# SECURE IDENTITY VERIFICATION USING CLOUD SERVICES AND FHE

**Deep Inder Mohan**

**Master of Technology Thesis**
June 2022

International Institute of Information Technology, Bangalore

# SECURE IDENTITY VERIFICATION USING CLOUD SERVICES AND FHE

Submitted to International Institute of Information Technology,
Bangalore
in Partial Fulfillment of
the Requirements for the Award of
Master of Technology

by

**Deep Inder Mohan**
**IMT2017013**

International Institute of Information Technology, Bangalore
June 2022

*Dedicated to*

*My Family: Mom, Dad, and Veera*

## Thesis Certificate

This is to certify that the thesis titled **Secure Identity Verification Using Cloud Services and FHE** submitted to the International Institute of Information Technology, Bangalore, for the award of the degree of **Master of Technology** is a bona fide record of the research work done by **Deep Inder Mohan**, **IMT2017013**, under my supervision. The contents of this thesis, in full or in parts, have not been submitted to any other Institute or University for the award of any degree or diploma.

Prof. Srinivas Vivek

Bengaluru,
The 2$^{\text{nd}}$ of June, 2022.

# SECURE IDENTITY VERIFICATION USING CLOUD SERVICES AND FHE

## Abstract

National digital identity verification systems have played a crucial role in the effective distribution of goods and services, particularly, in developing countries. Due to the cost involved in deploying and maintaining such systems, combined with the lack of in-house technical expertise, governments seek to outsource this service to third-party cloud service providers to the extent possible. This leads to increased concerns regarding individual users' personal data privacy. In this work, we propose a practical privacy-preserving digital identity verification protocol where the third-party cloud services process encrypted identity data using an asymmetric fully homomorphic encryption (FHE) scheme such as BFV. Though the role of a trusted entity such as government is not completely eliminated, our protocol significantly reduces the computation load on such parties. We implement our protocol using the Microsoft SEAL FHE library and demonstrate that the demographic and biometric matching queries and secure age comparisons can be efficiently performed on batched FHE ciphertexts.

# Acknowledgements

I would like to express my deepest gratitude to my advisor, Prof. Srinivas Vivek, not only for his guidance but also for his patience and support. I am extremely grateful to Mr. Sanath V. and the entire MOSIP team for their trust and for funding this research. I am deeply indebted to Prof. Shrisha Rao, who molded me into a better researcher by blessing me with his guidance and wisdom. I am thankful to Prof. Ashish Choudhury, who introduced me to the wonderful field of cryptography. I would also like to recognize all the IIITB professors I have had the pleasure of being taught by, as their lessons gave me the intellectual maturity needed to undertake this endeavor.

I could not have undertaken this journey without the help of my brother, Dr. Nitinder Mohan, who had the answer to every question I could ask. I would also like to acknowledge all my friends whose confidence and belief in me gave me the strength to persevere. Finally, I would like to thank my parents: my father, Dr. H. R. Verma, for inspiring me to be a great researcher, and my mother, Baljit Kaur, for her constant love and support.

# Contents

# List of Figures

# List of Tables

# List of Abbreviations

| | |
|---|---|
| **2PC** | Two-Phase Commit Protocols |
| **BFV** | Brakerski/Fan-Vercauteren |
| **BGV** | Brakerski-Gentry-Vaikuntanathan |
| **CKKS** | Cheon-Kim-Kim-Song |
| **CRT** | Chinese Remainder Theorem |
| **CS** | Central Server |
| **FHE** | Fully Homomorphic Encryption |
| **HE** | Homomorphic Encryption |
| **IND-CPA** | Indistinguishable under Chosen Plaintext Attack |
| **MB** | Megabyte |
| **PHE** | Partially Homomorphic Encryption |
| **SEAL** | Simple Encryption Arithmetic Library |
| **SIMD** | Single Instruction, Multiple Data |
| **SP** | Service Provider |
| **SWHE** | Somewhat Homomorphic Encryption |
| **TFHE** | Fast Homomorphic Encryption over the Torus |
| **TPS** | Third-Party Server |

# CHAPTER 1

# INTRODUCTION

## 1.1 Problem Description

A country's identification (ID) system plays a critical role in the effective delivery of public and private services. Governments are exploring the development of multipurpose foundational digital ID systems where individuals receive a unique identifier that they can use for identity verification. Over the last decade, some developing nations have pioneered their own digital identity verification systems, such as India with AADHAR [1]. The success of these systems has brought to the forefront the impact that national digital identity verification systems that use biometric verification can have on empowering the poor and most vulnerable sections of society; from the distribution of rations at subsidized rates to enabling adults to find employment through employment guarantee schemes, etc. Digital identity verification systems can ensure transparent and equitable distribution of the funds and benefits from government social welfare programs and thus help the governments of developing economies utilize their scarce development funds more effectively and efficiently. Systems such as AADHAR have become strategic policy tools for social and financial inclusion, public sector delivery reforms, managing fiscal budgets, increasing convenience, and promoting hassle-free people-centric governance [1].

A nation deploying a digital ID system must spend considerable resources for its design, deployment, and maintenance. Many developing countries may not be able to afford the design of such systems from scratch. Motivated by these needs, the Modular Open Source Identity Platform (MOSIP) project [2], anchored at IIIT Bangalore, assists governments and other user organizations in implementing a digital, foundational identity system in a cost-effective way. The Governments of Morocco, Philippines, and Ethiopia have adopted this platform, and many countries across Asia, Africa, and Latin America have expressed interest [3]. Currently, about 57 million users have been enrolled in MOSIP-based systems [2].

To ensure the availability of ID services and to reduce the burden of maintenance on central authorities, users' demographic and biometric data are often outsourced to third parties. The number of third-party cloud services involved depends on the usage of the ID service by service providers and the latter could bear the cost of these cloud services, thereby, making it economically viable for central authorities. However, this leads to serious privacy concerns and potential misuse of personal data [4]. In this work, we aim to provide a practical solution to the above privacy concern by making use of fully homomorphic encryption to provide confidentiality to the demographic and the biometric data outsourced to a Third-Party Server(s) (TPS).

We make use of a asymmetric-key word-wise FHE scheme such as BFV [5], and we still need a (fully trusted) Central Server (CS) (e.g., the ID issuing authority) to decrypt the encrypted data. But, we significantly reduce the burden on CS by outsourcing most of the computations on encrypted data to TPS. Although bit-wise FHE schemes such as TFHE [6] can homomorphically evaluate Boolean operations much faster than BGV/BFV, their storage requirements are a couple of orders of magnitude larger ($>$ 150 MB per user record). This is due to the encryption of each bit of the plaintext data individually, which becomes very expensive when the number of users runs into hundreds of millions. Even in scenarios where the number of users is limited, this high ciphertext expansion ratio makes it impractical to communicate ciphertexts over networks. On the contrary, we make use of ciphertext packing in BFV and, hence, make judicial use of the available plaintext space. While previous works have demonstrated that Partially Homomorphic (PHE) schemes can provide secure and efficient demographic and biometric matching [7, 8, 9, 10], our choice to go with FHE schemes is motivated by the need to design a scalable system that can handle a large variety of queries in the future. Our contributions are formally summarised in Section 1.3.

## 1.2 Existing Approaches

Homomorphic Encryption techniques have received wide-scale adoption in the creation of practical secure outsourced computation applications. HE based techniques have been applied to achieve privacy-preserving set operations [11, 12, 13, 14] and matrix operations [15, 16, 17]. They have also been successful in privacy-preserving machine learning and data mining, with HE-based algorithms being developed for regression [18, 19, 20] and classification [21, 22, 23, 24] tasks. HE-based approaches also exist for more advanced applications, such as artificial neural networks [25, 26] and secure image processing [27, 28, 29].

Although not in the context of national ID systems, many applications have used homomorphic encryption techniques for the purposes of biometric authentication. In [30], the authors provide a framework for the secure verification of multiple biometric templates. Their approach involves the fusion of multiple biometric templates by either concatenating each template vector or processing each template individually and normalizing the individual scores. They use the Pallier cryptosystem [31] to encrypt their templates and implement algorithms for both encrypted Euclidean distance and encrypted cosine similarity. Their scheme uses the fingercodes template and their secure algorithm for fingerprint matching performs without penalty to the error rate. Although their algorithm is fast, performing a comparison in only 0.5 ms with GPU acceleration, their use of PHE makes this scheme unsuitable for our requirements. Furthermore, this scheme requires the user's collected biometric template to be sent to the server for verification in plain, which exposes the scheme to leakage attacks from honest-but-curious adversaries.

In [32], they implement template quantization of the fingercodes template to achieve a low-bandwidth privacy-preserving Pallier-based scheme. However, their per-comparison time is $> 4$ seconds, making this protocol too slow for our task. In [33], the scheme is created to target a similar privacy model to ours. They also parallelize the execution of their Euclidean distance algorithm to achieve faster multi-threaded performance. However, their scheme requires multiple rounds of communication between the central server and the third-party server, making it impractical for our objective of minimizing CS involvement. Furthermore, their matching method execution time with 192-bits of security and a polynomial modulus value of 8192 is  1.2 seconds. In the same settings, our algorithm executes in about 0.3 seconds, making it nearly four times as fast.

In [34], the authors introduce GSHADE, a protocol for the privacy-preserving computation of distance metrics such as Euclidean distance. This protocol works in the 2PC setting and uses correlated Oblivious Transfer as a primitive to exchange information between the client and server. This protocol is used in [35] to develop a biometric authentication protocol. In this work, the authors develop a blinding technique to allow the fingerprint data to be obscured from the authenticator. Blinding allows their application to leverage incredibly fast distance computation from GSHADE for biometric authentication. However, the privacy-preserving login procedure needed in this protocol is stateful and required 3 rounds of communication, making it impractical for our task.

[36] presents a TFHE-based privacy-preserving fingerprint authentication system.

The authors implement arithmetic operations using logic gates to calculate Euclidean distance values across fingercode vectors. Their algorithms offer similar performance to ours, taking about 0.33 seconds per comparison. However, due to the use of TFHE, their ciphertext expansion ratio is extreme ($> 5000$) and the size of each ciphertext is in the order of megabytes. Since our system architecture requires all communication to be encrypted, this makes this protocol impractical for online-based authentication systems.

In addition to privacy-preserving distance computation, certain existing works also explore efficient HE-based comparison algorithms. In [37], the authors introduce a polynomial rank sort algorithm for efficiently sorting ciphertexts encrypted with the BGV [38] scheme. Their algorithm implements SIMD operations using the same CRT-batching technique as our work. However, their algorithm has a high circuit depth ($> 10$), and their comparison operations are too slow for real-time verification purposes. In [39], the authors implement fast integer comparison algorithms for BGV and BFV schemes. This result offers state-of-the-art performance for homomorphic comparison, with their scheme taking only 11 milliseconds to compare two 64-bit integers. However, the system uses a very specific integer representation which is not generalizable to other comparison tasks.

The task of privacy-preserving identity matching has also been tackled in existing research through various techniques. The OLYMPUS framework [40] implements a privacy-preserving identity management framework which is complaint with the EU GDPR. The primary technical contribution of this framework is to provide surveillance protection against a malicious identity provider by distributing its role to several virtual identity providers. The project, however, has been developed primarily for password-based authentication, making their techniques inapplicable for our stateless framework.

In [41], the authors present a privacy-preserving biometric identification scheme targeted toward IoT devices based on zero-knowledge proofs. Although the scheme is theoretically proven to be perfectly complete and perfectly zero-knowledge, it requires a large computation and communication overhead for each comparison, making it unsuitable for practical deployment. A similar protocol is presented in [42] which uses non-interactive zero-knowledge and permissioned blockchains. The zero-knowledge prover in this protocol executes its proof in $> 3$ in the 128-bit security setting, making it more than $10\times$ slower that the protocol presented in this thesis.

In [43], the authors present `BlindIdM`, a privacy-preserving cloud-based framework that achieves Identity Management as a Service (IDaaS). This scheme follows a similar model to ours as it involves a semi-honest third-party which can *blindly* verify user identity. The scheme is based on the Security Assertion Markup Language (SAML 2.0) and requires multiple rounds of communication between the Service Provider (SP) and the cloud-based third party for identity verification.

