UCLouvain

École polytechnique de Louvain

Privacy-Preserving Face Verification with Fully Homomorphic Encryption

Author: **Kerem OKYAY**
Supervisors: **François-Xavier STANDAERT, Thomas PETERS**
Reader: **Olivier PEREIRA**
Academic year 2024–2025
Master [120] in Computer Science and Engineering

Abstract

This thesis investigates the use of Fully Homomorphic Encryption (FHE) for privacy-preserving one-to-one face verification. Motivated by the sensitivity of biometric data and privacy regulations such as the General Data Protection Regulation (GDPR), we focus on enabling secure computation directly on encrypted embeddings using the Fast Fully Homomorphic Encryption over the Torus (TFHE) scheme and Zama's `Concrete` framework.

To address FHE's computational overhead, we propose an optimized preprocessing pipeline comprising Principal Component Analysis (PCA) for dimensionality reduction, min–max normalization, and integer quantization. We compare global and per-feature (per-dimension) strategies and find that global normalization with 4-bit global quantization offers the best trade-off between efficiency and performance metrics.

Our system performs encrypted comparisons in approximately 0.142 seconds, with ciphertext sizes as small as 1.27 KB. We further demonstrate practical viability by integrating this pipeline into a proof-of-concept client-server communication scenario using Elliptic Curve Diffie–Hellman (ECDH) key exchange, Schnorr zero-knowledge identification, and Hash-based Message Authentication Codes (HMAC).

Although slower than some previous approaches, our system achieves better performance on several metrics while significantly reducing memory usage, demonstrating the practicality of FHE for biometric authentication in low-throughput, high-security environments.

Contents

Chapter 1

Introduction

The proliferation of biometric technologies, particularly face recognition, into everyday applications offers unprecedented convenience and security. However, this rapid integration brings with it a significant problem: the very data designed to protect and identify us are inherently sensitive, immutable, and, if compromised, can lead to severe and irreversible privacy violations. Incidents such as the 2019 Biostar 2 breach, which exposed the biometric details of over a million individuals[1], serve as strong reminders of these vulnerabilities. Combined with increasingly strict regulatory rules such as the General Data Protection Regulation (GDPR) [1] and the California Consumer Privacy Act (CCPA) [2], the demand for robust and privacy-preserving mechanisms for dealing with biometric data has never been more urgent. Traditional face recognition/verification architectures, which often depend on centralized servers for storing and processing plaintext biometric embeddings, present a critical point of failure. Such systems are inherently vulnerable to internal and external threats, such as a compromised server or an untrusted operator, both of which can lead to catastrophic data leakage.

Considering these developments, the research community has intensified its efforts towards developing privacy-enhancing technologies that can safeguard biometric data without sacrificing system accuracy or operational efficiency. While various privacy-enhancing technologies have been explored, a critical challenge remains: how to perform complex biometric matching operations with strong security guarantees without making the system impractical due to computational overhead or data utility loss. Among the most promising privacy-enhancing technologies, Fully Homomorphic Encryption (FHE) stands out for its unique ability to perform arbitrary computations directly on encrypted data, thereby eliminating the need for decryption during processing. While the immense potential of FHE has been recognized, its practical adoption has been challenging due to significant performance overheads and implementation complexities. However, recent advancements, particularly in optimized libraries such as Zama's `Concrete` framework [3] and its underlying `tfhe-rs` implementation [4], have dramatically improved FHE's practicality. These breakthroughs are now narrowing the performance gap, making FHE a possible candidate for computationally intensive tasks like biometric authentication and enabling its evaluation in this domain.

This thesis follows these advancements to investigate the application of FHE to the domain of face verification. The core of this research is the systematic evaluation and optimization of data preprocessing strategies—specifically dimensionality reduction, data normalization, and integer quantization—to minimize the computational overhead in the encrypted domain operations. We believe that carefully tailored preprocessing of biometric features can significantly enhance the feasibility of FHE-based face verification by reducing ciphertext size and computational work, while maintaining good matching performance. This focus on optimizing the input to FHE operations, rather than only focusing on the cryptographic primitives themselves, represents an

[1] `www.theguardian.com/biostar2`

area for practical improvement. Leveraging the `Concrete` framework, this work assesses the impact of these preprocessing techniques on both biometric matching performance metrics and the computational demands of encrypted computations. Furthermore, this research benchmarks our optimized FHE-based approach against prior state-of-the-art encrypted face verification systems, aiming to demonstrate tangible improvements in matching performance metrics and ciphertext compactness. A key contribution of this work is to provide empirical evidence on the trade-offs involved and to offer practical guidelines for configuring FHE-based face verification systems using face embeddings.

To illustrate the practical viability of our findings, the thesis also presents a proof-of-concept communication scenario implementation within a client-server architecture. This demonstration incorporates complementary cryptographic tools to showcase a more complete, but prototypical, real-world integration for privacy-preserving biometric authentication. While a formal security analysis of the entire composed system is beyond the current scope and is not provided here, the prototype serves as a practical example of how these cryptographic components can be combined, relying on the established security properties of FHE for core data protection and privacy.

The thesis is organized as follows: Chapter 2 provides a comprehensive review of the existing literature and surveys the state-of-the-art in encrypted biometric authentication systems. In Chapter 3, we introduce the chosen dataset and detail our preprocessing pipeline, discussing the rationale and expected impact of each preprocessing stage. Chapter 4 describes the proposed client-server architecture, motivating its design choices and outlining the communication scenario for privacy-preserving face verification. Chapter 5 details the experimental setup designed to evaluate the matching performance and computational costs of the preprocessing pipeline in the clear and the encrypted domain, as well as comparative experiments against prior work. The results of these experiments are presented and discussed in Chapter 6. Finally, Chapter 7 concludes the thesis, discusses its limitations, and offers directions for future research.

Chapter 2

Review of Existing Literature

Contents

In this chapter, we review the existing literature in face recognition and examine the developments in privacy-preserving techniques, with a focus on homomorphic encryption. We structure this review into three parts: (1) the evolution of face recognition methods, (2) the privacy challenges they present, and (3) the state-of-the-art encrypted solutions.

2.1 Advances in Face Recognition

Face recognition has long been a challenging problem in computer vision. Early approaches for face recognition relied on hand-crafted features such as Eigenfaces [5] and Fisherfaces [6]. However, more modern approaches to face recognition generally follow the following steps:

1. **Face detection:** Locate a face in the input image.
2. **Face alignment:** Align the detected face for pose, scale, or orientation.
3. **Feature extraction:** Use a deep neural network model to extract a fixed-length representation of the face.
4. **Matching:** Compare two feature vectors to compute a similarity or distance score and compare with a predefined threshold.

The introduction of deep Convolutional Neural Networks (CNNs) brought significant improvements over earlier methods by enabling models to learn feature representations directly from data, thereby replacing manual feature extraction rules.

One of the first successful deep learning models for face recognition was Facebook's DeepFace [7] in 2014 . DeepFace followed the modern pipeline described above, improving the alignment and feature extraction steps. When evaluated on the well-known Labeled Faces in the Wild (LFW) benchmark [8], it achieved 97.35% accuracy, a 27% reduction in error compared to the previous state-of-the-art. This result effectively closed the gap to human performance on this benchmark, which is approximately 97.5%.

In 2015 Google introduced FaceNet [9], a different approach based on learning a direct face embedding optimized for face similarity. Instead of training a CNN to classify identities, FaceNet trains the network to map face images into a Euclidean feature space where the squared distance corresponds to face dissimilarity. Each face is thus encoded as a higher-dimensional embedding on the unit hypersphere (L2-normalized), which enables efficient similarity computation. FaceNet used a novel loss function called *triplet loss*, which aims to minimize the distance between pairs of same identity while maximizing the distance between pairs of different identities. This approach was heavily data-driven, trained on an in-house dataset containing between 100 and 200 million face images, spanning approximately 9 million different identities [10]. With the large training set, and a novel approach for face detection FaceNet achieved 99.63% accuracy on LFW benchmark, surpassing human performance.

By 2016, most state-of-the-art face recognition models were capable of producing embeddings that performed consistently well on benchmarks such as LFW. Since then, progress has been more incremental, with improvements primarily on enhancing the discriminative power of deep features through refined loss functions and classification layers.

Notable developments include SphereFace [11], which improved the softmax loss and achieved 99.43% on the LFW benchmark. CosFace [12] built on this by simplifying the loss function, reaching 99.73%, while ArcFace [13] refined the margin-based approach and achieved 99.83%.

These advancements have established a standard pipeline for modern face recognition systems: deep neural networks extract fixed-length embeddings from face images. These embeddings are then compared using similarity or distance metrics. A match is determined based on whether the computed distance falls below a predetermined threshold [14].

2.2 Privacy Concerns in Face Recognition

Face recognition has since become increasingly ubiquitous in security, authentication, and access control systems, ranging from smartphone unlocking to border control applications [15].

While these embeddings are not directly reversible to the face image, they are sensitive biometric data akin to fingerprints. If leaked, embeddings can compromise user privacy by allowing impersonation or inference of personal attributes [16, 17]. This risk is further increased in cloud-based or cross-enterprise scenarios where face data may be stored or processed by untrusted servers.

To address these privacy concerns, recent research has focused on Homomorphic Encryption (HE) as a solution for privacy-preserving face recognition/verification systems.

HE allows computations to be performed directly on encrypted data without the need for decryption, thereby eliminating one of the key vulnerabilities in traditional biometric systems: the exposure of sensitive data in untrusted environments. This capability directly addresses one of the major challenges identified by Zhang et al. [18] in their comprehensive survey, which emphasizes that performing biometric matching in plaintext on untrusted platforms presents a significant future risk for biometric authentication systems.

2.3 Encrypted Face Recognition Approaches

2.3.1 Early Approaches Using PHE and MPC

Early efforts in this space leveraged partially homomorphic encryption (PHE), often in conjunction with secure multiparty computation (MPC), to enable basic operations on encrypted biometric data. MPC is a cryptographic technique that allows multiple parties to jointly compute a function over their inputs while keeping those inputs private from one another. For instance, Erkin et al. [19] combined PHE with MPC to implement a privacy-preserving version of Eigenfaces [5]. In their system, one party holds a facial image and the other maintains a database, and the protocol securely determines whether there is a match, either returning a binary yes/no result or the identification number (ID) of the matched individual.

Complementing this, Lagendijk et al. [20] provided a tutorial synthesis of encrypted signal processing using both PHE and MPC, to illustrate how classical signal processing tasks like face recognition can be decomposed into encrypted linear operations and secure threshold comparisons.

Barni et al. [21] proposed a secure fingerprint-based authentication scheme (although not focused on facial data), in which feature extraction was performed using a bank of Gabor filters, followed by a secure Euclidean distance computation protocol involving multiple interactive sub-protocols between client and server.

Similarly, Upmanyu et al. [22] proposed a blind authentication protocol that leverages RSA's multiplicative homomorphism to enable secure biometric verification without revealing the underlying biometric data or classifier parameters to either party. The protocol is termed "blind" because the client and the server remain mutually oblivious to each other's inputs; only the final authentication outcome is revealed. To overcome the limitations of RSA's homomorphism—namely its inability to perform additions—the authors introduced a randomization-based protocol that enables secure computation of inner products without exposing intermediate values. Non-linear components of the classification function, such as those used in kernel support vector machines or neural networks, were either approximated using linear models or require increased client-side interaction and computation.

Due to the inherent limitations of PHE schemes, which support only a single type of operation in the encrypted domain, such as addition or multiplication, early biometric authentication systems based on PHE often needed to be supplemented with additional cryptographic techniques such as secure MPC, or required increased interaction between the server and client to overcome these functional constraints and maintain practical utility [23]. Considering these limitations, the use of FHE—which supports arbitrary computations directly on encrypted data—has emerged as a promising direction for biometric authentication.

2.3.2 The Emergence of Fully Homomorphic Encryption

The first FHE scheme was introduced in 2009 by Gentry [24], which meant that it became possible to compute arbitrarily many operations on encrypted data. In other words, this allowed any function to be evaluated in the encrypted domain.

Since then, a variety of FHE schemes have been developed, each with different design goals and trade-offs. These can be broadly categorized according to the type of arithmetic supported, as shown in Table 2.1

Scheme Type	Description and Examples
Binary Arithmetic	Supports bit-level operations, making it efficient for tasks such as logical comparisons and Boolean circuit evaluations. Examples: TFHE [25], FHEW [26].
Exact Arithmetic	Enables fast operations on integers, particularly for additions and multiplications. Examples: BGV [27], BFV [28].
Approximate Arithmetic	Operates on real numbers, well-suited for real-time privacy-preserving applications. Example: CKKS [29].

Table 2.1: Types of FHE schemes and their characteristics.

The invention of FHE opened new possibilities for facial biometric authentication systems without relying on additional privacy-preserving techniques. Over the years, many studies have explored the use of FHE in privacy-preserving face recognition systems. Most of these systems follow a similar client-server architecture, which can be outlined as follows:

1. **Feature extraction:** A trusted client device uses a deep learning model (e.g., FaceNet) to extract a fixed-length feature vector (embedding) from the user's facial image. This vector serves as a numerical representation of the face.

2. **Encryption:** The client encrypts the embedding using an FHE scheme. More precisely, it generates FHE keys (public, private, evaluation keys) and uses the public key to encrypt the user's embedding. The resulting ciphertext vector and evaluation keys are transmitted to the server for storage or comparison.

3. **Encrypted matching:** The server performs face comparison directly on encrypted data. It evaluates a similarity or distance function between the encrypted query and stored encrypted templates, producing an encrypted score. The server does not learn any information about the underlying data.

4. **Decryption and decision:** The client, or another authorized entity possessing the secret key, decrypts the result. The decrypted similarity score is compared against a threshold to determine whether the input matches an enrolled identity.

The steps above are also illustrated in Figure 2.1.

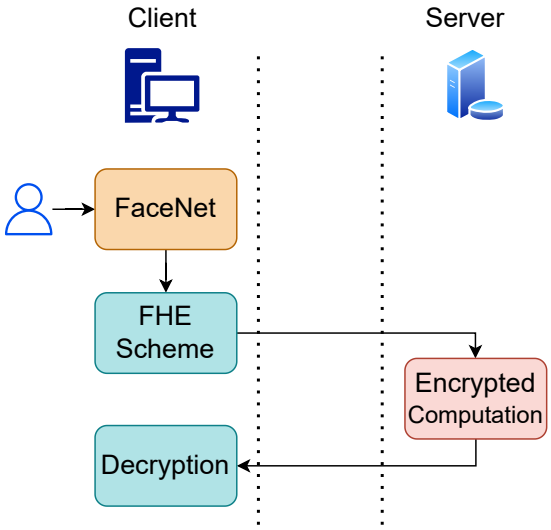

Figure 2.1: Overview of the typical client–server architecture in modern FHE-based biometric authentication systems. FaceNet is a deep neural network model and is used for the feature extraction step. The client uses the selected FHE scheme to generate keys and perform encryption and decryption tasks.

2.3.3 FHE-Based Face Verification Systems

One earlier approach for FHE-based facial verification is due to Troncoso-Pastoriza et al. [30]. It followed a similar client-server structure outlined above, employing handcrafted Gabor-based features with a novel statistical model for coefficient magnitudes as the feature extraction method (comparable to that of Barni et al. [21]). The authors used and improved a lattice-based extension of Gentry's scheme [31], adapted to support low-degree polynomial operations. However, the implementation was computationally expensive and encrypted template sizes—the resulting ciphertext vector of step 2 in the client-server outline above—reached 380 MB. Computing the score between two encrypted embeddings took around 100 seconds.

