Master's thesis

Håvard Borgen Myrekrok
Lasse Vad

Towards Privacy-Preserving Fingerprint Verification

Master's thesis in Communication Technology and Digital Security
Supervisor: Anamaria Costache
Co-supervisor: Pia Bauspieß
June 2022

NTNU
Norwegian University of Science and Technology
Faculty of Information Technology and Electrical Engineering
Dept. of Information Security and Communication Technology

Håvard Borgen Myrekrok
Lasse Vad

Towards Privacy-Preserving Fingerprint Verification

Master's thesis in Communication Technology and Digital Security
Supervisor: Anamaria Costache
Co-supervisor: Pia Bauspieß
June 2022

Norwegian University of Science and Technology
Faculty of Information Technology and Electrical Engineering
Dept. of Information Security and Communication Technology

NTNU
Norwegian University of
Science and Technology

Title:	Towards Privacy-Preserving Fingerprint Verification
Student:	Håvard Borgen Myrekrok, Lasse Vad

Problem description:

Fingerprint authentication is a widely used authentication method, as it is both accurate and difficult to forge. It also does not require users to store or manage a password, which is another advantage in using fingerprint authentication. Fingerprint authentication is also applicable as part of multi-factor authentication (MFA) as the inherence factor. Therefore, it is common to combine fingerprint verification with username and password to enhance security through MFA.

Fingerprints are regarded as sensitive personal information and should be protected in systems used for automated authentication. Fingerprint templates stored in plaintext in a database is at risk in the event of a data breach, but encryption can be applied when the fingerprint templates are at rest to mitigate the consequence of such an event. However, several fingerprint recognition algorithms are designed to compare fingerprint templates in plaintext, which requires the fingerprint templates to be decrypted before they can be compared. This exposes the fingerprint templates to additional risk of data leakage during the biometric transaction. Secure fingerprint comparison is therefore necessary for the use of fingerprints as authentication mechanism in applications provided in the cloud and other third parties over the internet. Fully Homomorphic Encryption (FHE) is an encryption technique that enables the possibility of computing on encrypted data, which is suitable in systems where calculations need to be performed on sensitive data. While there are existing comparison algorithms which balance accuracy and efficiency, the aspect of privacy and security in fingerprint recognition is still an active research area.

The goal of this thesis is to study several existing fingerprint comparison algorithms, which compare variable-sized fingerprint templates, and implement a privacy-preserving solution using FHE. The chosen algorithm will be one that can be modified in a way which is suitable to implement with FHE.

Date approved:	2022-01-27
Responsible professor:	Anamaria Costache, NTNU, IIK
Supervisor(s):	Pia Bauspieß, NTNU, IIK

Abstract

In offices and border controls, authentication is often verified by physical ID cards or passports, while in a digital setting, authentication is performed using passwords created by the users. Unfortunately, there are vulnerabilities related to most of these authentication methods. ID cards and passports are at risk of being stolen or forged, and passwords could be guessed or leaked as a result of a cyber attack. Fingerprints are currently not used for authentication in use cases that require a high level of security, even though they are one of the unique features of a human being. In order to verify a data subject's fingerprint, both an accurate and efficient fingerprint comparison algorithm is desired. However, using fingerprints for authentication is not without risk either. A challenge is the privacy aspect of using fingerprints for authentication in a digital setting, as it is regarded as sensitive information and should not be publicly exposed. The aim of this master thesis is to implement a fingerprint verification application that is privacy-preserving without compromising on accuracy.

The proposed solution uses a Fully Homomorphic Encryption (FHE) scheme, which has the property of computing on encrypted data, to achieve secure verification of fingerprints. An existing fingerprint comparison algorithm is used as the basis for the proposed solution's fingerprint templates and comparison algorithm. This thesis' contribution is a fingerprint verification application operating in the encrypted domain, using encrypted fingerprint templates as input. It is implemented in C++, and PALISADE, a library supporting a range of FHE schemes and operations, provides the FHE capabilities used in the implementation.

Having the implementation allows for gathering results through experimentation, testing, and several executions with different fingerprints. The results verify that the implementation has the same accuracy in the encrypted domain as in cleartext. Based on the results, the implementation is also discussed regarding runtime performance, and an analysis of the security and privacy of the system is conducted. In this final discussion, this implementation's deviations from the ideal solution are presented, leading to a conclusion and a section suggesting further work.

Sammendrag

I kontorlokaler og ved grensekontroller blir autentisering ofte gjennomført ved bruk av fysiske ID-kort eller pass, mens i en digital setting er det som oftest autentisering med brukerdefinerte passord dersom det ikke er et system som krever sterk autentisering. Dessverre er det sårbarheter knyttet til de fleste av disse autentiseringsmetodene. ID-kort og pass innehar en risiko for å bli stjålet eller forfalsket, og passord kan enten bli gjettet eller lekket som resultat av cyberangrep. Fingeravtrykk er foreløpig ikke brukt som autentisering i bruksområder som krever et høyt sikkerhetsnivå, til tross for at det er en av de mest unike karakteristikkene hos mennesker. For å verifisere fingeravtrykket til et subjekt er det ønskelig med en både presis og effektiv fingeravtrykksgjenkjenning. Imidlertid er bruken av fingeravtrykk som autentisering heller ikke risikofritt. En utfordring er personvernsaspektet ved bruk av fingeravtrykk som autentisering i digitale settinger, ettersom det er klassifisert som sensitiv informasjon og ikke bør bli lekket offentlig. Denne masteroppgaven forsøker å lage en fingeravtrykkgjenkjenningsapplikasjon som bevarer personvern uten at det går på bekostning av nøyaktighet.

Den foreslåtte løsningen bruker et Fullstendig Homomorft Krypterings (FHE) skjema, som muliggjør å kalkulere på kryptert data. Hensikten med det er å oppnå sikker gjenkjenning av fingeravtrykk. En eksisterende fingeravtrykksgjenkjenningsalgoritme blir brukt som utgangspunkt for den foreslåtte løsningens fingeravtrykksformat og gjenkjenningsalgoritme. Denne masteroppgaven sitt bidrag er en fingeravtrykksgjenkjenningsimplementasjon som opererer i det krypterte domenet, hvor krypterte fingeravtrykksformat blir brukt som inndata. Den er skrevet i C++, og PALISADE, et bibliotek som støtter en rekke FHE skjemaer og operasjoner, gir FHE funksjonaliteten som blir brukt i implementasjonen.

Implementasjonen gjør det mulig å samle resultater gjennom eksperimentering, testing, og en rekke kjøringer med ulike fingeravtrykk. Resultatene viser at implementasjonen har samme presisjon i det krypterte domenet og i klartekst. Basert på resultatene blir også implementasjonen diskutert med hensyn til kjøretidsytelse, og en analyse av systemets sikkerhet og personvern blir gjennomført. I den endelige diskusjonen blir avvikene denne implementasjonen har fra den ideelle løsningen presentert, som leder til en konklusjon og en seksjon om forslag til videre arbeid.

Preface

This master's thesis concludes our Master of Science Degree in Communication Technology and Digital Security at Norwegian University of Science and Technology (NTNU).

We want to express our gratitude to Anamaria Costache and Pia Bauspieß for the tremendous support that we have received these final two semesters of our master's degree. Your guidance and advice kept our spirit high and helped us navigate the challenges we faced during this thesis. Also, we thank the biometrics and internet security research group da/sec for their help with a field of study that was new to us.

Last but not least, we want to thank our family and friends for their support and for helping us maintain a healthy study-life balance.

Contents

List of Figures

List of Tables

List of Source Codes

List of Acronyms

3D Three-Dimensional.

AS Authentication Server.

BFV Brakerski-Fan-Vercauteren.

BGV Brakerski-Gentry-Vaikuntanathan.

CKKS Cheon-Kim-Kim-Song.

CS Computation Server.

CVP Closest Vector Problem.

DET Detection Error Tradeoff.

FHE Fully Homomorphic Encryption.

FMR False Match Rate.

FNMR False Non-Match Rate.

HE Homomorphic Encryption.

KPI Key Performance Indicators.

LFHE Leveled Fully Homomorphic Encryption.

LWE Learning With Errors.

MCC Minutiae Cylinder-Code.

NIST National Institute of Standards and Technology.

NTNU Norwegian University of Science and Technology.

PHE Partially Homomorphic Encryption.

RLWE Ring Learning With Errors.

SOD Separation of Duty.

SVP Shortest Vector Problem.

SWHE Somewhat Homomorphic Encryption.

TEE Trusted Execution Environment.

TMR True Match Rate.

TNMR True Non-Match Rate.

Chapter 1

Introduction

The following chapter describes the motivation, defined scope and identified challenges for the thesis. An overview of the proposed implementation, the methodology of the thesis, and the research questions the thesis aims to answer are also covered.

1.1 Motivation

Historically, fingerprint recognition has been used by governments to identify individuals [Col04]. Today, it is also used in devices that are part of everyday life such as smartphones and laptops, where built-in fingerprint verification has become prevalent. Fingerprint recognition offers an alternative to using a password for authentication, but it can also be utilised as a part of multi-factor authentication. A drawback to using passwords is that they can be easy to guess for an adversary since users must create and remember passwords for many services and devices. Thus, many users choose common passwords that are prone to dictionary attacks. An advantage of using a biometric characteristic like a fingerprint for authentication is that it removes the need for remembering a password, in addition to being unique for each individual.

The General Data Protection Regulation classifies fingerprints as personal data [GDPR16], and fingerprint templates must therefore be handled in a way that ensures privacy and security. Even though templates are an abstracted representation of original fingerprint images, they allow for synthetic reconstruction of fingerprint images sufficient for attacks [CMLM07]. Therefore, storing and processing fingerprint templates centrally can have severe consequences since it exposes them to an increased risk for data breaches. Each individual only has a maximum of 10 fingerprints to choose from in contrast to a password, and one can not change a fingerprint as one would do with a password after a data breach. Protecting the fingerprint templates from being disclosed is therefore essential. Smartphone and laptop operating system developers have solved security challenges related to fingerprints by handling the processing and comparison of fingerprints locally in each device. Examples of fingerprint recognition can be found in Android and Microsoft devices. For Android devices, a secure operating system called Trusty can handle the processing of fingerprints in a trusted execution environment [Ty20], while Microsoft has developed a biometrics security system called Windows Hello [Wh22]. In both solutions, hardware-based encryption protects the biometric data, and they are not shared with external parties. Therefore, fingerprints are only available for authentication in the devices they have been enrolled to.

Other use cases, however, require centralised storage of biometric data, such as automated border control at airports or watchlist searches by governmental institutions. Such cases involve enrolling fingerprint templates to a server instead of individual devices at a time. However, this introduces a security challenge, as sharing and distributing fingerprint templates must not impact their confidentiality and integrity. Therefore, the question arises whether fingerprint templates can be securely compared by an untrusted server such that the server does not learn the content of the templates it compares.

1.2 Scope

Biometric comparison in the encrypted domain has been researched and achieved for fixed-length biometric representations [Bod18] [KDG+20]. However, like other biometric traits, every fingerprint is unique and contains various amounts of information, and reducing this information to a fixed-length template can influence the comparison accuracy. The original representation for fingerprint templates is minutia templates, where minutiae are the unique features in fingerprints, defined by the ridges and valleys. The scope of this thesis is to compare variable-length minutiae-based fingerprint templates in the encrypted domain. The purpose of comparing in the encrypted domain is to ensure privacy protection for the fingerprint templates. The recognition accuracy for secure comparisons could be improved by using minutia-templates instead of fixed-size representations, since it would allow for more detailed comparisons between fingerprints. A dip in performance has been shown to occur with fixed-size representations [JPHP99]. We have implemented and adapted a state-of-the-art fingerprint comparison algorithm to take two encrypted minutia-based templates as input and verify whether they originate from the same data subject. This way, the algorithm will not access the personal information contained in the templates. We are not concerned with fingerprint identification, where one fingerprint template is compared to a potentially large database of fingerprint templates and the workload that comes with this transaction, but rather a one-to-one comparison between fingerprints.

FHE is an encryption technique that is suitable for solving this challenge. Ideally, the algorithm for comparing fingerprint templates should be partially or entirely implemented with FHE, making the computations of the comparison algorithm encrypted. The implementation should provide privacy in the specific area of focus, the comparison, and at the same time preserve efficiency and performance as much as possible. Additionally, the accuracy of the biometric comparison should not be reduced in the secure solution as this would mitigate the system's security. Creating fingerprint templates by extracting features from fingerprint images is out of scope for this thesis. Therefore, an existing feature extraction algorithm will be used to generate fingerprint templates.

1.3 Challenges

There exist several known challenges related to the task of this thesis. One is that FHE is known to add computational overhead, making the operations performed in the encrypted domain slower than in the cleartext domain, which will likely impact the usability of the implementation. FHE also has some limitations in terms of computational complexity and operations. While FHE, in theory, allows for the secure evaluation of arbitrary functions, conditional expressions are too complex to perform in the encrypted domain since the evaluated values are encrypted and therefore hidden. An example of a conditional expression is to evaluate whether a value is equal to or less than another value. Some information may, in that case, have to be accessed in cleartext to make decisions that are important for the algorithm. In order to implement an algorithm with FHE where the complete algorithm works within the encrypted domain, the complexity of the operations must not surpass the operations FHE support, meaning that every step of the algorithm should be manageable to perform encrypted using FHE. In this thesis, the algorithm used for comparing templates will be based on an existing fingerprint comparison algorithm that may be modified for compatibility with FHE.

Comparison of fixed-length representations in the encrypted domain can be performed using simple operations. Previous research has, for instance, used Euclidean distance to calculate the difference between fixed-length templates, which is an easy operation with FHE [JPHP99]. Comparing variable-length minutia-templates in the encrypted domain is a more challenging task, and it is still an open challenge that is being researched [ECJ19].

1.4 Research Questions

In this thesis, we aim to answer the following four research questions:

RQ1	Is it possible to implement an existing fingerprint comparison algorithm with FHE?
RQ2	Which factors influence the performance of the FHE fingerprint comparison algorithm most significantly?
RQ3	Can FHE be used to improve the privacy of fingerprint comparison algorithms?
RQ4	Can variable-sized fingerprint templates efficiently be compared in the encrypted domain?

Table 1.1: Research questions.

1.5 Methodology

The aim of this thesis was to implement a fingerprint comparison algorithm in the encrypted domain. To achieve this, the work was divided into five parts: 1) Research phase, 2) Cleartext implementation, 3) Testing and validation of cleartext implementation, 4) Encrypted implementation and 5) Testing and validation of encrypted implementation.

1.5.1 Research Phase

Research on existing fingerprint comparison algorithms was conducted in the research phase to find a suitable algorithm to implement in the encrypted domain. In order to determine which algorithms were suitable, the limitations of FHE were considered. Another criterion for a suitable algorithm was that it used minutia-templates, which consist of minutiae information from the fingerprints, such as coordinates, angles and minutia type.

This phase also involved research related to FHE, such as which operations FHE supports, and libraries that provide FHE capabilities. Some time was also spent on getting acquainted with the FHE libraries to decide which one was suitable and practical to use. The decision of which fingerprint comparison algorithm to implement and which FHE library to use concluded this task.

1.5.2 Cleartext Implementation

An implementation of the algorithm of choice was done in this phase. The approach of this phase was to follow the original paper describing the chosen algorithm and implement it as accurately as possible, preparing it for the later encryption.

1.5.3 Testing and Validation of Cleartext Implementation

Both during and after the implementation, several tests were conducted to evaluate the accuracy and performance of the algorithm on cleartext data. Different databases with fingerprint images were used to observe how well the implementation could handle different fingerprint image qualities, sizes, and alignments.

Analysis of the implementation's performance was done on data collected with a quantitative data collection method. For biometric comparisons, there is a selection of relevant key performance indicators that includes but is not limited to: 1) False Non-Match Rate (FNMR), 2) False Match Rate (FMR), and 3) the time to perform a comparison, in other words, its efficiency [ISO21]. FNMR are cases where the implementation falsely rejects a biometric claim that should have been accepted. This can be represented as a percentage of falsely rejected individuals compared to the total number of comparisons performed. FMR are the cases in which the implementation falsely accepts a biometric claim that should have been rejected and can be represented with a percentage of falsely accepted individuals compared to the total number of comparisons performed. These indicators were used to evaluate the cleartext implementation.

This phase concluded when the implementation satisfied the demands for practical usage so that accuracy would not be a concern when introducing encryption to the implementation.

1.5.4 Encrypted Implementation

The aim of this stage was to use the knowledge and resources available on FHE to incorporate an encryption layer in the current cleartext implementation to make it privacy-preserving. More specifically, the templates which describe the fingerprints should reveal as little information as possible in the comparison process of the algorithm, which is the part a potential third party server would perform. With this idea, the optimal scenario would be for a client to provide their fingerprint template in the encrypted format before sending it to the algorithm provider and then receive an acceptance/rejection message back in a secure format. Figure 1.1 illustrates the ideal usage of the encrypted solution.