[44] presents a formal protocol for outsourced privacy-preserving biometric matching using FHE. The protocol setup in this scheme is similar to the one presented in this thesis, with the absence of a trusted central server. Furthermore, this protocol is formally proved to be *IND-CPA* secure. However, the authors only create naive implementations of the comparisons, resulting in their algorithms requiring several minutes per comparison when using the TFHE encryption scheme.

## 1.3 Our Contributions

In this thesis, we propose a practical privacy-preserving digital identity verification protocol involving SP, CS, and TPS (see Figure FC3.1). Our protocol supports the matching of encrypted demographic and biometric data. Only the hash of the concatenated user data is in the clear to allow quick indexing, by TPS, to locate his/her encrypted data. Our protocol also supports private age *comparisons* as some services can only be offered to users in a certain age range.

The main technical idea used in making our protocol efficient on the CS side is to design an ”extended and query transparent” FHE decryption circuit that decrypts intermediate encrypted values resulting from low multiplicative-depth circuit evaluation on ciphertexts, thereby, outsourcing the rest of the computations to TPS or SP in a secure manner (see Section 4.2). The extended part of the decryption circuit basically does the necessary Boolean operations on plaintexts very efficiently. One of the advantages of an extended decryption circuit being independent of the type of query is that it facilitates cheaper hardware implementation. To facilitate this query transparent extended decryption, we propose a novel plaintext encoding mechanism, particularly, for the age comparison query.

We have implemented our protocol using the Microsoft SEAL library [45]. The ciphertext of each individual user's data is only 0.864 MB (for 192-bit security level) as we deploy FHE ciphertext batching and it takes at most 0.04 seconds of computing time of TSP for demographic match queries, 0.3 seconds for age comparison queries, and 0.3 seconds for the biometric match query. The computing time for CS is less than 5 milliseconds per query. The communication overhead is also very less due to the small size of data. We will make our implementation code publicly available after due approval.

The rest of this thesis is organized as follows: In Chapter 2, we give a brief description of Homomorphic Encryption and the different types of HE schemes. Chapter 3 goes over the proposed architecture for our system, giving detailed descriptions of each party involved. In Chapter 4, we describe the algorithms for homomorphically processing our query types, and give correctness proofs where necessary. Chapter 5 goes over our implementation results and processing times, and we give a detailed description of the scheme's security in Chapter 6.

# CHAPTER 2

# HOMOMORPHIC ENCRYPTION

## 2.1 What is Homomorphic Encryption?

Homomorphic Encryption, or HE, is a cryptographic primitive through which it becomes possible to carry out binary or arithmetic operations over encrypted data without needing to decrypt the data. Like other encryption schemes, HE schemes can also be based on either the symmetric or asymmetric key paradigm. For the purpose of this thesis, our focus will be on asymmetric HE schemes. These encryption schemes have separate keys for encryption (public key *pk*) and decryption (secret key *sk*) operations. However, both symmetric and asymmetric HE schemes can be used in both contexts, as methods for transforming symmetric HE to asymmetric HE and vice-versa have already been demonstrated in [46].

In general, asymmetric HE schemes can be said to involve the following operations:

- $\underline{\texttt{ParamGen}(\lambda, PT, K, B) \rightarrow params}$: This algorithm is used to instantiate the parameters for the HE scheme. Its inputs are:
  - $\lambda$: This is the security parameter that determines the security level of the scheme. For example, $\lambda = 128$ denotes 128-bits of security.
  - $PT$: This denotes the plaintext space. Examples of $PT$ for HE schemes are

  $\mathbb{Z}$ (integers), $\mathbb{Z}_p$ (integers modulo prime $p$) or extension rings/fields.

  – $K$: This denotes the dimension of the encrypted vector. This only applies if the scheme supports SIMD (Single Instruction, Multiple Data) operations across vectors. For the BFV HE scheme, this value is determined by the ciphertext modulus.

  – $B$: This controls the complexity of the circuits that can be executed using the HE scheme. Lower parameters support only smaller circuits or less complex algorithms. This allows the ciphertexts to be smaller and the evaluation to be more efficient, Higher parameters allow for computing more complex functions/algorithms but come with increased ciphertext sizes and evaluation times.

- $\underline{\texttt{KeyGen}(params) \rightarrow pk, sk}$: For input parameters *params*, this function outputs a public key *pk* and a secret key *sk*. Depending on the specific HE scheme, some other keys such as evaluation key or a Galois key may also be generated.

- $\underline{\texttt{Enc}_{pk}(m) \rightarrow c}$: For a plaintext message $m$, output its ciphertext encryption $c$.

- $\underline{\texttt{Dec}_{sk}(c) \rightarrow m}$: For a ciphertext $c$, output its plaintext decryption $m$.

- $\underline{\texttt{Eval}_{\star}(c_1, c_2) \rightarrow c_{\star}}$: An evaluation function for an operation '$\star$' that evaluates this operation on the underlying plaintexts for two ciphertexts $c_1$ and $c_2$, and outputs the corresponding ciphertext $c_{\star}$.

An asymmetric encryption scheme can be defined as being homomorphic over an operation '$\star$' if the above functions satisfy the following equation:

$$\texttt{Dec}_{sk}\Big(\texttt{Eval}_{\star}\big(\texttt{Enc}_{pk}(m_1), \texttt{Enc}_{pk}(m_2)\big)\Big) = m_1 \star m_2, \ \ \forall\, m_1, m_2 \in PT$$

## 2.2 Types of HE Schemes

Homomorphic encryption schemes can be generally categorized to belong to one of three types:

1. **Partially Homomorphic Encryption (PHE)**: These schemes allow for the computation of only one type of operation. However, these schemes allow this operation to be performed an unlimited number of times without compromising the accuracy of the ciphertexts. This operation can be integer addition, as the schemes by Pallier [31] and Benaloh [47]; integer multiplication as the RSA [48] and the El-Gamal [49] cryptosystems or binary addition as the public-key scheme by Goldwasser and Micali [50].

2. **Somewhat Homomorphic Encryption (SWHE)**: These schemes can allow multiple operations on the ciphertext, allowing the computation of more complex circuits. However, they come with the caveat of only being able to perform these operations a limited number of times. The first SWHE scheme was the BGN [51] scheme, which allows for the computation of unlimited additions, but only a single multiplication operation on the ciphertexts.

3. **Fully Homomorphic Encryption (FHE)**: FHE schemes support an unlimited number of operations on ciphertexts, and these can be applied an unlimited number of times. Most schemes only support two operations on ciphertexts, such as addition and multiplication. However, since every mathematical circuit can be represented using only these two operations, it becomes possible to evaluate every circuit with these schemes, making them *fully* homomorphic. The first FHE scheme was achieved by Gentry in 2009 [52] for which he created an innovative *bootstrapping* operation to preserve ciphertext sizes after multiplication. More recently, advances in FHE schemes have made them viable tools for preventing

the leakage of sensitive data such as biomedical or financial data.

## 2.3 SEAL: Simple Encrypted Arithmetic Library

The Simple Encrypted Arithmetic Library, or SEAL, is a free and open-source software library developed by Microsoft Research. MS SEAL implements the BFV and CKKS FHE schemes. Due to being free, open-source, cross-platform, and without external dependencies, SEAL is the ideal library for applications involving FHE. This library also includes automatic parameter selection and noise estimator tools. This makes SEAL easier to use than libraries such as HElib, which due to its low-level implementation requires sophisticated parameter selection.

The BFV scheme implements a fully homomorphic Ring-Learning With Errors (RLWE) based cryptosystem. BFV is instantiated over two separate rings - a plaintext ring and a ciphertext ring. This scheme supports addition and multiplication operations over modular integer ciphertexts. Furthermore, the BFV scheme supports SIMD (Single Instruction, Multiple Data) operations using a technique called CRT-batching [53]. Batching uses the Chinese Remainder Theorem to encode a vector of multiple plaintext messages as a single plaintext. All homomorphic operations performed on the encryption of this *batched* plaintext reflect back to each individual plaintext. Therefore despite the per-gate evaluation speed of BFV being slower than bitwise homomorphic schemes such as TFHE, we can achieve fast amortized costs for our algorithms by batching and parallel processing our inputs.

The CKKS scheme [54] allows for operations on encrypted real and complex numbers. Due to not having the algebraic constraints of BFV, CKKS ciphertexts can pack a lot more data. This makes this scheme the most efficient in terms of amortized cost per operation. However, the results yielded in this scheme are only approximate, making it unsuitable for our application.

# CHAPTER 3

# SYSTEM ARCHITECTURE

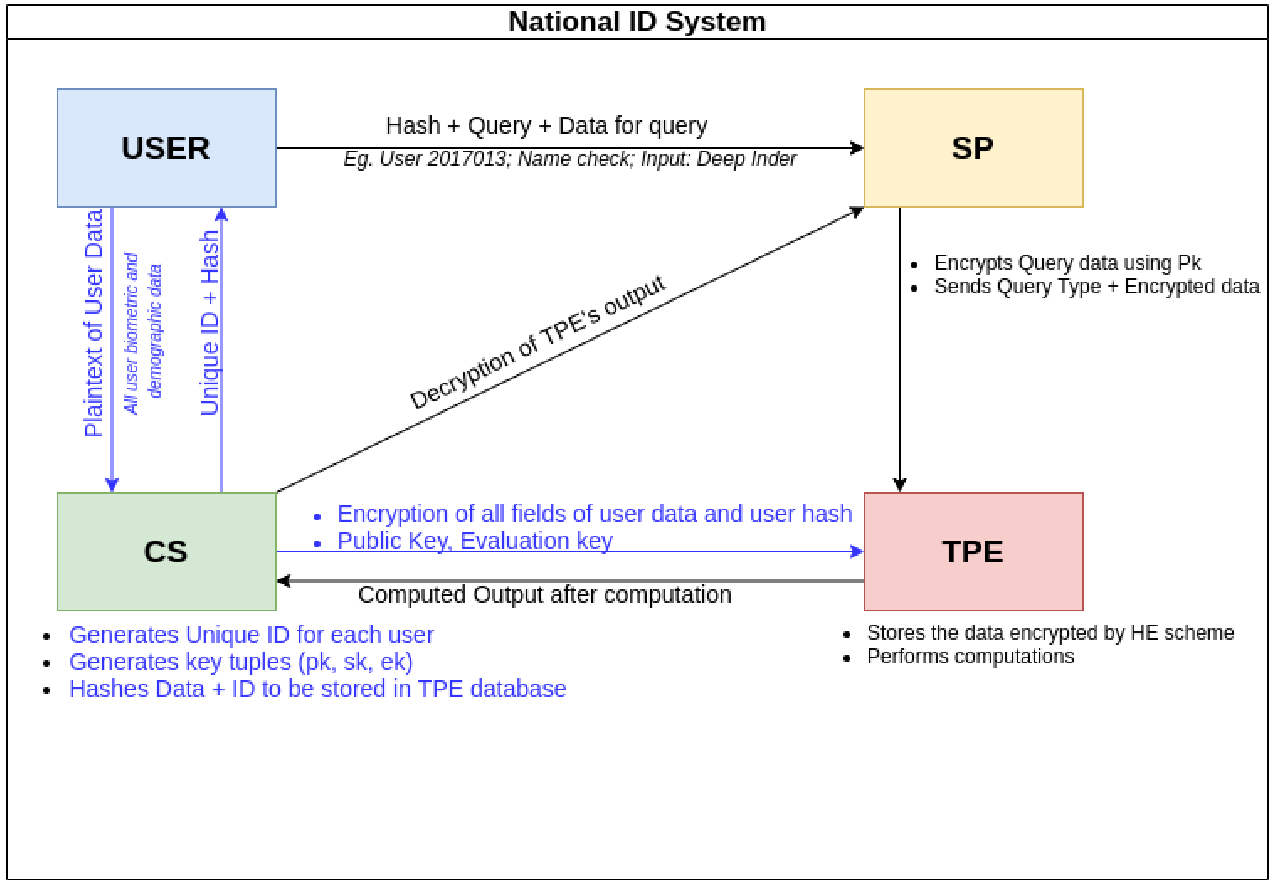
In this section, we describe the overall architecture of our system, including all the parties involved, their respective roles, and the format of data storage.