The analysis done by Boddeti [32] on the practicality of using FHE for face matching over encrypted face templates followed the client-server setup above and employed two different deep neural network based face representations for feature extraction, namely, Facenet [9] and Sphereface [33], with 128- and 512-dimensional embedding outputs, respectively. Additionally, dimensionality reduction with Principal Component Analysis (PCA) was used as an illustrative step to reduce the output to 64 dimensions. The FHE scheme Boddeti used is the BFV scheme [28] combined with a batching scheme to allow for homomorphic multiplication of multiple numbers for the cost of a single homomorphic multiplication. Boddeti then described two communication scenarios: enrollment and authentication. During enrollment, multiple images of a user are captured and features are extracted before being sent to the server for storage in encrypted form under an ID. During authentication, an encrypted query is sent with the claimed ID which is then matched against stored encrypted templates under the given claimed ID. The author provided performance metrics and accuracy metrics of the system implementing these communication scenarios. The analysis compared the implementation in-clear with the implementation in FHE, and found that accuracy is not affected by the addition of FHE. As for efficiency, there is a price. The author observed that FHE-based face matching is approximately 1000× slower than the in-clear implementation. However, it is reported that

addition of the batching scheme offers a significant speed up and saves memory. For 128 bits of security and 128-dimensional features, the batching scheme reduced the computational time of vector matching from 38.64 ms to 4 ms and lowered the memory requirement from 1 MB to 4 KB, all while maintaining a security level equivalent to 2^{128} operations against brute-force attacks.

Jindal et al. [34] extended the approach outlined by Boddeti [32]. By using the CKKS scheme on real numbers with batching, the authors addressed one shortcoming in the BFV scheme, which is the quantization needed to convert real number facial features to integers. As preprocessing steps, the authors first detected the face in a given image and cropped the face out for feature extraction and then employed deep learning features based on 128-dimensional FaceNet [9] embeddings. Similarly, they described enrollment and authentication scenarios, where in enrollment they captured only one image of the user and applied synthetic data augmentation to increase robustness during authentication phase, which follows a similar process to that described by Boddeti. The proposed method by Jindal et al. took about 2.83 ms to match a pair of encrypted, 128-dimensional facial features with competitive or higher performance metrics reported by Boddeti. However, the security level of the CKKS scheme used by Jindal et al. is reported to be at least 80 bits, compared to 128 and 256 bits of security analyzed by Boddeti for the BFV scheme. As for memory requirement of the scheme used by Jindal et al., for 128-dimensional features, 32.8 KB of memory is required for storage, which is higher than 4 KB reported by Boddeti. The Table 2.3 and Table 2.2 summarize the results from Jindal et al. and Boddeti.

Method	**Dim.**	**Security**	**Scheme**	**Comp. Time (ms)**	**Ciphertext Size (KB)**
Boddeti [32]	128	128 bits	BFV	4	4.0
Jindal et al. [34]	128	≥ 80 bits	CKKS	2.8	32.8

Table 2.2: Cryptographic parameters and encrypted-domain comparison costs reported in prior work. "Dim." indicates the dimensionality of the feature embeddings. "Security" refers to the estimated bit-level resistance against brute-force attacks. "Scheme" indicates the FHE scheme used. Computation time corresponds to the encrypted matching of a single vector pair and ciphertext size is the size of a single encrypted embedding.

Method	**TPR@0.01%**	**TPR@0.1%**	**TPR@1%**
Boddeti [32]	84.06	94.56	98.65
Jindal et al. [34]	86.58	94.28	96.10

Table 2.3: Face matching performance reported by Boddeti and Jindal et al., evaluated in the encrypted domain for the LFW dataset. True Positive Rate (TPR) is reported at various False Positive Rate (FPR) thresholds: 0.01%, 0.1%, and 1%. For detailed definitions, see the Appendix B.

2.3.4 FHE-Based Face Search and Identification

Drozdowski et al. [35] presented a system for one-to-many face search and identification in the encrypted domain, extending the ideas of face verification explored by Boddeti [32] and Jindal et al. [34]. Drozdowski et al. used 512-dimensional Facenet [9] embeddings as features and compared two FHE schemes, namely BFV and CKKS, both with batching capability. Additionally, they diverged slightly from the client-server structure outlined previously and

introduced a trusted third party that holds the FHE secret key. The client sends the encrypted probe to the server, which homomorphically computes the distance between the probe and every stored encrypted template. Rather than outputting all distances (which would be encrypted), the server applies a threshold to identify the minimum distance, all in encrypted form. The encrypted result of thresholding is then sent to the trusted third party, which decrypts it and communicates the final decision to the client. This architecture ensures the database never sees unencrypted features, and the client does not get access to raw database values either, only the final match decision is revealed. Using 128-bit security parameters, the authors reported public/secret key sizes under 1 MB and Galois rotation keys of about 10 MB, these extra keys enable non-parallel operations on packed ciphertexts for batching schemes used in BFV and CKKS. They found that using CKKS on the raw 512-dimensional float vectors was slow (≈ 5 s per encrypted distance computation) but that quantizing features to integers and using BFV dramatically sped this up to ≈ 850 ms per comparison. By choosing a sufficiently large quantization, the accuracy of the homomorphic matching yielded virtually the same biometric performance as plaintext.

To generalize the approach by Boddeti [32] for encrypted image search, Engelsma et al. [36] presented a method to search for a probe (or query) image representation against a large gallery in the encrypted domain, including biometric data such as fingerprints and face images. They describe enrollment and search protocols and require fixed-length feature representations of the data which are typical representations acquired via deep neural network-based features. Their method, called HERS (Homomorphically Encrypted Representation Search) consists of the following steps. First, they compress the fixed-length dimensionality towards its estimated intrinsic dimensionality. This step builds on the deep neural network-based non-linear mapping called Deep Multi Dimensional Scaling (DeepMDS) by Gong et al. [37] which achieves the mapping from ambient space to a lower-dimensional intrinsic space. Second, the authors use the BFV scheme to encrypt the compressed representations. Third, they search against a gallery of encrypted representations. The results from Engelsma et al. showed that practical and accurate image search in the encrypted domain is possible. They highlighted that the system finds a match in under 5 minutes against a gallery of 1 million images and it does it within 2% of the unencrypted accuracy.

2.3.5 Secure Client–Server Protocols with FHE

The work presented so far focuses on the lower-level matching operations the client and the server perform. The communication steps for enrollment and authentication between the client and the server are described without a formal security analysis.

Pradel et al. [38] proposed a privacy-preserving biometric authentication protocol using the TFHE scheme. Their protocol supports secure client-server face verification by computing encrypted Euclidean distances and thresholding without revealing either the query or the stored template. The system implements a conditional token release mechanism to authenticate the user, ensuring that only successful verifications result in a valid authentication token. Unlike prior work, this protocol is formally proven to achieve privacy and soundness in an honest-but-curious setting, where all parties follow the protocol correctly but may attempt to learn additional information from the data they receive. To simulate a real-world use case the authors chose to use synthetic biometric vectors of length 128, whose values are integers between 0 and 10, and evaluated the performance on these synthetic features. The TFHE implementation they use has significant computation overhead. They report that for an n-bit multiplication, it takes around 206 seconds without specifying a value for n. However, for vectors with values in the interval $[0, 10]$, the runtime of their full protocol is reported around 34765 seconds, which is far from practical.

2.3.6 Challenges and Opportunities

As highlighted in prior work, the computational cost of FHE operations can vary significantly. This variability stems from several factors, including implementation details, hardware capabilities, the choice of FHE scheme, and the specific optimizations applied to data representation. A straightforward and effective principle is that minimizing the number of operations performed in the encrypted domain can reduce execution time. One practical strategy to achieve this is to carefully analyze and optimize the structure and dimensionality of the input data prior to encryption.

For example, Gong et al. [37] showed that feature representations learned by deep convolutional neural networks tend to be highly redundant. Using their method, DeepMDS, the estimated intrinsic dimensionality of FaceNet embeddings is approximately 12 which is significantly lower than the original 128-dimensional output of the model they evaluated.

Despite the increasing interest in FHE-based facial authentication systems, several important challenges and opportunities remain underexplored. In particular, there is a lack of analysis regarding how input data characteristics and preprocessing strategies impact the performance of computations in the encrypted domain. Most existing works either use fixed embedding dimensions without applying dimensionality reduction or apply it as an illustrative example—e.g., Boddeti [32]— without providing an analysis, often overlooking the practical implications this has on computational efficiency under encryption.

These gaps create opportunities for further investigation in the following areas:

- The combination of preprocessing techniques—such as face detection and dimensionality reduction—to reduce computational costs in FHE-based systems.
- The trade-offs between matching accuracy and performance introduced by preprocessing strategies.
- The practical use of recently developed and optimized libraries for TFHE—such as Zama's `Concrete` framework [3]—to implement efficient, privacy-preserving facial recognition systems.

2.4 Positioning of This Thesis

This thesis builds upon prior work by analyzing preprocessing strategies aimed at reducing the computational load in FHE-based face verification systems. Specifically, it focuses on one-to-one face matching in the encrypted domain, covering both enrollment and authentication phases, as discussed in earlier studies.

The implementation leverages Zama's Python-based `Concrete` framework [3], which supports the TFHE scheme. A key emphasis of this work is on combining preprocessing techniques—specifically, face detection and dimensionality reduction using PCA—to minimize the number of operations required in the encrypted domain.

In addition to PCA, we investigate the effects of normalization and quantization on the embeddings prior to encryption. In particular, we compare global versus per-feature min–max normalization and global versus per-dimension quantization, analyzing and evaluating their influence on matching performance. This detailed analysis of how different preprocessing configurations affect encrypted computation fills a gap left by prior studies, which have largely treated these operations as fixed or secondary.

Lastly, to illustrate the potential for real-world deployment, a client–server communication scenario is presented. It combines the encrypted face matching pipeline with other cryptographic tools—such as elliptic curve Diffie–Hellman key exchange, Schnorr zero-knowledge identification

protocol, and message authentication codes—across the enrollment and authentication phases. While this proof-of-concept implementation does not include a formal security analysis, it demonstrates how FHE can be integrated into broader cryptographic protocols to explore trade-offs in performance, memory usage, and system design within a client–server architecture.

Work	**Scheme**	**Matching**	**Features**	**Preprocessing**
Boddeti (2018) [32]	BFV	1:1	embeddings	PCA
Drozdowski et al. (2019) [35]	BFV, CKKS	1:n	embeddings	–
Jindal et al. (2020) [34]	CKKS	1:1	embeddings	Face Detection
Engelsma et al. (2022) [36]	BFV	1:n	embeddings	DeepMDS++

Table 2.4: Overview of related systems for FHE-based biometric authentication. In the matching column, 1:1 refers to face verification against a known identity, while 1:n denotes identification against a gallery of enrolled templates. In Boddeti's work, PCA was applied as an illustrative example rather than as a core component of the preprocessing pipeline.

Chapter 3

Dataset and Preprocessing Pipeline

Contents

This chapter presents the dataset and preprocessing pipeline used to enable encrypted face matching with FHE. We begin by describing how raw facial images from the LFW dataset are transformed into fixed-length embeddings using the FaceNet model, following a face detection and alignment step with Multi-Task Cascaded Neural Networks (MTCNN). We then detail the application of PCA, normalization, and quantization to reduce the dimensionality and adapt the embeddings for encrypted computation. A brief overview of the TFHE scheme and the `Concrete` framework is provided to contextualize the encryption constraints. We evaluate the suitability of various similarity and distance functions for homomorphic comparison, and analyze how each preprocessing step may impact the pairwise ranking of embeddings, which is an important factor for matching performance.

3.1 The Labeled Faces in the Wild Dataset

The LFW dataset [8] was selected for this thesis due to its widespread adoption in benchmarking face recognition/verification algorithms. It is a publicly available collection of 13,233 face images of 5,749 individuals which were collected from the web to support research in unconstrained face recognition.

LFW poses real-world challenges including variations in pose, illumination, facial expression, age, and occlusion. Each image is annotated with the identity of the person, enabling both verification and identification tasks. To facilitate standardized preprocessing, the dataset includes several aligned and cropped variants in addition to the original images. These are aligned using either eye coordinates or the deep funneling method, and typically resized to fixed resolutions such as 250×250 or 112×96 pixels, making them suitable for deep learning pipelines.

This thesis uses the *non-funneled* version of the dataset with RGB color channels in the range $[0, 255]$. We opt for the non-funneled version as it better reflects real-world conditions where facial images are not pre-aligned, and it allows us to apply custom preprocessing techniques—such as face detection and alignment—tailored to our pipeline. We adopt the standard `pairs.txt` verification protocol, which defines a 10-fold cross-validation setup with 6,000 image pairs. This protocol comes with the dataset and each fold contains 600 pairs: 300 from the same identity (positive pairs) and 300 from different identities (negative pairs).

Due to its variability, standardization, and wide acceptance, LFW serves as a robust benchmark for evaluating face recognition systems under real-world conditions.

3.2 Face Embedding Extraction with FaceNet

For feature extraction, we use FaceNet, specifically the implementation and pre-trained model by Esler [39]. This model was trained on the VGGFace2 dataset [40], which is a dataset for recognizing faces across pose and age and contains a large number of images with high intra-class and inter-class variations. This pre-trained version of FaceNet maps each face image to a fixed-length L2-normalized 512-dimensional embedding, which serves as the initial fixed-length face representation.

As was shown by Jindal et al. [34], cropping the faces from images increases the matching accuracy. This approach is also followed in the implementation of FaceNet from Esler [39], which suggests applying face detection before computing embeddings to improve performance. In the same repository by Esler [39], he also provides the implementation and pre-trained model of MTCNN by Zhang et al. [41]. MTCNN is a deep learning framework designed for face detection and alignment that integrates multiple related tasks into a single cascaded network architecture. It was introduced to improve accuracy and efficiency in detecting faces and their key landmarks in images, even under challenging conditions like varied poses, lighting, and occlusions. These properties make MTCNN a good choice for the task of face detection and alignment on LFW.

To this end, before passing the LFW images through FaceNet, we first perform face detection and alignment using MTCNN. The aligned faces are then resized to 160×160 pixels, and their pixel values are normalized to be centered around zero, in accordance with FaceNet's expected input format. These aligned images are then fed into FaceNet, which outputs 512-dimensional embeddings. The full preprocessing and embedding extraction pipeline is illustrated in Figure 3.1.

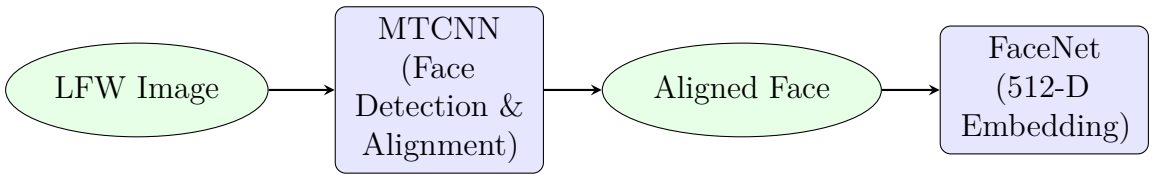

Figure 3.1: Pipeline for obtaining fixed-length face embeddings using FaceNet. The input is an RGB image from the LFW dataset. MTCNN detects and aligns the face, producing a cropped 160×160 RGB image centered around 0. This aligned image is passed to the FaceNet model, which outputs a 512-dimensional L2-normalized floating-point embedding.