1.5.5 Testing and Validation of Encrypted Implementation

The important part regarding biometric accuracy in the encrypted implementation was maintaining the same biometric accuracy as the cleartext implementation. Testing and validation of the encrypted implementation were conducted similarly to the testing and validation of the cleartext implementation, but with some additional elements. Accuracy was expected to be the same in the cleartext and encrypted

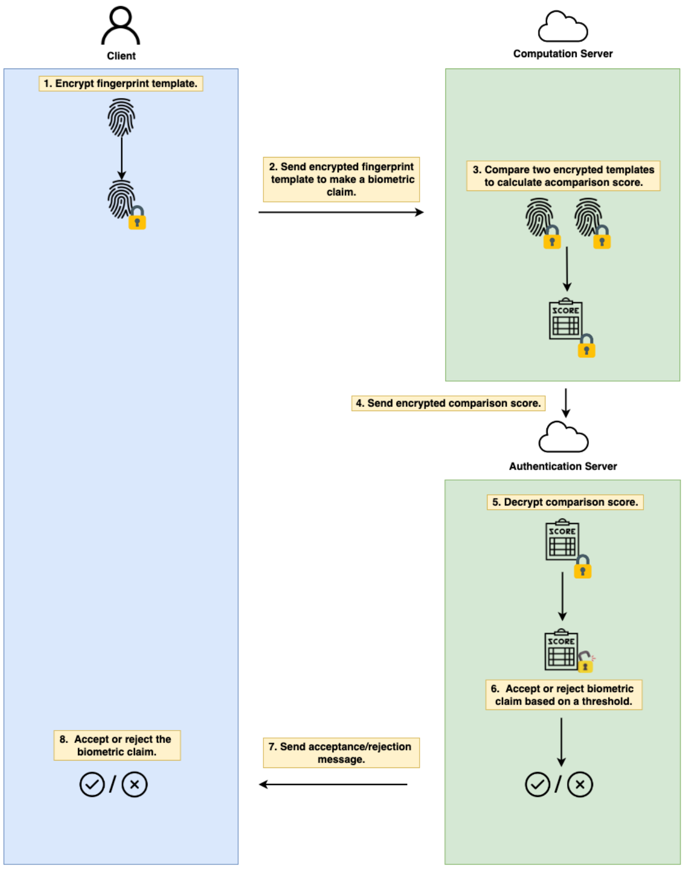

Client
Computation Server
1. Encrypt fingerprint template.
2. Send encrypted fingerprint template to make a biometric claim.
3. Compare two encrypted templates to calculate acomparison score.
SCORE
4. Send encrypted comparison score.
Authentication Server
5. Decrypt comparison score.
6. Accept or reject biometric claim based on a threshold.
7. Send acceptance/rejection message.
8. Accept or reject the biometric claim.

Figure 1.1: Simplification of the ideal architecture of the encrypted solution.

implementations, since FHE schemes typically only add a small error to encrypted values. In order to validate this, we compared the similarity scores from the encrypted and the cleartext implementation to ensure that they did not differ. If the scores were equal, a conclusion could be drawn that both implementations share the same accuracy, and the key performance indicators would also be equal.

Runtime performance was expected to differ significantly from cleartext to encrypted implementation due to the computational complexity of FHE. A quantitative test was run with several fingerprint comparisons to evaluate the drop in performance. Optimisations were also added to enhance its performance.

From the privacy perspective, qualitative analysis and evaluation were conducted to evaluate how privacy-preserving the solution is and to identify its shortcomings.

1.6 Outline

The remainder of the thesis is divided into five chapters with the following structure:

Chapter 2: Preliminaries Introduces biometric terminology, mathematical functions and cryptography definitions, which are relevant in the proposed solution.

Chapter 3: Related Work Addresses previous work from the same research area, which are either similar or related to the work conducted in this thesis.

Chapter 4: Proposed Solution Presents the proposed solution that is designed as part of this thesis. The chapter describes related libraries, algorithms and the implementation from both a high-level and a technical perspective.

Chapter 5: Results and Discussion Presents and discusses various results related to the implementation, including verification and time performance, as well as the security and privacy achieved. Technical specifications that made up the testing environment are also introduced.

Chapter 6: Conclusion Summarises the problem in light of the proposed solution and the generated results, and concludes by presenting to which degree this solution tackles the challenge of encrypted fingerprint verification. This chapter also includes further work.

Chapter 2 Preliminaries

The premise of this thesis comes from a combination of the scientific fields of biometrics and cryptography. The following chapter provides the reader with the necessary knowledge about the two fields to understand the work conducted in this thesis.

2.1 Biometrics

Biometrics is the automated recognition of individuals based on their observable characteristics [ISO17]. Biometric characteristics can be biological characteristics, for instance, fingerprint structures, iris patterns, or behavioural characteristics like walking or typing style. A common factor for these characteristics is that they are unique for every individual and can therefore be regarded as a valid authentication method to a certain degree of precision. The focus of this thesis is on fingerprint verification. Therefore, this section elaborates on how fingerprints are used for automated comparison and verification.

2.1.1 Fingerprint Features

Ridges in the skin are the information that optical or physical sensors can observe to enable automated fingerprint recognition. The ridges form together different patterns that become the basis for fingerprint verification. It is possible to look for a coarse classification of fingerprints or a more detailed view of a fingerprint with several identifiable points. Even smaller structures within the ridges, such as sweat pores, can also be analysed but are seldom used as they require a scanning device with high quality that outputs an image with a dots per inch value of 1000 in order to be accurate [ZZZL10]. To identify ridge patterns, it is sufficient to use images with dots per inch value of 500 [ZLZ+11]. Fingerprint verification can be divided into three levels: level 1) analysis of global patterns, level 2) analysis of Galton's details, and level 3) analysis of sweat pores.

Global Patterns

Singularities are points in the fingerprint that make up a global pattern based on their relative position to each other. Singularities can be used for both fingerprint classification and alignment of fingerprints, due to their rotation immutability. There are two types of singularities, as defined by ISO/IEC 19794-8 [ISO11]:

- Core — A singular point in the fingerprint, where the curvature of the ridges reaches a maximum.
- Delta — Structure where three fields of parallel ridge lines meet.

Figure 2.1 visualises where these global patterns are found in a fingerprint.

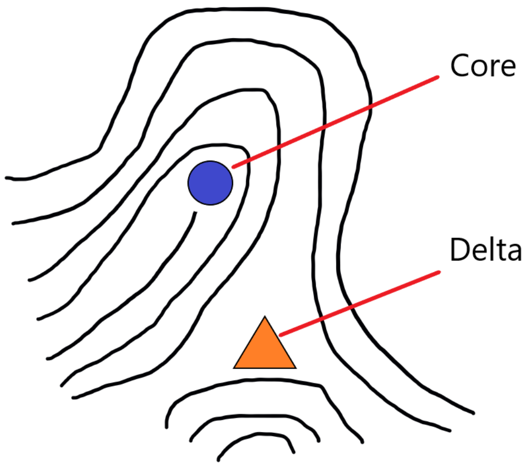


Figure 2.1: Singularities in a fingerprint.

Galton's Details

Minutiae points are located where a ridge line ends or splits into two or more ridge lines. Therefore, the different minutiae points are ridge ending, bifurcation and trifurcation, also known as Galton details. The composition of minutiae points serves as a more detailed classification of a fingerprint than global patterns since a good fingerprint image contains about 40-100 minutiae [Zae11]. The minutiae points extracted from a fingerprint can be used for fingerprint verification and is the most common approach for fingerprint verification today.

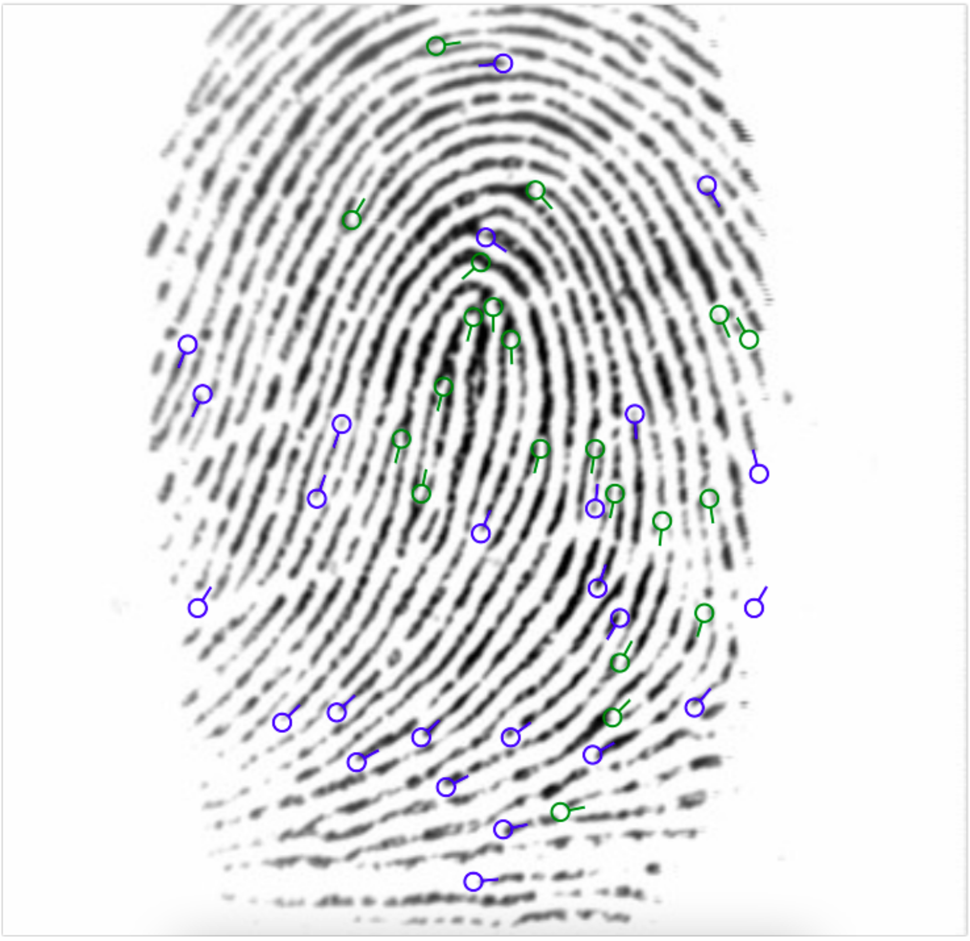


Figure 2.2: Minutiae in fingerprint. Image source: https://sourceafis.machinezoo.com/algorithm.

2.1.2 Fingerprint Templates

Fingerprint templates are the stored features extracted from a fingerprint. These have been standardised into specific encoding formats, for instance, ISO 19497-2 and ANSI 381. Which fingerprint template to use is decided by what information the fingerprint recognition requires, and some fingerprint algorithms may require specific templates. Using a standard encoding format ensures interoperability between the different subsystems involved in a fingerprint recognition system. If a more efficient fingerprint recognition algorithm became available and used the same fingerprint template as the fingerprint database, a transition to the new algorithm would not require a new enrolment of fingerprints.

Fingerprint templates with the same encoding formats are not necessarily the same length, as fingerprints contain a varying amount of minutiae. An example of a fingerprint template can be a set of minutiae with the following information for each minutia: [x-coordinate, y-coordinate, angle, minutia type]. FingerCode is another approach to fingerprint templates, where the local and global patterns in a fingerprint are compressed into a fixed length of 640 bytes [JPHP99].

2.1.3 Fingerprint Verification System

A fingerprint verification system can be generalised as consisting of two stages, the enrolment and verification stage. The enrolment stage is the initial stage. An individual, referred to as the data subject, presents their finger to a fingerprint capture device, either a fingerprint scanner or a camera. The captured fingerprint sample will usually be pre-processed to reduce unnecessary information and make the raw fingerprint sample better suited for feature extraction of the relevant features. Feature extraction results in a reference fingerprint template, consisting of the coordinates and angles of the minutia points, which is then stored in the system's reference database.

The second stage is the verification stage. Here, a data subject will attempt to be authenticated. The data subject presents their fingerprint as in the enrolment stage and makes a biometric claim. A probe fingerprint template is extracted from the data subject's captured fingerprint sample and is compared to the reference fingerprint template retrieved from the reference database corresponding to the subject's claimed identity. The comparison between a probe and a reference fingerprint template can be obtained in a number of ways, three of which will be elaborated upon in this thesis. A comparison score is derived from the comparison, and the data subject is authenticated if the comparison score is above a set threshold.

2.1.4 Challenges in Fingerprint Verification

Alignment

Fingerprint comparison algorithms must consider that each captured fingerprint sample is not consistently aligned in the same way during the capture process. Different techniques can be used to align the probe and reference fingerprint, and they are dependent on which features are extracted from the fingerprint image. One option is to use a global pattern in the reference fingerprint and rotate the probe fingerprint to match the global pattern if there is a similar global pattern in the probe fingerprint. Another possibility is to use Galton's details as proposed by [ZYZ05], by using multiple pairs of reference minutiae to align fingerprints prior to comparing. Instead of using a single minutia pair from the probe and reference template and aligning based on it, it uses multiple minutia pairs. This approach achieves a better alignment than if a single minutia pair is used. In comparison, [JPHP99] identifies a reference point as the maximum curvature of ridges and derives from that point a reference axis. Together, the reference point and axis establish an invariant reference frame.

Distortion

Fingerprint images are prone to non-linear distortion caused by uneven pressure applied to the fingerprint capture device. This can cause the fingerprint to appear different, which is critical when used for comparisons. A typical example is ink based fingerprint collection, where the subject might swipe ink over the fingerprint details by not holding the finger still. Another cause of distortion can come from having too much ink on the finger, which makes the fingerprint too thick and dark to obtain the important details. Similar non-linear distortions occur in digital sensors, where the problem of correcting the sensors remains. As a contribution to this problem, [CMM01] proposes a mathematical instrument for counteracting the distortion problem in fingerprints, allowing the development of more distortion-invariant fingerprint comparison algorithms.

Sample quality

Fingerprints are often collected as images, which adds the element of image quality as a potential challenge. If the image consists of too few pixels, or the contrasts of black and white are not clear enough, it might pose a challenge. The skin's surface may also be wet or dirty, negatively impacting the sample quality. The details on the outermost skin layer may also be affected if the individual has been in contact with rough surfaces, making the ridges in the skin less distinct. Preprocessing is performed to mitigate poor sample quality, but nevertheless it is one of the main reasons why biometric data is fuzzy.

2.2 Distance and Norm Functions

Calculating distances are used for biometric comparisons, as it can contribute towards determining similarity between vectors, which is a representation often used for biometric templates. In mathematics, a distance function is a function that can calculate the distance between two points or between each pair of points in two sets of elements.

Definition 2.1. Euclidean Distance Let $\vec{x}, \vec{y} \in \mathbb{R}^n$. Then the Euclidean distance is defined as:

$$d(\vec{x}, \vec{y}) = \sqrt{\sum_{n=1}^{n} (x_i - y_i)^2}. \tag{2.1}$$

It is more advantageous for several implementations of biometric comparison algorithms to work with squared Euclidean distance, due to the cost and complexity of performing the square root.

Definition 2.2. The Squared Euclidean Distance Let $\vec{x}, \vec{y} \in \mathbb{R}^n$. Then the squared Euclidean distance is defined as:

$$d(\vec{x}, \vec{y})^2 = \sum_{n=1}^{n} (x_i - y_i)^2. \tag{2.2}$$

Definition 2.3. Euclidean Norm On the n-dimentional Euclidean space $\mathbb{R}^n$, the length of the vector $x = (x_1, x_2, ..., x_n)$ is given by the formula:

$$\|x\| = \sqrt{x_1^2 + x_2^2 + ... + x_n^2}. \tag{2.3}$$

2.3 Cryptography

Cryptography is the practice and study of techniques for secure communication in the presence of adversarial behaviour [Riv90]. Cryptography has significantly evolved since its humble beginnings in antiquity when a deterministic algorithm of shifting a letter by ten places could be sufficient protection for a message. More advanced algorithms are necessary to withstand today's computing resources, and most cryptography relies on problems that are hard to solve even for a computer. This section presents definitions and cryptographic algorithms that are relied upon in this thesis.

2.3.1 Cryptosystem

In its simplest form, a cryptosystem is a specified set of algorithms and alphabets that are used to hide or reveal messages by using defined keys [SP18]. It can be summarised as a tuple $(\mathcal{P}, \mathcal{C}, \mathcal{K}, \mathcal{E}, \mathcal{D})$, where:

- **Plaintext space** $\mathcal{P}$ — Defines the alphabet of symbols that messages can be composed of.
- **Ciphertext space** $\mathcal{C}$ — Defines the alphabet of symbols that encrypted messages can be composed of. The ciphertext space might or might not be equal to the plaintext space.
- **Key space** $\mathcal{K}$ — Defines the set of all possible (distinct) keys. There can be different sets of keys for specific purposes, such as encryption or decryption.
- **Encryption algorithm** $\mathcal{E}$ — Defines the algorithm that converts a string from the plaintext space into a string from the ciphertext space. It usually takes an encryption key as input.

- **Decryption algorithm** $\mathcal{D}$ — Defines the algorithm that converts a string from the ciphertext space into a string from the plaintext space. It usually takes a decryption key as input.

Cryptosystems in the FHE category have additional algorithms that support the evaluation of ciphertexts, key switching, and other algorithms that implement the desired functionality specific to the FHE cryptosystem. Thus, cryptosystems vary concerning their capabilities and the functionality they offer. However, most modern cryptosystems are designed to have four basic properties:

- **Confidentiality** — The property that sensitive information is not disclosed to unauthorized entities [Bar20].
- **Integrity** — The property that sensitive data has not been modified or deleted in an unauthorized and undetected manner since it was created, transmitted or stored [Bar20].
- **Authentication** — A process that assures the source and integrity of information in communications sessions, messages, documents or stored data or that assures the identity of an entity interacting with a system [Bar20].
- **Non-repudiation** — Assurance that the sender of information is provided with proof of delivery and the recipient is provided with proof of the sender's identity, so neither can later deny having processed the information [Bar20].

2.3.2 Lattice-based Cryptography

The encryption schemes that are used in this thesis are lattice-based. Hence, a brief introduction to lattice-based cryptography and the hardness the security is based on is provided in this section.

A lattice is a subgroup of $\mathbb{R}^n$ of the form

$$\Lambda = \{\sum_{i=1}^{r} a_i \boldsymbol{b}_i \mid a_1, \ldots, a_r \in \mathbb{Z}\},$$

where $\boldsymbol{b}_1, \ldots, \boldsymbol{b}_r$ are linearly independent vectors in $\mathbb{R}^n$.