## 3.1 System Architecture

The following are the entities that are involved in our proposed system. Their roles and relationships are also summarised in Figure FC3.1.

- **User**: The user is the owner of the demographic data. After the initial registration process, the user will provide his demographic/biometric information to SPs for identity verification and availing of various services.
- **Service Providers (SP)**: These are entities that require identity verification for providing a service. They can be government organizations, such as ration distribution centers and license & registration offices, or private companies such as telecom or gas providers. They collect user data and generate queries on this data, which in turn get forwarded to TPSs.
- **Third-Party Servers (TPS)**: These are third-party servers that may be located anywhere on earth. They provide storage and computation services to the system.

Figure FC3.1: Flowchart of system architecture. The blue text/arrows represent the flow during initial user registration, and the black text/arrows represent the flow during query computation


National ID System
USER
SP
CS
TPE
Hash + Query + Data for query
Eg. User 2017013; Name check; Input: Deep Inder
Plaintext of User Data
All user biometric and demographic data
Unique ID + Hash
Decryption of TPE's output
Encrypts Query data using Pk
Sends Query Type + Encrypted data
Encryption of all fields of user data and user hash
Public Key, Evaluation key
Computed Output after computation
Generates Unique ID for each user
Generates key tuples (pk, sk, ek)
Hashes Data + ID to be stored in TPE database
Stores the data encrypted by HE scheme
Performs computations

They receive the queries from SPs, fetch the appropriate user records and perform computations on the encrypted data itself. They then send the output of these encrypted computations to the government CS.

- **Central Servers (CS)**: These are the secure servers that will be owned by the governments of the nations deploying the ID system. Since the system uses the public-key encryption paradigm, these are the only entities that will possess the secret keys for our FHE algorithm, and hence will be the only point at which decryption may be carried out. They will receive the encrypted query output from TPS and perform an ”extended” decryption operation on it. They can then pass the decrypted output through a deterministic circuit which will output if the query has passed or failed. This result can then be forwarded to the SP that issued the query. By making our post decryption computations deterministic and query agnostic, these servers can be made very efficiently and will require minimal hardware investment to erect.

## 3.2 Need for TPS and CS

In the proposed architecture, the government-owned secure Central Server (CS) continues to be needed for processing every query. Even with the existence of Third-Party Servers (TPS), the processing at CS continues to be linear in the number of queries. This prompts the question - why use a TPS at all? Why not do all the processing at CS? In addition to the security guarantees we achieve by having all communication between parties be encrypted (see Chapter 6), the following reasons make it impractical for such a system to be implemented without a TPS:

- In our proposed architecture, the job of storing and processing the data is handled by the TPS, and that of decryption is handled by the CS. If a single server were to be assigned all these tasks, it would need to be immensely powerful. This server would need to be capable of handling hundreds of queries concurrently. The initial investment that would be required by the governments to erect such a server would be massive, in both the money spent and the time consumed. Since the proposed architecture and system are specifically targeted at poorer economies, sustaining such huge initial costs may put them off deploying this kind of system in the first place.

- It may be hard for governments to accurately estimate the performance requirements of such a system. If they overestimate these requirements, they may purchase expensive computing hardware that remains underutilized, making it a massive waste of computing resources and capital. If the performance requirements are underestimated, it would result in a large capital requirement to retroactively upgrade a system already in deployment.
- In addition to the initial capital required for purchasing and setting up the server equipment, there would also be the need for regular maintenance and upkeep of the hardware of such a system. This has two significant implications:
  - The cost of maintaining and upgrading such a system may balloon over the years. It would also require the governments to have constant access to highly skilled personnel capable of diagnosing and fixing the problems with such a system. This would cause such a system to be a constant drain on the nation's already limited resources, which may cause the governments to abandon the project as a whole.
  - In case the system hardware does need maintenance, the downtime of these servers may cripple the provision of fundamental government services. For such a digital ID verification system to become ubiquitous, governments must ensure that its services are always available. Since a major use-case of such a system would be the provision of government welfare schemes like

daily ration distribution, the system being down may not just be an inconvenience, but rather become perilous to human life.

Furthermore, there are certain advantages to using third-party servers from cloud service providers. These are as follows:

- **Ease and flexibility of deployment:** Third-party cloud-based service providers (such as Amazon Web Services) have servers specifically designed with ease and flexibility in mind. It is incredibly convenient to set such servers up for SaaS-based applications with the exact configuration of the operating system and database type as needed. This can allow governments to save a fortune on set-up costs in such a system.
- **Cost-effectiveness and scalability:** Third-party servers require you to pay only for the computing resources that the application actually uses. This makes these highly cost-effective, as computing resources practically never go to waste. These servers are also constantly upgraded with state-of-the-art hardware, thus ensuring that the application has the best possible performance. Furthermore, these services also allow you to scale the available storage and computing resources based on the demand for the application. This means that irrespective of the number of concurrent query requests, these servers will never be overwhelmed.
- **Reliability:** Third-party servers can provide a much higher degree of reliability than in-house servers. With the availability of backup servers to take over operations in case of any failure and 24/7 support for managing both hardware and software issues, these servers allow for the consistent provision of service without the need of purchasing redundant hardware.

Table TC3.1: Summary of the user data and the way it is stored in ciphertext.

| **Field** | **Plaintext Size (bytes)** | **Vector Index** | **Storage Format** | **Size on Disk at TPS (kB)** |
|---|---|---|---|---|
| User ID | 16 | - | Plaintext | 0.016 |
| Name | 50 | [0, 399] | Encrypted Demographic Data Vector | 432 |
| Gender | 1 | [400, 407] | | |
| Pincode | 6 | [408, 455] | | |
| Phone Number | 13 | [456, 559] | | |
| Email ID | 20 | [560, 799] | | |
| Date-of-birth | 6 | [800, 1599] | | |
| Biometric Template | 640 | [0, 640] | Encrypted Biometric Data Vector | 432 |

## 3.3 User Data and Encryption

User data collected during enrollment is of two types: demographic and biometric. Demographic data includes a user name, date-of-birth, gender, email, phone number, and pin code (see Table TC3.1), while the biometric data is the user's fingerprint information stored as a fingercodes vector [55, 56]. The user will be issued a unique ID number after enrollment.

In order to make computations on large ciphertexts more efficient, we use *Batching*, a technique that is integrated into MS SEAL via the `BatchEncoder` class. Let $N$ denote the degree of the polynomial modulus and $T$ denote the plaintext modulus. Batching allows the BFV plaintext polynomials to be viewed as 2-by-$(N/2)$ matrices, with each element an integer modulo $T$. In the matrix view, encrypted operations act element-wise on encrypted matrices, allowing us to obtain speeds-ups of several orders of magnitude by making our computations fully vectorizable.

MS SEAL's BFV implementation uses Gaussian noise to mask the plaintext, and the noise variance grows with each multiplication. Hence each ciphertext has a *noise budget*, which defines the total amount of *noise* each ciphertext can tolerate before it loses its accuracy during decryption. The size of this noise budget varies with the SEAL parameters, namely $N$ and $T$. Since in BFV, noise is only introduced for multiplication operations, we optimize the selection of our SEAL parameters based on the highest multiplicative depth of the circuits we aim to evaluate on ciphertexts. In the proposed system, $N$ is set to 8192, and $T$ is set to a 22-bit prime number. Our encrypted data is thus stored as two batched vectors, one for demographic data and one for biometric data. Each of these vectors is of size 8192. Table TC3.1 summarises the data attributes stored for each user, its size in plaintext, and the corresponding vector indices. Note that since most of both the plaintext and ciphertext vectors are empty, we can add data for multiple fingerprints, or more detailed demographic data if needed without changing the ciphertext sizes.

# CHAPTER 4

# SYSTEM FUNCTIONALITY

## 4.1 Query Types and Algorithms

The following are the types of queries that can be currently addressed in our system. It is worth noting, however, that by using FHE, we can address new query types that may arise without any changes to our encrypted data.

### 4.1.1 Direct Demographic Data Comparison

These are the queries that directly match the demographic data. Since a query only passes on an exact match of data, these queries can be easily addressed with integer arithmetic.

We store our demographic data in a single *batch vector*, where each index of the vector stores 1 bit of data. SEAL operations of multiplication and vector rotation allow us to isolate any part of this vector. So for instance, in the case of a query on the comparison of a user's pin code (which is stored in the index positions [408, 455] of the batch vector), we can multiply the vector with the encryption of a vector of all 0's except in the index positions [408, 455], where it has a 1. Then on left-shifting the resulting vector by 408 positions, we achieve the encryption of a vector having the pin code in index [0, 47]. Let the vector created above be $v_{\text{enc}}$. Then the comparison algorithm follows:

**Algorithm 1** Demographic data matching algorithm

```
At Service Provider (SP)

Require: User data string $s$          ▷ Data here can be any of the demographic fields.
 1: procedure ENCODEDATA($s$)
 2:     $d \leftarrow$ binary encoding of $s$
 3:     $d_{enc} \leftarrow Enc_{pk}(d)$
 4:     $d_{enc}$ is sent to the TPS as query data.
 5: end procedure

At TPS

Require: User ID $id$, Encrypted input data vector $d_{enc}$, query type $q$
 6: function COMPAREDEMOGRAPHIC($d_{enc}, id, q$)          ▷ Assume $q$ is Email data
 7:     $v_{enc} \leftarrow$ encrypted demographic data vector corresponding to $id$ from database
 8:     $u_{enc} \leftarrow Enc_{pk}(\langle 0, 0, \cdots, 1, \cdots, 1, 0, \cdots, 0, \rangle)$   ▷ Vector of all 0s, with 1s in the index position $[560, 799]$, where the email address is stored in $Dec_{sk}(v_{enc})$
 9:     $u_{enc} \leftarrow u_{enc} \times v_{enc}$          ▷ Isolating only the email data from $v_{enc}$
10:     $u_{enc} \leftarrow$LeftShift($u_{enc}, 560$)          ▷ Shift encrypted email data from index $[560, 799]$ to index $[0, 239]$
11:     $u_{enc} \leftarrow u_{enc} - d_{enc}$
12:     Send $u_{enc}$ to CS.
13: end function

At CS

Require: $u_{enc}$ from TPS, secret-key $sk$
14: procedure CHECKOUTPUT($u_{enc}$)          ▷ Procedure to check if query data was a match
```

```
15:     $u \leftarrow Dec_{sk}(u_{enc})$
16:     **if** Index $[0,400]$ of $u$ are 0 **then**
17:         **return** "Query data match successful!"
18:     **else**
19:         **return** "Query data match failed"
20:     **end if**
21: **end procedure**
```

- **At SP**:Encode the input pincode in a vector, say $u$, in index positions [0, 47]. Encrypt this vector to generate $u_{\text{enc}}$, and send $u_{\text{enc}}$ to TPS.
- **At TPS**: Calculate $v_{\text{enc}} - u_{\text{enc}}$ homomorphically. Send the output to CS.
- **At CS**: Check if the output vector is $\langle 0,0,0,\cdots,0\rangle$. If yes, then send "Query Passed" message to SP.

Hence we can process the direct demographic matching queries with a circuit having a 0 multiplicative depth.

### 4.1.2 Biometric Data - Threshold Comparison

Currently, our system supports the comparison of fingerprints. We record the fingerprint data as a fingercodes vector, which is a vector of size 640 bytes. This vector is encoded as a batch vector with each vector element representing 1 byte of data.