3.3 Dimensionality Reduction

Gong et al. [37] observed that feature representations learned by deep convolutional neural networks tend to be highly redundant. This observation is particularly relevant in the context of FHE, where reducing computational cost in the encrypted domain is a key objective. A natural question arises: What are the trade-offs involved in operating on lower-dimensional embeddings, rather than the standard 512-dimensional output produced by models like FaceNet?

Boddeti [32] offered an illustrative example in which 128-dimensional embeddings were reduced to 64 dimensions using PCA, without any noticeable degradation in performance. However, this example was not accompanied by a detailed analysis of the impact of PCA on encrypted computation and performance metrics.

More recently, Engelsma et al. [36] proposed DeepMDS++, an enhanced version of the DeepMDS method introduced by Gong et al. [37]. These methods apply non-linear transformations to reduce dimensionality and are trained specifically for this task. Notably, Gong et al. [37] estimated the intrinsic dimensionality of FaceNet embeddings to be as low as 12 dimensions, which is a striking contrast with the commonly used 128- or 512-dimensional forms.

Given these findings, it is important to understand how a classical linear method like PCA compares to more sophisticated neural network–based approaches such as DeepMDS and DeepMDS++. To explore this, we adopt PCA as our method for dimensionality reduction and assess its effectiveness for preserving performance while reducing the encrypted domain cost.

3.3.1 Principal Component Analysis

We briefly review the theory behind PCA, following the tutorial by Shlens [42], and discuss its relevance to our setting.

Overview of PCA

PCA is a classical statistical technique for identifying the most informative directions in high-dimensional data by projecting it onto a lower-dimensional subspace that preserves maximal variance. Let $X \in \mathbb{R}^{m \times n}$ denote a dataset with m features and n observations. In our application, each column of X corresponds to a 512-dimensional embedding produced by FaceNet, representing the facial features of a single image.

Before applying PCA, we assume that the dataset has been *mean-centered*, i.e., the empirical mean of each feature (row of X) has been subtracted—for example, `scikit-learn`'s PCA implementation [43] centers the data by default—. This ensures that the covariance matrix reflects the variance and covariances around the origin.

The sample covariance matrix is defined as:

$$C_X = \frac{1}{n} X X^\top.$$

The entry C_{ij} measures the covariance between the i-th and j-th features. Diagonal entries represent the variance of individual features, while off-diagonal terms quantify redundancy between different dimensions.

PCA seeks a linear transformation $P \in \mathbb{R}^{m \times m}$ such that the transformed data $Y = PX$ has a diagonal covariance matrix:

$$C_Y = \frac{1}{n} Y Y^\top = \frac{1}{n} P C_X P^\top.$$

The goal is to find a new orthonormal basis in which the features are uncorrelated, and the data's variance is concentrated in as few dimensions as possible.

Since C_X is a real symmetric matrix, it admits an eigendecomposition:

$$C_X = W \Lambda W^\top,$$

where $W \in \mathbb{R}^{m \times m}$ is an orthonormal matrix whose columns are the eigenvectors of C_X, and $\Lambda \in \mathbb{R}^{m \times m}$ is a diagonal matrix containing the corresponding eigenvalues in descending order. Setting $P = W^\top$, we obtain:

$$C_Y = \Lambda.$$

Thus, the principal components are the eigenvectors of the covariance matrix C_X, and the associated eigenvalues represent the variance captured by each component.

To reduce dimensionality, we retain only the top d components, forming a truncated projection matrix $P_d \in \mathbb{R}^{d \times m}$, where $d < m$. The resulting transformed data $Y_d = P_d X$ lies in a d-dimensional subspace that captures the largest portion of the dataset's total variance.

PCA is widely used in practice due to its simplicity, computational efficiency, and ability to decorrelate features. However, it comes with several limitations. One key assumption is that directions of high variance correspond to meaningful structure in the data, which may not always hold true, especially in cases where important features lie in low-variance subspaces. Additionally, PCA captures only linear relationships and is therefore unable to model complex, non-linear patterns that may exist in the data. Its effectiveness can also be influenced by the scale of the input features, making it sensitive to differences in units or magnitudes across dimensions, as well as to the presence of outliers, which can disproportionately affect the direction of the principal components.

Despite these limitations, PCA remains a strong baseline for dimensionality reduction, particularly when efficiency and reduced complexity are essential, as is the case in encrypted computations.

PCA on FaceNet Embeddings

The FaceNet model outputs 512-dimensional embeddings that are L2-normalized, meaning each embedding lies on the unit hypersphere and all features are on a comparable scale. Our goal is to reduce their dimensionality using PCA.

There are two common approaches to PCA preprocessing:

1. **Standardized PCA:** Apply PCA to data that has been standardized to have zero mean and unit variance per feature. This is commonly used when features are on different scales (e.g., height vs. weight).

2. **Raw PCA:** Apply PCA directly to mean-centered data without scaling the variance, preserving the original variance structure across features.

Standardization is helpful when features vary in scale, as it prevents those with larger magnitudes from dominating the principal components. However, since FaceNet embeddings are already L2-normalized, their features are on a similar scale by design.

Therefore, in this thesis we apply PCA directly to the mean-centered FaceNet embeddings without standardization. This approach preserves the inherent variance structure of the embeddings while reducing dimensionality.

How Many Components to Keep?

A central question in applying PCA is how many principal components should be retained to achieve a meaningful reduction in dimensionality without losing critical information. Since PCA orders components by the amount of variance they capture, one natural criterion is to select the smallest number of components that together explain a sufficiently large proportion of the total variance in the data.

A widely used heuristic is to retain enough components to explain between 90% and 95% of the total variance. This approach balances two competing goals: minimizing the dimensionality of the data for computational efficiency, and preserving as much of the original structure as possible.

This heuristic is justified by the observation that in many high-dimensional datasets, especially those with redundancy or noise, most of the meaningful variation is concentrated in the first few ordered components. The remaining components often capture noise or small-scale fluctuations that do not contribute significantly to downstream tasks, such as similarity computation. Thus, discarding them can reduce computational cost and noise sensitivity without compromising overall performance. We discuss more about this heuristic choice in Chapter 6.

In practice, the cumulative explained variance ratio is computed as:

$$\text{ExplainedVariance}(d) = \frac{\sum_{i=1}^{d} \lambda_i}{\sum_{i=1}^{m} \lambda_i},$$

where λ_i denotes the i-th highest eigenvalue of the covariance matrix, and d is the number of components retained and m is the initial dimensionality. A scree plot or cumulative variance plot (also known as the elbow curve) is often used to visualize this trade-off and identify an "elbow point" beyond which additional components yield diminishing returns.

In our setting, selecting the appropriate number of components is particularly important due to the constraints of operating in the encrypted domain, where each additional dimension directly increases the computational cost. This trade-off will be examined quantitatively in the experiments and results presented in Chapter 5 and Chapter 6.

3.4 TFHE Scheme and the `Concrete` Framework

We now describe the homomorphic encryption scheme used in this thesis, TFHE, and its implementation via the `Concrete` framework. The TFHE implementation provided by Zama's `Concrete` framework supports both Boolean and integer arithmetic. The `Concrete` stack simplifies the use of FHE by allowing developers to write FHE programs in Python and compile them using an LLVM-based TFHE compiler.

This compiler abstracts away many of the low-level details of FHE (such as key management, bootstrapping, and encoding), making it easier to develop privacy-preserving applications

However, depending on the use case, understanding these low-level details and their implementation can help identify further optimization opportunities. In the following, we will discuss some background on TFHE and implementation details in the `Concrete` framework.

3.4.1 Overview of the TFHE Scheme

For a good guide to TFHE we refer to the article by Joye [44] and for a detailed treatment of noise growth in TFHE we refer to the article by Klemsa [45].

TFHE is built upon the hardness assumptions of lattice-based cryptography, particularly the Learning With Errors (LWE) problem and its ring variant (RLWE). Security of the scheme ultimately relies on the difficulty of solving certain lattice problems in high dimensions. In particular, the *LWE problem* (recovering a secret vector given noisy inner products with that secret) is believed to be computationally intractable.

There are two important properties of LWE ciphertexts: additive homomorphism and presence of noise. The additive homomorphism property allows addition of two ciphertexts $\bar{c}_1, \bar{c}_2$ encrypting m_1, m_2 respectively. Their summation is calculated as $\bar{c}_1 + \bar{c}_2$ encrypting $m_1 + m_2$. Each ciphertext contains a small amount of noise to ensure security. However, the addition of ciphertexts increases this noise level and if it increases too much, the messages encrypted by the ciphertexts may not be decrypted correctly.

TFHE addresses the issue of noise growth through a process known as bootstrapping. This operation refreshes a ciphertext by reducing its noise level while preserving the underlying encrypted message. Notably, in TFHE, bootstrapping can also homomorphically evaluate a custom Look-Up Table (LUT), enabling the application of arbitrary functions in the encrypted domain. To support efficient bootstrapping, TFHE uses several types of ciphertexts derived from the LWE and RLWE problems. These ciphertexts include built-in structural redundancy, which facilitates bootstrapping but also results in increased ciphertext sizes.

To sum up, TFHE provides two operations: *homomorphic addition* and *bootstrapping*. Importantly, bootstrapping in TFHE inherently supports programmable LUT evaluations, enabling the evaluation of arbitrary functions during the refresh process. These two operations are sufficient to achieve *full homomorphism* in the TFHE scheme—that is, the ability to compute any function over encrypted data.

3.4.2 The `Concrete` Compiler Framework

`Concrete` framework, developed by Zama [3], is a high-level framework for writing TFHE-based programs in Python. It abstracts away low-level implementation and cryptographic details that would otherwise need to be managed manually by the developer. At its core, `Concrete` relies on the `tfhe-rs` library [4], an optimized Rust implementation of the TFHE scheme, also maintained by Zama.

In `Concrete`, there are two types of operations:

- **Linear operations** include basic arithmetic computations such as additions, subtractions, and scalar multiplications (i.e., multiplications by plaintext integers). These operations are computationally efficient, require no bootstrapping, and are directly supported by the additive homomorphism of TFHE ciphertexts.

- **Non-linear operations** include more complex computations, such as comparisons or non-linear functions such as activation functions, and are implemented via LUTs. These require programmable bootstrapping, which not only refreshes the ciphertext noise but also applies a specified function to the encrypted input. As a result, non-linear operations are significantly more computationally intensive.

The functions targeted for encrypted face comparison are similarity/distance functions and they fall into the category of non-linear operations, thus relying on LUTs. A critical parameter influencing the performance of these operations is the *Maximum Bit-Width* (MBW), which determines the size of the LUT associated with the function. The MBW is dictated by the

largest intermediate or final integer value the function encounters over its input domain, and directly impacts the complexity and runtime of the associated bootstrapping procedure.

`Concrete` supports both Boolean and integer datatypes. For unsigned integers, the full bit-width of the LUTs is available for representing values. However, when using signed integers, one bit is reserved for the sign, effectively either increasing the LUT MBW size by one bit or reducing the number of precision bits available. Using signed integers also introduces additional computational overhead during LUT evaluations, such as handling two's complement indexing. Further implementation details can be found in the related source file [46].

In the next section, we will review the similarity/distance functions that could be applied to compare embedding pairs to continue the discussion on the preprocessing pipeline after PCA.

3.5 Similarity Metrics and Distance Functions

After reducing the dimensionality of the FaceNet embeddings, a similarity or distance function is needed to compare them. In this section, we will review similarity and distance functions and in the following section, we will discuss their suitability in the encrypted domain.

3.5.1 Similarity Metrics

In general, a *similarity metric* (or a *similarity measure*) is a function

$$S : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$$

defined over a domain $\mathcal{X}$, which assigns a real-valued score to a pair of objects indicating their degree of resemblance. A similarity metric typically satisfies the following intuitive properties:

- **Symmetry**: $S(x, y) = S(y, x)$, i.e., the similarity score is symmetric with respect to its arguments.
- **Maximum self-similarity**: $S(x, x) \geq S(x, y)$ for all $y \in \mathcal{X}$, meaning an object is maximally similar to itself.

The codomain of similarity scores is often normalized, commonly to the interval $[0, 1]$ or $[-1, 1]$, depending on the specific application. Higher values indicate greater similarity, while lower values correspond to lesser similarity or dissimilarity.

Cosine Similarity

A widely used similarity measure is the *cosine similarity*, which quantifies the similarity between two vectors by measuring the cosine of the angle between them. It is defined as the dot product of the vectors divided by the product of their Euclidean norms.

Given two n-dimensional vectors $\mathbf{a}, \mathbf{b} \in \mathbb{R}^n$, the cosine similarity is expressed as:

$$S_C(\mathbf{a}, \mathbf{b}) := \frac{\mathbf{a} \cdot \mathbf{b}}{\|\mathbf{a}\| \, \|\mathbf{b}\|}$$

where $\|\mathbf{a}\|$ denotes the Euclidean norm of vector $\mathbf{a}$, i.e., $\|\mathbf{a}\| = \sqrt{\sum_{i=1}^{n} a_i^2}$.

The codomain of $S_C(\cdot, \cdot)$ lies in the closed interval $[-1, 1] \subset \mathbb{R}$, where values closer to 1 indicate a higher degree of similarity (i.e., vectors pointing in the same direction), values near 0 indicate orthogonality (no similarity), and values near -1 indicate opposite directions (dissimilarity).

3.5.2 Distance Functions

A *distance function* $d : \mathbb{F} \times \mathbb{F} \to \mathbb{R}_{\geq 0}$ on a set $\mathbb{F}$ is a function that quantifies the dissimilarity between two elements of the set. To be a valid metric, the function $d(\cdot, \cdot)$ must satisfy the following four properties for all $x, y, z \in \mathbb{F}$:

1. **Non-negativity:** $d(x, y) \geq 0$. Distances are always nonnegative.
2. **Identity of indiscernibles:** $d(x, y) = 0 \iff x = y$. Two distinct points must have a strictly positive distance.
3. **Symmetry:** $d(x, y) = d(y, x)$. The order of comparison does not affect the distance.
4. **Triangle inequality:** $d(x, z) \leq d(x, y) + d(y, z)$. The direct distance between two points is less than or equal to the sum of distances via an intermediate point.

Cosine Distance

The *cosine distance* is derived from the cosine similarity measure and quantifies dissimilarity based on the angle between two vectors. Given two non-zero vectors $\mathbf{a}, \mathbf{b} \in \mathbb{R}^n$, the cosine distance is defined as:

$$D_C(\mathbf{a}, \mathbf{b}) := 1 - \frac{\mathbf{a} \cdot \mathbf{b}}{\|\mathbf{a}\| \, \|\mathbf{b}\|},$$

where $\mathbf{a} \cdot \mathbf{b}$ denotes the dot product of the vectors and $\|\mathbf{a}\|$ is the Euclidean (L2) norm of $\mathbf{a}$. The value of D_C lies in the interval $[0, 2]$, where $D_C = 0$ indicates identical direction (maximum similarity), and values closer to 1 indicate orthogonality, while values near 2 represent opposite directions and maximum dissimilarity.