A basis for a lattice Λ is any set of linearly independent vectors $\boldsymbol{c}_1, \ldots, \boldsymbol{c}_r$, such that $\Lambda = \{\sum_i a_i \boldsymbol{c}_i \mid a \in \mathbb{Z}^n\}$.

From a basis $\boldsymbol{b}_1, \ldots, \boldsymbol{b}_r$ we can construct a $r \times n$ matrix B by letting the ith row be $\boldsymbol{b}_i$, $i = 1, 2, \ldots, r$. Then

$$\Lambda = \{aB \mid a \in \mathbb{Z}^n\}. \tag{2.4}$$

Hard Problems

The security of using lattices in cryptography can be based on two fundamental hard problems, Shortest Vector Problem (SVP) and Closest Vector Problem (CVP).

Definition 2.4. Shortest Vector Problem Given a lattice Λ in $\mathbb{R}^n$, find a vector $\boldsymbol{x} \in \Lambda$, $\boldsymbol{x} \neq (0, \dots, 0)$, such that the Euclidean norm $\| \boldsymbol{x} \|$ is minimised. The $\boldsymbol{x}$ is then a shortest vector in Λ.

The SVP has been proved to be NP-hard [Kho05].

Definition 2.5. Closest Vector Problem Given a lattice Λ in $\mathbb{R}^n$ and a vector $\boldsymbol{y} \in \mathbb{R}^n$ that is not in Λ, find a vector $\boldsymbol{x} \in \Lambda$ such that $\| \boldsymbol{x} - \boldsymbol{y} \|$ is minimised. Vector $\boldsymbol{x}$ is then a closest vector to $\boldsymbol{y}$ in Λ.

The CVP has been proven to be NP-hard [DKRS03].

The Learning With Errors (LWE) problem is the basis for security in several lattice-based public-key cryptosystems, and it can be reduced to SVP and CVP. LWE is believed to be post-quantum secure since, at the moment, there is no known quantum speed-up that solves it.

Definition 2.6. Learning With Errors Distribution — Let q be a prime and n an integer. A discrete random variable $\mathbf{E}$ with probability distribution χ is defined on the set $\mathbb{Z}_q$, and m samples $(\boldsymbol{a^i}, b^i) \in (\mathbb{Z}_q^{n+1})$, where the m samples are constructed from a secret $\boldsymbol{s} = (s_1, s_2, \dots, s_n) \in (\mathbb{Z}_q)^n$. For $1 \leq i \leq m$, $\boldsymbol{a^i} = (a_1^i, \dots, a_n^i)$ is chosen uniformly at random from $(\mathbb{Z}_q)^n$, e^i is chosen according to the probability distribution χ and

$$b^i = e^i + \sum_{j=1}^{n} a^i s_j \pmod{q}. \tag{2.5}$$

Search LWE Given a set of samples sampled according to the LWE distribution, find the secret $(s_1, s_2, \dots, s_n)$.

Decision LWE Distinguish valid LWE samples from uniformly random samples.

Good and bad bases

Lattices have infinitely many bases, which we can categorise into good and bad basis vectors. Good basis vectors are close to being orthogonal and have small Euclidean norms. A bad basis has longer Euclidean norms, and its vectors are skewed. Orthogonal bases allow to solve SVP and CVP efficiently and correctly, while there exist no known algorithms that solve SVP and CVP for a bad basis when the dimension of the lattice is high.

Lattices in public-key cryptography

Hard lattice problems can serve as a foundation for public-key cryptography since they make it possible to construct a trapdoor one-way function [GGH97]. Using public-key cryptography for the confidentiality of data usually means having an asymmetric key pair consisting of one private key and one public key. The public key can be distributed openly and is used for encrypting data, while the private key is kept private and used for decrypting the data. A bad and good basis serves as a public and private key in lattice-based cryptography, respectively. Therefore, the trapdoor one-way function uses the bad basis to calculate a point in the lattice and add some error to it. Adding the error shifts the original point to a new point, and retrieving the original point is reduced to the CVP. Goldreich et al. shows that this is hard to do with a bad basis [GGH97]. Reducing a hard basis to a good basis is also a hard problem, namely the SVP. Knowledge of the good basis is what unlocks the trapdoor, and with the good basis finding the original point, essentially removing the error, becomes more trivial. Retrieving the encrypted data is easy when the original point in the lattice is known.

2.3.3 Homomorphic Encryption

Homomorphic Encryption (HE) is a class of encryption schemes that has the property of computing on encrypted data without having to decrypt it [RAD78]. Computing in this context means performing general arithmetic on the data, such as addition and multiplication. When the data is decrypted, the result is the same as it would have been if the computations were done in plaintext. There are several types of HE schemes, categorised by how many types of operations they support and how many times they can be performed in the encrypted domain. These types are: Partially Homomorphic Encryption (PHE), Somewhat Homomorphic Encryption (SWHE), Leveled Fully Homomorphic Encryption (LFHE) and FHE. PHE allows a single selected function to be performed on the ciphertext unlimited times [AAUC18]. In a SWHE scheme, multiple operations are allowed on the ciphertext, but only a limited amount of times [AAUC18]. In a LFHE scheme, different kinds of operations are supported, but a pre-determined depth is necessary [BGV11], meaning the number of operations to be executed. FHE schemes allow for the evaluation of arbitrary circuits with an unbounded depth and with multiple different functions [AAUC18]. This means that an FHE scheme must allow every kind of operation on the ciphertext unlimited times.

The problem of creating an FHE scheme was first proposed in 1978 by Rivest et al. [RAD78] and remained an open problem for more than 30 years. In 2009, Craig Gentry proposed the first ever FHE scheme [Gen09], which allows for the addition and multiplication of ciphertexts by evaluating circuits over the encrypted data. Gentry's solution uses lattice-based cryptography, which has become an important candidate for secure post-quantum encryption due to the complexity of its problem [MR09]. Gentry's scheme is derived from a SWHE scheme by demonstrating how to make it bootstrappable, which means that it can evaluate its own decryption circuit, as well as one additional operation [Gen09]. Since this breakthrough, multiple other FHE schemes has been proposed, including Brakerski-Gentry-Vaikuntanathan (BGV) [BGV11], Brakerski-Fan-Vercauteren (BFV) [FV12] [Bra12] and Cheon-Kim-Kim-Song (CKKS) [CKKS17]. BGV was introduced in 2011 as a radical new way of implementing FHE. The encryption is achieved without the bootstrapping approach by Gentry and improves performance in arbitrary polynomial-size circuits while basing the encryption on weaker assumptions. BGV's security is based on the hard problem LWE. BFV, introduced in 2012, is another FHE scheme using the LWE-based FHE technique presented by Brakerski [Bra12]. Instead of using LWE for security, BFV uses Ring Learning With Errors (RLWE), a different version of the LWE problem based on polynomial rings. BFV also includes optimisation of computation time and simplification of bootstrapping. CKKS was introduced in 2017 and is one of the newest schemes and aims at FHE for approximate arithmetics, contrary to the previous two, which are meant for exact arithmetics on integers.

Chapter 3 Related Work

This chapter introduces the most important work and research related to this project. The focus is directed explicitly toward publications that explore the combination of biometrics and cryptography. This is a relatively new area of research, which both proposes challenges and opportunities.

3.1 Fingercode Templates and Homomorphic Encryption

Barni et al. describe a fingerprint recognition system which preserves the user's privacy by computing the comparison task in the encrypted domain using HE [BBC+10]. The system relies on Fingercode templates instead of minutiae-based fingerprint templates. The system's architecture consists of two actors, the client and the server, where the main idea is not to let the server know the client's fingerprint in plaintext.

The first step in their system is to extract features from the client's fingerprint and store them as a Fingercode template. This is done by estimating a reference point of the fingerprint and applying Gabor filters [BBC+10] [Gab46], a well-known tool in texture analysis of images, which can capture both local and global features of the fingerprint. After the feature extraction, a reduction of the number of features in the Fingercode template is performed. In addition, a quantisation of the template is carried out. These operations are performed to reduce the complexity as preparation for comparison in the encrypted domain. This was done together with a study of the results to minimise the negative effect encryption had on the quality of the results. The next step is encryption of the template, which strongly relies on additively HE. For their specific solution, they use the Pailier [Pai99] encryption scheme together with the ElGamal encryption scheme [ElG85] ported on elliptic curves, and the encrypted result is stored as a vector. At this point, the encrypted template is sent to the server side of the system to be compared with its database of fingerprint templates. Specifically, the square of the Euclidean distance is calculated between the client's vector and the vectors in the database, exploiting the HE properties. The values from the calculations are communicated to the client several times, and a threshold is used to determine whether or not there is a match in the comparison. This result is not available to the server in cleartext. Figure 3.1 from the paper shows the architectural design of the system.

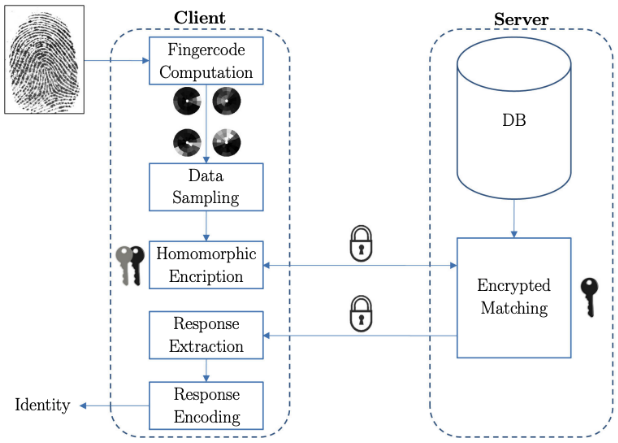


Figure 3.1: Architecture of the system proposed in [BBC+10].

3.2 Hamming Distance on Encrypted Fingerprint Templates Represented as Binary Vectors

Similarly to what was described by [BBC+10], Yang et al. proposed a fingerprint authentication system in [YWY+20] using HE with the same idea of a client side and a server side. The feature extraction technique used in this publication is minutiae-based, contrary to the Fingercodes used in [BBC+10]. The fingerprint template is transformed into binary vectors on the client side, then encrypted using the Paillier encryption scheme before being sent to the server. The binary vectors are then compared in the encrypted domain by the Authentication Server (AS) using hamming distance, where the homomorphic traits are exploited. Finally, the results are sent back to the client [YWY+20]. In the future work section of this paper, they explicitly state that research should be done on more efficient HE algorithms for this purpose in order to decrease the computational time.

3.3 Iris-based Comparison with Homomorphic Encryption

Studies have also been done on iris-biometric verification and identification, which share many challenges with fingerprint verification and identification. Kolberg et al. described in [KBG+19] a strategy for efficient iris template comparison in the encrypted domain, which utilises bit codes and hamming distance in a similar manner as proposed in [YWY+20]. Gomez-Barrero et al. proposed a system in [GMG+17] for multi-biometric template protection using homomorphic encryption, where both iris and fingerprint templates are considered. The system utilises the Paillier cryptosystem to achieve the homomorphic traits and compares the templates based on Euclidean distance and cosine distance in the encrypted domain.

3.4 Garbled Circuits and the Bozorth Algorithm

Zhang and Koushanfar published a paper in 2016 proposing a privacy-preserving fingerprint comparison system, which uses minutiae-based fingerprint templates [ZK16]. The fundamental comparison mechanism used is a customisation of the Bozorth algorithm developed by National Institute of Standards and Technology (NIST). The system's privacy is achieved by using Garbled Circuits, a cryptographic protocol enabling two parties to evaluate a function without trusting each other or a third party. Garbled Circuits were first introduced by Yao in 1986 [Yao86].

3.5 Fully Homomorphic Encryption Fingercode

De Fuentes et al. proposes an automated privacy-preserving fingerprint verification system using FHE [DKOK20]. The paper extends the Fast FHE over the Torus library to include the calculation of the Euclidean distance. The distance is calculated between two encrypted vectors containing double-precision floating-point numbers [CGGI20]. It also performs filterbank-based fingerprint verification on FingerCodes [JPHP00]. Once a data subject has been enrolled, the fingerprint templates are stored and processed in encrypted form. The fingerprint comparison process takes 166 seconds ($\pm 0,564$ seconds), and bootstrapping is the major contributor to slowing the process down.

3.6 Private Fingerprint Matching

Cryptographic primitives can be used to build a fully private fingerprint comparison protocol, as demonstrated by Shahandasht et al. [SSO12]. In this proposed protocol, Alice and Bob each have a private fingerprint that they can compare with each other to verify whether they have the same fingerprint or not. HE provides the necessary protection so that the fingerprints are not revealed to the other party. However, an HE scheme in itself does not support enough computations for the comparison to be calculated. Therefore, a primitive called aided computation is proposed to achieve the same computations capability as for an FHE scheme. Aided computation allows

Alice to help Bob with computing in the encrypted domain but does not give Alice information about what she helps to compute.

Chapter 4
Proposed Solution

This chapter provides an in-depth description of the proposed solution for the thesis. It includes the complete architecture of the proposed system and flowcharts describing the important parts. The choice of FHE library and fingerprint comparison algorithm are also covered.

4.1 Fully Homomorphic Encryption Library

PALISADE is the FHE library used in this proposed solution, enabling all the homomorphic functionalities [Pde17]. It is an open-source lattice cryptography library with contributors from several institutions and companies. PALISADE supports multiple different FHE schemes, including BFV, CKKS and BGV, as well as some others. PALISADE provided the capabilities needed in order to encrypt, decrypt and operate on the fingerprint templates. The broad range of computations it supports, such as additions, multiplications, rotations and polynomial arithmetics [Pde17], were a key factor in choosing PALISADE for this implementation. In addition to this, the library is quite user friendly for people that are not FHE experts, with its fairly simple usage and detailed documentation. The library is mainly compatible with C++, and it also provides a starting point for writing the implementation with a pre-configured compiler and code setup. Therefore, the implementation is also written in C++ as it made using PALISADE straightforward. Every operation that is performed on the encrypted data in this implementation, are using methods from PALISADE.

4.1.1 Other Libraries

Before deciding to use PALISADE for our project, four other available libraries were also researched and considered.

Microsoft SEAL

Microsoft SEAL is an open-source project created by Microsoft. It is written in C++ and C# and can be run in several environments. SEAL currently only supports the CKKS and BFV schemes [SEL20], which is one drawback to using SEAL. Some testing using SEAL was done prior to starting this thesis' work, and the documentation and usability of SEAL were not on par with the experience of using PALISADE.

HElib, HEAAN and Lattigo

HElib is an open source HE library developed by IBM, which supports the BGV and CKKS schemes [Heb22]. HEAAN is a library developed by researchers from Seoul National University, which is exclusively created for approximate numbers using CKKS [Han22]. Lattigo, is an HE library developed for the Go programming language [Lo22], and supports the CKKS and BFV encryption schemes. These libraries could potentially fit the application in this thesis. However, having tested PALISADE and SEAL prior to researching these libraries, a decision was made not to spend time familiarising with these libraries in the same manner as with PALISADE and SEAL since these two were fitting libraries to use.

4.2 Fingerprint Comparison Algorithms

Several algorithms have been developed for fingerprint comparison, many of which are developed for standard usage in the cleartext domain. These algorithms do not address the difficulties of performing computations with a limited set of operations available, which is the case if the algorithms were to be evaluated on homomorphically encrypted data. Therefore, a careful investigation of the known fingerprint comparison algorithms was done to consider which would fit this thesis's use case. A central part of the evaluation was the complexity of their operations due to the computational limitations of HE. The algorithms described below were found through studies as the most promising ones for performing in the encrypted domain with HE.

4.2.1 The SourceAFIS Algorithm

SourceAFIS is a package available in both .NET and Java that provides a complete fingerprint verification implementation [Važ21]. SourceAFIS supports feature extraction from fingerprint images and will firstly process and enhance the fingerprint images to increase the likelihood of precise minutiae identification. After processing an image, the algorithm produces a fingerprint template containing the identified set of minutia. With the fingerprint template as input, an edge table of one array for each minutia respectively is populated. Each minutia's array contains several arrays, where each array contains information about the edges connecting the referenced minutia with its closest neighbour minutiae, such as edge length, neighbour's ID, and each of the angles between the minutiae and the edge. These edges are invariant to both translations and rotations, making them suitable for comparison. Using these edges, the algorithm will attempt to find a similar edge in both the probe fingerprint and reference fingerprint. This will be the starting point for building a pairing tree recursively between the probe and the reference. It will attempt several pairing trees and calculate a score based on a long list of parameters. The resulting best match is whichever pairing achieves the highest score.

4.2.2 The Minutia Cylinder-Code Algorithm

Minutiae Cylinder Code (MCC) is a fingerprint comparison system which was first published in 2010 by Cappelli et al. [CFM10]. It was introduced as a new way of approaching the fingerprint comparison challenge as it represents the fingerprint features as Three-Dimensional (3D) cylinders, providing the data with fixed length and invariant attributes, which further enables a simple and effective comparison of fingerprints. The algorithm uses the classical minutiae-based template and focuses specifically on the position and angle of each minutia.

Each minutia of a fingerprint is represented as a 3D cylinder, a datastructure created with its minutia as the centre. Each cylinder contains a radius and height. These values are determined from fixed parameters, making each cylinder the same size. Each cylinder is built by creating several cells, where each cell describes a building block of the cylinder using the coordinates: i, j and k. The cells are all identical in terms of length, width and height. The complete cylinder is then described by all cells at each layer k. When n cylinders are created based on the n minutiae of a fingerprint, each cell in each cylinder will calculate a contribution score, which describes the probability of finding a minutia near its centre, excluding the belonging cylinder minutia. Each cylinder can then be described by a vector containing all the contribution scores of its cells. These vectors are the essential component in the comparison part.

