloud computing security challenges amp; solutions-a survey*, in 2018 IEEE 8th Annual Computing and Communication Workshop and Conference (CCWC), 2018, pp. 347–356.

[9] Z. Brakerski, *Fully homomorphic encryption without modulus switching from classical gapsvp*, in Annual Cryptology Conference, Springer, 2012, pp. 868–886.

[10] Z. Brakerski, C. Gentry, and S. Halevi, *Packed ciphertexts in lwe-based homomorphic encryption*, in International Workshop on Public Key Cryptography, Springer, 2013, pp. 1–13.

[11] Z. Brakerski, C. Gentry, and V. Vaikuntanathan, *Fully homomorphic encryption without bootstrapping.* Cryptology ePrint Archive, Paper 2011/277, 2011. `https://eprint.iacr.org/2011/277`.

[12] Z. Brakerski and V. Vaikuntanathan, *Efficient fully homomorphic encryption from (standard) lwe*, SIAM Journal on computing, 43 (2014), pp. 831–871.

[13] L. Chen, L. Chen, S. Jordan, Y.-K. Liu, D. Moody, R. Peralta, R. Perlner, and D. Smith-Tone, *Report on post-quantum cryptography*, vol. 12, US Department of Commerce, National Institute of Standards and Technology ..., 2016.

[14] J. H. Cheon, A. Kim, M. Kim, and Y. Song, *Homomorphic encryption for arithmetic of approximate numbers*, in International conference on the theory and application of cryptology and information security, Springer, 2017, pp. 409–437.

[15] J. H. Cheon, D. Kim, D. Kim, H. H. Lee, and K. Lee, *Numerical method for comparison on homomorphically encrypted numbers*, in International Conference on the Theory and Application of Cryptology and Information Security, Springer, 2019, pp. 415–445.

[16] I. Chillotti, N. Gama, M. Georgieva, and M. Izabachène, *TFHE: Fast fully homomorphic encryption library*, August 2016. https://tfhe.github.io/tfhe/.

[17] I. Chillotti, N. Gama, M. Georgieva, and M. Izabachène, *Faster fully homomorphic encryption: Bootstrapping in less than 0.1 seconds*. Cryptology ePrint Archive, Paper 2016/870, 2016. `https://eprint.iacr.org/2016/870`.

[18] L. Ducas and D. Micciancio, *Fhew: Bootstrapping homomorphic encryption in less than a second*. Cryptology ePrint Archive, Paper 2014/816, 2014. `https://eprint.iacr.org/2014/816`.

[19] J. Fan and F. Vercauteren, *Somewhat practical fully homomorphic encryption*. Cryptology ePrint Archive, Paper 2012/144, 2012. `https://eprint.iacr.org/2012/144`.

[20] C. Gentry, *A fully homomorphic encryption scheme*, Stanford university, 2009.

[21] C. Gentry, C. Peikert, and V. Vaikuntanathan, *Trapdoors for hard lattices and new cryptographic constructions*, in Proceedings of the fortieth annual ACM symposium on Theory of computing, 2008, pp. 197–206.

[22] C. Gentry, A. Sahai, and B. Waters, *Homomorphic encryption from learning with errors: Conceptually-simpler, asymptotically-faster, attribute-based*. Cryptology ePrint Archive, Paper 2013/340, 2013. `https://eprint.iacr.org/2013/340`.

[23] R. Goldschmidt, *Applications of division by convergence. master's thesis*, MIT, (1964).

[24] B. D. U. G. W. Group et al., *Un handbook on privacy-preserving computation techniques*, 2019.

[25] S. Halevi and V. Shoup, *Design and implementation of helib: a homomorphic encryption library*, Cryptology ePrint Archive, (2020).

[26] Y. Iqbal, S. Tahir, H. Tahir, F. Khan, S. Saeed, A. M. Almuhaideb, and A. M. Syed, *A novel homomorphic approach for preserving privacy of patient data in telemedicine*, Sensors, 22 (2022), p. 4432.

[27] J. M. Khurpade, D. Rao, and P. D. Sanghavi, *A survey on iot and 5g network*, in 2018 International Conference on Smart City and Emerging Technology (ICSCET), 2018, pp. 1–3.

[28] A. Kim, Y. Polyakov, and V. Zucca, *Revisiting homomorphic encryption schemes for finite fields*, in International Conference on the Theory and Application of Cryptology and Information Security, Springer, 2021, pp. 608–639.