Euclidean Distance

The *Euclidean distance* is the standard metric in $\mathbb{R}^n$, representing the length of the straight line connecting two points. For two vectors $\mathbf{a}, \mathbf{b} \in \mathbb{R}^n$, the Euclidean distance is defined as:

$$D_E(\mathbf{a}, \mathbf{b}) := \|\mathbf{a} - \mathbf{b}\| = \sqrt{\sum_{i=1}^{n} (a_i - b_i)^2}.$$

This metric satisfies all four axioms of a distance function and provides an intuitive notion of spatial separation in Euclidean space.

3.6 Suitability in the Encrypted Domain

In this section, we examine how to select an appropriate similarity or distance function (as introduced in Section 3.5) for use with FHE. This discussion is grounded in the implementation constraints of the `Concrete` framework, as detailed in Section 3.4. Then, we outline the considerations required to prepare the reduced embeddings for use in the encrypted domain.

3.6.1 Choosing Comparison Functions for Encrypted Evaluation

To discuss the computational cost in the encrypted domain, we explore the commonly used vector comparison functions. Specifically, we focus on cosine similarity and Euclidean distance due to their popularity and relative simplicity.

When PCA is applied to reduce dimensionality, the L2-norms of the transformed vectors are not preserved if not all principal components are retained (more details given in Section 3.7.2).

This affects the behavior of norm-sensitive functions such as cosine similarity, and should be taken into account when selecting a comparison function in a privacy-preserving setting.

We now analyze the computational cost of cosine similarity and Euclidean distance in the context of FHE, where operations such as ciphertext–ciphertext multiplication, ciphertext–ciphertext division, and non-linear functions (e.g., square roots) require costly LUT evaluations (through bootstrapping in TFHE).

As a simple exercise, assume two vectors $\mathbf{x}, \mathbf{y} \in \mathbb{Z}^3$ represented with integers. The costs of computing cosine similarity ($S_C(\mathbf{x}, \mathbf{y})$) and Euclidean distance ($D_C(\mathbf{x}, \mathbf{y})$) are summarized in Table 3.1.

Step	**C–C Mults**	**Adds/Subs**	**Other**
Cosine similarity			
L2-norm (for 2 vectors)	6*	4	2* (sqrt)
Dot product	3*	2	0
Norms multiplication + division	1*	0	1* (C-C Div)
Total (cosine)	**10***	**6**	**3***
Euclidean distance			
Component-wise subtraction	0	3	0
Squaring each component	3*	0	0
Sum of squares	0	2	0
Square root	0	0	1*
Total (Euclid)	**3***	**5**	**1***

$*$ Requires a LUT evaluation (non-linear operation).

Table 3.1: Analysis of ciphertext operations for Euclidean distance and Cosine similarity. "C-C Mults" denotes ciphertext-ciphertext multiplication, "C-C Div" refers to ciphertext-ciphertext division, "sqrt" indicates square root operations. Operations requiring LUT evaluation are marked with $*$.

From the basic complexity analysis in Table 3.1, Euclidean distance appears more FHE-friendly than cosine similarity. It involves fewer ciphertext operations and fewer costly LUT-based evaluations, which are typically the main performance bottleneck in homomorphic computation.

Furthermore, computing the *squared* Euclidean distance avoids the expensive square root entirely, making it an attractive option for encrypted similarity comparisons. This simplification is particularly useful when strict adherence to metric properties is not essential, as the squared version violates the triangle inequality and is no longer a true metric.

3.6.2 Impact of Signedness and Quantization

The `Concrete` framework operates on encrypted booleans and integers, whereas the reduced FaceNet embeddings consist of floating-point vectors, which may include both positive and negative values. To perform homomorphic computations on these embeddings, a set of preprocessing steps is required to convert them into integer-valued representations prior to encryption.

This conversion process involves two critical considerations: (i) how to handle the signedness of the data, and (ii) how to quantize the floating-point values into discrete integers without excessively distorting their structure.

(i) Signed vs. Unsigned Encoding.

Since `Concrete` supports both signed and unsigned integer types, a design choice must be made regarding the preservation of negative values. One option is to retain the full value range by using signed integers. However, using signed types consumes one bit for the sign, thereby reducing the effective precision. It also increases the complexity of homomorphic operations through LUTs, which are needed for evaluating non-linear functions. An alternative approach is to transform the data into a nonnegative domain using *min-max normalization*. This can be applied either globally (across the entire dataset) or independently per-feature dimension. The choice of normalization method can influence the geometric relationships between embeddings. For instance, per-feature min-max normalization may distort angular relationships more than global min-max normalization. Thus, for min-max normalization the following two strategies will be considered:

Global min-max normalization. A single minimum m and maximum M are computed over the entire reduced embedding dataset matrix $X \in \mathbb{R}^{d \times n}$, where d is the feature dimension and n is the number of embeddings. Each value is scaled as:

$$x_{ij}^{\text{scaled}} = \frac{x_{ij} - m}{M - m}, \quad \text{where } m = \min(X),\ M = \max(X)$$

Per-feature min-max normalization. Each feature dimension i is scaled independently using its own minimum and maximum values:

$$x_{ij}^{\text{scaled}} = \frac{x_{ij} - m_i}{M_i - m_i}, \quad \text{where } m_i = \min_j(x_{ij}),\ M_i = \max_j(x_{ij})$$

(ii) Quantization and Rounding Effects.

Once the data has been scaled (or left as-is in the signed case), it must be discretized into integers via *quantization*. This process typically involves multiplying by a scaling factor and rounding to the nearest integer. As with min-max normalization, quantization can be applied in two ways: either using a single global quantization factor for the entire embedding vector, or applying a unique scaling factor to each feature dimension independently. The global approach preserves the overall structure of the vector space, while per-dimension quantization may provide finer control over the precision of individual features, but potentially at the cost of distorting inter-feature relationships. We therefore consider the following two strategies:

Global quantization. A single global scaling factor $s \in \mathbb{R}$ is applied to all values:

$$x_{ij}^{\text{int}} = \left\lfloor s \cdot x_{ij}^{\text{float}} \right\rceil$$

Here, s is chosen such that the output fits within the target integer range (e.g., target bit-width b):

$$s = \frac{2^b - 1}{M - m}, \quad \text{where } m = \min(X),\ M = \max(X)$$

Per-dimension quantization. Each feature dimension is assigned its own scaling factor s_i, allowing finer control of quantization precision:

$$x_{ij}^{\text{int}} = \left\lfloor s_i \cdot x_{ij}^{\text{float}} \right\rceil$$

Similarly, s_i is chosen such that the output fits within the target integer range (e.g., target bit-width b) for the given feature dimension i:

$$s_i = \frac{2^b - 1}{M_i - m_i}, \quad \text{where } m_i = \min_j(x_{ij}),\ M_i = \max_j(x_{ij})$$

Stage	Strategy	Summary
(i) Min-max normalization	Global	Scales all values using a single global min and max, preserves overall structure but may waste dynamic range for some features.
	Per-feature	Each feature is scaled independently, improves range utilization but may distort inter-feature relationships.
(ii) Quantization	Global	Uses a single scaling factor across all dimensions, simple and consistent but may under-represent low-variance features.
	Per-dimension	Assigns a custom scaling factor to each feature, improves precision per dimension but may affect similarity/distance metrics.

Table 3.2: Summary of design choices for min-max normalization and quantization.

3.7 Analysis of Preprocessing Operations

In this section, we construct the preprocessing pipeline and analyze how it affects the relative ordering of embeddings. Specifically, we propose the following preprocessing steps on 512-dimensional embeddings extracted by FaceNet.

1. Apply PCA to reduce the dimensionality to d, where $0 < d < 512$.
2. Apply min-max normalization.
3. Apply quantization to convert float-valued embeddings to integer-valued ones.

We now examine how each step affects the relative ranking of embedding pairs, as defined below.

3.7.1 Ranking the LFW Pairs

Given two m-dimensional FaceNet embeddings $\mathbf{x}, \mathbf{y} \in \mathbb{R}^m$, we score their similarity with the squared Euclidean distance

$$f(\mathbf{x}, \mathbf{y}) \ = \ \|\mathbf{x} - \mathbf{y}\|^2,$$

which is FHE-friendly and assigns smaller values to same-identity pairs. Where $\|\cdot\|$ denotes the L2-norm.

Let $X \in \mathbb{R}^{m \times n}$ be the full dataset, with each column vector as embeddings $\mathbf{x}_1, \ldots, \mathbf{x}_n$; in our case $m = 512$ since we use 512-dimensional FaceNet embeddings. We evaluate $f(\mathbf{x}_i, \mathbf{x}_j)$ for every pair, rank the resulting scores, and—using the ground-truth labels in `pairs.txt` protocol—select a threshold that best separates the two classes (e.g., Youden's J statistic in Appendix B.2.1).

Applying this procedure to rank all the pairs on the original $m = 512$-dimensional embeddings yields a *clear-domain baseline ranking*. In the rest of the section, we try to get an intuition on

how each preprocessing step could alter this baseline ranking. This relative ranking is closely tied to the final matching performance: any deviation introduced by preprocessing may shift some embeddings across the decision threshold, affecting classification.

3.7.2 Effect of PCA

As previously discussed, applying PCA involves constructing a projection matrix $P \in \mathbb{R}^{d\times 512}$, where d denotes the number of principal components to retain and 512 is the initial dimensionality of the embeddings. This matrix projects the original dataset $X \in \mathbb{R}^{512\times n}$ onto a lower-dimensional subspace defined by the rows of P. The resulting projection is:

$$X' = PX \in \mathbb{R}^{d\times n}$$

Here, each column vector $\mathbf{x}_i \in \mathbb{R}^{512}$ of X, for $1 \leq i \leq n$, is transformed into its lower-dimensional representation $\mathbf{x}'_i = P\mathbf{x}_i \in \mathbb{R}^d$.

We now examine how the choice of d impacts the relative ordering of pairwise distances between embeddings when measured using the squared Euclidean distance:

$$f(\mathbf{x}, \mathbf{y}) = \|\mathbf{x} - \mathbf{y}\|^2 = \sum_{i=1}^{d} (x_i - y_i)^2$$

Consider two embeddings from the dataset $\mathbf{x}, \mathbf{y} \in \mathbb{R}^{512}$. If the full dimensionality is retained (i.e., $d = 512$), then $P \in \mathbb{R}^{512\times 512}$ is an orthonormal matrix (i.e., $P^\top P = I_{512}$). The projected vectors are $\mathbf{x}' = P\mathbf{x}$ and $\mathbf{y}' = P\mathbf{y}$, and their squared distance becomes:

$$\begin{aligned} f(\mathbf{x}', \mathbf{y}') &= \|P(\mathbf{x} - \mathbf{y})\|^2 \\ &= (P(\mathbf{x} - \mathbf{y}))^\top (P(\mathbf{x} - \mathbf{y})) \\ &= (\mathbf{x} - \mathbf{y})^\top P^\top P(\mathbf{x} - \mathbf{y}) \\ &= (\mathbf{x} - \mathbf{y})^\top (\mathbf{x} - \mathbf{y}) \\ &= f(\mathbf{x}, \mathbf{y}) \end{aligned}$$

This shows that when $d = 512$, PCA preserves all pairwise distances exactly. As d decreases, this exact preservation no longer holds, as we are in the situation where:

$$P^\top P \neq I_{512}$$

To the extent to which rankings are preserved depends on how much variance the top k components capture. If high proportion of the variance is retained, the approximation error will be minimal.

$$f(\mathbf{x}', \mathbf{y}') \approx f(\mathbf{x}, \mathbf{y})$$

This indicates that the ranking is not guaranteed to stay the same, however, the expected effect of PCA should be minimal if the retained components capture a sufficient proportion of the dataset variance.

Another observation is that, the L2-norm of FaceNet embeddings are not preserved after PCA when fewer than all components are retained since the projection discards part of the original vector's variance, effectively shortening its length in the lower-dimensional space

3.7.3 Effect of Min-Max Normalization

After applying PCA, the next step is min-max normalization. Let the reduced dataset be $X' \in \mathbb{R}^{d\times n}$, where d is the reduced dimensionality with $0 < d < 512$ and n is the number of embeddings. Consider two embedding vectors from the reduced dataset, $\mathbf{x}, \mathbf{y} \in \mathbb{R}^d$.

Applying per-feature min-max normalization yields normalized vectors $\mathbf{x}', \mathbf{y}' \in \mathbb{R}^d_{\geq 0}$, where each feature is scaled according to its individual minimum and maximum. The squared Euclidean distance between the normalized vectors becomes:

$$f(\mathbf{x}', \mathbf{y}') = \sum_{i=1}^{d} \left(\frac{x_i - y_i}{M_i - m_i} \right)^2 \quad \text{where } m_i = \min_j(x_{ij}), \quad M_i = \max_j(x_{ij})$$

Since the feature ranges $(M_i - m_i)$ may vary across dimensions, this transformation introduces a feature-wise scaling effect that can distort the geometry of the space. As a result, per-feature min-max normalization does not preserve the original distances:

$$f(\mathbf{x}', \mathbf{y}') \neq f(\mathbf{x}, \mathbf{y})$$

which implies that the relative ordering of pairwise distances may also change.

By contrast, applying global min-max normalization computes a single minimum m and maximum M over the entire dataset, and uses the same scaling factor $(M - m)$ across all features. In this case, the normalized vectors are:

$$x'_i = \frac{x_i - m}{M - m}, \quad y'_i = \frac{y_i - m}{M - m} \quad \text{for all } i = 1, \ldots, d$$

and the squared Euclidean distance becomes:

$$f(\mathbf{x}', \mathbf{y}') = \frac{1}{(M - m)^2} \sum_{i=1}^{d} (x_i - y_i)^2$$

Since all distances are scaled uniformly by a positive constant, the relative ordering between embedding pairs is preserved:

$$\text{If } f(\mathbf{x}_a, \mathbf{y}_a) < f(\mathbf{x}_b, \mathbf{y}_b) \quad \Rightarrow \quad f(\mathbf{x}'_a, \mathbf{y}'_a) < f(\mathbf{x}'_b, \mathbf{y}'_b)$$

Here, each pair $(\mathbf{x}_a, \mathbf{y}_a)$ and $(\mathbf{x}_b, \mathbf{y}_b)$ corresponds to a pair of embeddings selected from the dataset. The vectors $\mathbf{x}'_a, \mathbf{y}'_a$ and $\mathbf{x}'_b, \mathbf{y}'_b$ denote their respective normalized versions obtained through global min-max normalization.

3.7.4 Effect of Quantization

With the dataset $X' \in \mathbb{R}^{d \times n}_{\geq 0}$ already scaled to a nonnegative range (via min-max normalization), the next step is to convert floating-point values into integers to enable encrypted computation in the `Concrete` framework, which operates natively on integers.

As with normalization, float-to-integer quantization can be applied using either a single global scaling factor or per-dimension scaling factors.