In order to compare two fingerprints using MCC, the first step is to compare local similarities between two minutiae, represented as cylinders, where one is from the probe and one from the reference. As an optimisation mechanism, only cylinders that are determined to be `matchable` will be compared. The comparison will result in a local similarity score in the area [0, 1]. When several local similarity scores are obtained, a global score can be calculated from these, which will be the score that needs to be above a certain threshold in order to conclude that two fingerprints represent the same identity. The original MCC paper does not provide one specific algorithm for determining the global score but rather introduces several possible algorithms that could be used, highly inspired by the existing literature.

4.2.3 The Bozorth Algoritm

The Bozorth algorithm is a minutia-based fingerprint comparison algorithm developed by NIST and has the properties of being both rotation and translation invariant. This reduces the amount of pre-processing alignment work on the fingerprint templates, enhancing practicality.

The Bozorth Algorithm can be divided into three steps. The first step is to construct intra-fingerprint minutia-pair tables, consisting of the relative comparison between the minutiae in a fingerprint within a specific distance. Six values describe each minutia pair: the distance between the two minutiae, the orientation of the first minutia, the orientation of the second minutia, their absolute angle, the index of the first minutia, and the index of the second minutia. The translation and rotation invariant property is obtained because the distance between the points, and their relative orientation with respect to the line connecting them, are almost constant regardless of how much the fingerprint is rotated or translated. The second step is to construct inter-fingerprint minutia-pair compatibility tables, which use the previously constructed comparison tables. The two minutia pairs are compatible if the corresponding distance and relative minutia angles of two entries in the two separate comparison tables are within a specific threshold. The information about the two minutia pairs is incorporated and entered into the compatibility table. When this step is finished, a table consisting of compatible associations between minutia-pairs from the different fingerprints is created, and the last step can be done. This step consists of traversing the compatibility table to find the longest possible path of associated minutia pairs. The longer the path, the higher the similarity [ZK16].

4.2.4 Evaluation

The choice of comparison algorithm for the proposed solution was considerably influenced by its compatibility with the FHE methods. There is not one clearly superior solution among the candidates mentioned above, as they all have various trade-offs.

The SourceAFIS algorithm was somewhat complex in its public open-source implementation, using several functions in its comparison procedure that PALISADE does not support. Examples of such functions are sine, cosine, minimum, and maximum. The algorithm description lacks details on its official website, with no citation of a corresponding research paper, which limits the information available. This makes it hard to implement the algorithm from scratch, and relying on the current implementation would be undesirable. Although the algorithm could possibly be implemented with FHE, the lack of information and the dependency on complex operations made us eliminate this algorithm for our implementation.

The Bozorth algorithm has a quite simple description, which certainly is an advantage considering the complex task of implementing the comparison of fingerprints in the encrypted domain. However, there are still operations within the Bozorth algorithm that are not compatible with FHE. One of them is determining if an inter-fingerprint minutiae-pair distance and relative angles are within a threshold, which is a conditional statement. Another one is traversing the table of compatible minutia pairs. These tasks, combined with the lack of public, descriptive information about the algorithm, as the original paper from NIST describing the algorithm was not to be found, made the algorithm less desirable for the implementation.

The Minutiae Cylinder-Code (MCC) algorithm is the most complex of the three in terms of the number of operations, as it represents each minutia in each fingerprint as a complex 3D datastructure. This is costly for performance, but the complexity of the critical operations is pretty simple. MCC is well documented in previous research papers, making it easier to understand on a deep level, which is a critical factor for achieving the goals of this thesis. Although the 3D datastructures that the algorithm proposes are challenging, they can be created and processed in the enrolment stage before encryption of the relevant information. Furthermore, the comparison of the local values from the different minutiae could be made with simple arithmetic supported by PALISADE and FHE, making MCC the most interesting for our solution.

4.3 General Overview

The proposed solution is, in short, a modification of the MCC algorithm [CFM10], where parts of the algorithm are encrypted using FHE to achieve a more privacy-preserving solution. The solution is proposed as a biometric verification use case, where there is a one-to-one comparison between two templates to determine whether they stem from the same data subject. This differs from biometric identification, which is a one to many comparison to search for a single individual in a biometric enrolment database and return a pointer to the individual. To ensure that the description of the proposed solution is clear, its architecture is first presented with a high-level overview before being described in a more detailed manner where the technical details are included.

The system can be described as a client-server architecture, as visualised in Figure 4.1. The client is the entity that provides a fingerprint probe sample and a biometric claim, or in other words, provides a fingerprint to be used for comparison. The biometric claim gives the reference sample to use for the 1:1 comparison, which can, for example, be an ID or the name of the client. The Computation Server (CS) is the entity that performs the necessary operations for calculating a comparison score. The AS assists the CS by decrypting some information needed in the verification stage, and lastly, it determines the validity of the biometric claim based on the comparison score. Communication in the system happens between these three entities. The AS has three keys, private, public, and evaluation, and makes both the public and evaluation keys publicly available. Having two different servers is to ensure that the server receiving encrypted templates does not hold the private key. Furthermore, Separation of Duty (SOD) is a common design in authentication schemes. That way, the risk of a compromised CS is mitigated.

Firstly, the client extracts features from a fingerprint image, creates a valid template and encrypts it using FHE with the AS's public key. Then, the client makes a biometric claim by sending the encrypted template to the CS. The CS compares the encrypted template provided by the client with the stored encrypted reference template. The publicly available evaluation key is used for the operations in the encrypted domain. There is some communication between the CS and AS during the verification stage because the AS holds the private key. Afterwards, the AS receives the comparison score, which it compares to a threshold to determine whether to accept or reject the biometric claim. The result of the biometric verification decision is sent to the client.

In the implementation[1], these operations are all executed by the same program on one computer for simplicity's sake, but it resembles the flow of a client-server architecture.

Ideally, the global similarity score between two fingerprint templates should be calculated without decrypting any information from the fingerprint templates. This is, however, not achieved in this proposed solution. This proposal has three types of information exposed in cleartext during the comparison. These are: 1) the angular difference between two cylinders, 2) the number of cells in the two cylinders that are `matchable`, and 3) the Euclidean norm of each contribution vector from both cylinders, in addition to the Euclidean distance of the two contribution vectors. Figure 4.1 illustrates the actual flow in the proposed solution, which differs from the ideal solution presented in Figure 1.1.

4.4 Technical Description

The technical description presents each stage of the proposed solution chronologically, giving details about operations, parameters and more.

[1]https://github.com/TTM4502-FHE/Proof-Of-Concept

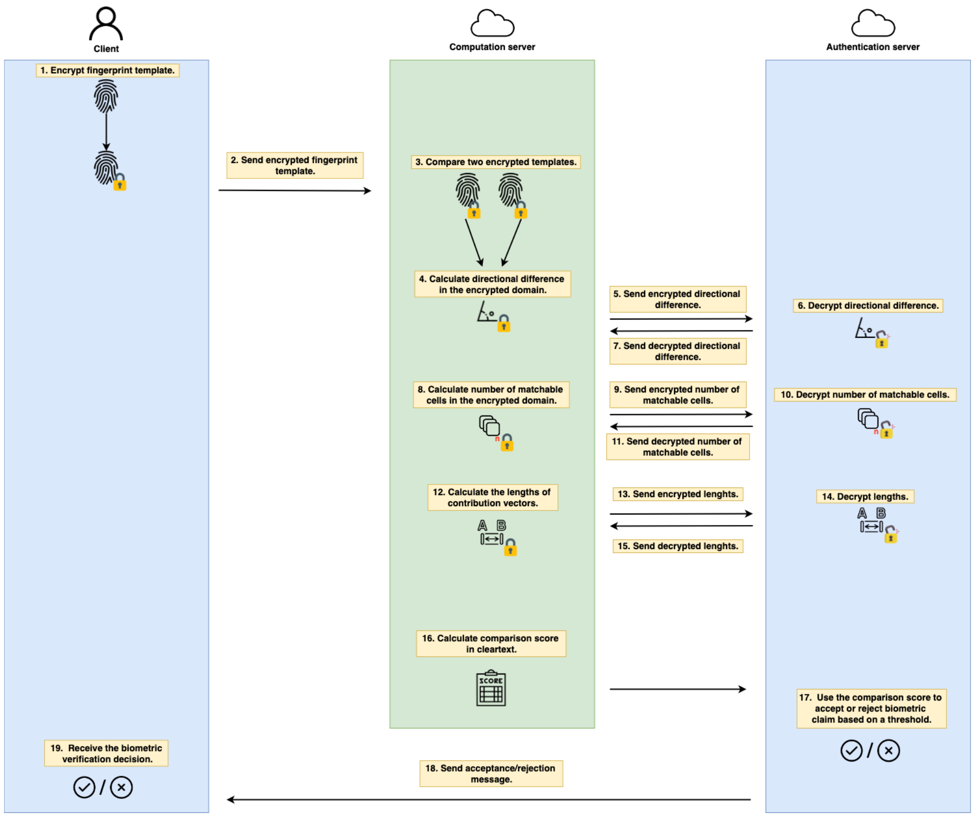

Client
Computation server
Authentication server
1. Encrypt fingerprint template.
2. Send encrypted fingerprint template.
3. Compare two encrypted templates.
4. Calculate directional difference in the encrypted domain.
5. Send encrypted directional difference.
6. Decrypt directional difference.
7. Send decrypted directional difference.
8. Calculate number of matchable cells in the encrypted domain.
9. Send encrypted number of matchable cells.
10. Decrypt number of matchable cells.
11. Send decrypted number of matchable cells.
12. Calculate the lengths of contribution vectors.
13. Send encrypted lenghts.
14. Decrypt lengths.
15. Send decrypted lenghts.
16. Calculate comparison score in cleartext.
17. Use the comparison score to accept or reject biometric claim based on a threshold.
18. Send acceptance/rejection message.
19. Receive the biometric verification decision.
SCORE
A B

Figure 4.1: Simplified flowchart of the proposed solution.

4.4.1 Feature Extraction

Feature extraction is technically not a part of the proposed solution. As long as the important information is included, there is no definite answer as to how minutia features from the fingerprint should be extracted. The required template for this implementation is an array with minutiae information on the following format $\{\{x_1, y_1, a_1\}, \{x_2, y_2, a_2\}, \ldots, \{x_n, y_n, a_n\}\}$, where x is the x-coordinate, y is the y-coordinate, and a is the angle of the minutia. In this thesis, a part of the SourceAFIS algorithm was used for the feature extraction [Važ21].

4.4.2 Establishing the PALISADE CryptoContext

CryptoContext

A `CryptoContext` in PALISADE is the class that provides all PALISADE encryption functionality [Use17]. The `CryptoContext` object must be initialised in the beginning of every application that uses the HE functionality of PALISADE. Depending on the HE scheme of choice, there are certain parameters that must be defined in the establishment of the `CryptoContext`. For this thesis, the `CryptoContext` is established with the CKKS scheme, and usually takes in the parameters: `multDepth`, `scaleFactorBits`, `batchSize`, `securityLevel` and `ringDim`. Additionally, one more optional arguments were added for this use case: `rsTech`. The motivation for using CKKS was that it supports direct encryption of vectors containing float values, which was how several of the data structures in the cleartext implementation was written.

Parameters

`multDepth` is the multiplicative depth, and describes the maximum amount of multiplications allowed in sequence [ACC+21] [Pgh21]. Minimising the multiplicative depth has a positive effect on runtime performance. This implementation has `multDepth` set to 2.

`scaleFactorBits` is the parameter used in CKKS to scale the real values to integers before computation [Pgh21]. A scaling factor of 1000 means that the original real value is rounded to 1000 times itself. `scaleFactorBits` then indicates the number of bits in the scaling factor and not the scaling factor itself. This implementation has a `scaleFactorBits` value of 30, which provides sufficient precision for this purpose.

`batchSize` is the number of plaintext slots that are used in the ciphertext [Pgh21]. The `batchSize` in the CKKS implementation is set to 1536. This corresponds to the number of cells in each cylinder and therefore the number of slots in the plaintext of the contribution vectors.

`securityLevel` determines the level of security, which typically is defined by a 128-bit, 192-bit, or 256-bit key size [ACC+21] [Pgh21]. For the purpose of demonstration, 128-bit security is used in order to balance security and efficiency. This could easily be modified in accordance to security requirements.

`ringDim` is the ring dimension from the RLWE problem and is typically a power of two. The ring dimension is assigned automatically in the `CryptoContext` of this implementation by giving the parameter the value 0. With the other parameters set as described in this section, the `ringDim` parameter is given the value 8192. The ring dimension also sets the upper limit for the batch size.

`rsTech` describes the rescaling technique that PALISADE uses to scale the integers whenever it is necessary. For instance, two ciphertexts with a scaling factor of D will have a scaling factor of D^2 when multiplied. Rescaling will then adjust the scaling factor back to D before further operations. The rescaling technique used in this implementation is called `APPROXAUTO` and is a built-in function in PALISADE that does the rescaling automatically [Pgh21].

4.4.3 Probe and Reference Template Creation

The implementation takes two fingerprint templates as input. Therefore, the process that is explained in this section is performed twice. Firstly, the fingerprint template is parsed and stored as minutia objects in a vector. This vector is then passed to the fingerprint template constructor, which generates the complete 3D data structure of the fingerprint based on the MCC algorithm. After the 3D data structure is created in the constructor, the process of encrypting the template is started. Encrypting the template essentially means extracting the crucial information from the 3D data structure and storing the information as ciphertext. Each fingerprint template will contain several encrypted cylinders. Recall that one cylinder from the MCC algorithm represents one minutia point from the fingerprint. This means that n cylinders corresponds to n minutiae. Each of the encrypted cylinder objects will contain three ciphertexts. The first one is the cylinder angle encrypted, which inherits the minutia's angle. The second encrypted value is the c_m vector, which holds the contribution values of every cell related to minutia m. This corresponds to equation (4) in the MCC paper [CFM10]. The third and final value stored as a ciphertext is the $c_{m_{validity}}$ vector, which represents the validity of each cell related to minutia m. These three ciphertexts are the information needed for the comparison part. Figure 4.2 illustrates the components of the encrypted fingerprint template, while Figure 4.3 shows the complete flowchart of encrypting the fingerprint template. The result of doing this process twice is a probe and reference template that are used for comparison.

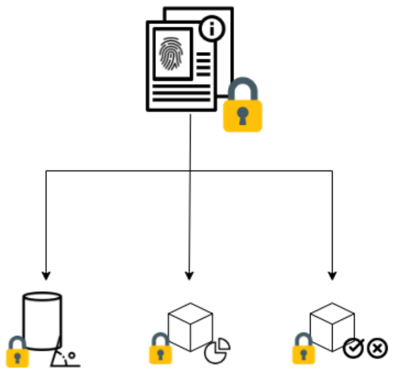


Figure 4.2: The encrypted template struct consists of encrypted cylinder angles, cell contributions, and cell validities.

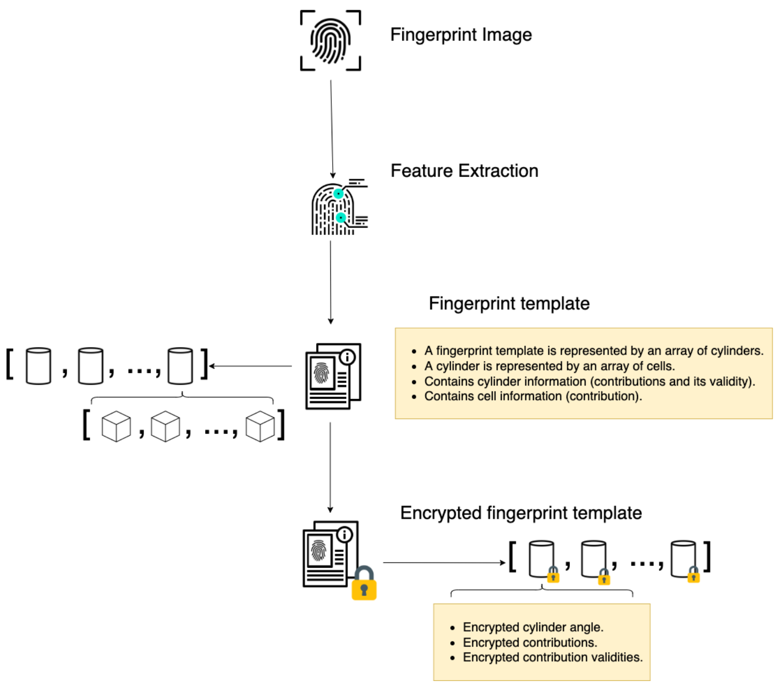

Fingerprint Image
Feature Extraction
Fingerprint template
A fingerprint template is represented by an array of cylinders.
A cylinder is represented by an array of cells.
Contains cylinder information (contributions and its validity).
Contains cell information (contribution).
Encrypted fingerprint template
Encrypted cylinder angle.
Encrypted contributions.
Encrypted contribution validities.

Figure 4.3: Flowchart representing the process of creating an encrypted fingerprint template.

4.4.4 Comparison of Probe and Reference Templates

Comparing the encrypted templates consists of first calculating every local similarity between cylinders and then, based on these, calculating the global similarity score between the templates.

Suppose the probe template contains n minutiae, and the reference template contains m minutiae. In that case, the number of local similarity calculations will be $n \cdot m$ as every cylinder from the probe will be compared with every cylinder from the reference.