[29] R. Lai, X. Fang, P. Zheng, H. Liu, W. Lu, and W. Luo, *Efficient fragile privacy-preserving audio watermarking using homomorphic encryption*, in International Conference on Artificial Intelligence and Security, Springer, 2022, pp. 373–385.

[30] K. Laine, *Simple encrypted arithmetic library 2.3. 1*, Microsoft Research https://www. microsoft. com/en-us/research/uploads/prod/2017/11/sealmanual-2-3-1. pdf, (2017).

[31] A. López-Alt, E. Tromer, and V. Vaikuntanathan, *On-the-fly multiparty computation on the cloud via multikey fully homomorphic encryption*, in Proceedings of the forty-fourth annual ACM symposium on Theory of computing, 2012, pp. 1219–1234.

[32] B. McFee, A. Metsai, M. McVicar, S. Balke, C. Thomé, C. Raffel, F. Zalkow, A. Malek, Dana, K. Lee, O. Nieto, D. Ellis, J. Mason, E. Battenberg, S. Seyfarth, R. Yamamoto, viktorandreevichmorozov, K. Choi, J. Moore, R. Bittner, S. Hidaka, Z. Wei, nullmightybofo, A. Weiss, D. Hereñú, F.-R. Stöter, L. Nickel, P. Friesch, M. Vollrath, and T. Kim, *librosa/librosa: 0.9.2*, June 2022. Zenodo, `https://doi.org/10.5281/zenodo.6759664`.

[33] O. Regev, *On lattices, learning with errors, random linear codes, and cryptography*, Journal of the ACM (JACM), 56 (2009), pp. 1–40.

[34] R. L. Rivest, L. Adleman, M. L. Dertouzos, et al., *On data banks and privacy homomorphisms*, Foundations of secure computation, 4 (1978), pp. 169–180.

[35] K. Rohloff, D. B. Cousins, and D. Sumorok, *Scalable, practical voip teleconferencing with end-to-end homomorphic encryption*, IEEE Transactions on Information Forensics and Security, 12 (2016), pp. 1031–1041.

[36] *Microsoft SEAL (release 3.7)*. `https://github.com/Microsoft/SEAL`, Sept. 2021. Microsoft Research, Redmond, WA.

[37] C. Shi, H. Wang, Y. Hu, Q. Qian, and H. Zhao, *A speech homomorphic encryption scheme with less data expansion in cloud computing*, KSII Transactions on Internet and Information Systems (TIIS), 13 (2019), pp. 2588–2609.

[38] T. Shortell and A. Shokoufandeh, *Secure signal processing using fully homomorphic encryption*, in International Conference on Advanced Concepts for Intelligent Vision Systems, Springer, 2015, pp. 93–104.

[39] ——, *Secure fast fourier transform using fully homomorphic encryption*, (2016).

[40] R. Shrestha and S. Kim, *Integration of iot with blockchain and homomorphic encryption: Challenging issues and opportunities*, in Advances in Computers, vol. 115, Elsevier, 2019, pp. 293–331.

[41] N. P. Smart and F. Vercauteren, *Fully homomorphic simd operations*, Designs, codes and cryptography, 71 (2014), pp. 57–81.

[42] Y. Tang, B. Zhu, X. Ma, P. T. Mathiopoulos, X. Xie, and H. Huang, *Decoding homomorphically encrypted FLAC audio without decryption*, in ICASSP 2019-2019 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP), IEEE, 2019, pp. 675–679.

[43] G. Tzanetakis and P. C. Gtzan, *genre collection*. web resource, 2001. `http://marsyas.info/downloads/datasets.html`.

[44] M. van Dijk, C. Gentry, S. Halevi, and V. Vaikuntanathan, *Fully homomorphic encryption over the integers*. Cryptology ePrint Archive, Paper 2009/616, 2009. https://eprint.iacr.org/2009/616.

[45] M. V. Wilkes, D. J. Wheeler, and S. Gill, *The Preparation of Programs for an Electronic Digital Computer: With special reference to the EDSAC and the Use of a Library of Subroutines*, Addison-Wesley Press, 1951.

[46] Q.-Y. Zhang and Y.-J. Ba, *An adaptive speech homomorphic encryption scheme based on energy in cloud storage*, International Journal of Network Security, 24 (2022), pp. 628–641.

[47] M. Zuber, S. Carpov, and R. Sirdey, *Towards real-time hidden speaker recognition by means of fully homomorphic encryption*, in International Conference on Information and Communications Security, Springer, 2020, pp. 403–421.