The fingercodes comparison algorithm declares two fingerprints to be a match if the Euclidean distance between their corresponding vectors is less than a certain threshold $\beta$. Having access to addition, multiplication, and vector rotation, we can easily calculate the Euclidean distance between two encrypted vectors in MS SEAL. In order to check if the value is less than the threshold, we try to introduce a 0 at some index position in the vector. The complete algorithm is described in Algorithm 2 and is summarised as follows:

- **At SP**: Collect the fingerprint data from the user using some secure hardware. Encode this as a fingercodes vector with the first 640 indices populated. Encrypt this vector to obtain the input vector
- **At TPS**:
  - Modify the vector $v_{\text{enc}}$ using rotation and addition operations to a vector of the form $\langle e, e+1, e+2, \cdots, e+\beta, 0, 0, \cdots, 0\rangle$.
  - Create a vector $u$ of the form $\langle \beta, \beta, \cdots, \beta\rangle$. Encrypt it to form $u_{\text{enc}}$.
  - Output the homomorphic subtraction $v_{\text{enc}} - u_{\text{enc}}$ to CS.
- **At CS:** We know that if $e \leq \beta$, then $\exists x$, such that $x < \beta$ and $e + x - \beta = 0$. Hence we can decrypt the vector received from TPS. If it contains a 0 on any of its index positions, we send a "Query Passed" message to the SP.

**Algorithm 2** Biometric data threshold comparison algorithm

**At Service Provider (SP)**

```
1: procedure ENCODEFINGERPRINTDATA
2:     $u \leftarrow$ encoding of the user's fingerprint data    ▷ $u$ here is a fingercodes vector with the first 640 positions populated with 1 byte of data each.
3:     $u_{enc} \leftarrow Enc_{pk}(u)$
4:     $u_{enc}$ is sent to the TPS as query data.
5: end procedure
```

**At TPS**

**Require:** User ID *id*, Encrypted biometric data vector $u_{enc}$, query type $q$, threshold value $\beta$

```
6: function CompareBiometric($u_{enc}, id, q$)    ▷ $q$ here will be “Biometric”
7:     $v_{enc} \leftarrow$ encrypted biometric data vector corresponding to $id$ from database
8:     procedure CalculateEuclideanDistance($u_{enc}, v_{enc}$)
9:         $e_{enc} \leftarrow u_{enc} - v_{enc}$
10:        $e_{enc} \leftarrow e_{enc} \times e_{enc}$
11:        $temp \leftarrow e_{enc}$
12:        $i \leftarrow 0$
13:        while $(i < 640)$ do
14:            $temp \leftarrow$ LeftShift$(temp, 1)$
15:            $e_{enc} \leftarrow e_{enc} + temp$
16:            $i \leftarrow i + 1$
17:        end while
18:        $temp \leftarrow Enc_{pk}(\langle 1, 0, 0, \cdots, 0 \rangle)$    ▷ Encryption of vector of all 0s except at index position 0
19:        $e_{enc} \leftarrow e_{enc} \times temp$    ▷ $e_{enc}$ is now of the form $\langle \text{ED}, 0, 0, \cdots, 0 \rangle$, where ED is the Euclidean distance.
20:    end procedure
21:
22:    procedure CompareThreshold($e_{enc}, \beta$)
23:        $i \leftarrow 0$
24:        $temp \leftarrow e_{enc}$
25:        while $(i \leq \beta)$ do
26:            $temp \leftarrow$ RightShift$(temp, 1)$
27:            $e_{enc} \leftarrow e_{enc} + temp + i$
28:            $i \leftarrow i + 1$
29:        end while    ▷ $e_{enc}$ is now of the form $\langle \text{ED}, \text{ED} + 1, \text{ED} + 2, \cdots, \text{ED} + \beta, 0, 0, \cdots, 0 \rangle$
30:        $b_{enc} \leftarrow Enc_{pk}(\langle \beta, \beta, \cdots, \beta \rangle)$
31:        $e_{enc} \leftarrow e_{enc} - b_{enc}$
32:        return $e_{enc}$
33:    end procedure
```

```
34: end function
```

**At CS**

```
Require: $e_{enc}$ from TPS, secret-key $sk$
35: procedure CHECKOUTPUT($e_{enc}$)        ▷ Procedure to check is query data was a match
36:     $e \leftarrow Dec_{sk}(e_{enc})$
37:     if $e$ contains one 0 then
38:         return "Query data match successful!"
39:     else
40:         return "Query data match failed"
41:     end if
42: end procedure
```

**Proof of correctness:** As per the algorithm at TPS, after step 29, the encrypted vector $e_{enc}$ will be of the form $\langle \mathrm{ED}, \mathrm{ED}+1, \mathrm{ED} + 2, \cdots, \mathrm{ED}+\beta, 0, 0, \cdots, 0\rangle$, where ED is the Euclidean distance between our two biometric vectors. As per the fingercodes algorithm, the two fingerprints may be declared a match if $\mathrm{ED} < \beta$, where $\beta$ is the threshold value. We know that if $\mathrm{ED} < \beta$, then $(\mathrm{ED}+x) - \beta = 0$, where $x \in [0, \beta - 1]$. Hence in step 31, we try and introduce this 0 in our vector by calculating $e_{enc} - b_{enc}$, where $b_{enc}$ is of the form $\langle \beta, \beta, \cdots, \beta\rangle$. The resulting vector will be $\langle \mathrm{ED} - \beta, \mathrm{ED} + 1 - \beta, \mathrm{ED} + 2 - \beta, \cdots, \mathrm{ED} + \beta - \beta, p - \beta, p - \beta, \cdots, p - \beta\rangle$, where $p$ is the chosen plaintext modulus. Since $\beta$ is non-zero, $p - \beta$ will also be non-zero. Therefore at CS, if the vector contains a 0, then we know that $\mathrm{ED} < \beta$.

### 4.1.3 Logic Gates With BFV

In order to execute our age comparison algorithm, we needed a method to homomorphically execute logic gates across batch vectors. The simplest solution to this would be to set the plaintext modulus to 2 and then execute homomorphic addition and multiplication to carry out an OR gate and an AND gate respectively. However, since the ciphertext noise budget is tied directly to the plaintext modulus, our plaintext modulus is set as some 20-bit prime number to enable our ciphertexts to handle suitable multiplicative circuit depth. For our purposes, however, it is sufficient to design algorithms for logic gates that assume that each index in the input vectors is set to either 0 or 1.

- **AND gate**: Assuming two batch vectors have each of their elements as either 0 or 1, the product of these vectors will also have each element as either 0 or 1, and will be equivalent to the AND of the corresponding indices:

$$\begin{aligned} u &= \langle 1,0,0,1,0,1,1,0,\cdots\rangle \\ v &= \langle 0,1,0,0,1,1,0,0,\cdots\rangle \\ w = u\times v &= \langle 0,0,0,0,0,1,0,0\cdots\rangle \\ \text{Here, } w_i &= u_i \wedge v_i \end{aligned}$$

  Hence the multiplicative depth of an AND gate is 1. Note that the same approach will not work for the OR gate with addition. Since the plaintext modulus isn't 2, $1+1=2\neq 1\vee 1$.

- **NOT gate**: Given an encrypted batch vector with each of its indices set to 0 or 1 (say $u$), we use Algorithm 3 to invert the values at its indices:-

  *Correctness:* Suppose $u$ is set to $\langle 1,0,0,1,0\rangle$. We first create $v_1$ as $\langle p-1,p-1,\cdots\rangle$, which is equivalent to the vector $\langle -1,-1,-1,-1,-1\rangle$ (since every vector element is $mod\, p$). Now, $u\times v_1$ will be the same binary sequence as $u$ except with every 1 set to -1, i.e., $u\times v_1=\langle -1,0,0,-1,0\rangle$. If we add 1 to every element in this product, all indices containing $-1$ will become 0, and all indices with 0 will become 1. Hence, given $v2=\langle 1,1,1,1,1\rangle$, $(u\times v_1)+v_2=\langle 0,1,1,0,1\rangle=\overline{u}$.

  Hence NOT can be evaluated with a circuit having 1 multiplicative depth.

**Algorithm 3** Homomorphic NOT gate

```
Require: Input vector $u$ (Encrypted batch vector), public key $pk$
1: procedure NOT($u$)
2:     $v_1 \leftarrow Enc_{pk}(\langle p-1, p-1, \cdots, p-1 \rangle)$        ▷ $p$ is the chosen plaintext modulus
3:     $v_2 \leftarrow Enc_{pk}(\langle 1, 1, \cdots, 1 \rangle)$
4:     $u \leftarrow u \times v_1$
5:     $\overline{u} \leftarrow u + v_2$
6:     return $\overline{u}$ is NOT($u$)
7: end procedure
```

- **OR and XOR gate**: Having access to both AND and NOT, we have effectively created a NAND gate, which is a universal logic gate. Therefore, we can simulate OR and XOR gates using the following, well-known logic circuits:

$$v_1 \vee v_2 = \overline{\overline{v_1} \wedge \overline{v_2}}$$
$$v_1 \oplus v_2 = (v_1 \vee v_2) \wedge \overline{v_1 \wedge v_2}$$

  Hence we can homomorphically evaluate the OR and the XOR gate on SEAL's BFV batched ciphertexts using circuits having multiplicative depths 3 and 4 respectively.

### 4.1.4 Age Comparison

Many government and private applications require age verification. We need our system to be able to answer queries of the form ”Is the user above the age of 18?” or ”Is the user below the age of 65?” homomorphically. This requires us to create a comparison algorithm, which is especially challenging with BFV since it supports only modular arithmetic. We formulate these query types in a more generic manner as follows:

> Given 2 date-of-births $d_1$ and $d_2$, can we encode them as vectors and perform a set of operations on these vectors such that the resulting vector has properties that only hold when $d_1$ comes before $d_2$?

Our solution to this problem requires the use of logic gates on batch vectors. Given two encrypted batch vectors $v_1$ and $v_2$ such that each element of these vectors is either 0 or 1, we want to evaluate logic gates such as AND and OR on these binary sequences. The implementation details of these logic gates using BFV are given in Section 4.1.3. Given an encrypted *binary* vector $v$, our algorithm for computing $\bar{v}$ is as follows:

– Calculate the encoding and encryption of a vector, say $u$ with each of its elements being $T-1$ ($T$ is the plaintext modulus), i.e., $u = Enc(\langle T-1, T-1, \cdots, T-1\rangle)$. Since BVF operates in modular arithmetic, this is equivalent to having each element set to -1.

– Calculate the encoding and encryption of a vector, say $w$ with each of its elements being 1, i.e., $w = Enc(\langle 1, 1, 1, \cdots, 1\rangle)$.

– Calculate $w + (u \times v)$ homomorphically. This is the required encrypted vector $\bar{v}$.

Now we explain how the date-of-birth data is encoded as a vector:

- We pick a pivot date of 1 Jan 1900. Each date-of-birth is then expressed as the distance from this pivot in years and days. For instance, the date 6 April 1999, can be thought of as being 99 years after 1900, and 96 days after 1 Jan. So it can be expressed as the number 099-096. In this way, every date can be expressed as the number of years from 1900, $y$, and the number of days from 1 Jan, $d$.
- In our date-of-birth vector, we use 800 index positions to store this date. Of these, [0, 399] indices contain $y$ encoded in unary, and [400, 799] indices are the vector

$\langle d, d+1, d+2, \cdots, d+399 \rangle$. For instance, 6 April, 1999 is encoded in these indices as as:

$$\langle \underbrace{1,1,1,\cdots,1}_{\text{upto index 98}}, \underbrace{0,0,\cdots,0}_{\text{upto index 399}}, 96, 97, 98, \cdots, 495 \rangle$$

(Note that the above mentioned "date-of-birth vector" is not stored as a separate ciphertext vector, but is instead just a part of the demographic data vector in the index positions [800, 1599].)

During query execution, we can isolate each of the two parts of this encrypted vector using a similar technique to direct demographic comparison. We create vectors $y_{enc}$ and $d_{enc}$ such that:

$$y_{enc} = Enc(\langle 1,1,1,\cdots,1,0,0,\cdots,0 \rangle)$$
$$d_{enc} = Enc(\langle 96,97,98,\cdots,494,495 \rangle)$$

To compare this date-of-birth to some input date-of-birth $dob$, we create the corresponding vectors $y'_{enc}$ and $d'_{enc}$ for this date-of-birth. Here, $y'_{enc}$ is created similar to $y_{enc}$, but $d'_{enc}$ does not increment as in $d_{enc}$, i.e., if the number of days from 1 Jan for the input date-of-birth are $x$, then $d'_{enc}$ is constructed as $Enc(\langle x,x,x,\cdots,x \rangle)$. The comparison algorithm is described in detail in Algorithm 4.