Cylinder Similarity

The cylinder similarity between two cylinders a and b, denoted as $\gamma(a, b)$, is calculated as a function where the two cylinders are used as input parameters. The function for calculating each cylinder similarity implements Equation 4.1, which is equation (17) in the MCC paper [CFM10]. Due to the limitations of FHE, the process is divided into several steps in order to fulfil the operational and conditional requirements from the MCC algorithm.

$$\gamma(a,b) = \begin{cases} 1 - \frac{\|\mathbf{c_{a|b}} - \mathbf{c_{b|a}}\|}{\|\mathbf{c_{a|b}}\| + \|\mathbf{c_{b|a}}\|}, & \text{if } C_a \text{ and } C_b \text{ are } \texttt{matchable}. \\ 0, & \text{otherwise.} \end{cases} \tag{4.1}$$

The cylinder similarity can only be calculated if two cylinders are considered `matchable`. The requirements for cylinders being `matchable` are [CFM10]:

- The directional difference between the two minutiae is not greater than $\frac{\pi}{2}$.
- At least 60% corresponding elements in the two vectors c_a and c_b are `matchable`.
- $\|\mathbf{c_{a|b}}\| + \|\mathbf{c_{b|a}}\| \neq 0$.

The first condition requires a calculation of the directional difference between two cylinders and to check if it is below $\frac{\pi}{2}$. This is done by subtracting the two minutiae's encrypted angles in the encrypted domain. This difference is then decrypted in order to evaluate the conditional statements from equation (9) in the MCC paper, which defines the directional difference outcome. After the directional difference is available in cleartext, the algorithm checks if it is below $\frac{\pi}{2}$ and returns 0 if it is not. Evaluating encrypted conditional statements is very complex and therefore avoided in this proposal. The difference between the minutia angles becomes the first part of information related to the fingerprints that is exposed in cleartext at the server side.

The next step is calculating the $c_{a|b}$ and $c_{b|a}$ vectors. These vectors describe all contributions in the two cylinders where corresponding cells from both cylinders are valid. $c_{a|b}$ consists of all contributions from c_a where c_b has contribution from corresponding cells, and $c_{b|a}$ consists of all contributions from c_b where c_a has contribution from corresponding cells. To combine and organise encrypted ciphertexts from two templates, a new representation is created which holds three ciphertexts: 1) $c_{a|b}$, 2) $c_{b|a}$, and 3) common validity vector (cvv). cvv contains 1s in the corresponding indexes where both c_b and c_a has a value, and 0s in the corresponding indexes where neither, or only one of c_b and c_a has a value. This vector is calculated in the encrypted domain by multiplying the encrypted validity vectors from the two cylinders. As these vectors consists of only 0s and 1s, the multiplication will be equivalent to the binary AND operation. cvv is then multiplied with each of the two encrypted vectors c_a and c_b, which filters away the contribution of cells that should not be taken into account in the calculation of the cylinder score. The following is an example of the calculation of $c_{a|b}$, $c_{b|a}$, and cvv:

$$\begin{aligned} c_a &= [0,0,3,2,8,0,5] & \qquad c_b &= [2,1,7,0,0,4,4] \\ c_{a_{validity}} &= [0,0,1,1,1,0,1] & \qquad c_{b_{validity}} &= [1,1,1,0,0,1,1] \end{aligned}$$

$$cvv = c_{a_{validity}} \cdot c_{b_{validity}} = [0, 0, 1, 0, 0, 0, 1]$$

$$c_{a|b} = c_a \cdot cvv = [0, 0, 3, 0, 0, 0, 5] \qquad c_{b|a} = c_b \cdot cvv = [0, 0, 7, 0, 0, 0, 4].$$

The second condition is to check if over 60% of the corresponding elements in c_a and c_b are `matchable`. In order to check this, we first need to find the number of corresponding `matchable` elements in the two vectors. This is achieved by calculating the squared Euclidean norm of cvv, which is computed in the encrypted domain using rotation and addition [BOK+22]. Finally, this value is decrypted in order to evaluate the condition, which makes it the second piece of information accessed in cleartext by the server. If the amount of `matchable` elements in the two vectors is below 60% of the total amount of elements, the cylinders are not `matchable`.

Following the two conditions, the denominator and numerator in Equation 4.1 are calculated. The numerator, which consists of calculating $||c_{a|b} - c_{b|a}||$, is done by first subtracting the two vectors in the encrypted domain, then the squared Euclidean norm is calculated encrypted before decrypting the value and taking the square root of it. Decryption is necessary due to the complexity of computing a square root in the encrypted domain. This exposes the length of the difference between the two vectors in cleartext to the server, which is part of the third piece of information that is revealed in cleartext. The result of these operations is the Euclidean norm $||c_{a|b} - c_{b|a}||$. In a similar manner, the denominator of the fraction is calculated primarily in the encrypted domain, with a decryption in the end to find the square root. The denominator is $||c_{a|b}|| + ||c_{b|a}||$. The exposed information is then the lengths of each of the vectors, $||c_{a|b}||$ and $||c_{b|a}||$.

The third condition condition is checked just before calculating $\gamma(a, b)$, where the purpose is to verify that the sum of the two vector lengths is not zero. This is straightforward with the decrypted length values.

Finally, if all conditions are satisfied, the cylinder similarity is calculated in cleartext and returned as the final step of the cylinder similarity method. This whole method is repeated $n \cdot m$ times until all cylinders from the probe template are compared with all cylinders from the reference template.

Global Score

When all cylinder scores are obtained, the next and final step of the template comparison is to calculate the global score. This is accomplished using the local similarity sort algorithm. Local similarity sort consists of calculating the average of the top n_p cylinder scores, where n_p denotes the amount of cylinder scores to consider, determined by equation (24) in the MCC paper [CFM10]. First, the list of cylinder similarity scores is sorted, then n_p is calculated with its selected parameters from the paper [CFM10], and finally the average is calculated.

The global score between the two fingerprint templates can be compared with a threshold score that will determine whether to accept or reject the biometric claim. An illustration of the complete comparison process is shown in Figure 4.4.

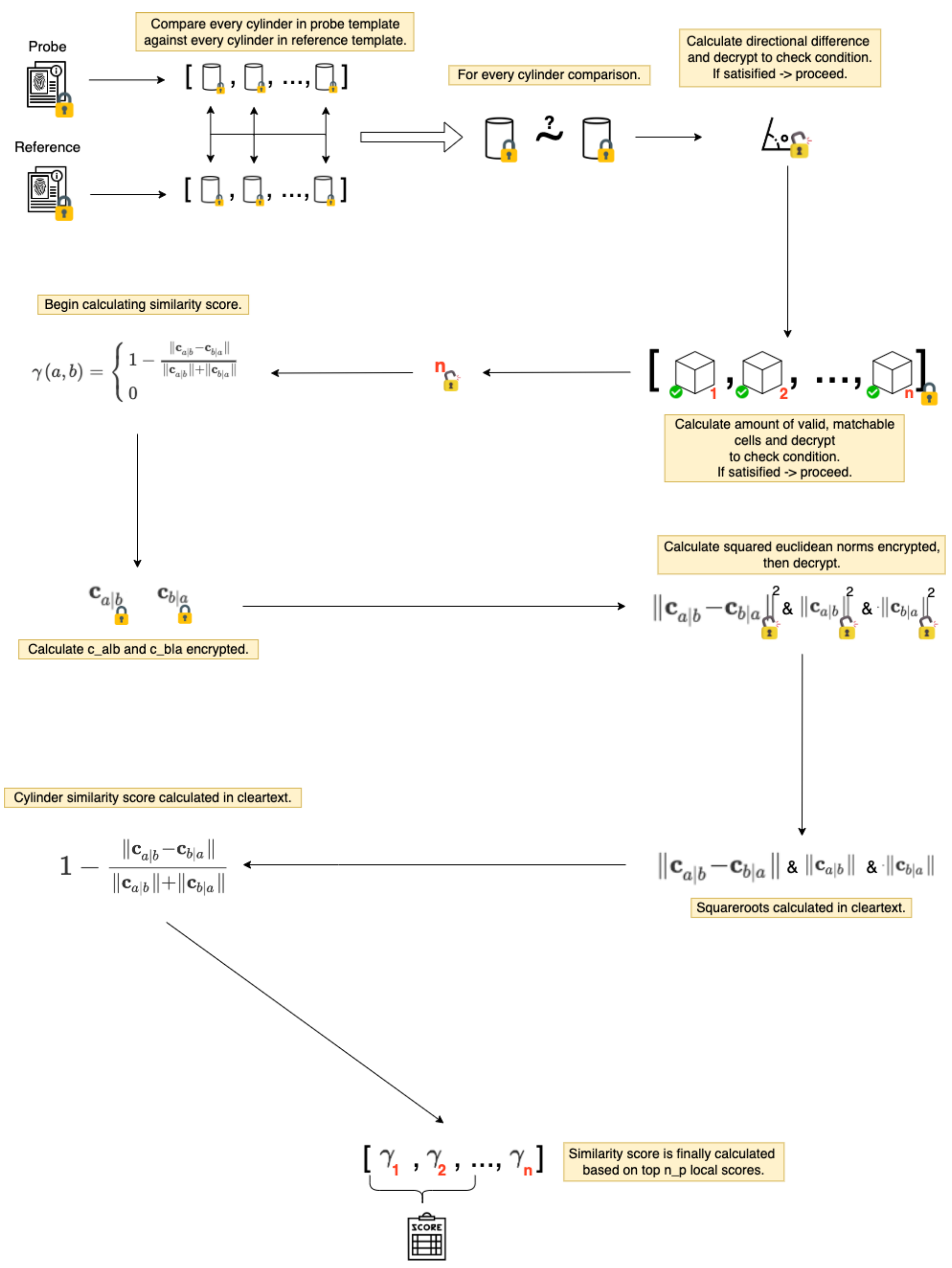

Probe
Reference
Compare every cylinder in probe template against every cylinder in reference template.
For every cylinder comparison.
Calculate directional difference and decrypt to check condition. If satisfied -> proceed.
Calculate amount of valid, matchable cells and decrypt to check condition. If satisified -> proceed.
Begin calculating similarity score.
$\gamma(a,b) = \begin{cases} 1 - \frac{\|\mathbf{c}_{a|b} - \mathbf{c}_{b|a}\|}{\|\mathbf{c}_{a|b}\| + \|\mathbf{c}_{b|a}\|} \\ 0 \end{cases}$
$\mathbf{c}_{a|b}$ $\mathbf{c}_{b|a}$
Calculate c_alb and c_bla encrypted.
Calculate squared euclidean norms encrypted, then decrypt.
$\|\mathbf{c}_{a|b} - \mathbf{c}_{b|a}\|^2$ & $\|\mathbf{c}_{a|b}\|^2$ & $\|\mathbf{c}_{b|a}\|^2$
$\|\mathbf{c}_{a|b} - \mathbf{c}_{b|a}\|$ & $\|\mathbf{c}_{a|b}\|$ & $\|\mathbf{c}_{b|a}\|$
Squareroots calculated in cleartext.
Cylinder similarity score calculated in cleartext.
$1 - \frac{\|\mathbf{c}_{a|b} - \mathbf{c}_{b|a}\|}{\|\mathbf{c}_{a|b}\| + \|\mathbf{c}_{b|a}\|}$
$[\gamma_1, \gamma_2, \ldots, \gamma_n]$
Similarity score is finally calculated based on top n_p local scores.
SCORE

Figure 4.4: Flowchart of the encrypted comparison process.

Chapter 5
Results and Discussion

This chapter includes the results generated from the implementation and discusses these results, as well as the implementation itself. Several experimental tests have been conducted using different fingerprints with different modifications in the code, which together makes the collection of results presented in this chapter. Details regarding each result are included.

The chapter consists of four sections: Technical Specifications, Fingerprint Verification, Performance, and Security and Privacy. The first section describes the environment used for testing the solution. The second section discusses the proposed solutions' ability to accept or reject fingerprints, including quantitative results for analysis. The third section discusses the implementation's runtime performance, which is somewhat a discussion on usability and contains numerical results. This section also includes experimentation conducted to improve the performance. The fourth and final section analyses how secure and privacy-preserving the solution is.

5.1 Testing Environment

The results in this section were produced by executing the implementation[1] on a server hosted by NTNU with the following specifications:

- **Operating System** — Ubuntu.
- **Version** — 1.13.0-1ubuntu1.1.
- **RAM size** — 128 GB.
- **CPU1** — 2 GHz size and capacity. 8 cores.
- **CPU2** — 2 GHz size and capacity. 8 cores.

[1]https://github.com/TTM4502-FHE/Proof-Of-Concept

Fingerprint templates have been generated from two sources of fingerprint images. The first source consists of 4 databases made available in conjunction with the Fingerprint Verification Competition 2002 [FVC02]. This source is referred to as the FVC database in this thesis. The FVC database is made up of 399 data subjects, each of which has eight fingerprint captures. The second source was NeuroTechnology's fingerprint sample database [SFD30], which is referred to as the Crossmatch database. This database provided 51 data subjects, with eight fingerprint captures for each data subject.

5.2 Fingerprint Verification

The implementation in this thesis is written based on [CFM10], and its fundamental task is to calculate a comparison score between two fingerprints. The implementation has been tested with templates generated from the images in the FVC and CrossMatch databases. These results will be evaluated to decide whether the implementation accomplishes its fundamental task. The results gathered in this section were generated using the cleartext implementation. The encrypted implementation has, from observations, very little approximation loss, and therefore we argue that the results from the cleartext implementation in terms of verification are approximately the same. The cleartext implementation is much more efficient than the encrypted implementation, making it possible to perform more comparisons to get larger samples of comparison scores.

5.2.1 Comparison Scores

A comparison score indicates the degree of similarity between two fingerprints, and it ranges from 0 to 1, where 0 is the lowest and 1 is the highest possible score. A high score indicates a high degree of similarity between two fingerprints. The figures in this section divide comparison scores into two categories: mated and non-mated scores. Mated scores stem from comparisons between an enrolled reference template against several probe templates from the same data subject. A mated score should generally have a high comparison score, but distortion and sample quality can reduce the comparison score since two captures of a fingerprint will not be identical. A non-mated score results from a comparison between two enrolled reference templates from different data subjects. Therefore, non-mated scores should be lower than mated scores.

Figure 5.1 presents the results from using the implemented comparison algorithm to compare templates generated from the FVC database. The y-axis shows the probability density of a comparison score, and the x-axis is the comparison score. The graph has been normalised to provide a better understanding of the distribution

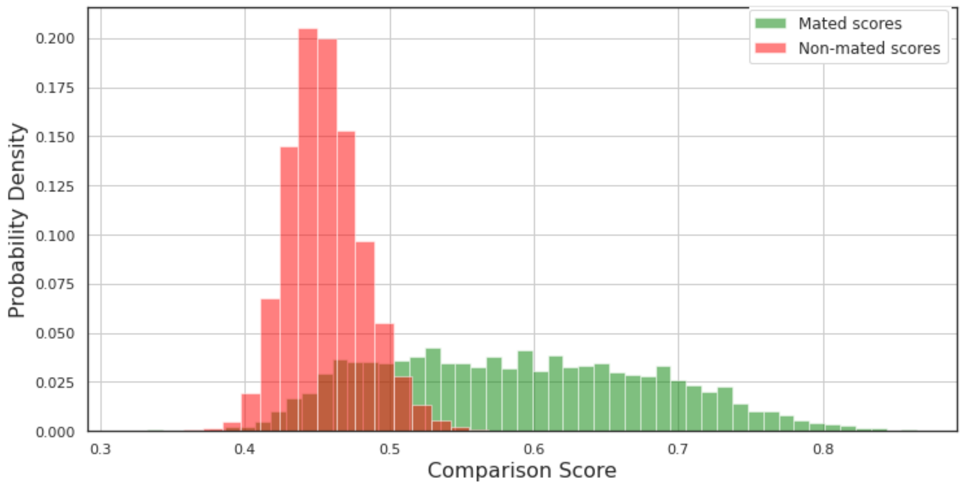


Figure 5.1: Normalised graph with comparison scores resulting from comparisons between templates from the FVC database.

of comparison scores since the number of non-mated comparisons is greater than mated comparisons.

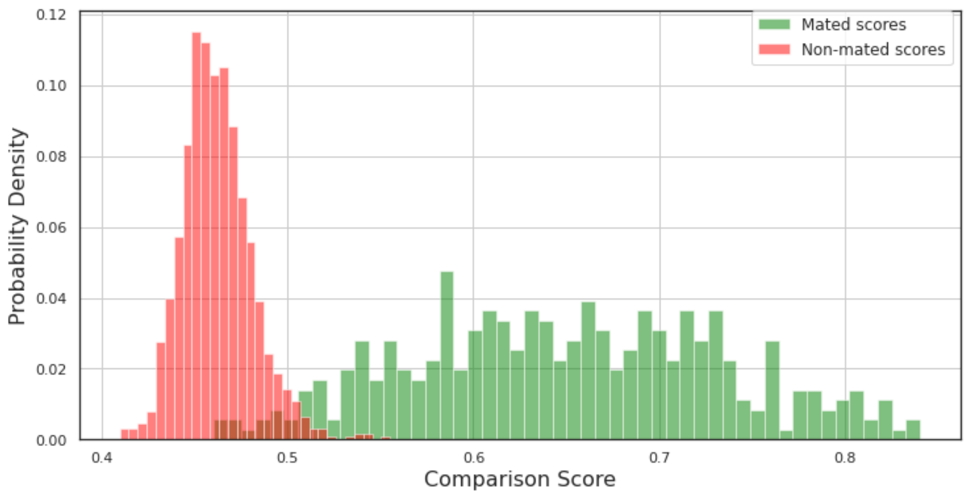


Figure 5.2: Normalised graph with comparison scores resulting from comparisons between templates from the Cross-Match database.

Figure 5.2 shows the results from comparing templates from the CrossMatch database. The clearer separation between mated and non-mated scores indicates that the implementation performs better with fingerprint images from the CrossMatch database.

5.2.2 FVC Verification Performance

In Figure 5.1, the majority of the non-mated scores are between 0,4 and 0,5, with a distinct peak at approximately 0,45. However, most of the mated scores range from 0,45 to 0,75, which is a much larger spread and is not recognised with an explicit peak.