Per-dimension quantization. Each feature dimension is scaled independently using a dimension-specific scaling factor s_i for a target bit-width b. It is defined as:

$$s_i = \frac{2^b - 1}{M_i - m_i}, \quad \text{for } i = 1, \ldots, d \quad \text{where } m_i = \min_j(x_{ij}), \quad M_i = \max_j(x_{ij})$$

The quantized embedding $\mathbf{x}^{\text{int}} \in \mathbb{Z}^d$ is then computed elementwise as:

$$x_i^{\text{int}} = \lfloor s_i \cdot x_i \rceil$$

This approach maximizes the use of the available integer range in each dimension, improving precision locally. However, since each feature is scaled independently, the inter-feature ratios are not preserved, which might influence the relative ranking of the embedding pair.

Global quantization. In this case, a single global scaling factor s is applied across all features:

$$s = \frac{2^b - 1}{M - m}, \quad \text{where } M = \max(X), \quad m = \min(X)$$

The entire embeddings are quantized uniformly:

$$x_i^{\text{int}} = \lfloor s \cdot x_i \rceil, \quad \text{for all } i = 1, \ldots, d$$

This preserves the relative magnitudes across dimensions, maintaining the original geometry of the vector space up to a global scale. Therefore the quantized version preserves the relative ordering between embedding pairs. However, global quantization may underutilize the integer range in low-variance dimensions, leading to higher rounding error in those components.

3.7.5 Summary of Effects

As summarized in Table 3.3, each step in the preprocessing pipeline introduces potential distortions that may affect the relative ranking of embedding pairs. Our goal is to empirically evaluate the significance of these effects in practice in Chapter 5 and Chapter 6.

Step	**Purpose**	**Effect on Pairwise Ranking / Distortion Source**
PCA	Reduce dimensionality	If all components are retained, ranking is preserved. When $d < 512$, distortion depends on the amount of retained variance.
Min-max normalization	Scale values to a non-negative range	Relative distances may shift, especially with per-feature normalization due to inconsistent scaling across features. Global min-max preserves rankings, as all dimensions are scaled uniformly.
Float-to-integer quantization	Convert to integer representation for FHE compatibility	Per-dimension quantization improves local precision but may distort similarity relationships. Global quantization maintains pairwise ranking consistency but may sacrifice fine-grained precision in certain low-variance features due to rounding errors.

Table 3.3: Summary of preprocessing steps and their effects on relative embedding rankings.

Chapter 4

Client-Server Architecture

Contents

In this chapter, we describe the client-server architecture used in this thesis. We detail the enrollment and authentication steps in an encrypted setting using FHE. Finally, to demonstrate the integration of FHE with other cryptographic tools, we present an example communication scenario between the client and the server. This scenario is not accompanied by a formal security analysis, but serves to assess the practicality of such an integration in the following chapters.

4.1 Motivation for a Client-Server Architecture

The client-server architecture is a common paradigm in practical systems, particularly for applications involving biometric authentication. In the context of FHE, this architecture must be reconsidered, as the server is unable to access or interpret any data in plaintext. Although this introduces limitations in conventional client-server settings, it provides an opportunity in privacy-focused biometric applications.

By leveraging FHE, it becomes possible to offload both computation and storage to a cloud-based server while ensuring that biometric data remains encrypted at all times. This offers strong privacy guarantees and is especially attractive in high-security scenarios where data leakage or misuse must be avoided.

An example of such a setting is biometric access control in secure buildings, where personal information must not be stored on the premises. In this setting, tamper-proof devices can be used on-site to collect and process sensitive data locally, while all other computations are performed in encrypted domain on the server.

As illustrated in Figure 4.1, a camera sensor at the enterance to the building collects an image of the user along with a claimed ID. The initial preprocessing steps, such as face detection and feature extraction, are performed locally in plaintext. The resulting face embedding is then encrypted using the public FHE key and sent to the server.

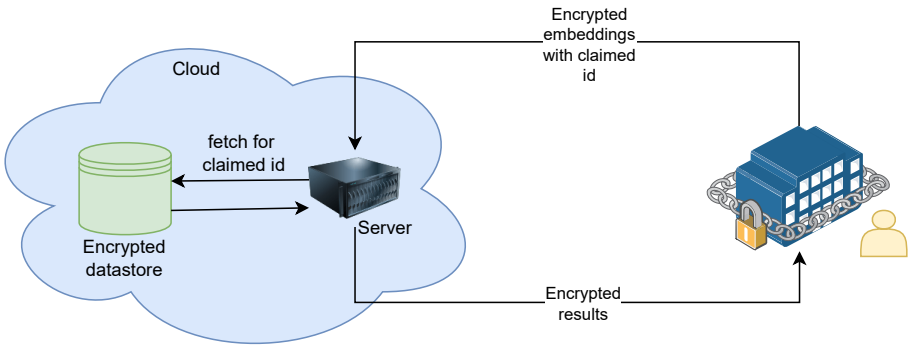

Figure 4.1: Biometric access example to a secure building using encrypted computation in the cloud.

On the server, the encrypted query embedding is compared against an encrypted database of stored embeddings, using a matching algorithm operating entirely in the encrypted domain. The server returns an encrypted similarity score or match result, which is then decrypted by the on-site device to make the final access decision. This setup ensures that sensitive biometric data are never exposed in plaintext, neither during transmission nor in storage or computation.

It is also straightforward to extend the concept illustrated in Figure 4.1 to a per-client setting. In such a scenario, each client maintains their own cryptographic key pair and shares their public key with the server during the initial registration or setup phase. For example, this approach is particularly suitable for access control systems within a corporate infrastructure, where each employee is assigned a dedicated company computer. Before gaining access, the employee would need to authenticate using a biometric method, after which the computer verifies the authentication result, decrypts the session data on the computer, and grants or denies access accordingly.

Another interesting application of the per-client setup is in enhancing multifactor authentication for high-security operations. For example, banking applications could incorporate FHE-based biometric verification as an additional step before authorizing large transactions or digitally signing sensitive documents. This added layer of security could significantly reduce the risk of unauthorized actions.

These examples illustrate the practical motivation for adopting a client-server architecture that leverages FHE for privacy-preserving biometric authentication. In the following, the system we describe is designed to support one-to-one matching, where each user is identified by a unique ID. To enable practical deployment of such a system, it is necessary to define clear enrollment and authentication protocols that govern the interaction between the client and the server.

4.2 Core Protocol Functionalities

This section outlines the two core functionalities of the system: enrollment and authentication. Since the system performs one-to-one matching, the server must maintain a record of individual clients. This is achieved by having the server generate random client IDs, which are then assigned to clients for future interactions. The enrollment and authentication protocols are described below at a high level.

We assume that all communication between the client and server takes place over an authenticated and encrypted channel, such as using Transport Layer Security (TLS). Furthermore, in practical face recognition systems, it is essential to incorporate a liveness detection step prior to image capture. This is a critical safeguard against spoofing attacks using photographs or

video recordings. Although the implementation of liveness detection falls outside the scope of this thesis, we assume its presence as part of the overall system setup.

4.2.1 Enrollment

During the enrollment phase (Figure 4.2), the client requests a unique identifier from the server. Once a unique ID is obtained, the client collects facial images from the user and applies the preprocessing steps described in Chapter 3. These steps produce feature embeddings that are ready for encryption. The client then generates an FHE keys (public, private, evaluation keys), retaining the private key locally and sending the evaluation key to the server. The client encrypts the preprocessed embeddings using its public FHE key and transmits them to the server, along with the assigned ID. The server stores the encrypted embeddings in its database indexed by the client's ID.

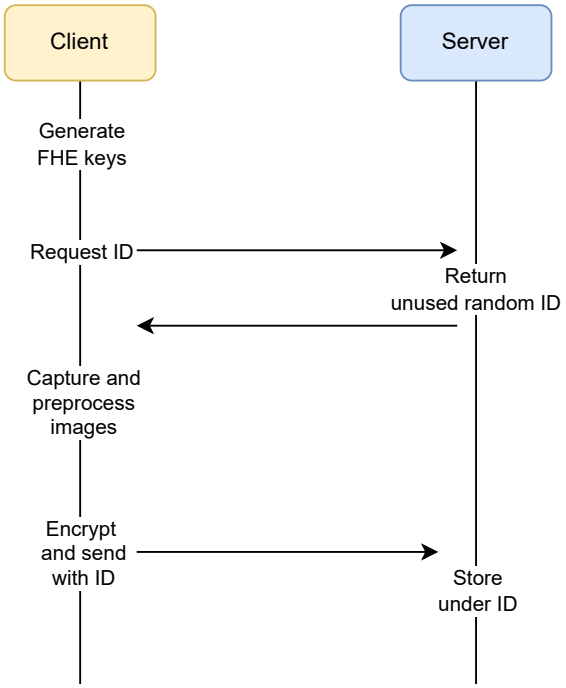

Figure 4.2: Client-server enrollment flow using FHE.

4.2.2 Authentication

In the authentication phase (Figure 4.3), the client captures a new facial image and applies the same preprocessing steps as during enrollment. The resulting embeddings are encrypted with the client's public FHE key and sent to the server, along with the user's claimed ID.

Upon receiving the encrypted query and the claimed ID, the server retrieves the associated stored encrypted embeddings. It then performs the encrypted-domain computations, such as squared Euclidean distance, to compare the query and the stored data. Since all the data remain encrypted, the server learns nothing about the actual values. Finally, the server returns the encrypted comparison result to the client. The client decrypts this result using its private key and verifies whether the score satisfies a predefined threshold for successful authentication.

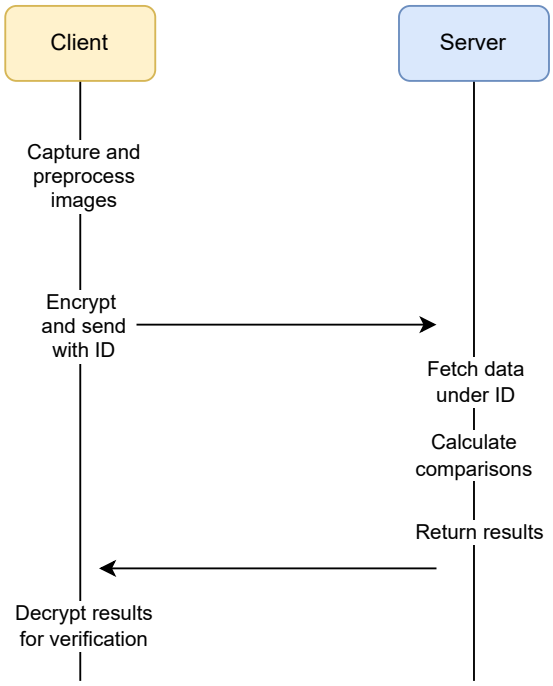

Figure 4.3: Client-server authentication flow using FHE.

4.3 Communication Scenario

In the previous section, we outlined the high-level steps involved in the enrollment and authentication procedures. Building on this, we now describe a practical communication scenario that illustrates how FHE can be integrated alongside traditional cryptographic tools. The aim of this section is to demonstrate, through a concrete example, how FHE can be employed in conjunction with other mechanisms for a client-server interaction.

It is important to note that this scenario is presented for illustrative purposes only and is not accompanied by a formal security analysis. While it borrows from well-established cryptographic practices, we make no claims regarding its security under formal adversarial models. Instead, the focus is on examining practical deployment considerations and how FHE fits into the wider cryptographic landscape. We assume that communication between the client and the server takes place over an authenticated channel, such as one established using TLS.

To ground the discussion in a concrete use case, consider a secure banking application. In a typical scenario, users authorize a transaction using a password or PIN. This system extends the authentication step with a privacy-preserving face verification mechanism using FHE. The user captures their facial image locally, encrypts the corresponding feature vector, and initiates a biometric verification protocol with the server.

The remainder of this section describes the cryptographic building blocks and explains how they are combined to support this interaction model.

4.3.1 Underlying Cryptographic Tools

Elliptic Curve Cryptography (ECC)

ECC is an approach to public key cryptography that offers strong security guarantees with relatively small key sizes, making it well-suited for resource-constrained environments such as mobile or embedded systems. Compared to traditional schemes like RSA, ECC achieves equivalent security with significantly less computational overhead and bandwidth consumption, which is particularly beneficial in client-server scenarios involving biometric authentication.

In this work, we adopt the widely used elliptic curve `secp256k1`. This curve is defined over a 256-bit prime field and is notable for its efficient arithmetic and widespread adoption in practice, most famously in Bitcoin. It is a Koblitz curve defined by the equation $y^2 = x^3 + 7 \mod p$, where p is the large prime and its security is based on the hardness of the Elliptic Curve Discrete Logarithm Problem (ECDLP).

Elliptic Curve Diffie-Hellman Key Exchange (ECDH)

ECDH is a key exchange protocol that allows two parties to establish a shared secret over an insecure channel using ECC. Each party selects a private key as a random scalar from a finite field and computes the corresponding public key by multiplying the scalar with a publicly agreed-upon generator point on the elliptic curve. They then exchange their public keys and compute the shared key by multiplying each other's public keys with their secret keys, as illustrated in Figure 4.4 with details in Table 4.1. This shared point can then be used as input to a key derivation function to obtain a symmetric key for further communication.

Symbol	Description	Secret or Public?
$\mathcal{E}$	Elliptic curve defined over a finite field $\mathbb{F}_p$	Public
G	Base point (generator) on the curve $\mathcal{E}$	Public
n	Order of the base point G	Public
d_A	Alice's private key, a random integer in $[1, n-1]$	**Secret**
d_B	Bob's private key, a random integer in $[1, n-1]$	**Secret**

Table 4.1: Public and secret parameters in the ECDH key exchange protocol.

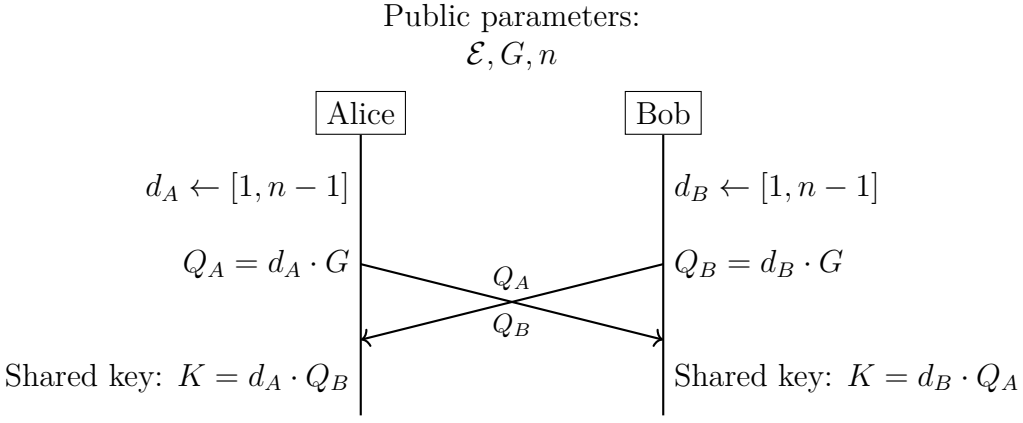

Figure 4.4: ECDH key exchange.

Key Derivation Function (KDF)

KDF is an essential component of cryptographic systems: Its goal is to take a source of initial keying material, usually containing some good amount of randomness, but not distributed uniformly or for which an attacker has some partial knowledge, and derive from it one or more cryptographically strong secret keys. The result of the Diffie-Hellman key exchange is a shared secret K, known only to the two participating parties. While this value is computationally secure, it may not be uniformly random and may not meet the input requirements of certain algorithms. To transform this raw shared secret into one or more high-entropy cryptographic keys suitable for further use, a KDF is employed.