**Algorithm 4** Date-of-birth comparison algorithm

**At Service Provider (SP)**

```
Require: User date-of-birth $dob$
1: procedure ENCODEDATEOFBIRTH($dob$)
2:     $y \leftarrow$ no. of years from 1900 in $dob$
3:     $d \leftarrow$ no. of days from 1 Jan in $dob$
4:     $y'_{enc} \leftarrow Enc_{pk}(\langle 1,1,1,\cdots,1,0,0,\cdots,0\rangle)$    ▷ $y$ no. of 1s, followed by all 0s upto index 400
5:     $d'_{enc} \leftarrow Enc_{pk}(\langle d,d,d,\cdots,d\rangle)$    ▷ Vector of all $d$s upto index 400
6:     $y'_{enc}$ and $d'_{enc}$ are sent to TPS as query data.
7: end procedure
```

**At TPS**

```
Require: User ID $id$, $y'_{enc}$ and $d'_{enc}$ from SP.
8: function COMPAREDATEOFBIRTH($id, y'_{enc}, d'_{enc}$)
9:     $demo_{enc} \leftarrow$ encrypted demographic data vector corresponding to $id$ from database
10:                                ▷ The year and date in $demo_{enc}$ are stored in indices [800,1199] and [1200,1599] respectively.
11:    $y_{enc} \leftarrow \langle 0,0,\cdots,0, \underbrace{1,1,\cdots,1,}_{\text{index [800, 1199]}} 0,\cdots,0\rangle$
12:    $d_{enc} \leftarrow \langle 0,0,\cdots,0, \underbrace{1,1,\cdots,1,}_{\text{index [1200, 1599]}} 0,\cdots,0\rangle$
13:    $y_{enc} \leftarrow \text{LeftShift}(demo_{enc} \times y_{enc}, 800)$
14:    $d_{enc} \leftarrow \text{LeftShift}(demo_{enc} \times d_{enc}, 1200)$
15:
16:    $\text{temp}_1 \leftarrow y_{enc} \wedge (y_{enc} \oplus y'_{enc})$    ▷ All logic gates used are homomorphic and element-wise across vectors
17:    $\text{temp}_2 \leftarrow (d'_{enc} - d_{enc}) \times \overline{(y_{enc} \oplus y'_{enc}) \wedge y'_{enc}}$
18:    $\text{temp}_2 \leftarrow \text{RightShift}(\text{temp}_2, 400)$
19:    $\text{out}_{enc} \leftarrow \text{temp}_1 + \text{temp}_2$
```

```
20:     return out$_{enc}$ to CS
21: end function
```

**At CS**

```
Require: out$_{enc}$ from TPS, secret-key $sk$
22: procedure CheckOutput($e_{enc}$)    ▷ Procedure to check if input DoB lies after user's DoB
      in the database
23:     out $\leftarrow Dec_{sk}(\text{out}_{enc})$
24:     if Index $[0,399]$ of out are all 0s and index $[400,799]$ of out contain st least one 0 then
25:         return "Input DoB lies after the user's DoB"
26:     else
27:         return "Input DoB lies before the user's DoB"
28:     end if
29: end procedure
```

**Correctness:** Let $y$ and $d$ be the values of the distance from the year and date pivots for the user's DoB, and $y'$ and $d'$ be these distances for the input DoB. Let $y_{enc}, d_{enc}, y'_{enc}, d'_{enc}$ be the corresponding excrypted vector encodings of these values as used in Algorithm 3. We can break-up the functionality of our algorithm into 4 cases.

- **Case 1:** $y > y'$

  Let the user's date of birth on record is 6th April, 1999. In the representation method for Algorithm 4, $y = 99$ and $d = 96$ (99 years from 1900, 96 days from 1 Jan). Hence the vectors $y_{enc}$ and $d_{enc}$ will be:-

$$y_{enc} = Enc_{pk}(\langle \underbrace{1,1,\cdots,1}_{\text{Index [0, 98]}}, \underbrace{0,0,\cdots,0}_{\text{Index [99, 399]}} \rangle)$$

$$d_{enc} = Enc_{pk}(\langle 96,97,98,\cdots,495 \rangle)$$

Now suppose the input date of birth to be compared is 2nd February, 1994. Hence $y' = 94$ and $d' = 33$. As per lines 4 and 5 of Algorithm 4, $y'_{enc}$ and $d'_{enc}$ are:-

$$y'_{enc} = Enc_{pk}(\langle \underbrace{1,1,\cdots,1}_{\text{Index [0, 93]}}, \underbrace{0,0,\cdots,0}_{\text{Index [94, 399]}} \rangle)$$

$$d'_{enc} = Enc_{pk}(\langle 33,33,33,\cdots,33 \rangle)$$

As per algorithm line 16, the vector $\text{temp}_1$ contains 1s if vector $y$ has more 1s than the vector $y'$. In other words, the first 400 index positions of the output vector contain a non-zero index only if the input DoB comes before the user's DoB. In the example values above, the value of $\text{temp}_1$ will be:

$$\text{temp}_1 = y_{enc} \wedge (y_{enc} \oplus y'_{enc}) = Enc_{pk}(\langle \underbrace{0,0,\cdots,0}_{\text{Index [0, 93]}},1,1,1,1,1, \underbrace{0,0,\cdots,0}_{\text{Index [99, 399]}} \rangle)$$

Since the first 400 index positions will not be 0s, as per line 24 of the algorithm, the output will declare input DoB to come before the user's DoB. Hence due to the check on $\text{temp}_1$, the algorithm works correctly in all cases when $y > y'$.

- **Case 2:** $y = y$ and $d > d'$

  When $y$ and $y'$ are equal, $y_{enc}$ and $y'_{enc}$ will be the same vector. Hence $y_{enc} \oplus y'_{enc} = \langle 0,0,\cdots,0 \rangle$. Suppose the user's DoB is 6th April, 1999 and the input DoB is 3rd February, 1999. Then $d$ and $d'$ will be the same as in case 1.

  Since $y = y'$, the value of $\overline{(y_{enc} \oplus y'_{enc}) \wedge y'_{enc}}$ will be $\langle 1,1,\cdots,1 \rangle$. The value of the vector $\text{temp}_2$ will therefore be the same as $d'_{enc} - d_{enc}$ as per algorithm line 17.

  We know that if $d > d'$, then $d' - (d+i)$ will be non-zero for all positive integers $i$. Hence none of the indices of the vector $d'_{enc} - d_{enc}$ will be zero. As per line 24 of the algorithm, the output will therefore be correct in all cases when $y = y'$ and $d > d'$

- **Case 3:** $y = y'$ and $d < d'$

As discussed in Case 2, $y_{enc} \oplus y'_{enc}$ will be zero, and the output will be determined by temp$_2 = d'_{enc} - d_{enc}$. Let the user's DoB is 3rd February, 1999 and the input DoB is 6th April, 1999. Then the values of $d_{enc}$ and $d'_{enc}$ will be the opposite of their values in cases 1 and 2, i.e.,

$$d_{enc} = \langle 33, 34, 35, \cdots, 432 \rangle$$
$$d'_{enc} = \langle 96, 96, 96, \cdots, 96 \rangle$$

We know that if $d < d'$, then $\exists\, i \geq 0$ such that $d' - (d+i) = 0$. Also, $i$ will also be smaller than the maximum possible value of $d'$. The vector $d_{enc}$ encodes the value $d+i$ in each of its indices, with $0 \leq i < 400$. Since the maximum possible distance any date can have from 1 Jan is 365 (number of days in a year) and 365 < 400, we know that exactly one index of the vector $d'_{enc} - d_{enc}$ will contain a 0 when $d < d'$. As per line 24 of the algorithm, the first 400 indices of the output vector will be all 0s ($y = y'$), and the next 400 indices will contain at least one 0. The algorithm will declare the user's DoB to lie before the input DoB. Hence the algorithm output is correct in all cases when $y = y'$ and $d < d'$.

- **Case 4:** $y < y'$ Let the user's DoB on record is in the year 1994, and the input DoB is in the year 1999. Then $y = 94$ and $y' = 99$. The value of the encrypted vectors will then be:

$$y_{enc} = Enc_{pk}(\langle \underbrace{1, 1, \cdots, 1}_{\text{Index [0, 93]}}, \underbrace{0, 0, \cdots, 0}_{\text{Index [94, 399]}} \rangle)$$
$$y'_{enc} = Enc_{pk}(\langle \underbrace{1, 1, \cdots, 1}_{\text{Index [0, 98]}}, \underbrace{0, 0, \cdots, 0}_{\text{Index [99, 399]}} \rangle)$$

Since $y \neq y'$, as in Case 1, $y_{enc} \oplus y'_{enc}$ will contain 1s in the index positions [94, 98]. However since the extra 1s are in $y'_{enc}$, temp$_1$ as per line 16 of the algorithm will be:

$$\text{temp}_1 = y_{enc} \wedge (y_{enc} \oplus y'_{enc}) = Enc_{pk}(\langle 0,0,0,\cdots,0,0\rangle)$$

In cases 2 and 3, the value of temp$_2$ has been independent of $\overline{y'_{enc} \wedge (y_{enc} \oplus y'_{enc})}$, since this has been a vector of all 1s. However in this case, since the number of 1s in $y_{enc}$ is less than the number of 1s in $y'_{enc}$, this vector will be of the form:

$$\overline{y'_{enc} \wedge (y_{enc} \oplus y'_{enc})} = Enc_{pk}(\langle \underbrace{1,1,\cdots,1}_{\text{Index [0,93]}},0,0,0,0,0,\ \underbrace{1,1,\cdots,1}_{\text{Index [99, 399]}}\ \rangle)$$

Due to this, as per line 17 of our algorithm, the value of the second half of the output vector will be:

$$\begin{aligned}\text{temp}_2 &= (d'_{enc} - d_{enc}) \times \overline{y'_{enc} \wedge (y_{enc} \oplus y'_{enc})} \\ &= Enc_{pk}(\langle \underbrace{X,X,\cdots,X}_{\text{Index [0,93]}},0,0,0,0,0,\underbrace{X,X,\cdots,X}_{\text{Index [99, 399]}}\rangle)\end{aligned}$$

Here, $X$ is the value of that index in the vector $d'_{enc} - d_{enc}$, which may of may not be 0. By multiplying this vector by $\overline{y'_{enc} \wedge (y_{enc} \oplus y'_{enc})}$, we introduce 0s in the second half of our output vector whenever $y < y'$. Therefore, irrespective of the values of $d$ and $d'$, the output vector will contain at least one 0 in the index positions [400, 799] every time $y < y'$. As per line 24 of the algorithm, the output will be correct in all cases when $y < y'$.

The above-described cases document algorithm 4's behavior for all possible values of $y, d, y'$ and $d'$. Hence algorithm 4 is correct.

## 4.2 Central Server - Query Agnostic Processing

In all the algorithms described above, at the Central Server, processing involves decryption of the TPS output followed by simple vector comparison logic. By simple manipulations on the TPS output for our algorithms, we can make this vector comparison logic completely query agnostic. This means that the CS would be able to determine the query pass/fail status by executing a deterministic comparison circuit on TPS output independent of query type and data. This would allow the CS to simply carry out an *extended decryption circuit* for each query, making it an O(1) process at CS for each query.