Table 5.1 summarises statistics of the results. It shows a minimum score of 0,33 and 0,32 for respectively mated and non-mated comparisons of fingerprints from the FVC database. This means that the lowest score between two non-mated fingerprint templates and the lowest score between two mated fingerprint templates are approximately the same. This is not ideal, as the more distinction between the two categories, the better. Additionally, the maximum non-mated score is well into the range of what is expected from mated scores. Preferably the implementation should not output high scores for non-mated comparisons. From the mean data row, we see that the mated scores have a mean score of 0,59 while the non-mated scores have a mean of 0,46, indicating that in general there is a difference between the mated and non-mated scores. The standard deviation is 0,10 for the mated and 0,03 for the non-mated, which makes sense as there is a great difference in spread for the two types of scores.

Database	FVC		Crossmatch	
Score category	Mated	Non-Mated	Mated	Non-Mated
Observations	2793	79401	357	1275
Min. score	0,33	0,32	0,46	0,41
Max. score	0,87	0,71	0,84	0,56
Mean	0,59 ± 0,10	0,46 ± 0,03	0,65 ± 0,08	0,46 ± 0,02

Table 5.1: Comparison scores statistics.

The non-mated scores are much more concentrated around their mean value. Therefore, the implementation is arguably better at determining a non-mated score than a mated score. However, an essential element is the images and to which degree they are, in practice, possible to determine as the same fingerprints. The FVC database is created for the Fingerprint Verification Competition, which has intentionally made multiple fingerprint images challenging to read and compare in order to test how good algorithms tackle this challenge. The MCC algorithm is claimed to be translation and rotation invariant [CFM10], which should enhance the performance of misaligned fingerprint captures. However, by visually observing the images generating the poorer scores, the trend was that poorer scores came from misaligned images. It is hard to tell if this is a malfunction in this implementation or if the algorithm itself performs better with images that are more similarly aligned.

5.2.3 CrossMatch Verification Performance

Figure 5.2 illustrates the distribution of the non-mated and mated scores resulting from comparisons of fingerprints from the CrossMatch database. At first glance, a greater separation between the two categories can be observed. The non-mated scores are still concentrated in the same area as the FVC database, approximately between 0,4 and 0,5, and still hold a clearly defined peak. The most notable difference between the two databases is the mated scores, which in these results are shifted more towards the right side, representing higher scores, and have most scores between 0,5 and 0,8, compared to 0,45 and 0,75 in FVC. An important difference between the two sets of results is the overlap between the mated and non-mated scores. In FVC, we saw a rather large chunk of the mated scores in the non-mated scoring area, which generates a large portion of false rejections. There is significantly less overlap in the results from the CrossMatch database.

From Table 5.1, the number of mated comparisons made with the CrossMatch database are 357, while 2793 were done with the FVC database. For non-mated comparisons, 1275 were completed with CrossMatch, while 79401 were done with FVC. These differences are because of the differences in the number of fingerprint images in each database. The FVC database contains 399 subjects with eight images per subject, while the CrossMatch database only contains 51 subjects with eight images per subject. The minimum and maximum scores of the non-mated CrossMatch comparisons are 0,41 and 0,56. For mated, these are 0,46 and 0,84. Here we see that the maximum non-mated score from the CrossMatch results is significantly lower than the corresponding score from the FVC database, indicating a more stable set of scores. The mean score for non-mated comparisons is the same as for FVC, 0,46, while the mean score for mated comparisons is 0,65. This results in a difference of 0,19 between non-mated and mated means, compared to the difference of 0,13 in the FVC results. The standard deviation is also lower for the CrossMatch results, which indicate more concentrated scores.

Claims can be made that the fingerprint images from the CrossMatch database, in general, perform better than the fingerprint images from the FVC database. These claims are supported by the fact that there is a more significant distinction between mated and non-mated scores, making room for error smaller. The number of false rejections and false acceptances is smaller, meaning that it is possible to verify or reject a subject with a higher certainty. However, as the number of comparisons made with the CrossMatch database is significantly lower than the FVC results, the confidence that these results are a correct representation of performance is lower.

5.2.4 FVC versus CrossMatch

Why does the CrossMatch database perform better? In all likelihood, because of the images' quality. The images from the CrossMatch database have a higher resolution and less distinction between the different images from the same finger. This might result from better preprocessing, but unlike the FVC images, the CrossMatch database does not intentionally try to make it difficult to compare the images. Another factor is the technology used to capture fingerprint images. The CrossMatch database images are captured exclusively with the optical sensor "Cross Match Verifier 300" [Neu22]. The FVC database images, however, are captured with four different technologies; optical sensor "TouchView II" by Identix, optical sensor "FX2000" by Biometrika, capacitive sensor "100 SC" by Precise Biometrics, and synthetic fingerprint generation [FVC02]. This again is an effort in FVC to challenge the algorithm in comparing images of different qualities with different characteristics. The difference in results between the two databases is arguably what was expected.

5.2.5 Detection Error Tradeoff Curves

Detection Error Tradeoff (DET) curves provide a visual representation of the relationship between FMR and FNMR. As the name of the curve may indicate, there is a tradeoff with regards to usability when deciding on a FMR. The lower the FMR, the higher the FNMR, thus leading to more rejected claims that should have been accepted. Having a low FMR means having a more secure biometric recognition system. Therefore the use case for the biometric recognition system will impact what FMR is acceptable. In this thesis, the scope is not set to any particular use case, and thus the FMR can be set to 1% to illustrate an example. Figure 5.3 shows the DET curve for the FVC database, and with an FMR of 1%, the FNMR is approximately 32%. That means a rejection for about $\frac{1}{3}$ of actual matches, which is not a usable biometric recognition system. The DET curve for the CrossMatch database in Figure 5.4 indicates an FNMR of 5% for the same FMR, which is a clear improvement. With that database, the same level of security provides a more usable biometric recognition system.

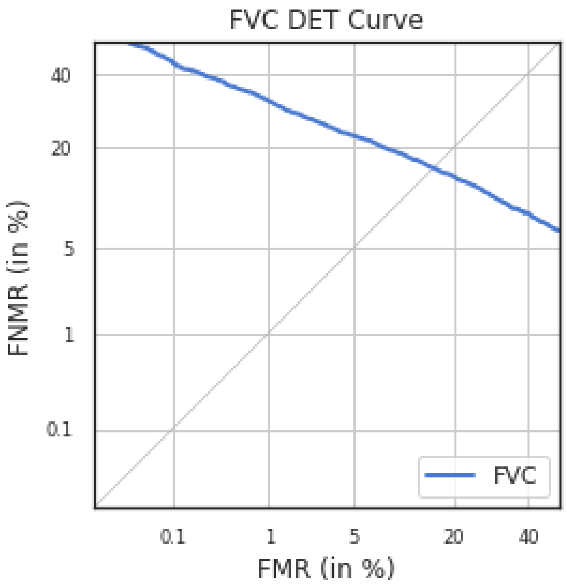


Figure 5.3: FVC Detection Error Tradeoff Curve.

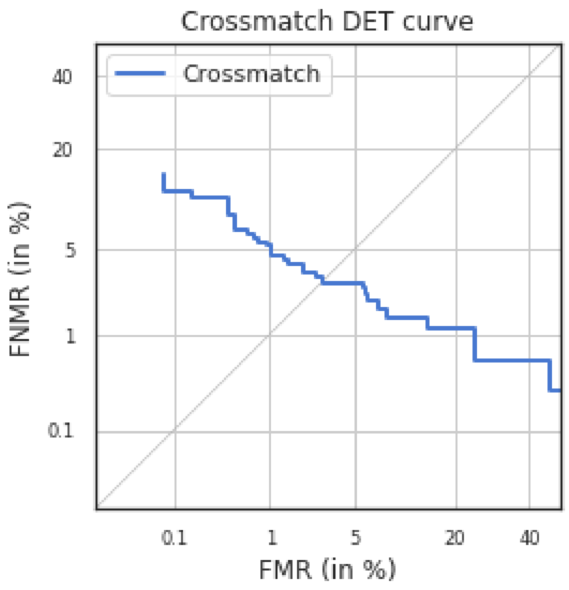


Figure 5.4: CrossMatch Detection Error Tradeoff Curve.

5.2.6 Threshold

Whether to reject or accept a biometric claim is decided based on a set threshold. If the comparison results in a score above the set threshold, then the biometric claim is accepted. The thresholds have been set separately for the two databases to achieve a FMR of 1%. This yielded a calculated threshold of 0,514795 for the CrossMatch database and 0,5288 for the FVC database. Figure 5.6 and 5.5 visualise how many of the biometric rejections and acceptances were false rejections, false acceptances, true rejections and true acceptances based on the calculated thresholds. These figures are also presented in number format in Table 5.2 and 5.3, along with the FMR, True Match Rate (TMR), FNMR, and True Non-Match Rate (TNMR).

Predicted/**Actual**	**Mated**	**Non-Mated**
Mated	1903 (68,13 %)	796 (1,00 %)
Non-Mated	890 (31,87 %)	78605 (98,99 %)

Table 5.2: FVC Confusion Matrix.

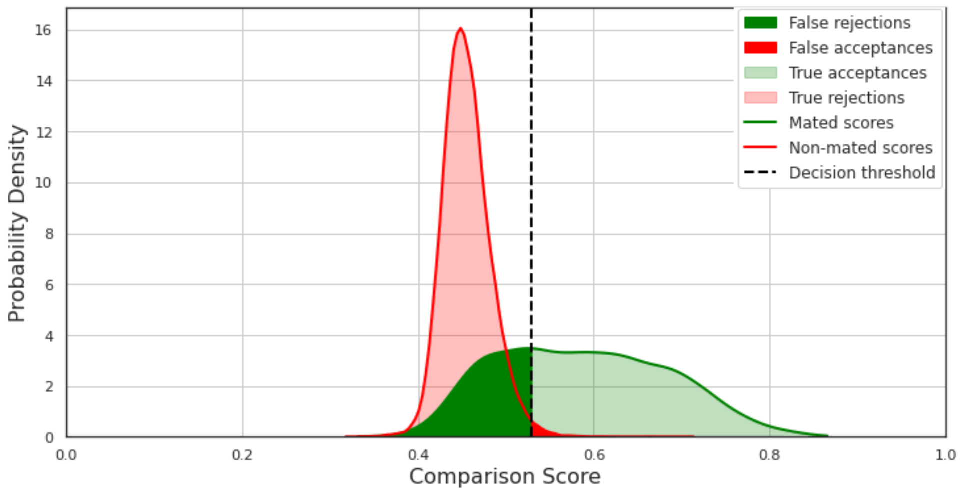


Figure 5.5: FVC comparison scores with the threshold set to 0,5288.

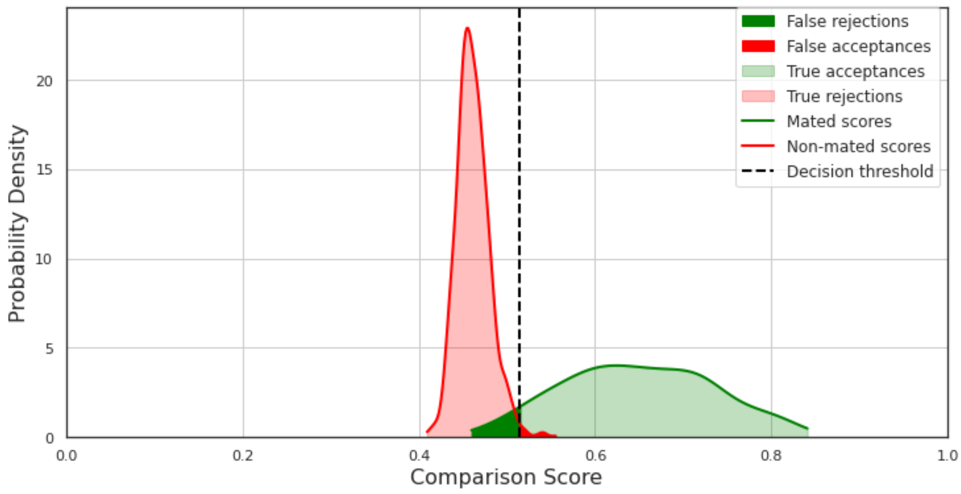


Figure 5.6: CrossMatch comparison scores with the threshold set to 0,514795.

Predicted/**Actual**	**Mated**	**Non-Mated**
Mated	337 (94,40 %)	12 (0,94 %)
Non-Mated	20 (5,60 %)	1263 (99,06 %)

Table 5.3: CrossMatch Confusion Matrix.

5.3 Runtime Performance

In order to evaluate the usability of this proposed solution, it is necessary to do an analysis of its performance in terms of time. The time it takes to enrol a fingerprint and verify it heavily influences the perceived usability of a fingerprint verification system. This section discusses both the cost of operations for PALISADE's methods, as well as the cost of the two stages of the implementation. Also, a subset of the functions in the implementation will be presented to quantify their contribution to the overall performance in the verification stage. The execution times referred to in this section were gathered during comparisons of fingerprints from the FVC database.

5.3.1 Cost Of PALISADE Methods

PALISADE provided the methods needed to shift the implementation from the cleartext domain to the encrypted domain. Table 5.4 show the relative cost of six key PALISADE methods that were utilised. The cheapest operation is the `Add` operation, and this operation sets the point of reference for the cost of the other operations. The average cost for the `Add` operation in the implementation is 106,72 μs, and the cost for the `Multiply` operation is thus $\approx$ 4909 μs. These costs provide some context for the following sections about the enrolment and verification stage.

Method	Add	Substract	Rotate	Decrypt	Multiply	Encrypt
Relative cost	1	5	24	33	46	52

Table 5.4: Relative Cost of Palisade Methods.

5.3.2 Enrolment Stage

The average time to enrol a fingerprint was 527,98 ms. This is an increase from the cleartext implementation, which had an average of 123,48 ms. The difference between enrolment for the encrypted and cleartext implementation is the encryption of the template as an added last step in the enrolment stage. For each cylinder in a fingerprint template, three encryptions are performed, and each encryption takes about 6 ms.

5.3.3 Verification Stage

Performing a complete comparison between two templates is time-consuming, and thus the runtime performance sample size for the verification stage is not large. Some of the observed runtimes are summarised in the following list: 7296 s, 8867 s, 9325 s, 9443 s, 11704 s, 11346 s, 12689 s, 13747 s, 14023 s. The cleartext implementation's average comparison time was 132,63 ms $\approx$ 0,13 s. What functions in the encrypted implementation causes this increase in runtime performance?

When a probe template is compared to a reference template, every cylinder in the probe template is compared against every cylinder in the reference template. To illustrate with an example, if the probe and reference template has 30 and 30 cylinders, respectively, then the number of comparisons done between two cylinders is 900. Therefore, every operation that is executed when comparing two cylinders has a significant impact on the overall performance. In some cases, a cylinder comparison will not complete the full set of operations due to certain conditions, for instance, if the angle difference between the two cylinders is too high. Observations showed that about 75% of the cylinder comparisons execute the complete cycle, while 25% are returned after the angle difference is evaluated. These numbers will be used as a reference for the following functions' runtime contribution in the verification stage.

Evaluating the angle difference between two cylinders is the first step of every cylinder comparison, and is therefore executed exactly one time for each cylinder comparison. The average observed runtime performance for Source Code 5.1 was 4129,81 µs, which adds ≈ 3,72 s to the total comparison time for two templates. 25% of the cylinder comparisons are returned after this function, which is why this condition is evaluated first.

```cpp
double cylinderDirectionalDifference(Ciphertext<DCRTPoly> &angle1,
                                       Ciphertext<DCRTPoly> &angle2,
                                       CryptoContext<DCRTPoly> &cc,
                                       LPKeyPair<DCRTPoly> &keys) {
  auto encDelta = cc->EvalSub(angle1, angle2);
  Plaintext ptDelta;
  cc->Decrypt(keys.secretKey, encDelta, &ptDelta);
  double delta = ptDelta->GetRealPackedValue()[0];

  // Equation (9) from the MCC paper.
  if (-M_PI <= delta && delta < M_PI) {
    return delta;
  } else if (delta < -M_PI) {
    return 2 * M_PI + delta;
  } else if (delta >= M_PI) {
    return -2 * M_PI + delta;
  }

  return 0;
}
```

Source Code 5.1: Directional difference between two cylinders.

Calculating the squared Euclidean distance between two vectors and the squared Euclidean norm for a single vector can be done using the same function. The function in Source Code 5.2 passes a subtracted vector to the function in Source Code 5.3, where the Euclidean distance norm is calculated. The observed average runtime for the function in Source Code 5.3 was 3800 ms. This function is executed 4 times for 75% of the cylinder comparisons, making its contribution as big as 170 minutes to the total comparison time if the number of cylinder comparisons is 900. This is the bottleneck for the verification stage, and there have been unsuccessful attempts at calculating the Euclidean norm in a different way. As the Table 5.4 shows, the rotation method is expensive, and it is executed 1536 times in every call to the function in Source Code 5.3.

```
Ciphertext<DCRTPoly> edOfEncryptedVector(Ciphertext<DCRTPoly> &cmA,
                        Ciphertext<DCRTPoly> &cmB,
                        Ciphertext<DCRTPoly> &validityVector,
                        CryptoContext<DCRTPoly> &cc) {
  Ciphertext<DCRTPoly> deltaVector = cc->EvalSub(cmA, cmB);

  return edOfEncryptedVector(deltaVector, validityVector, cc);
}
```

Source Code 5.2: Substraction of vectors and calling the squared Euclidean norm function.

```
Ciphertext<DCRTPoly> edOfEncryptedVector(
                    Ciphertext<DCRTPoly> &encVector,
                    Ciphertext<DCRTPoly> &validityVector,
                    CryptoContext<DCRTPoly> &cc) {
  int vectorSize = 1536;

  Ciphertext<DCRTPoly> squaredVector =
      cc->EvalMult(encVector, encVector);

  squaredVector =
      cc->EvalMult(squaredVector, validityVector);

  Ciphertext<DCRTPoly> rotatedVector =
      cc->EvalAtIndex(squaredVector, 1);

  Ciphertext<DCRTPoly> encED = cc->EvalAdd(squaredVector,
                                            rotatedVector);
  for (int i = 0; i < vectorSize - 1; i++) {
    rotatedVector = cc->EvalAtIndex(rotatedVector, 1);
    encED = cc->EvalAdd(encED, rotatedVector);
  }

  return encED;
}
```

Source Code 5.3: Calculating Euclidean norm.