A widely adopted choice is the HMAC-based Key Derivation Function (HKDF) from Krawczyk [47], standardized in RFC 5869 [48]. HKDF is designed to extract uniform randomness from the shared secret and expand it into keys of arbitrary length.

It operates in two stages:

- **Extract:** A pseudorandom key (PRK) is derived from the shared secret and an optional non-secret salt using HMAC.
- **Expand:** The PRK is then used to generate output key material.

This separation of extraction and expansion phases ensures that even if the original shared secret has some structure or bias, the derived keys are cryptographically sound.

Hash-based Message Authentication Code (HMAC)

The HMAC is a cryptographic construction used to verify both the integrity and authenticity of a message by combining a cryptographic hash function (such as SHA-256) with a secret key. It is defined as $\text{HMAC}(K, m) = H((K' \oplus \text{opad}) \,\|\, H((K' \oplus \text{ipad}) \,\|\, m))$, where K is the secret key, m is the message, H is the hash function, and ipad and opad are fixed inner and outer padding constants. HMAC provides strong security guarantees: it ensures message integrity (any modification of m results in a different HMAC) , authenticity (only parties with the secret key K can produce a valid HMAC), and resistance to collision and forgery. In this work, HMAC is employed to allow a client to prove possession of a derived symmetric key—obtained via ECDH and HKDF—by computing an HMAC over a challenge payload (namely, nonce, timestamp, request ID), thereby ensuring both authenticity and freshness of client requests.

Σ-Protocols

Σ-protocols are a family of efficient, 3-move, and honest-verifier zero-knowledge protocols. One of the most well-known examples is Schnorr's zero-knowledge identification protocol. In the following we provide an overview about the protocol from the course LMAT2450 [49].

Schnorr's Identification Protocol The goal is for the prover to demonstrate knowledge of a secret x such that $h = x\,G$, without revealing x. The interaction proceeds as in Figure 4.5 with details given in Table 4.2.

The Schnorr identification protocol in the elliptic curve setting also satisfies *special soundness* and *honest-verifier zero-knowledge*. Special soundness states that if a malicious prover can respond correctly to two different challenges e and e' using the same initial commitment $R = r\,G$, then the verifier can extract the secret x. Given two valid responses $s = r + e \cdot x \mod n$ and $s' = r + e' \cdot x \mod n$, subtracting the equations gives $s - s' = (e - e') \cdot x \mod n$, which allows computing the secret as $x = \frac{s-s'}{e-e'} \mod n$. This theoretical extraction, possible in simulation via rewinding, forms the basis for soundness. Moreover, the protocol is honest-verifier zero-knowledge: if the verifier chooses the challenge e uniformly at random and behaves honestly,

Symbol	Description	Secret or Public?
$\mathcal{E}$	Elliptic curve defined over a finite field $\mathbb{F}_p$	Public
G	Generator point on the elliptic curve $\mathcal{E}$	Public
n	Order of the point G	Public
X	Prover's public key, computed as $X = x \cdot G$	Public
x	Prover's private key, a random integer in $[1, n-1]$	**Secret**

Table 4.2: Public and secret parameters in the EC-Schnorr identification protocol.

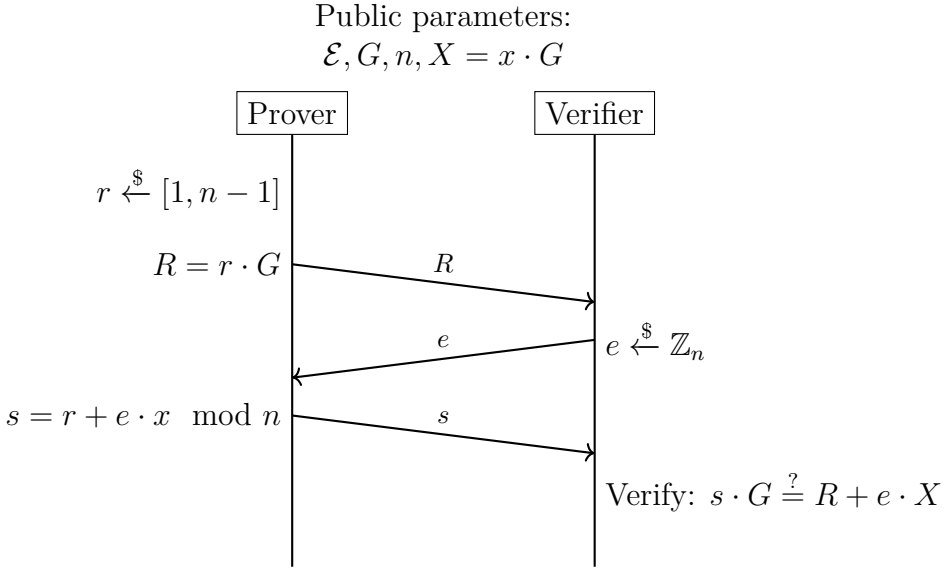

Figure 4.5: Schnorr Zero-Knowledge Identification Protocol

then a simulator who only knows the public key $P = x\,G$ (but not the secret x) can simulate a transcript (R, e, s) that is indistinguishable from one generated in a real protocol execution. Therefore, the verifier learns nothing about x beyond the fact that the prover possesses it.

4.3.2 Scenario Construction Using the Tools

To demonstrate how the primitives introduced earlier can be combined into a full communication workflow, we present in Figure 4.6 a protocol that supports both the enrolment and the authentication of a client.

The protocol has four distinct phases:

- **Client Introduce.** The client and server perform an ECDH exchange on the `secp256k1` curve to derive a shared secret. This secret is then expanded using HKDF into a session key K, which is then bound to a client ID. The server also records (ID, Q_c) for later use.
- **Schnorr Identification.** To prove possession of the private key corresponding to the stored public key Q_c under client ID, the client engages in Schnorr's zero-knowledge identification protocol. Upon successful verification of the proof, the server generates a challenge message by concatenating a freshly generated nonce and timestamp. This challenge is stored and sent to the client, bound to the client's ID.
- **Client Enroll.** To enroll to the server, the client transmits its encrypted face embeddings $\mathbf{E}$ and the corresponding TFHE evaluation keys $\mathbf{K_e}$, and a confirmation in the form of an HMAC over the challenge message received from the Schnorr identification concatenated with the user ID. The server verifies the HMAC confirmation, then stores $\{\mathbf{E}, \mathbf{K_e}\}$ under the client's ID.
- **Client Authenticate.** For each authentication attempt, the client repeats the Schnorr identification to obtain a fresh challenge message then concatenates its ID and computes the HMAC over it. It then sends new encrypted face embeddings $\mathbf{E}^*$ and the HMAC confirmation. The server verifies the confirmation, homomorphically compares $\mathbf{E}$ and $\mathbf{E}^*$ using its stored TFHE evaluation keys under client's ID, and returns the result of the comparison.

The full implementation of the communication scenario is available in the thesis GitHub repository. [50]

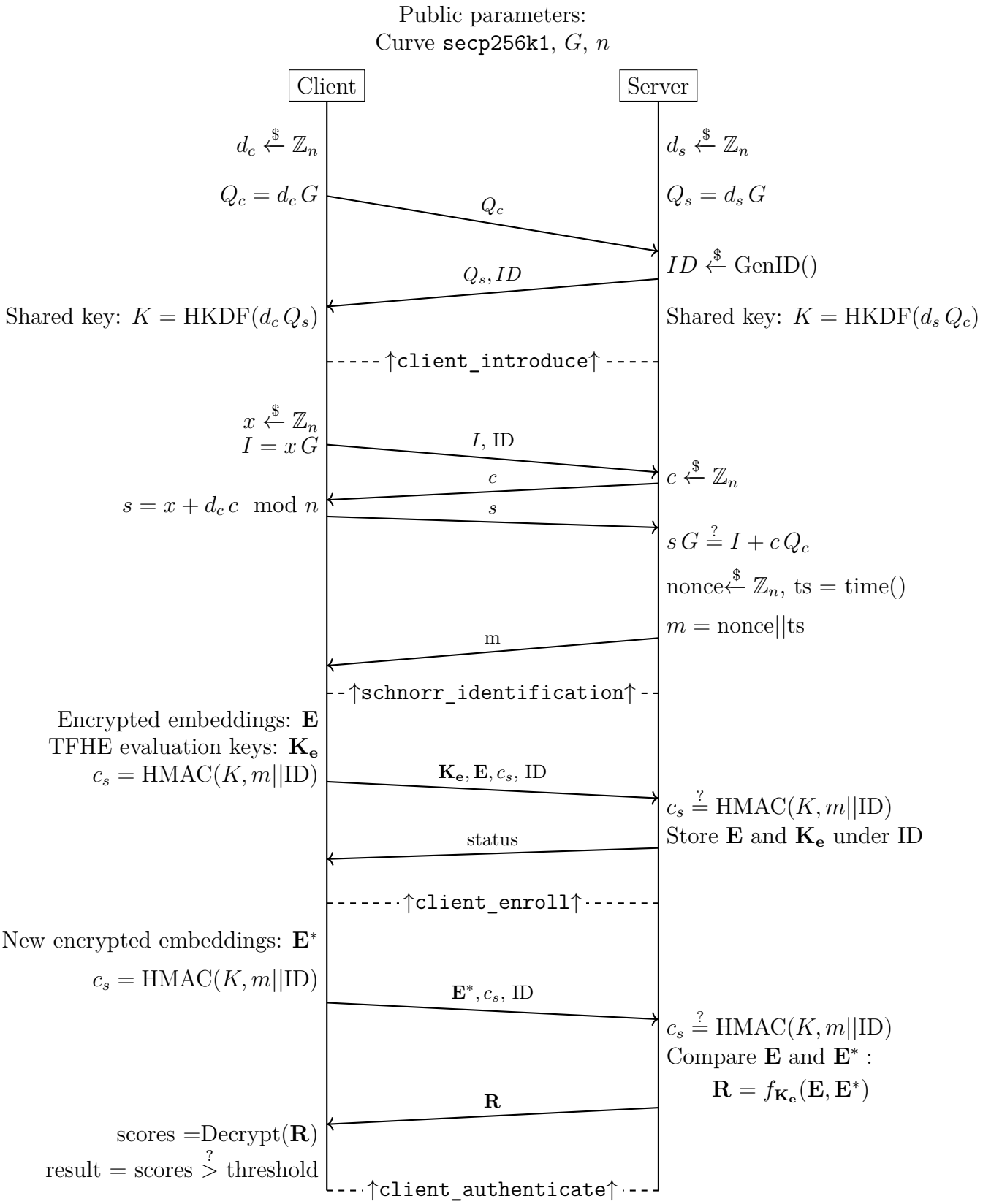

Figure 4.6: Overview of the communication scenario. The scenario integrates ECDH, Schnorr identification, HMAC-based challenge confirmation, and homomorphic encrypted face embedding comparison using TFHE. Each of the four phases—`client_introduce`, `schnorr_identification`, `client_enroll`, and `client_authenticate`—is marked at the point where it completes. Before each `client_enroll` and `client_authenticate`, the client performs `shnorr_identification`. The diagram illustrates both message exchanges and local computations performed by the client and server.

4.4 Privacy Gains of the Proposed Scenario

In the context of biometric data, the overarching goal of privacy-preserving techniques is to enable essential computations while ensuring that sensitive data remains inaccessible to unauthorized entities. As noted in the comprehensive survey by Zhang et al. [18], performing biometric matching on plaintext data, especially on platforms that may not be fully trusted, poses a significant long-term risk to the integrity and privacy of biometric authentication systems.

In the communication scenario presented above, it is evident, even in the absence of a formal security proof, that no sensitive information is exposed to the server. The client's biometric data is transformed into preprocessed embeddings, which are then encrypted using the TFHE scheme. This ensures that the server interacts solely with encrypted representations and never gains access to raw or intermediate biometric data.

This privacy-by-design approach demonstrates how homomorphic encryption can be used to construct biometric authentication systems. When combined with complementary cryptographic tools, these systems can be implemented entirely within the encrypted domain. If supported by formal security analysis, such frameworks have the potential to meaningfully reduce the privacy risks inherent in biometric recognition technologies.

Chapter 5

Experiments of the Preprocessing Pipeline

Contents

In this chapter, we present the experiments carried out to evaluate the proposed preprocessing pipeline and overall system. In the following chapter, we provide the results of these experiments and discuss them.

The experiments begin by analyzing the impact of each preprocessing stage in the clear (plaintext) domain, highlighting how these steps influence face matching performance. Next, we assess how the same preprocessing operations behave in the encrypted domain, focusing on their implications for computational cost and matching performance. Finally, we replicate key experimental setups from prior work to enable meaningful comparisons and to evaluate the effectiveness of our approach under similar conditions.

5.1 Evaluation in the Plaintext Domain

5.1.1 Baseline Setup and Evaluation Metrics

We begin by establishing the baseline performance of the 512-dimensional FaceNet embeddings, evaluated in the clear domain without any preprocessing.

Performance benchmarking is conducted on the LFW dataset using its standard `pairs.txt` protocol, which defines a 10-fold cross-validation split. We use the `scikit-learn`'s utility[1] to fetch the dataset and its associated partitioning.

To assess biometric verification performance, we use the standard metrics of True Positive Rate (TPR) and False Positive Rate (FPR). TPR measures the proportion of genuine users correctly accepted by the system, while FPR quantifies the proportion of impostors incorrectly accepted. These metrics are computed based on a decision threshold applied to the similarity or distance scores between image pairs. In biometric systems, reporting TPR at fixed FPR levels is especially informative, as it reflects the system reliability under varying security constraints. Specifically, we report TPR at FPR thresholds of 0.01%, 0.1%, and 1%. This evaluation is particularly relevant for high-security applications, where minimizing FPR is critical. For example, TPR@0.01% denotes the proportion of genuine matches correctly accepted when the system is tuned to allow only 0.01% false positives. Details on the computation of these metrics and the 10-fold cross-validation protocol can be found in the Appendix B.

Additionally, in the encrypted domain, we evaluate the computational efficiency of the system by measuring the execution time required for one-to-one embedding comparisons, as well as the memory overhead associated with storing encrypted embeddings and evaluation keys. All evaluations—both in the clear and encrypted domains—are conducted using the squared Euclidean distance as the comparison function.

5.1.2 Principal Component Analysis

The first step of the preprocessing pipeline is the application of PCA. In this section, we design an experiment to address the question raised in Chapter 3: How many principal components do we need?

Variance Retention: Elbow Curve and Scree Plot

A common heuristic in dimensionality reduction is to retain enough principal components to capture at least 90% to 95% of the total variance. This threshold offers a practical trade-off between preserving the most informative features and reducing model complexity. In our case, this approach is particularly suitable, as the face images are first aligned and cropped using MTCNN, which effectively removes background variation and concentrates the remaining variance on facial features.

To identify the optimal number of components, we use both the elbow curve and the scree plot. The elbow curve helps us visually determine the point at which the addition of further components no longer significantly improves the cumulative explained variance. In other words, it marks the point where the marginal gain in the explained variance becomes negligible. Similarly, the scree plot illustrates the individual contribution of each principal component, making it easier to pinpoint the components that contribute the most to explaining the variance, and where the explained variance diminishes.

By analyzing these plots, we can effectively determine the number of components that could be retained, ensuring a meaningful reduction in dimensionality without sacrificing important information.