To recap the CS query PASS condition for each algorithm, given $v$ is the decrypted vector output from TPS:

- **Demographic Comparison**: Query PASS if index [0, 399] of $v$ are all 0s.
- **Biometric Comparison**: Query PASS if index [0, $\beta - 1$] of $v$ contain at least one 0 ($\beta$ here is the fingercodes threshold).
- **Date-of-birth Comparison**: Query PASS if index [0, 399] of $v$ are all 0s, and index [400, 799] contain at least one 0. (*Note*: PASS case here is the input DoB lying after the user's DoB)

In the configuration of MS SEAL used in this project, the size of each vector is 4096, and the size of the fingercodes threshold $\beta$ is 3000. Therefore, we can carry out homomorphic operations at TPS on the final vector $v_{enc}$ to create a modified vector $v'_{enc}$ for each algorithm as follows:

- **Demographic Comparison**: $v'_{enc} = v_{enc} \times Enc_{pk}(\langle \underbrace{1, 1, \cdots, 1}_{\text{Index [0, 399]}}, \underbrace{0, 0, \cdots, 0}_{\text{Index [400, 400+}\beta\text{]}} \rangle)$

- **Biometric Comparison**: $v'_{enc} = \text{RightShift}(v_{enc}, 400)$
- **Date-of-birth Comparison**: $v'_{enc} = v_{enc} + Enc_{pk}(\langle \underbrace{0, 0, \cdots, 0}_{\text{Index [0, 799]}}, \underbrace{1, 1, \cdots, 1}_{\text{Index [800, 400 + }\beta\text{]}} \rangle)$

By performing these computations at TPS before sending the final vector (now $v'$) to CS, the following check can be used for all query types at CS:

> If index [0, 400] of $v'$ are all 0s and index [400, 400+$\beta$] contain at least one 0, query PASS. Else query FAIL.

Using the above-stated logic at CS will correctly determine the query output for all query types. This can allow us to make the processing at CS very efficient.

# CHAPTER 5

# EXPERIMENTS AND RESULTS

As described in Section 3.3, we choose our polynomial modulus $T$ as 8192, and our plaintext modulus $N$ as a 22-bit prime. This allows us to encode each ciphertext as a SIMD vector of size 4096, with each SIMD slot being *mod*(22-bit prime). SEAL automatically picks the remaining parameters to achieve 192-bit security by default, which can be changed to 128-bit or 256-bit as needed. The third-party server uses $<$ 1MB of storage for each user's data. Each message sent from SP to TPS, and from TPS to CS is a ciphertext of size 432 kB. For these experiments, we do not provide any estimates based on expected network latency.

Table TC5.1: Timing data of different query types at TPS and decryption at CS. In the fingercodes implementation used, $\beta$ is set to 3000.

| **Operation** | **Multiplicative Depth** | **No. of Rotations** | **No. of Additions** | **Time (milliseconds)** |
|---|---|---|---|---|
| CS: decryption | - | - | - | 4.66 |
| Name match | 1 | 1 | 1 | 22.82 |
| Gender match | | 2 | | 35.39 |
| Pincode match | | | | 40.83 |
| Phone Number match | | | | 35.36 |
| Email ID match | | | | 35.17 |
| Date-of-birth comparison | 7 | 3 | 6 | 217.73 |
| Biometric Template comparison | 3 | $\lceil \log_2(640) \rceil + \lceil \log_2(\beta) \rceil$ | $\lceil \log_2(640) \rceil + \lceil \log_2(\beta) \rceil + 2$ | 286.74 |

In our biometric comparison algorithm, we implement further optimizations to greatly speed up the algorithm. In the loops at lines 13 and 25 of Algorithm 2, we use the ‘fast exponentiation’ trick to complete the loop operations in $\lceil \log_2(640) \rceil$ and $\lceil \log_2(\beta) \rceil$ steps respectively.

We run our experiments on a laptop with an Intel i7-7700HQ processor running at 2.8 GHz. All operations are run as single-threaded tasks. For our experiments, we execute each query 1000 times and use the C++ `chrono` class to collect timing data in microseconds. The average time taken over 1000 iterations for each query is summarised in Table TC5.1.

As the results indicate, our most compute-intensive query takes $< 0.3$ to execute, making it perfectly feasible for real-world deployment. We can expect a further speedup in these timings by using Intel’s Homomorphic Encryption Acceleration Library (HEXL) [57] along with an AVX512-IFMA52 compatible processor.

# CHAPTER 6

# SYSTEM SECURITY & ALTERNATE LIBRARIES

In the security analysis of this scheme, the Central Server (CS) is assumed to be fully trusted. It is still preferable for CS to maintain a backup of user data in encrypted form to safeguard against malware attacks (though there could still be leakage during decryption). We can provably claim that our system will have all the same security guarantees as the underlying HE scheme exclusively in the case of an honest-but-curious adversary controlling only the TPS. The TPS and SPs are assumed to be honest-but-curious and they both could collude. The hash function used needs to be collision resistant and pre-image resistant. By making the decryption of intermediate computations available to only the CS, our system is resistant to large-scale leakage analysis attacks. The SP would learn only the result of the ID verification. Because the hash of the user data is available to TSP for efficient indexing, it will only be able to track that some unknown user underwent ID verification using the recorded set of queries at the recorded times.

## 6.1 Security Guarantees

### 6.1.1 Semi-honest setting

As stated above, our system offers the same security guarantees as the underlying BFV scheme against an honest-but-curious TPS. This is because the TPS only has access to encrypted data, and never gets to see the data in plaintext. Even as a semi-honest adversary, we cannot make any claims for security against SP as the user data may be prevented to the SP as plaintext. There are workarounds to prevent this, which can include using secure collection hardware that immediately encrypts data on collection. However, these methods are beyond the scope of the thesis. Furthermore, with minor modifications, this scheme can be made secure in the semi-honest setting if the role of the trusted central server is taken over by another third party. To ensure this, however, it is necessary that this party is unable to collude with the TPS. Even if this untrusted central server is able to collude with a semi-honest SP, the scheme can continue to be secure. By storing the encrypted user data exclusively at the TPS, we can isolate the user data and the secret key. In this setting, the third-party 'central' server would only have access to the decrypted TPS output.

In this setting, however, it becomes possible for the adversary to know both the input query data, as well as the decrypted TPS output. With the current algorithmic implementations, this enables the adversary to carry out leakage attacks which may reveal user data in plain. The nature of these attacks and the techniques that may be used to prevent them are discussed in section 6.2.3. Furthermore, there are multiple other challenges that can come about by handing over the secret key to a potentially untrusted adversary, which is why this architecture is not the recommended setup.

### 6.1.2 Malicious Setting

In the current setup, we do not make any security guarantees for our model in the malicious setting. It may always be possible for a malicious adversary controlling any one of the three involved parties to determine the private data of a particular citizen. A simple extension to our scheme can be implementing a timeout based on the number of queries issued against a specific user hash (ID). This can allow the system to be secure against large-scale inference attacks, even in the malicious adversary setting.

## 6.2 Leakage Analysis

In this section, we discuss the types of breaches and leakages that can occur for each involved party and their implications. We also discuss methods to mitigate the danger from these leakages where possible.

### 6.2.1 Leakage at SP

The data collected at every SP is in plaintext and is susceptible to a leakage attack. The data a user provides for identity verification to SP may be intercepted by some adversary who is able to breach the SP. In this case, however, the data breached is limited to only the information provided by the user. The actual user data on record, as well as the data of all the other users, remains safe from leakage. The only way this kind of attack may be mitigated is by using secure collection hardware at SP, which encrypts the data immediately after collection. An example of such a device may be a biometric collection device with an in-built encryption circuit.

### 6.2.2 Leakage at TPS

All the data at TPS, including the query data from SP and the user data from the database, is fully encrypted, and hence, secure from leakage attacks. This is strictly necessary as, in ideal system functionality, the entire verification process should continue to remain secure even if TPS is an honest-but-curious adversary. Hence by using FHE, our system makes it impossible for a data breach to occur from the TPS.

### 6.2.3 Leakage at CS

There are two main possible data leaks that may take place at the CS:

1. **Secret key leak:** This is the most critical data held at CS. If the secret key is leaked, the adversary gets the ability to decrypt and view in plain any user's demographic and biometric data. The only protection in the case of secret key leakage stems from the fact that the encrypted database is not held by the CS, only by the TPS. Therefore in order to completely breach the system, the adversary needs to breach not only the CS but also the TPS. In case the adversary cannot breach the TPS, and only has access to the TPS output message to CS after query processing, then it will be able to view the TPS output in plain for each query. This will have the exact same implications as the next type of leak.

2. **Leakage of decryption of TPS output:** Since the primary job of CS is to decrypt the TPS output, this decrypted output may also be leaked to an adversary in case of a breach at CS. Depending on the query type, this may have different implications:

   (a) ***Direct demographic matching queries***: Since we use rudimentary operations to determine if the demographic data is a match, if an adversary can find out the query data from the SP, as well as the plaintext decryption of TPS output from CS, it would be possible for this adversary to recover the user's demographic details. For instance, for checking if a user's name is a match, the TPS operation is:

   $$\texttt{TPS\_output} = \texttt{query\_data} - \texttt{user\_data}$$

   Hence an adversary knowing the `TPS_output` and the `query_data` in plain can easily determine the `user_data`.

   A possible way to mitigate this attack is to follow up the subtraction operation at TPS with some additional operations. The following are the best attack mitigation strategies that can be implemented at TPS:

- *Random vector multiplication:* In this strategy, we multiply the final encrypted vector with some random vector at TPS before sending it to CS. TPP can generate an encrypted vector of the form $Enc_{pk}(\langle r_1, r_2, \cdots, r_4096\rangle)$, where $r_i \in_{\text{random}} [0, p-1]$. Here $p$ is the plaintext modulus. Since our query PASS/FAIL decision making happens only on the basis of the presence of 0s in the output vector, this strategy does not impact query output. By multiplying with a random vector, we can ensure that there is no information leaked from the non-zero indices of the plaintext vector at CS.
- *Rotate-and-add*: The following 'rotate-and-add' strategy can be used to prevent data leakage, specifically for demographic data -

```
Require: Encrypted TPS output vector $v$
  // Operations at TPS
  for $k$ iterations do
      $v' = \text{Copy}(v)$
      $r \leftarrow \text{generateRandomNumber}()$
      $v \leftarrow \text{RightShift}(v, r)$
      $v \leftarrow v + v'$
  end for
```

Since the average time cost of one rotation operation is 5 ms and one addition operation is 6 ms in MS SEAL, these operations together repeated $k$ times will add $11k$ ms to the query processing time. $k$ here can be as large as needed since neither addition nor rotation consume our ciphertext noise budget and can be repeated indefinitely.

Proof of Correctness: Since the query passes only in the case when the vector output is all 0s, performing any number of rotate-and-add operations on a PASS vector will not change this vector. However, in a query fail vector, the location of the 0s in this vector indicates partial matches between the query input name and the name on record. However, by rotate-and-add operations, these positions will instead get populated by the values in other index positions, and will no longer remain decipherable.

(b) ***Biometric data comparison queries:*** In the process of the biometric data comparison algorithm, we calculate the Euclidean distance between the input biometric vector and the template biometric vector. The output vector indicates a comparison between this Euclidean distance and the threshold value. If this output vector is leaked, the adversary may be able to determine this Euclidean distance.

One possible mitigation strategy for this attack can be for the TPS to randomise the encrypted vector $e_{enc}$ during its generation in line 27 of Algorithm 2. Instead of generating an ordered vector of the form $\langle \mathrm{ED}, \mathrm{ED}+1, \mathrm{ED}+2, \cdots, \mathrm{ED}+\beta, 0, 0, \cdots, 0 \rangle$, the TPS generates a permuted version of this vector: $\langle \mathrm{ED}+r_1, \mathrm{ED}+r_2, \mathrm{ED}+r_3, \cdots, \mathrm{ED}+r_\beta, 0, 0, \cdots, 0 \rangle$, where $r_1, r_2, \cdots, r_\beta$ are values picked uniformly randomly and without repetition from the range $[0, \beta]$. This vector will continue to maintain the correctness property of our algorithm, i.e., it will contain a 0 iff $\mathrm{ED} \leq \beta$. However, due to the randomness of the vector, the index position of this 0 reveals no information to the adversary about the value of the Euclidean distance.

(c) ***Age data comparison queries:*** As we know from Algorithm 3, the output vector for age queries is composed of two vectors, temp1 and temp2.

- *temp1*: This vector has the same number of 1s as the difference $y - y'$, if $y - y' > 0$, and is all 0s otherwise (see Algorithm 3 line 16). Therefore, if this vector is leaked, it may reveal the exact year of birth of the user.