5.3.4 Optimisation

Since the implementation was based on a research paper and had to be implemented from the ground up, there were several decisions regarding how the implementation should be built. Some of the paper's algorithms were straightforward, while others, especially in the encrypted implementation, required modifications in order to work. A discussion regarding some of these choices, successful and unsuccessful attempts at optimising, and an analysis of potential other ways to optimise runtime are included in this section.

Asynchronous Functions

Asynchronous, or simply async, is a C++ integrated function template that enables multithreading of functions, meaning that operations that are usually happening in sequence can be done in parallel by utilising multiple of the processor cores. This is useful in scenarios where parts of a program are not directly dependent on each other.

Asynchronous functions were tested to improve both the enrolment and verification stage runtime performance. The theory was that the available computing power was underutilised, and the benefit of having multiple cores could be further utilised by introducing asynchronous functions. In the verification stage, the cylinder comparisons between two templates are not dependent on each other, and they could therefore be done in parallel. Also, in the enrolment stage, the creation of cylinders is not dependent on each other. Both cases are a fitting scenario for asynchronous functions.

The gain in performance in the enrolment stage was a 23% decrease in runtime as shown in Table 5.5. For a single enrolment to take 404,43 ms instead of 527,98 ms is not a noticeable difference. However, if the system were to scale up and enrol thousands of new fingerprints simultaneously, the reduction of 123,55 ms would be more significant. A single comparison between two templates was executed both with and without asynchronous functions to test the verification stage. The asynchronous functions decreased the time to compare these two templates by 75%. While still not fast enough to be usable, it was a noteworthy improvement. It can be argued that using extensive computing resources to speed up the runtime is not an accurate representation of its actual performance. Therefore the discussion in Section 5.3 used results from the implementation without asynchronous functions. Still, if this implementation was deployed in centralised servers, it is feasible that they would have large computing services available and would likely be optimised by running functions in parallel.

Stage\Mode	**Without asynchronous functions**	**With asynchronous functions**
Enrolment	527,98 ms	404,43 ms
Verification	12689 s	3219 s

Table 5.5: Runtime performance with and without asynchronous functions.

BFV

As mentioned previously, the proposed solution in this thesis is implemented with the CKKS encryption scheme. The implementation was tested with BFV since it had the potential to improve the performance. Benchmarking results outside of the implementation have shown that BFV was faster than CKKS for HE operations, including rotation, addition, multiplication and decryption. The benchmark results were also tested for several depths [JJ22].

An important difference between BFV and CKKS is that BFV is built for computations on integers and not real numbers. Therefore, the implementation had to be altered to convert its real numbers to integers before applying the encryption scheme on the data. The real values in the contribution vectors have several decimal places, and had to be scaled up to not lose information about each minutia's contribution. This was solved by scaling the contribution vectors by a large factor before encrypting the template. The BFV scheme in PALISADE requires a `PlaintextModulus` parameter to be set to the highest possible value that can occur. Due to the calculation done in the encrypted domain, the `PlaintextModulus` had to be large, which impacted the performance negatively. Observations made during testing of BFV was that it performed slower than CKKS, in addition to having a accuracy loss in the verification stage. CKKS was therefore decided on as the better option for this implementation and use case.

BGV

Kim et al. implemented improved variants of BGV and BFV in PALISADE [KPZ21]. Their results suggested that BGV performed better with large plaintext moduli. Motivated by this paper, the implementation was re-written to use BGV to evaluate whether this was correct for the standard PALISADE version as well, or if it was down to the improvements made by Kim et al..

The observation made after testing the implementation with BGV and several plaintext moduli of different sizes was a performance slower than with BFV and CKKS. Thus using BGV for this implementation was not an improvement neither in terms of performance or accuracy.

Adjust Parameters

Within HE, parameters have a significant influence on the performance of the encryption schemes. Some of these parameters include `ringDim`, `batchSize`, `multDepth` and `scaleFactorBits`. The implementation's runtime improved when these parameters were changed to be better accustomed to the plaintext data.

Allowing PALISADE to assign the `ringDim` based on the other parameters instead of manually assigning it was one measure which optimised the implementation's performance.

Aiming for a low `multDepth` would also influence the performance positively since a high value was associated with an increased cost. The earlier versions of the verification stage required a `multDepth` set to 3. However, by adjusting the operations in the verification stage, `multDepth` set to 2 was sufficient to perform the multiplications and handle the noise growth. The operations were changed so that instead of doing multiplications sequentially, they were organised as a binary tree multiplication. For example, multiplying sequantially gives a multiplicative depth of 3: $a \cdot b \cdot c \cdot d$. But organising it the following way lowers the multiplicative depth to 2: $a \cdot b = e,\ c \cdot d = f,\ e \cdot f$.

Changing the `scaleFactorBits` increased performance most significantly. The earliest versions of the implementation had `scaleFactorBits` set to 50, which was unnecessarily high for the data evaluated by the implementation. Through some trial and error, observations suggested that `scaleFactorBits` set to 30 bits gave the necessary precision and an increased runtime performance. The reduction from 50 bits to 30 bits increased performance equal to approximately 1/3 of the original time for the verification and enrolment stage.

Reduce Number of Operations

Reducing the number of operations should, in theory, improve the performance as each operation is costly in terms of time. The second variable revealed in cleartext, representing the number of matchable cells in two cylinders, is calculated using the squared Euclidean norm function (Source Code 5.3) on *cvv*. This function squares the inputted vector, which should be an unnecessary operation for *cvv* since it contains binary values. When this was identified, a new function (Source Code 5.4) was written to instead only do addition of the values in *cvv*. Runtime data were collected before and after the change to see how much time would be saved by removing one squaring, and the results were not as expected. Table 5.6 show the time to calculate the number of matchable cells in two cylinders with and without the squaring. The runtimes suggest that removing the squaring operation increased the runtime. This was an unexpected behaviour, and it had to be researched to figure out what caused it.

```
Ciphertext<DCRTPoly> edOfValidityVector(
                    Ciphertext<DCRTPoly> validityVector,
                    CryptoContext<DCRTPoly> &cc) {
  int vectorSize = 1536;

  Ciphertext<DCRTPoly> ciphertextRot =
      cc->EvalAtIndex(validityVector, 1);

  auto encED = cc->EvalAdd(validityVector, ciphertextRot);

  for (int i = 0; i < vectorSize - 1; i++) {
    ciphertextRot = cc->EvalAtIndex(ciphertextRot, 1);
    encED = cc->EvalAdd(encED, ciphertextRot);
  }
  return encED;
}
```

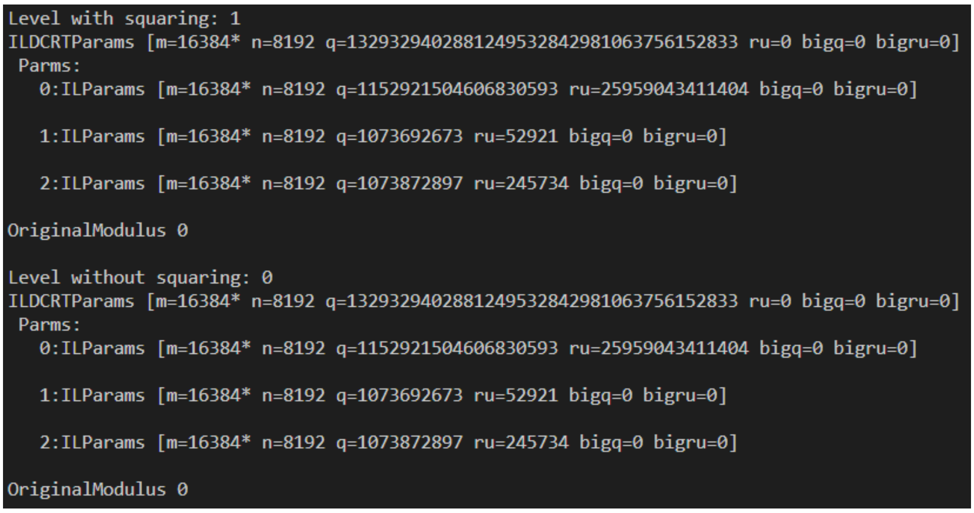
Source Code 5.4: Addition of values in the common validity vector.

With Squaring	5106 ms	4139 ms	3787 ms	3771 ms	3485 ms
Without Squaring	7273 ms	6245 ms	5950 ms	6028 ms	5925 ms

Table 5.6: Euclidean norm with and without squaring runtime performance.

The implementation's configuration had the CKKS scheme in addition to the Leveled SWHE capability. Therefore, a decreasing ladder of ciphertext moduli was set based on this implementation's fixed depth of computation. Figure 5.7 shows the level and parameters for each level for the two scenarios, with and without squaring. Squaring the *cvv* lead to a shift from level 0 to 1, thus decreasing the modulus from 1152921504606830593 to 1073692673. The level 0 modulus is a 60-bit integer, while the level 1 modulus is a 30-bit integer. In the scenario with no squaring, the level was kept at 0, therefore having a significantly higher modulus. The reduction in modulus caused the following 3072 operations in Source Code 5.3 to be executed faster, and this was the cause for the unexpected runtime increase when the squaring was removed. Rotation and addition does not increase the level, but multiplication do. Thus, the implementation kept the function (Source Code 5.3) performing the squaring operation.

Other examples of ways to optimise the code to function equally with fewer or simpler operations are likely to be found by studying the implementation further.


```
Level with squaring: 1
ILDCRTParams [m=16384* n=8192 q=1329329402881249532842981063756152833 ru=0 bigq=0 bigru=0]
 Parms:
   0:ILParams [m=16384* n=8192 q=1152921504606830593 ru=25959043411404 bigq=0 bigru=0]

   1:ILParams [m=16384* n=8192 q=1073692673 ru=52921 bigq=0 bigru=0]

   2:ILParams [m=16384* n=8192 q=1073872897 ru=245734 bigq=0 bigru=0]

OriginalModulus 0

Level without squaring: 0
ILDCRTParams [m=16384* n=8192 q=1329329402881249532842981063756152833 ru=0 bigq=0 bigru=0]
 Parms:
   0:ILParams [m=16384* n=8192 q=1152921504606830593 ru=25959043411404 bigq=0 bigru=0]

   1:ILParams [m=16384* n=8192 q=1073692673 ru=52921 bigq=0 bigru=0]

   2:ILParams [m=16384* n=8192 q=1073872897 ru=245734 bigq=0 bigru=0]

OriginalModulus 0
```

Figure 5.7: Level and parameters for ciphertexts with and without the squaring operation.

Multiple CryptoContexts

The batch size used in the `CryptoContext` was manually assigned to fit the application. An issue with this is that PALISADE allows for only one `CryptoContext` in a PALISADE application, which means that only one batch size can be set, independent of how many different variables are supposed to be encrypted. In this implementation, there are three encrypted vectors, two of which consist of 1536 values. The last vector consists of only one value, but is still encrypted with the `CryptoContext` assigned a `batchSize` capable of 1536 values. If two or more `CryptoContexts` could be made, each of them could be optimised for the variable it should encrypt and compute on. A challenge with this is communicating these values between different PALISADE applications.

Pack Values Together

Packing a cylinder's 1536 contribution values into a single plaintext instead of individual plaintexts improved the enrolment stage's performance. The initial idea in the implementation was to encrypt each of the contribution values individually and store them for further computations as 1536 individual HE ciphertexts. When working on optimising the code for performance, it was observed that the cost of one encryption operation had a high cost. Additionally, it did not matter whether a vector had 1536 values or one value regarding runtime. Therefore, reducing the number of encryptions from 1536 to one significantly impacted the runtime, especially in the enrolment stage. Instead of storing every contribution as double values, one vector held 1536 double values and was encrypted only once. Before the change was introduced, a single enrolment process could take 1-2 minutes to complete. After the change, the average enrolment runtime was 527,98 ms.

The idea of packing values together instead of encrypting them individually was implemented where it was possible. It was impractical to do so for the cylinder angles. The difference between multiple cylinder angles from the probe and reference templates must be calculated and revealed in cleartext. This requires the program to know which cylinder angles belong to which cylinders, which, when packed together as a single ciphertext, is not feasible since the ciphertext cannot be indexed. Therefore the cylinder angles was kept in separate ciphertexts.

Early Return Conditions

Conditional statements make up a considerable fraction of the MCC algorithm. When calculating the similarity score between two cylinders, three conditional statements must be valid for a score to be calculated. All these statements return 0 if the conditions are not satisfied, indicating that the cylinder comparison has no contribution to the similarity between the templates. The cylinder similarity function was structured so that the least costly conditions are evaluated as early as possible, making them return 0 before any redundant, costly calculations are performed. This avoids the scenario where the two cylinders are `non-matchable` but are not declared as `non-matchable` until their cylinder similarity score is already calculated, wasting time on unnecessary calculations.

5.4 Security and Privacy

This section focuses on the security and privacy aspect of the implementation and includes a discussion of how secure the overall solution is, and potential vulnerabilities in the system.

5.4.1 Indistinguishability under Adaptive Chosen Ciphertext Attack

Regarding the security of the proposed solution in this thesis, it is necessary also to address the security of FHE in general. Encryption schemes that support any form of homomorphic operation cannot be secure against chosen ciphertext attacks [LMSV11]. The reason for this is because every encryption scheme that support homomorphic operations are malleable [LMSV11]. Being malleable means that it is possible to transform a ciphertext into another ciphertext that decrypts into a related plaintext [DDN00], and FHE schemes are therefore not able to achieve the Indistinguishability Under Adaptive Chosen Ciphertext Attack security [LMSV11].

5.4.2 Separation of Duty

The design choice of dividing the server into two entities, the AS and the CS, was to achieve SOD. SOD is a principle that no single entity should have the capability to misuse a system on its own. Translated to this use case, it means that the server that receives encrypted templates and computes on them should not hold the private key. With this design, a compromised CS can not decrypt the encrypted templates. Likewise, a compromised AS does not have access to encrypted templates, and can therefore not decrypt them even though it has controls the private key. The scenario that an adversary compromises both servers is more unlikely than compromising a single server, and the proposed solution therefore achieves SOD.

5.4.3 Information Exposed Through Decryption

The variables decrypted during the execution of the implementation are the directional difference between cylinders, the number of matchable cells in the two cylinders and three different Euclidean norms. The reasoning for decrypting these variables have been discussed as part of the proposed solution in Chapter 5, and this section discusses to which degree they contribute to a lack of privacy for users. Note that in this section, it is assumed that the values are revealed to an adversary, but in the proposed solution, these values are not transmitted to the client at any point.

The directional difference between two cylinders is the first variable to be available in cleartext. This value describes a difference between two angles and does not expose each of the cylinders' angles. One could argue that an infinite number of angle combinations could result in any given difference. Therefore, the difference in itself is not sensitive and does not expose information about the fingerprint. However, some risk exists related to exposing this value since it opens up an opportunity for an exploit. Hill-Climbing attack against fingerprint recognition systems uses multiple attempts to choose and modify a biometric input so that the result surpasses the threshold score [MFA+06]. This could, for instance, be to send random templates as input until a template is approved or modify each minutia in one template until that template is approved. As the conditional requirement for the directional difference in the MCC algorithm is known, an attacker could modify the angles of the minutiae until a higher comparison score is achieved, using the decrypted directional difference as guidance.

The number of matchable cells between two cylinders exposes in some sense the degree of overlap between two minutiae relative to their surroundings in the fingerprint. This could be used by an adversary to gather information about the surroundings of each cylinder, and use this maliciously to understand the composition of the minutiae in the fingerprint as a whole. This is also a challenging task for a potential adversary as every cylinder is compared with every other cylinder. The application does not expose which cylinders represent the same minutia in the two fingerprint images. Neither does the variable expose any information about which type of minutiae the cylinders represent, or at which coordinates they lie. Regarding this, exposing the number of matchable cells between two minutiae does not cause any obvious risk but could become a helpful tool for an adversary.

The Euclidean norms are the numerical values from which the cylinder similarity score is calculated, as described in equation (17) from the official MCC paper [CFM10]. These values exposes the squared length of the contribution vectors $c_{a|b}$, $c_{b|a}$ and $c_{a|b} - c_{b|a}$. It is infeasible to obtain any of these three vectors from the lengths, as the lengths are calculated based on 1536 squared values between 0 and 1 added together. As the length values themselves do not reveal anything about the fingerprints, these values do not heavily impact the privacy aspect.

5.4.4 Parameters

Some parameter choices influence the degree of security in the application. The implementation uses 128-bit security level, which offers 128-bit security against classical computers [Pde17]. This is the lowest security level considered secure at the time of writing. To compensate for a high performance time, multiple parameters have been chosen so that they are, in general, considered secure, but does not apply any extra security.

Chapter 6 Conclusion

Verification of fingerprints in the encrypted domain is a challenging task. As the world becomes more digital and centralised, privacy and security in the digital world receive increasing attention from everyday people. People might be concerned when enrolling sensitive information such as fingerprints in a service. However, using fingerprints as an application authentication method can benefit users, as they will not have to remember a password. Additionally, using fingerprints guarantees a unique entry for every user. If fingerprints could be authenticated efficiently without compromising privacy and security, it could enhance users' experience.