Impact of Dimensionality Reduction on Matching Performance

The previous experiment on PCA gives us a component range on which we can test further. After determining this range for evaluation, the next step is to evaluate performance using

[1]Available at `https://scikit-learn.org/stable/modules/generated/sklearn.datasets.fetch_lfw_pairs.html`

embeddings in the clear. Concretely, we apply PCA to reduce the 512-dimensional FaceNet embeddings to the selected dimension and calculate the performance metrics.

5.1.3 Min-Max Normalization Strategies

In the second step of the pipeline, we choose to work with nonnegative embeddings. Therefore, we use min-max normalization to transform the data into a nonnegative range. As described in Chapter 3 there are two methods to perform min-max normalization: global and per-feature. They are shown in Figure 5.1.

We perform the PCA to reduce the dimensionality of the embeddings to a selected dimension (dimensions keeping 90 to 95% of the explained variance) and apply the selected min-max normalization strategy on reduced embeddings and report performance metrics.

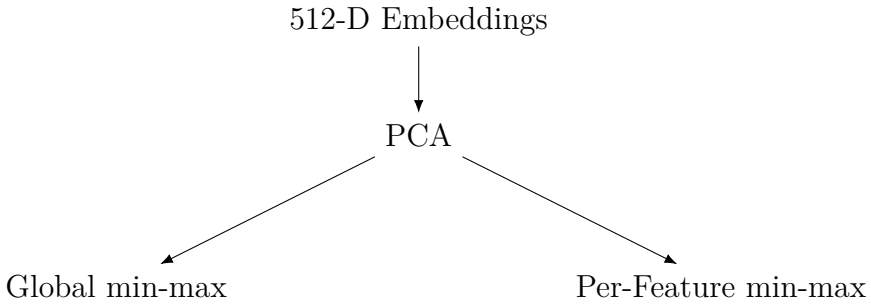

Figure 5.1: Experimental design for evaluating min-max normalization strategies. Starting from the original 512-dimensional FaceNet embeddings, PCA is applied for dimensionality reduction. Two normalization approaches are then tested: global min-max and per-feature min-max. Each leaf node represents a distinct experimental configuration used to assess the effect of normalization on face matching performance.

5.1.4 Quantization Strategies

In the third step of the pipeline, we convert float-valued embeddings to integer-valued embeddings. This conversion introduces quantization errors. As noted in Chapter 3, there are two main methods for performing this conversion: per-dimension quantization and global quantization. Notably, these options mirror the ones available for min-max normalization.

To assess the impact of quantization, we conduct a series of experiments. As shown in Figure 5.2, we apply the full preprocessing pipeline with different strategies and report the performance metrics for the dimension range corresponding to 90% to 95% explained variance.

An important observation from Chapter 3 is that global min-max normalization better preserves the relative ordering between embedding pairs compared to per-feature normalization. A similar trade-off applies to quantization: per-dimension quantization minimizes local quantization error but may distort interdimensional relationships, while global quantization, despite introducing possibly higher quantization error, maintains relative magnitudes across dimensions. Both strategies are evaluated and compared in this experiment.

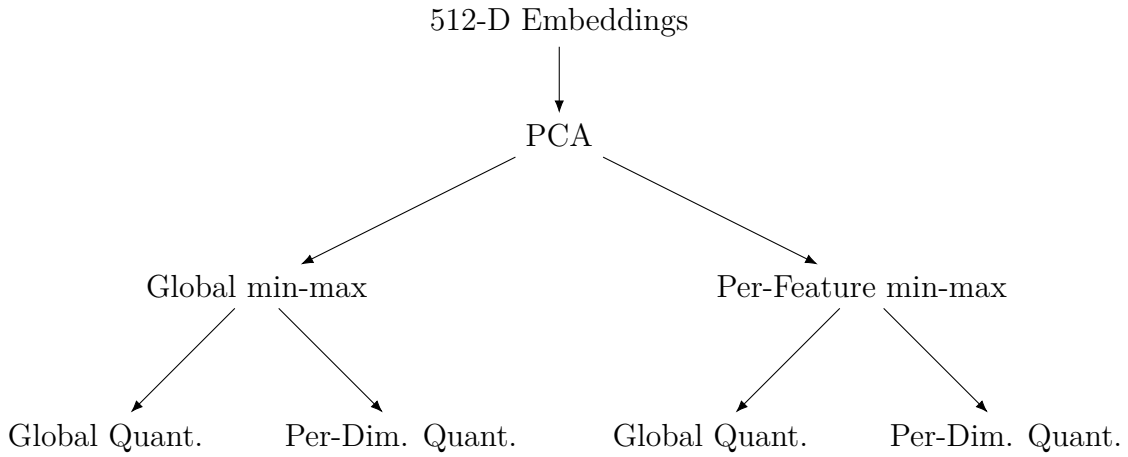

Figure 5.2: Experimental design for evaluating quantization strategies in the preprocessing pipeline. After applying PCA to the original 512-dimensional FaceNet embeddings, two normalization methods—global min-max and per-feature min-max—are considered. Each is followed by one of two quantization strategies: global quantization or per-dimension quantization. The leaf nodes represent all combinations of preprocessing strategies tested to assess their impact on matching performance.

5.2 Evaluation in the Encrypted Domain

After assessing the performance of different pipeline strategies in-clear domain across various preprocessing stages, we now evaluate its behavior within the encrypted domain.

5.2.1 Evaluation Setup and Circuit Execution

To evaluate encrypted domain performance, we apply the full preprocessing pipeline to the original 512-dimensional FaceNet embeddings, preparing them for execution under FHE.

We use the same biometric performance metrics as in the clear domain. In addition, we report the one-to-one comparison time for each pair and memory costs evaluated under encrypted execution. We apply the same 10-fold evaluation setup as in the clear domain using the `pairs.txt` protocol.

In the communication scenario, a threshold needs to be set on the client side to decide if decrypted results match with the data on the server. We set this threshold using the Receiver Operating Characteristic (ROC) curve. The optimal threshold is chosen on the ROC curve using Youden's J statistic. Details for these can be found in Appendix Section B.2 and Section B.2.1, respectively.

5.2.2 Pipeline Performance

As shown in Figure 5.2, we apply a similar approach to test out different combinations of preprocessing strategies for the preprocessing pipeline. The pipeline is summarized for reference in Figure 5.3. We investigate the permutations of the preprocessing strategies the same way as in the clear domain. Similarly, we report TPR at FPR thresholds 0.01%, 0.1%, and 1%.

5.2.3 Circuit Runtime: Quantization Bit-Width and LUT Impact

As discussed in Chapter 3, computing the squared Euclidean distance between two encrypted vectors in the `Concrete` framework involves non-linear operations. These are implemented

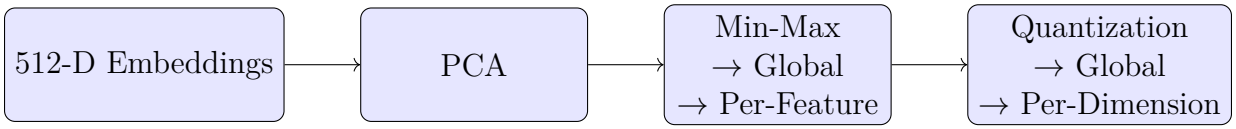

Figure 5.3: Preprocessing pipeline: PCA followed by min-max normalization and Quantization strategies.

using LUT operations, which are a core part of how `Concrete` handles non-linear functions over encrypted data.

The size of these LUTs depends on the input range of the function being evaluated. Wider input ranges require larger LUTs, which in turn increase both memory usage and computation time during encrypted evaluation. Therefore, choosing an appropriate quantization bit-width has a direct impact on the efficiency of the compiled FHE circuit.

With the experiments, we analyze how the choice of quantization bit-width affects the MBW of the compiled circuit, and how this translates to performance in the encrypted domain. By quantizing the inputs to different integer precisions, we examine the trade-off between circuit MBW and encrypted computation time.

5.2.4 Effect of Removing Min-Max Normalization

The `Concrete` framework also supports working with signed integers. However, this adds some overhead in calculations in the encrypted domain and uses one bit for signedness. To investigate this, we remove min-max normalization from the preprocessing pipeline (Figure 5.3) and report how working with signed integers in the encrypted domain changes the MBW requirements and performance metrics.

5.2.5 Memory Cost

TFHE ciphertexts are typically large, and their memory footprint can become a significant constraint when deploying FHE applications in real-world scenarios. This raises concerns about the practicality of the system, particularly in settings where client-server communication must be efficient and scalable.

In this experiment, we analyze the memory requirements of the encrypted pipeline by measuring the size of serialized ciphertexts corresponding to preprocessed embeddings, as well as the size of the associated evaluation keys used in the circuit that needs to be transmitted to the server that computes on encrypted data.

To exercise a realistic use case, we follow the client-server communication scenario described in Chapter 4, and measure the memory usage associated with each phase of the scenario. This includes both data transmission and key handling on the client and server sides.

The `Concrete` framework provides two serialization options for ciphertexts and keys: with compression and without. This setting must be selected at the time of circuit generation. We report the impact of this choice on memory consumption and evaluate whether compressed serialization provides a practical reduction in size without compromising functionality. This is because it adds an assumed cost of decompression within the encrypted circuit.

5.3 Optimizations in the `Concrete` Framework

In this section we will point out some of the configuration options in the `Concrete` framework. This is not an exhaustive discussion of all the options provided in the documentation[2] but given as opportunities for further optimization in the encrypted domain.

Some of these options are enabled by default such as loop parallelization and LUT fusing. Loop parallelization makes the tensor operation parallel through the use of OpenMP[3] and LUT fusing reduces the number of LUTs in the circuit. However, there are other options the users could investigate. One of these options is the LUT error probability parameter p_{error}. Recall that LUTs are performed with an FHE operation called Programmable Bootstrapping (PBS). PBSs have a certain probability of error: when these errors happen, it gives inaccurate results. The probability of this error can be configured through the p_{error} configuation option. For example, setting it to 0.01 will mean that every LUT in the circuit will have a 99% chance (or more) of being exact. Reducing this, results in less complex circuits (monitored through the *complexity* value reported by the compiled circuit in the `Concrete` framework). By default, the error probability value is 10^{-5}. We experiment with probabilities $\{10^{-4}, 10^{-3}, 10^{-2}, 10^{-1}\}$ and report how performance metrics, complexity value, and computation speed change.

5.4 Comparison with Prior Work

To position our approach relative to the state-of-the-art, we compare it with the two most relevant systems in the literature: Boddeti [32] and Jindal et al. [34]. Both works address one-to-one face matching in the encrypted domain and employ distinct preprocessing strategies alongside different FHE schemes—BFV in the case of Boddeti and CKKS for Jindal et al. These schemes support either exact or approximate arithmetic over encrypted data.

Boddeti evaluates performance on 128-dimensional embeddings under a 128-bit security level, while Jindal et al. report results for 128-dimensional embeddings with at least 80-bit security. For a fair comparison, we replicate experiments using 128-dimensional embeddings in our system and additionally report results under more aggressive dimensionality reduction, retaining 95% of the variance, to evaluate its impact on performance and efficiency.

The comparison includes the following criteria: TPR at FPR thresholds 0.01%, 0.1%, and 1%, ciphertext memory cost, embedding dimensionality, computational runtime and the cryptographic security level. This enables a meaningful assessment of the trade-offs and improvements offered by our approach relative to prior work.

[2]Available at `https://docs.zama.ai/concrete/guides/configure`

[3]`https://www.openmp.org/`

Chapter 6

Results of the Preprocessing Pipeline

Contents

In this chapter, we present the results of the experiments described in Chapter 5. For evaluations in the clear domain, we report standard biometric verification metrics: TPR at FPR levels of 0.01%, 0.1%, and 1%. In the encrypted domain, we extend the analysis by also reporting the execution time of one-to-one comparisons between encrypted embeddings, as well as the memory overhead of ciphertexts and evaluation keys.

All results are obtained using the standard 10-fold cross-validation protocol defined by the LFW `pairs.txt` protocol. Details on metric computation and evaluation methodology can be found in Appendix B. Throughout all experiments—both in the clear (plaintext) and encrypted settings—the squared Euclidean distance is used as the comparison function.

The chapter concludes with a summary of key findings across preprocessing strategies and execution domains. The full implementation of all experiments is available in the thesis GitHub repository [50].

6.1 Matching Performance in the Plaintext Domain

6.1.1 Baseline Performance

The baseline is set by evaluating the performance metrics on the 512-D embeddings from Facenet. The results are given in Table 6.1.

Metric	Value
TPR@0.01%	85.63±4.82
TPR@0.1%	97.77±1.33
TPR@1%	99.50±0.48

Table 6.1: Baseline performance metrics for 512-dimensional FaceNet embeddings aquired after the pipeline given in Figure 3.1. The squared Euclidean distance is used as comparison function. The values are given percentages with standard deviations.

6.1.2 Principal Component Analysis

To identify an appropriate range for dimensionality reduction, we analyze the explained variance using the elbow and scree plots as shown in Figure 6.1. According to the heuristic of retaining between 90% and 95% of the total variance, this corresponds to retaining approximately 40 to 44 components. From the scree plot we observe that beyond 48 components, the individual contribution of each additional component becomes marginal, indicating diminishing returns in representational capacity.

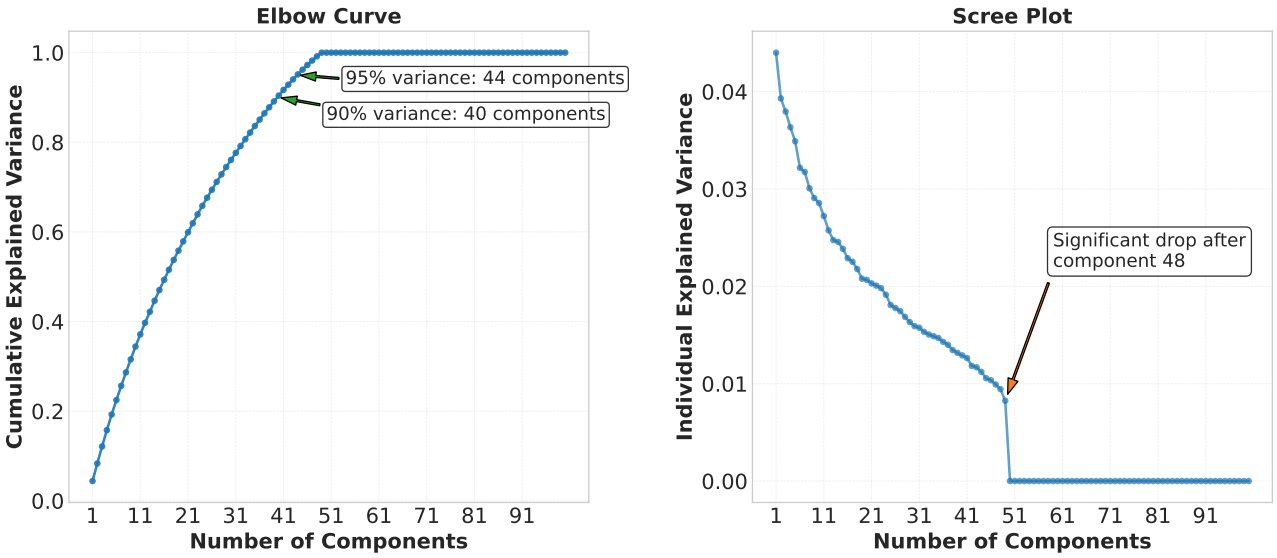

Figure 6.1: Explained variance analysis for PCA. **Left:** Elbow curve illustrating the cumulative explained variance with heuristic thresholds at 90% and 95%. **Right:** Scree plot showing the individual explained variance per component, highlighting the point beyond which additional components contribute minimally.