Since query output from this vector may be the PASS case only when it is all 0s, we can use the same rotate-and-add strategy as in the direct demographic matching case to hide the contents of this vector. If the initial vector has $n$ non-zero indices, by performing $k$ rotate-and-add operations, the number of non-zero indices will be in the range $[n, n \times 2^k]$. These non-zero indices may be further obfuscated by using the random multiplication strategy as described above.

- *temp2*: This vector may have two types of values:
  - In the case when $y \leq y'$, this vector indicates by the presence of a 0 if $d < d'$. The index at which this 0 is present may reveal the day of the year on which the user was born. This can be easily prevented by generating a random permutation of the vector $d_{enc}$ at TPS in Algorithm 4. The nature of this permutation can be similar to the randomization of the vector in the biometric comparison algorithm, as explained above. As in the case of the Euclidean distance-vector, this step will hide the information leaked from the specific index position of a 0 in our vector.
  - In the case when $y > y'$, this vector contains a contiguous range of indices having the value 0 such that the length of this range is $y - y'$. As with the vector temp1, these 0s can be used to determine the user's year of birth. We cannot use the rotate-and-add strategy to hide this vector, as these operations decrease the number of 0s, which may change the query output. An alternate rotate-and-multiply strategy may be used here, where the vector may be multiplied by a randomly rotated version of itself repeatedly. While this strategy would be effective in preventing the year-of-birth leakage from this vector, it is highly impractical. Each multiplication operation would need to be followed by an expensive ciphertext relinearization, and would also

consume the finite noise budget of our ciphertexts.

## 6.3 Alternate Libraries

### 6.3.1 Pallier Cryptosystem

The Pallier cryptosystem is a partially homomorphic (PHE) scheme that supports homomorphic integer addition. It is the most lightweight and efficient HE scheme, being able to encrypt and decrypt 32-bit integers in 18 milliseconds, and carry out a homomorphic addition in $< 0.1$ milliseconds [58]. This has made it a suitable candidate for many real-world applications, including biometric verification [30, 32, 33]. However, this cryptosystem is unsuitable for our architecture for the following reasons:

- Being only additively homomorphic, it is not possible to compute the Euclidean distance between two vectors in a purely encrypted fashion. Existing schemes for biometric verification either use other, less accurate, distance metrics or make one of the vectors available in clear. Since our model involves only communicating with the TPS in ciphertext, it becomes impossible for us to compute Euclidean distance while still retaining all of our security guarantees.
- Our age comparison circuit involves multiplications. While it may be possible to design a comparison algorithm for age using purely additions, it would most likely require a higher degree of processing after decryption at CS. This goes directly against our objective of keeping CS involvement beyond decryption minimum.

### 6.3.2 TFHE

TFHE [6] is a fast, open-source FHE library that allows for the homomorphic evaluation of binary gates. Being fully homomorphic, all of our algorithms are implementable using TFHE. The library is also the fastest FHE library, requiring only 13 milliseconds per binary gate. However, in the 128-bit security setting, TFHE expands 1-bit of plaintext 2016 bytes as ciphertext. This means that each user record requires 165 MB of storage in the database. Furthermore, since only ciphertext messages are exchanged between CS, TPS, and SP, this also makes this ciphertext sizes impractical due to network bandwidth limitations.

### 6.3.3 Palisade/HElib

Palisade and HElib are other FHE libraries that implement the BFV FHE scheme. Of these Palisade, after limited experimentation, offers similar ciphertext expansion and gate evaluation times to MS SEAL, and could be a suitable replacement. However, the rotation operations in Palisade are not as robust as MS SEAL, making SEAL the better choice for our algorithms. HELib is a lightweight implementation of BGV/BFV and is written in the C programming language. Its low-level implementation makes this library highly portable, and it offers similar gate-evaluation performance as SEAL [58]. However, its decryption operation is nearly three times slower, which is non-ideal for reducing CS load. Furthermore, HElib requires sophisticated manual parameter selection for optimal performance, which makes it highly impractical.

# CHAPTER 7

# CONCLUSIONS

This thesis presents a practical, privacy-preserving digital identification protocol targeted at poorer economies. Our protocol aims to make it possible to deploy a national digital ID system without the need for a large initial investment. The protocol works on an outsourced computation model, where we delegate the bulk of our storage and computation needs to a potentially untrustworthy third party. The role of a trusted central server is not entirely eliminated, but by keeping CS involvement at a minimum we can minimize the initial investment needed to set up the system.

We use Microsoft SEAL Fully Homomorphic Encryption (FHE) library to outsource our computation. We describe and prove algorithms for carrying out identity queries using SEAL - matching demographic or biometric data, or comparing encrypted user age against any threshold. We further show how these algorithms can be designed to make CS processing query agnostic. Our scheme is secure against a semi-honest TPS, and we present countermeasures against leakage-based attacks. Our experiments show that each query can be executed in under 0.3 seconds, and needs $< 5$ milliseconds of processing at CS. This makes the presented scheme fast enough for real-world deployment.

For future work, it will be interesting to design a digital ID verification protocol where the role of (a fully trusted) CS is only during the enrollment of users. Secure Multi-Party Computation (MPC)-based protocols could provide a practical solution to this problem assuming the existence of two or more non-colluding TPS (note that we need to minimize the role of CS, so it is best to avoid computation and communication load on it). Another practical concern in many developing countries is the availability of reliable Internet services. Hence, it will be useful to develop protocols that make very limited or no use of communication between entities, particularly, from/to SP. A solution for the latter problem could potentially make use of tamper-resistant hardware, thereby, increasing the cost of deployment and maintenance.

# Bibliography

[1] What is AADHAR? https://uidai.gov.in/my-aadhaar/about-your-aadhaar.html, . Accessed: 2022-03-07.

[2] Modular Open Source Identity Platform (MOSIP). https://mosip.io, . Accessed: 2022-03-07.

[3] The open-source, identity platform MOSIP hits a new milestone. https://medium.com/omidyar-network/the-open-source-identity-platform-mosip-hits-a-new-milestone/-ff9137610bed, . Accessed: 2022-03-07.

[4] Rs 500, 10 minutes, and you have access to billion Aadhaar details. https://www.tribuneindia.com/news/archive/nation/rs-500-10-minutes-and-you-have-access-to-billion-aadhaar-details-523361, . Accessed: 2022-03-07.

[5] Junfeng Fan and Frederik Vercauteren. Somewhat practical fully homomorphic encryption. Cryptology ePrint Archive, Report 2012/144, 2012. https://ia.cr/2012/144.

[6] Ilaria Chillotti, Nicolas Gama, Mariya Georgieva, and Malika Izabachène. TFHE: Fast fully homomorphic encryption library, August 2016. https://tfhe.github.io/tfhe/.

[7] Marta Gomez-Barrero, Emanuele Maiorana, Javier Galbally, Patrizio Campisi, and Julian Fierrez. Multi-biometric template protection based on homomorphic encryption. *Pattern Recognition*, 67:149–163, 2017. ISSN 0031-3203. doi: https://doi.org/10.1016/j.patcog.2017.01.024. URL `https://www.sciencedirect.com/science/article/pii/S0031320317300249`.

[8] Yang Yang, Xindi Huang, Ximeng Liu, Hongju Cheng, Jian Weng, Xiangyang Luo, and Victor Chang. A comprehensive survey on secure outsourced computation and its applications. *IEEE Access*, 7:159426–159465, 2019. doi: 10.1109/ACCESS.2019.2949782.

[9] Mauro Barni, Tiziano Bianchi, Dario Catalano, Mario Di Raimondo, Ruggero Donida Labati, Pierluigi Failla, Dario Fiore, Riccardo Lazzeretti, Vincenzo Piuri, Alessandro Piva, and Fabio Scotti. A privacy-compliant fingerprint recognition system based on homomorphic encryption and fingercode templates. In *2010 Fourth IEEE International Conference on Biometrics: Theory, Applications and Systems (BTAS)*, pages 1–7, 2010. doi: 10.1109/BTAS.2010.5634527.

[10] Zihao Shan, Kui Ren, Marina Blanton, and Cong Wang. Practical secure computation outsourcing: A survey. *ACM Comput. Surv.*, 51(2), feb 2018. ISSN 0360-0300. doi: 10.1145/3158363. URL `https://doi.org/10.1145/3158363`.

[11] Hao Chen, Kim Laine, and Peter Rindal. Fast private set intersection from homomorphic encryption. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, CCS '17, page 1243–1255, New York, NY, USA, 2017. Association for Computing Machinery. ISBN 9781450349468. doi: 10.1145/3133956.3134061. URL `https://doi.org/10.1145/3133956.3134061`.

[12] Keith Frikken. Privacy-preserving set union. In Jonathan Katz and Moti Yung, editors, *Applied Cryptography and Network Security*, pages 237–252, Berlin,

Heidelberg, 2007. Springer Berlin Heidelberg. ISBN 978-3-540-72738-5. doi: 10.1007/978-3-540-72738-5_16.

[13] Alex Davidson and Carlos Cid. An efficient toolkit for computing private set operations. Cryptology ePrint Archive, Report 2016/108, 2016. https://ia.cr/2016/108.

[14] Arisa Tajima, Hiroki Sato, and Hayato Yamana. Outsourced private set intersection cardinality with fully homomorphic encryption. In *2018 6th International Conference on Multimedia Computing and Systems (ICMCS)*, pages 1–8, 2018. doi: 10.1109/ICMCS.2018.8525881.

[15] Dung Hoang Duong, Pradeep Kumar Mishra, and Masaya Yasuda. Efficient secure matrix multiplication over lwe-based homomorphic encryption. *Tatra Mountains Mathematical Publications*, 67(1):69–83, September 2016. ISSN 1210-3195. doi: 10.1515/tmmp-2016-0031. Publisher Copyright: © 2016 Mathematical Institute, Slovak Academy of Sciences.

[16] Jarin Firose Moon, Shamminuj Aktar, and M. M. A. Hashem. Securely outsourcing large scale eigen value problem to public cloud. *2015 18th International Conference on Computer and Information Technology (ICCIT)*, pages 490–494, 2015.

[17] Valeria Nikolaenko, Stratis Ioannidis, Udi Weinsberg, Marc Joye, Nina Taft, and Dan Boneh. Privacy-preserving matrix factorization. In *ACM Conference on Computer and Communications Security*, pages 801–812, 2013. doi: 10.1145/2508859.2516751.

[18] Haomiao Yang, Weichao He, Qixian Zhou, and Hongwei Li. Efficient and secure outsourced linear regression. In *Algorithms and Architectures for Parallel Processing*, pages 89–102. Springer International Publishing, 2018. doi: 10.1007/978-3-030-05057-3_7. URL https://doi.org/10.1007/978-3-030-05057-3_7.

[19] Valeria Nikolaenko, Udi Weinsberg, Stratis Ioannidis, Marc Joye, Dan Boneh, and Nina Taft. Privacy-preserving ridge regression on hundreds of millions of records. pages 334–348, 05 2013. ISBN 978-1-4673-6166-8. doi: 10.1109/SP.2013.30.

[20] Jung Cheon, Duhyeong Kim, Yongdai Kim, and Yongsoo Song. Ensemble method for privacy-preserving logistic regression based on homomorphic encryption. *IEEE Access*, PP:1–1, 08 2018. doi: 10.1109/ACCESS.2018.2866697.

[21] Thore Graepel, Kristin Lauter, and Michael Naehrig. Ml confidential: Machine learning on encrypted data. In Taekyoung Kwon, Mun-Kyu Lee, and Daesung Kwon, editors, *Information Security and Cryptology – ICISC 2012*, pages 1–21, Berlin, Heidelberg, 2013. Springer Berlin Heidelberg. ISBN 978-3-642-37682-5. doi: 10.1007/978-3-642-37682-5_1.

[22] Raphael Bost, Raluca A. Popa, Stephen Tu, and Shafi Goldwasser. Machine learning classification over encrypted data. *IACR Cryptol. ePrint Arch.*, 2014:331, 2015. doi: 10.14722/ndss.2015.23241.

[23] B. B. Gupta, Shingo Yamaguchi, Zhiyong Zhang, and Konstantinos E. Psannis. Guest editorial: Recent advances on security and privacy of multimedia big data in the critical infrastructure. *Multimedia Tools Appl.*, 77(23):31517–31524, dec 2018. ISSN 1380-7501. doi: 10.1007/s11042-018-6426-2. URL `https://doi.org/10.1007/s11042-018-6426-2`.