This thesis presents a solution for privacy-preserving fingerprint verification based on the MCC fingerprint comparison algorithm. The implementation is written in the programming language C++ and uses the FHE library PALISADE for the functionality related to the encrypted domain. The implementation enrols fingerprint templates based on lists of minutiae and encrypts the fingerprint templates using a public key pair. It can then compare two encrypted templates partially in the encrypted domain to obtain a comparison score in cleartext. This comparison score is compared against a threshold to decide whether two templates were from the same data subject or not. The implementation is available on Github[1].

In order to evaluate the implementation, it was tested and analysed using fingerprint images from two fingerprint databases. The degree to which it could distinguish mated and non-mated fingerprints in both the cleartext and encrypted domain, its efficiency, and privacy and security were discussed. The verification results were quantitative, based on the two different fingerprint image databases: the FVC2002 database with 399 subjects [FVC02], and the CrossMatch database with 51 subjects [Neu22]. Whether the implementation achieved its intended privacy and security property was evaluated based on qualitative analysis.

Distinguishing between mated and non-mated fingerprints was accomplished to a

[1]https://github.com/TTM4502-FHE/Proof-Of-Concept

varying degree. The fingerprint images from the FVC2002 database are intentionally made difficult to correctly accept or reject in biometric verification. The images from the CrossMatch database are of higher quality with less distinction between fingerprint images from the same data subject. This made the verification results significantly better with the CrossMatch database. With an FMR of 1 %, the Crossmatch and FVC database had a TMR of 94,40 % and 68,13 % respectively.

The goal was to have the same accuracy in the encrypted domain as the cleartext domain, and it was achieved with no observed loss in accuracy. However, the encrypted implementation's practicality is negatively affected by its time-consuming functions. This is mainly due to MCC requiring a costly amount of operations when implemented with FHE and is hard to overcome without any improvements in time costs in FHE or PALISADE. A fully privacy-preserving implementation of MCC is also a difficult issue that was not entirely solved in this thesis, as MCC requires operations such as conditional statements and sorting. These operations are not directly compatible with FHE and was it was deemed that they could not be avoided. Therefore, this was solved by decryption of selected values, which impacted the intent of creating a privacy-preserving solution negatively.

6.1 Further Work

This section consists of different suggestions for improvement within the application and the issue of privacy-preserving fingerprint recognition in general.

6.1.1 New FHE and PALISADE Capabilities

As potentially new FHE schemes supporting a broader range of operations are defined, more complex comparisons can be performed in the encrypted domain. Not having an accessible way to sort or evaluate conditional statements in the encrypted domain was a limiting factor in this thesis, but further development could solve this problem. PALISADE is also under development, and based on [KPZ21] it can be assumed PALISADE has a potential for improvement with regards to both runtime performance and support of new encryption schemes.

6.1.2 Different Fingerprint Comparison Algorithm

As discussed in Chapter 4, the MCC algorithm was decided to be most compatible for encrypted implementation with FHE among the researched candidates. Still, there are more algorithms to explore since this thesis did not include an exhaustive search for every fingerprint comparison algorithm, and the biometrics field is in active development. There will be new algorithms which could substitute MCC and be implemented using FHE.

6.1.3 Multi-Party Computation

Steps can also be taken away from FHE in order to accomplish a privacy-preserving fingerprint verification solution. Multi-party computation is a well-known subfield within cryptography that amongst others enables computations of arithmetic circuits over a finite field $\mathbb{F}_{p^k}$ [DPSZ12]. This could potentially be implemented with a fingerprint comparison algorithm like MCC. Some research already exists in this area. Bringer et al. describe in [BCP13] techniques that could be used to perform biometric algorithms while still securing the biometric data, using multi-party computation.

References

[ACC+21] M. Albrecht, M. Chase, *et al.*, «Homomorphic encryption standard», in *Protecting Privacy through Homomorphic Encryption*, K. Lauter, W. Dai, and K. Laine, Eds. Cham: Springer International Publishing, 2021, pp. 31–62. [Online]. Available: https://doi.org/10.1007/978-3-030-77287-1_2.

[Bar20] E. Barker, «Recommendation for key management:: Part 1 - general», National Institute of Standards and Technology, Gaithersburg, MD, Tech. Rep., 2020, (last visited: 06. April, 2022).

[BBC+10] M. Barni, T. Bianchi, *et al.*, «A privacy-compliant fingerprint recognition system based on homomorphic encryption and fingercode templates», *2010 Fourth IEEE International Conference on Biometrics: Theory, Applications and Systems (BTAS)*, Sep. 2010. [Online]. Available: https://ieeexplore.ieee.org/document/5634527.

[BCP13] J. Bringer, H. Chabanne, and A. Patey, «Privacy-preserving biometric identification using secure multiparty computation: An overview and recent trends», *IEEE Signal Processing Magazine*, vol. 30, no. 2, pp. 42–52, 2013.

[BGV11] Z. Brakerski, C. Gentry, and V. Vaikuntanathan, «Fully homomorphic encryption without bootstrapping», *Electron. Colloquium Comput. Complex.*, vol. 18, p. 111, 2011.

[Bod18] V. N. Boddeti, «Secure face matching using fully homomorphic encryption», in *Proc. Intl. Conf. on Biometrics Theory, Applications and Systems (BTAS)*, IEEE, 2018, pp. 1–10.

[BOK+22] P. Bauspieß, J. Olafsson, *et al.*, «Improved homomorphically encrypted biometric identification using coefficient packing», in *Proc. Intl. Workshop on Biometrics and Forensics (IWBF)*, 2022.

[Bra12] Z. Brakerski, «Fully homomorphic encryption without modulus switching from classical gapsvp», *Proceedings of Advances in Cryptology-Crypto*, vol. 7417, Aug. 2012.

[CFM10] R. Cappelli, M. Ferrara, and D. Maltoni, «Minutia cylinder-code: A new representation and matching technique for fingerprint recognition», *IEEE transactions on pattern analysis and machine intelligence*, vol. 32, pp. 2128–41, Dec. 2010.

[CGGI20] I. Chillotti, N. Gama, *et al.*, «Tfhe: Fast fully homomorphic encryption over the torus», *Journal of Cryptology*, vol. 33, pp. 34–91, Jan. 2020.

[CKKS17] J. H. Cheon, A. Kim, *et al.*, «Homomorphic encryption for arithmetic of approximate numbers», in *Advances in Cryptology – ASIACRYPT 2017*, T. Takagi and T. Peyrin, Eds., Cham: Springer International Publishing, 2017, pp. 409–437.

[CMLM07] R. Cappelli, D. Maio, *et al.*, «Fingerprint image reconstruction from standard templates», *IEEE transactions on pattern analysis and machine intelligence*, vol. 29, no. 9, pp. 1489–1503, 2007.

[CMM01] R. Cappelli, D. Maio, and D. Maltoni, «Modelling plastic distortion in fingerprint images», in *Advances in Pattern Recognition — ICAPR 2001*, S. Singh, N. Murshed, and W. Kropatsch, Eds., Berlin, Heidelberg: Springer Berlin Heidelberg, 2001, pp. 371–378.

[Col04] S. A. Cole, «History of fingerprint pattern recognition», in *Automatic Fingerprint Recognition Systems*, N. Ratha and R. Bolle, Eds. New York, NY: Springer New York, 2004, pp. 1–25. [Online]. Available: https://doi.org/10.1007/0-387-21685-5_1.

[DDN00] D. Dolev, C. Dwork, and M. Naor, «Nonmalleable cryptography», *SIAM Journal on Computing*, vol. 30, no. 2, pp. 391–437, 2000. [Online]. Available: https://doi.org/10.1137/S0097539795291562.

[DKOK20] J. M. De Fuentes, T. Kim, *et al.*, «Efficient privacy-preserving fingerprint-based authentication system using fully homomorphic encryption», Hindawi, 2020. [Online]. Available: https://doi.org/10.1155/2020/4195852.

[DKRS03] I. Dinur, G. Kindler, *et al.*, «Approximating cvp to within almost-polynomial factors is np-hard», *Combinatorica*, vol. 23, pp. 205–243, Apr. 2003.

[DPSZ12] I. Damgård, V. Pastro, *et al.*, «Multiparty computation from somewhat homomorphic encryption», in *Advances in Cryptology – CRYPTO 2012*, R. Safavi-Naini and R. Canetti, Eds., Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, pp. 643–662.

[ECJ19] J. J. Engelsma, K. Cao, and A. K. Jain, *Learning a fixed-length fingerprint representation*, 2019. [Online]. Available: https://arxiv.org/abs/1909.09901.

[ElG85] T. ElGamal, «A public key cryptosystem and a signature scheme based on discrete logarithms», *IEEE transactions on information theory*, vol. 31, no. 4, pp. 469–472, 1985.

[FV12] J. Fan and F. Vercauteren, «Somewhat practical fully homomorphic encryption», *IACR Cryptol. ePrint Arch.*, vol. 2012, p. 144, 2012.

[FVC02] Fingerprint Verification Competition, 2002. [Online]. Available: http://bias.csr.unibo.it/fvc2002/default.asp (last visited: Apr. 25, 2022).

[Gab46] D. Gabor, «Theory of communication. part 1: The analysis of information», *The journal of the Institution of Electrical Engineers.*, vol. 93, no. 26, pp. 429–441, 1946.

[GDPR16] *What is considered personal data under the eu gdpr?* [Online]. Available: https://gdpr.eu/eu-gdpr-personal-data/ (last visited: Jan. 11, 2022).

[Gen09] C. Gentry, «Fully homomorphic encryption using ideal lattices», in *Proceedings of the Forty-First Annual ACM Symposium on Theory of Computing*, ser. STOC '09, Bethesda, MD, USA: Association for Computing Machinery, 2009, pp. 169–178. [Online]. Available: https://doi.org/10.1145/1536414.1536440.

[GGH97] O. Goldreich, S. Goldwasser, and S. Halevi, «Public-key cryptosystems from lattice reduction problems», in *Advances in Cryptology — CRYPTO '97*, B. S. Kaliski, Ed., Berlin, Heidelberg: Springer Berlin Heidelberg, 1997, pp. 112–131.

[GMG+17] M. Gomez-Barrero, E. Maiorana, *et al.*, «Multi-biometric template protection based on homomorphic encryption», *Pattern Recognition*, vol. 67, pp. 149–163, Jul. 2017. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0031320317300249.

[Han22] *Heaan*, Jan. 2022. [Online]. Available: https://github.com/snucrypto/HEAAN.

[Heb22] IBM, *Helib v2.2.1*, Jan. 2022. [Online]. Available: https://github.com/homenc/HElib.

[ISO11] ISO/IEC JTC1 SC37 Biometrics, *Information technology — biometric data interchange formats — part 8: Finger pattern skeletal data*, International Organization for Standardization, 2011.

[ISO17] ——, *Iso/iec 2382-37:2017 information technology - vocabulary - part 37: Biometrics*, International Organization for Standardization, 2017.

[ISO21] ——, *ISO/IEC 19795-1:2021. information technology – biometric performance testing and reporting – part 1: Principles and framework*, International Organization for Standardization, Jun. 2021.

[JJ22] L. Jiang and L. Ju, *Fhebench: Benchmarking fully homomorphic encryption schemes*, 2022. [Online]. Available: https://arxiv.org/abs/2203.00728.

[JPHP00] A. Jain, S. Prabhakar, *et al.*, «Filterbank-based fingerprint matching», *IEEE Transactions on Image Processing*, vol. 9, no. 5, pp. 846–859, 2000.

[JPHP99] A. Jain, S. Prabhakar, *et al.*, «Fingercode: A filterbank for fingerprint representation and matching», vol. 2, Jan. 1999, 193 Vol. 2.

[KBG+19] J. Kolberg, P. Bauspieß, *et al.*, «Template protection based on homomorphic encryption: Computationally efficient application to iris-biometric verification and identification», in *2019 IEEE International Workshop on Information Forensics and Security (WIFS)*, 2019, pp. 1–6.

[KDG+20] J. Kolberg, P. Drozdowski, *et al.*, «Efficiency analysis of post-quantum-secure face template protection schemes based on homomorphic encryption», in *Intl. Conf. of the Biometrics Special Interest Group (BIOSIG)*, Gesellschaft für Informatik e.V., Sep. 2020, pp. 175–182.

[Kho05] S. Khot, «Hardness of approximating the shortest vector problem in lattices», *J. ACM*, vol. 52, no. 5, pp. 789–808, Sep. 2005. [Online]. Available: https://doi.org/10.1145/1089023.1089027.

[KPZ21] A. Kim, Y. Polyakov, and V. Zucca, «Revisiting homomorphic encryption schemes for finite fields», in *Advances in Cryptology – ASIACRYPT 2021*, M. Tibouchi and H. Wang, Eds., Cham: Springer International Publishing, 2021, pp. 608–639.

[LMSV11] J. Loftus, A. May, *et al.*, «On cca-secure somewhat homomorphic encryption», vol. 7118, Aug. 2011, pp. 55–72.

[Lo22] *Lattigo v2.4.0*, Jan. 2022. [Online]. Available: https://github.com/ldsec/lattig o.

[MFA+06] M. Martinez-Diaz, J. Fierrez, *et al.*, «Hill-climbing and brute-force attacks on biometric systems: A case study in match-on-card fingerprint verification», Nov. 2006, pp. 151–159.

[MR09] D. Micciancio and O. Regev, «Lattice-based cryptography», in *Post-Quantum Cryptography*, D. J. Bernstein, J. Buchmann, and E. Dahmen, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2009, pp. 147–191. [Online]. Available: https://doi.org/10.1007/978-3-540-88702-7_5.

[Neu22] Neurotechnology, *Cross match verifier 300 single fingerprint scanner*, 2022. [Online]. Available: https://www.neurotechnology.com/fingerprint-scanner-cr oss-match-verifier-300-classic.html.

[Pai99] P. Paillier, «Public-key cryptosystems based on composite degree residuosity classes», in *Advances in Cryptology — EUROCRYPT '99*, J. Stern, Ed., Berlin, Heidelberg: Springer Berlin Heidelberg, 1999, pp. 223–238.

[Pde17] *PALISADE Lattice Cryptography Library (release 1.11.2)*, Mar. 2017. [Online]. Available: https://palisade-crypto.org/.

[Pgh21] *Palisade v1.11.2*, May 2021. [Online]. Available: https://gitlab.com/palisade /palisade-development.

[RAD78] R. Rivest, L. Adleman, and M. Dertouzos, *ON DATA BANKS AND PRIVACY HOMOMORPHISMS*. 1978. [Online]. Available: https://luca-giuzzi.unibs.it/c orsi/Support/papers-cryptography/RAD78.pdf.

[Riv90] R. L. Rivest, «Cryptography», in *Handbook of Theoretical Computer Science, Volume A: Algorithms and Complexity*, J. van Leeuwen, Ed., Elsevier and MIT Press, 1990, pp. 717–755.

[SEL20] *Microsoft SEAL v3.6.0*, Nov. 2020. [Online]. Available: https://github.com /Microsoft/SEAL.

[SFD30] Sample Fingerprint Database. [Online]. Available: https://www.neurotechnol ogy.com/download.html (last visited: Apr. 25, 2022).

[SP18] D. R. Stinson and M. B. Paterson, *Cryptography — Theory and Practice*, Fourth edition. CRC Press, 2018.

[SSO12] S. F. Shahandashti, R. Safavi-Naini, and P. Ogunbona, «Private fingerprint matching», in *Information Security and Privacy*, W. Susilo, Y. Mu, and J. Seberry, Eds., Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, pp. 426–433.

[Ty20] *Trusty tee*, Sep. 2020. [Online]. Available: https://source.android.com/security/trusty (last visited: Mar. 28, 2022).

[Use17] UserManual.wiki, *Palisade manual*, Dec. 2017. [Online]. Available: https://usermanual.wiki/Document/palisademanual.1250596869/help.

[Važ21] R. Važan, *Sourceafis fingerprint matcher v3.13.0*, Apr. 2021. [Online]. Available: https://sourceafis.machinezoo.com/ (last visited: Jan. 18, 2022).

[Wh22] *Windows hello biometrics in the enterprise*, Feb. 2022. [Online]. Available: https://docs.microsoft.com/en-us/windows/security/identity-protection/hello-for-business/hello-biometrics-in-enterprise (last visited: Mar. 28, 2022).

[Yao86] A. C.-C. Yao, «How to generate and exchange secrets», in *27th Annual Symposium on Foundations of Computer Science (sfcs 1986)*, 1986, pp. 162–167.

[YWY+20] W. Yang, S. Wang, *et al.*, «Secure fingerprint authentication with homomorphic encryption», *2020 Digital Image Computing: Techniques and Applications (DICTA)*, Nov. 2020. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/9363426.

[Zae11] N. Zaeri, «Minutiae-based fingerprint extraction and recognition», in Jun. 2011.

[ZK16] Y. Zhang and F. Koushanfar, «Robust privacy-preserving fingerprint authentication», in *2016 IEEE International Symposium on Hardware Oriented Security and Trust (HOST)*, 2016, pp. 1–6.

[ZLZ+11] D. Zhang, F. Liu, *et al.*, «Selecting a reference high resolution for fingerprint recognition using minutiae and pores», *Instrumentation and Measurement, IEEE Transactions on*, vol. 60, pp. 863–871, Apr. 2011.

[ZYZ05] E. Zhu, J. Yin, and G. Zhang, «Fingerprint matching based on global alignment of multiple reference minutiae», *Pattern Recognition*, vol. 38, no. 10, pp. 1685–1694, 2005. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0031320305001329.

[ZZZL10] Q. Zhao, D. Zhang, *et al.*, «Adaptive fingerprint pore modeling and extraction», *Pattern Recognition*, vol. 43, no. 8, pp. 2833–2844, 2010. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0031320310000944.

[AAUC18] A. Acar, H. Aksu, *et al.*, «A survey on homomorphic encryption schemes», *ACM Computing Surveys*, vol. 51, no. 4, pp. 1–35, Sep. 2018.