[24] Francisco-Javier González-Serrano, Adrián Amor-Martín, and Jorge Casamayón-Antón. Supervised machine learning using encrypted training data. *International Journal of Information Security*, 17(4):365–377, June 2017. doi: 10.1007/s10207-017-0381-1. URL `https://doi.org/10.1007/s10207-017-0381-1`.

[25] Pengtao Xie, Misha Bilenko, Tom Finley, Ran Gilad-Bachrach, Kristin E. Lauter, and Michael Naehrig. Crypto-nets: Neural networks over encrypted data. *CoRR*,

abs/1412.6181, 2014. doi: 10.48550/arXiv.1412.6181. URL http://dblp.uni-trier.de/db/journals/corr/corr1412.html#XieBFGLN14.

[26] Ehsan Hesamifard, Hassan Takabi, and Mehdi Ghasemi. Cryptodl: Deep neural networks over encrypted data, 2017. URL https://arxiv.org/abs/1711.05189.

[27] Chao-Yung Hsu, Chun-Shien Lu, and Soo-Chang Pei. Image feature extraction in encrypted domain with privacy-preserving sift. *IEEE Transactions on Image Processing*, 21(11):4593–4607, 2012. doi: 10.1109/TIP.2012.2204272.

[28] Dongmei Li, Xiaolei Dong, Zhenfu Cao, and Haijiang Wang. Privacy-preserving outsourced image feature extraction. *J. Inf. Secur. Appl.*, 47(C):59–64, aug 2019. ISSN 2214-2126. doi: 10.1016/j.jisa.2019.03.020. URL https://doi.org/10.1016/j.jisa.2019.03.020.

[29] Haomiao Yang, Yunfan Huang, Yong Yu, Mingxuan Yao, and Xiaosong Zhang. Privacy-preserving extraction of HOG features based on integer vector homomorphic encryption. In *Information Security Practice and Experience*, pages 102–117. Springer International Publishing, 2017. doi: 10.1007/978-3-319-72359-4_6. URL https://doi.org/10.1007/978-3-319-72359-4_6.

[30] Marta Gomez-Barrero, Emanuele Maiorana, Javier Galbally, Patrizio Campisi, and Julian Fierrez. Multi-biometric template protection based on homomorphic encryption. *Pattern Recognition*, 67, 01 2017. doi: 10.1016/j.patcog.2017.01.024.

[31] Pascal Paillier. Public-key cryptosystems based on composite degree residuosity classes. In Jacques Stern, editor, *Advances in Cryptology — EUROCRYPT '99*, pages 223–238, Berlin, Heidelberg, 1999. Springer Berlin Heidelberg. ISBN 978-3-540-48910-8. doi: 10.1007/3-540-48910-X_16.

[32] Mauro Barni, Tiziano Bianchi, Dario Catalano, Mario Di Raimondo, Ruggero Donida Labati, Pierluigi Failla, Dario Fiore, Riccardo Lazzeretti, Vincenzo

Piuri, Alessandro Piva, and Fabio Scotti. A privacy-compliant fingerprint recognition system based on homomorphic encryption and fingercode templates. In *2010 Fourth IEEE International Conference on Biometrics: Theory, Applications and Systems (BTAS)*, pages 1–7, 2010. doi: 10.1109/BTAS.2010.5634527.

[33] Ferhat Ozgur Catak, Sule Yildirim Yayilgan, and Mohamed Abomhara. A privacy-preserving fully homomorphic encryption and parallel computation based biometric data matching, 07 2020.

[34] Julien Bringer, Herve Chabanne, Melanie Favre, Alain Patey, Thomas Schneider, and Michael Zohner. GSHADE: Faster Privacy-Preserving Distance Computation and Biometric Identification . In *Proceedings of the 2nd ACM workshop on Information hiding and multimedia security - MMSec '14*. ACM Press, 2014. doi: 10.1145/2600918.2600922. URL `https://doi.org/10.1145/2600918.2600922`.

[35] Siddhant Deshmukh, Henry Carter, Grant Hernandez, Patrick Traynor, and Kevin Butler. Efficient and secure template blinding for biometric authentication. In *2016 IEEE Conference on Communications and Network Security (CNS)*, pages 480–488, 2016. doi: 10.1109/CNS.2016.7860539.

[36] Taeyun Kim, Yongwoo Oh, and Hyoungshick Kim. Efficient privacy-preserving fingerprint-based authentication system using fully homomorphic encryption. *Security and Communication Networks*, 2020:4195852, Feb 2020. ISSN 1939-0114. doi: 10.1155/2020/4195852. URL `https://doi.org/10.1155/2020/4195852`.

[37] Gizem S. Çetin, Erkay Savaş, and Berk Sunar. Homomorphic sorting with better scalability. *IEEE Transactions on Parallel and Distributed Systems*, 32(4):760–771, 2021. doi: 10.1109/TPDS.2020.3030748.

[38] Zvika Brakerski, Craig Gentry, and Vinod Vaikuntanathan. (leveled) fully homomorphic encryption without bootstrapping. *ACM Trans. Comput. Theory*, 6(3), jul

2014. ISSN 1942-3454. doi: 10.1145/2633600. URL https://doi.org/10.1145/2633600.

[39] Ilia Iliashenko and Vincent Zucca. Faster homomorphic comparison operations for bgv and bfv. *Proceedings on Privacy Enhancing Technologies*, 2021(3):246–264, 2021. doi: doi:10.2478/popets-2021-0046. URL https://doi.org/10.2478/popets-2021-0046.

[40] Rafael Torres Moreno, Jesús García Rodríguez, Cristina Timón López, Jorge Bernal Bernabe, and Antonio Skarmeta. Olympus: A distributed privacy-preserving identity management system. In *2020 Global Internet of Things Summit (GIoTS)*, pages 1–6, 2020. doi: 10.1109/GIOTS49054.2020.9119663.

[41] Lin You, Qiang Zhu, and Gengran Hu. A novel nizk-based privacy preserving biometric identification scheme for internet of things. Cryptology ePrint Archive, Report 2022/460, 2022. https://ia.cr/2022/460.

[42] Hasini Gunasinghe, Ashish Kundu, Elisa Bertino, Hugo Krawczyk, Suresh Chari, Kapil Singh, and Dong Su. Prividex: Privacy preserving and secure exchange of digital identity assets. In *The World Wide Web Conference*, WWW '19, page 594–604, New York, NY, USA, 2019. Association for Computing Machinery. ISBN 9781450366748. doi: 10.1145/3308558.3313574. URL https://doi.org/10.1145/3308558.3313574.

[43] David Nuñez and Isaac Agudo. Blindidm: A privacy-preserving approach for identity management as a service. *International Journal of Information Security*, 13(2):199–215, Apr 2014. ISSN 1615-5270. doi: 10.1007/s10207-014-0230-4. URL https://doi.org/10.1007/s10207-014-0230-4.

[44] Gaetan Pradel and Chris Mitchell. Privacy-preserving biometric matching using homomorphic encryption. In L Zhao, N Kumar, R C Hsu, and Z Zou, editors, *Proceedings: 2021 IEEE 20th International Conference on Trust, Security and Pri-*

*vacy in Computing and Communications, TrustCom 2021, Shenyang, China*. IEEE Press, October 2021. ISBN 978-1-6654-1658-0. doi: 10.1109/TrustCom53373.2021.00079.

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

[46] Ron Rothblum. Homomorphic encryption: From private-key to public-key. In Yuval Ishai, editor, *Theory of Cryptography*, pages 219–234, Berlin, Heidelberg, 2011. Springer Berlin Heidelberg. ISBN 978-3-642-19571-6. doi: 10.1007/978-3-642-19571-6_14.

[47] Laurent Fousse, Pascal Lafourcade, and Mohamed Alnuaimi. Benaloh's dense probabilistic encryption revisited. In *Progress in Cryptology – AFRICACRYPT 2011*, volume 6737, 08 2010. ISBN 978-3-642-21968-9. doi: 10.1007/978-3-642-21969-6_22.

[48] R. L. Rivest, A. Shamir, and L. Adleman. A method for obtaining digital signatures and public-key cryptosystems. *Commun. ACM*, 21(2):120–126, feb 1978. ISSN 0001-0782. doi: 10.1145/359340.359342. URL `https://doi.org/10.1145/359340.359342`.

[49] T. Elgamal. A public key cryptosystem and a signature scheme based on discrete logarithms. *IEEE Transactions on Information Theory*, 31(4):469–472, 1985. doi: 10.1109/TIT.1985.1057074.

[50] Shafi Goldwasser and Silvio Micali. Probabilistic encryption; how to play mental poker keeping secret all partial information. In *Proceedings of the Fourteenth Annual ACM Symposium on Theory of Computing*, STOC '82, page 365–377, New York, NY, USA, 1982. Association for Computing Machinery. ISBN 0897910702. doi: 10.1145/800070.802212. URL `https://doi.org/10.1145/800070.802212`.

[51] Dan Boneh, Eu-Jin Goh, and Kobbi Nissim. Evaluating 2-dnf formulas on ciphertexts. In Joe Kilian, editor, *Theory of Cryptography*, pages 325–341, Berlin, Heidelberg, 2005. Springer Berlin Heidelberg. ISBN 978-3-540-30576-7. doi: 10.1007/978-3-540-30576-7_18.

[52] Craig Gentry. Fully homomorphic encryption using ideal lattices. In *Proceedings of the Forty-First Annual ACM Symposium on Theory of Computing*, STOC '09, page 169–178, New York, NY, USA, 2009. Association for Computing Machinery. ISBN 9781605585062. doi: 10.1145/1536414.1536440. URL `https://doi.org/10.1145/1536414.1536440`.

[53] Yarkın Doröz, Gizem S. Çetin, and Berk Sunar. On-the-fly homomorphic batching/unbatching. In Jeremy Clark, Sarah Meiklejohn, Peter Y.A. Ryan, Dan Wallach, Michael Brenner, and Kurt Rohloff, editors, *Financial Cryptography and Data Security*, pages 288–301, Berlin, Heidelberg, 2016. Springer Berlin Heidelberg. ISBN 978-3-662-53357-4. doi: 10.1007/978-3-662-53357-4_19.

[54] Jung Hee Cheon, Andrey Kim, Miran Kim, and Yongsoo Song. Homomorphic encryption for arithmetic of approximate numbers. In Tsuyoshi Takagi and Thomas Peyrin, editors, *Advances in Cryptology – ASIACRYPT 2017*, pages 409–437, Cham, 2017. Springer International Publishing. ISBN 978-3-319-70694-8. doi: 10.1007/978-3-319-70694-8_15.

[55] A.K. Jain, S. Prabhakar, Lin Hong, and S. Pankanti. Fingercode: a filterbank for fingerprint representation and matching. In *Proceedings. 1999 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (Cat. No PR00149)*, volume 2, pages 187–193 Vol. 2, 1999. doi: 10.1109/CVPR.1999.784628.

[56] Lifeng Sha, Feng Zhao, and Xiaoou Tang. Improved fingercode for filterbank-based fingerprint matching. In *Proceedings 2003 International Conference on*

*Image Processing (Cat. No.03CH37429)*, volume 2, pages II–895, 2003. doi: 10.1109/ICIP.2003.1246825.

[57] Fabian Boemer, Sejun Kim, Gelila Seifu, Fillipe D.M. de Souza, and Vinodh Gopal. Intel hexl: Accelerating homomorphic encryption with intel avx512-ifma52. In *Proceedings of the 9th on Workshop on Encrypted Computing & Applied Homomorphic Cryptography*, WAHC '21, page 57–62, New York, NY, USA, 2021. Association for Computing Machinery. ISBN 9781450386562. doi: 10.1145/3474366.3486926. URL https://doi.org/10.1145/3474366.3486926.

[58] Vasily Sidorov, Ethan Yi Fan Wei, and Wee Keong Ng. Comprehensive performance analysis of homomorphic cryptosystems for practical data processing, 2022. URL https://arxiv.org/abs/2202.02960.