In Table 6.2, we report performance metrics after PCA for 40- to 50-dimensional embeddings. A more extensive version, covering dimensions from 10 to 49, is provided in Table A.1. At 49 components, the PCA transformation retains 100% of the original variance, and the resulting

metrics match those of the raw 512-dimensional baseline. From Figure 6.2 (visualizing the trends of mean of metrics from Table A.1), we see that between 40 and 45 retained components—corresponding to 90% and 95% explained variance respectively—the performance metrics tend to stabilize.

dim	TPR@0.01%	TPR@0.1%	TPR@1%	Variance
40	86.77±4.81	97.67±1.30	99.40±0.57	90.45±1.68
41	88.13±4.42	97.87±1.32	99.37±0.60	91.70±1.18
42	87.80±4.29	97.87±1.36	99.40±0.55	92.89±1.12
43	87.30±4.39	98.20±1.07	99.40±0.59	94.05±0.71
44	86.87±4.34	98.27±1.02	99.37±0.57	95.17±0.62
45	86.93±4.30	98.23±1.05	99.37±0.60	96.22±0.94
46	86.90±4.29	98.10±1.17	99.40±0.59	97.25±1.04
47	86.73±4.21	97.97±1.28	99.47±0.52	98.24±1.18
48	85.97±4.43	97.87±1.36	99.53±0.45	99.18±0.87
49	85.63±4.82	97.77±1.33	99.50±0.48	100.0±0.00
50	85.63±4.82	97.77±1.33	99.50±0.48	100.0±0.00

Table 6.2: TPR at FPR at thresholds 0.01%, 0.1%, and 1% across reduced dimensions. The metrics are reported in percentages with standard deviations.

Gong et al. [37] showed that convolutional-network embeddings contain substantial redundancy and the authors estimated the intrinsic dimensionality of FaceNet embeddings to be 12 dimensions. Although, our PCA analysis achieves significant reduction in dimensionality we are not able to go as low as 12 dimensions. With 49 principal components we capture essentially 100% of the variance in the dataset while reproducing the baseline TPR@FPR values. Applying the common "retain 90–95% of the variance" heuristic allows an even further reduction to 40–44 components, again without significant loss in any metric, as shown in Table 6.2.

The heuristic for selecting the number of principal components is based on the idea that certain dimensions may encode variable attributes—such as hairstyle or lighting—that introduce noise rather than improving identity discrimination, as these features can change even across images of the same individual. PCA alone does not indicate which component corresponds to a particular semantic attribute, nor does it necessarily isolate factors such as lighting variations into lower-variance components. In our application, we would like to achieve further dimensionality reduction without explicit semantic labels for components, a conservative strategy—such as discarding the least significant components accounting for approximately 5–10% of the variance—is practical in such a setting. Our empirical analysis supports this choice, as evidenced by the stable performance metrics within the 90–95% variance retention range in Figure 6.2. This stability demonstrates that removing a modest number of components, irrespective of their precise semantic meaning, effectively eliminates minimal identity-related information while achieving a more compact representation.

Going even further, Figure 6.2 suggests that starting from 30 components (explaining 76.13% of the variance) the LFW benchmark yields promising results (see Table A.1 for exact values). However, due to the unsupervised nature of PCA, we do not know which features we are discarding by lowering the retained explained variance. We therefore adopt the more conservative heuristic of retaining 90–95% of the explained variance for the subsequent experiments.

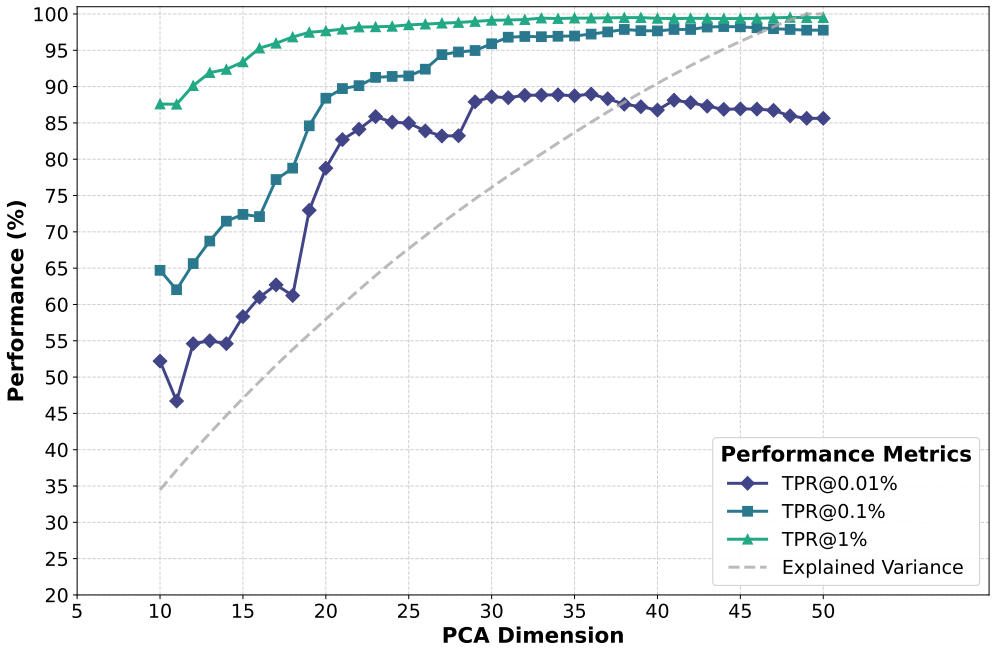

Figure 6.2: Mean of performance metrics as a function of PCA dimensionality, illustrating the trends in Table A.1. For each retained-component count, the curves report TPR at three operating points FPR = 0.01%, 0.1%, and 1%. The dashed line shows the cumulative explained variance. Together, the curves illustrate that performance plateaus once 40–45 components (≈ 90–95% of variance) are kept.

6.1.3 Min-max Normalization Strategies

In Table 6.3 we see the results of the experiment setups presented in the previous chapter in Figure 5.1 for components 40 and 44. For dimensions 41 to 43 refer to Table A.2.

Dim	Method	TPR@0.01%	TPR@0.1%	TPR@1%
40	G	86.77±4.81	97.67±1.30	99.40±0.57
	PF	84.80±4.26	96.67±1.42	99.17±0.70
44	G	86.87±4.34	98.27±1.02	99.37±0.57
	PF	86.27±4.32	97.17±1.34	99.03±0.77

Table 6.3: Comparison of global (G) and per-feature (PF) min-max normalization methods after applying PCA. All values are reported in percentages (%) with standard deviations.

From Chapter 3, global min–max normalization was expected to preserve the relative rankings of the embeddings and to outperform the per-feature variant. The empirical results confirm both expectations. First, we can compare Table 6.2 and Table 6.3 to confirm the global min-max results match with the results after PCA. Second, we can see from Table 6.3 that per-feature min-max normalization slightly underperforms compared to global min-max normalization.

6.1.4 Quantization Strategies

In this section, we provide the results of the full experimental pipeline presented in Figure 5.2 in Chapter 5. The following tables correspond to different combinations of normalization and quantization methods applied after PCA: Table 6.4 presents results for global min-max normalization

followed by global quantization; Table 6.5 corresponds to global min-max normalization followed by per-dimension quantization; Table 6.6 shows results for per-feature min-max normalization with global quantization; and Table 6.7 reports on per-feature min-max normalization followed by per-dimension quantization. Results for dimensions 41 to 43 for each configuration can be found in Table A.3, Table A.4, Table A.5, and Table A.6 respectively.

Dim	BW	TPR@0.01%	TPR@0.1%	TPR@1%
40	2	45.73±13.06	64.27±8.42	83.30±3.90
	3	74.40±11.91	90.33±3.29	96.87±1.49
	4	84.70±5.55	96.97±1.41	99.20±0.62
	5	87.23±3.85	96.23±2.15	99.17±0.72
44	2	54.07±15.12	70.27±6.42	85.57±2.25
	3	76.93±9.51	91.10±2.84	97.63±1.12
	4	85.10±5.91	97.33±1.34	99.13±0.81
	5	88.03±3.91	96.63±2.03	99.27±0.66

Table 6.4: Matching performance in the plaintext domain as a function of quantization bit-width (BW). For PCA-reduced embeddings across different dimensions, this table reports TPR at FPR = 0.01 %, 0.1 %, and 1 % under the pipeline: PCA→global min–max normalization→global quantization. Values are shown in percentages with standard deviations.

Dim	BW	TPR@0.01%	TPR@0.1%	TPR@1%
40	2	35.60±10.33	46.27±5.40	69.70±4.65
	3	76.93±6.99	85.87±2.50	96.00±1.27
	4	83.87±3.57	94.83±1.59	98.70±0.90
	5	82.47±6.70	96.33±1.57	99.07±0.81
44	2	36.57±9.10	45.83±3.47	70.77±3.01
	3	79.70±5.81	87.70±2.24	95.77±1.80
	4	84.57±4.54	95.20±2.28	98.90±0.73
	5	85.03±5.68	96.27±1.73	98.97±0.85

Table 6.5: Matching performance in the plaintext domain as a function of quantization bit-width (BW). For PCA-reduced embeddings at Dim = 40 and 44, this table reports TPR at FPR = 0.01 %, 0.1 %, and 1 % under the pipeline: PCA→global min–max normalization→per-dimension quantization. Values are shown in percentages with standard deviations.

From Table 6.4 and Table 6.5, we can investigate the effect of quantization strategy followed after global min-max normalization. We observe that the global quantization outperforms per-dimension quantization. This was expected as per-dimension quantization introduces distortions in quantized embeddings.

Dim	BW	TPR@0.01%	TPR@0.1%	TPR@1%
40	2	53.47±6.40	60.63±5.96	85.63±3.34
	3	80.83±7.92	92.83±2.36	97.80±0.95
	4	84.83±5.62	96.30±1.82	99.17±0.73
	5	83.10±4.40	94.20±1.80	98.37±0.78
44	2	60.37±7.54	68.20±6.50	86.93±3.34
	3	83.90±7.31	93.30±1.35	97.87±1.35
	4	85.90±4.89	96.80±1.69	99.00±0.80
	5	85.83±3.72	94.73±2.29	98.60±0.77

Table 6.6: Matching performance in the plaintext domain as a function of quantization bit-width (BW). For PCA-reduced embeddings at Dim = 40 and 44, this table reports TPR at FPR = 0.01 %, 0.1 %, and 1 % under the pipeline: PCA→per-feature min–max normalization→global quantization. Values are shown in percentages with standard deviations.

Dim	BW	TPR@0.01%	TPR@0.1%	TPR@1%
40	2	45.00±8.53	54.57±5.75	76.80±3.49
	3	74.37±8.10	86.93±3.05	96.63±1.66
	4	80.77±4.20	92.10±2.47	97.87±1.31
	5	78.67±5.07	91.70±2.73	97.60±1.09
44	2	49.13±9.57	60.47±6.48	78.90±2.32
	3	80.47±5.15	89.10±2.22	96.40±1.87
	4	85.00±2.46	93.93±2.11	97.63±1.52
	5	83.27±4.32	92.23±1.72	97.33±1.22

Table 6.7: Matching performance in the plaintext domain as a function of quantization bit-width (BW). For PCA-reduced embeddings at Dim = 40 and 44, this table reports TPR at FPR = 0.01 %, 0.1 %, and 1 % under the pipeline: PCA→per-feature min–max normalization→per-dimension quantization. Values are shown in percentages with standard deviations.

Observing the results in Table 6.6 and Table 6.7, we find again that applying global quantization after per-feature min–max normalization yields better performance than per-dimension quantization.

To further isolate the effect of the normalization strategy, we fix the quantization method to global quantization and compare Table 6.4 with Table 6.6. This comparison reveals that global min–max normalization outperforms its per-feature counterpart. These findings reinforce our earlier hypothesis from Chapter 3: global normalization strategies tend to better preserve the relative rankings between embeddings, which we see empirically that it is critical for maintaining and matching the baseline performance.

In the following section, we evaluate whether these trends hold in the encrypted domain.

6.2 Matching Performance in the Encrypted Domain

Building on the results observed in the plaintext (clear) domain, we now evaluate the full pipeline under encryption. Our preliminary findings in clear domain indicated that using a bit-width of 4 or more preserves matching performance close to the baseline. In this section, we extend the analysis to the encrypted setting to assess the impact of homomorphic computation.

6.2.1 Pipeline Performance

We repeat the strategies given in Figure 5.2 in the encrypted domain. Therefore, the following tables correspond to different combinations of normalization and quantization methods applied after PCA: Table 6.8 presents results for global min-max normalization followed by global quantization; Table 6.9 corresponds to global min-max normalization followed by per-dimension quantization; Table 6.10 shows results for per-feature min-max normalization with global quantization; and Table 6.11 reports on per-feature min-max normalization followed by per-dimension quantization. Results for dimensions 41 to 43 for each configuration can be found in Table A.7, Table A.8, Table A.9, and Table A.10 respectively.

Dim	BW	TPR@0.01%	TPR@0.1%	TPR@1%
40	4	84.70±5.55	96.97±1.41	99.2±0.62
	5	87.93±4.33	97.37±1.29	99.37±0.66
44	4	85.10±5.91	97.33±1.34	99.13±0.81
	5	88.13±3.91	97.87±1.33	99.33±0.61

Table 6.8: Matching performance in encrypted domain as a function of quantization bit-width (BW). For PCA-reduced embeddings at Dim = 40 and 44, this table reports TPR at FPR = 0.01 %, 0.1 %, and 1 % under the pipeline: PCA→global min–max normalization→global quantization. Values are shown in percentages with standard deviations.

Dim	BW	TPR@0.01%	TPR@0.1%	TPR@1%
40	4	83.87±3.57	94.83±1.59	98.7±0.90
	5	83.77±5.09	96.37±1.55	99.1±0.82
44	4	84.57±4.54	95.2±2.28	98.9±0.73
	5	85.47±5.36	96.47±1.62	99.03±0.81

Table 6.9: Matching performance in encrypted domain as a function of quantization bit-width (BW). For PCA-reduced embeddings at Dim = 40 and 44, this table reports TPR at FPR = 0.01 %, 0.1 %, and 1 % under the pipeline: PCA→global min–max normalization→per-dimension quantization. Values are shown in percentages with standard deviations.

Comparing Table 6.8 and Table 6.9 to investigate the effect of quantization strategy after global min-max normalization, we observe that global quantization performs better than per-dimension quantization. This indicates that the distortions introduced by per-dimension quantization—particularly its tendency to disrupt the relative relationships between embeddings—have more impact on matching performance than the extra quantization errors introduced by using a single global scaling factor. As was the case in the plaintext domain.

Now moving onto assessing the quantization strategies when per-feature min-max operation is applied on PCA-reduced embeddings.

Dim	BW	TPR@0.01%	TPR@0.1%	TPR@1%
40	4	84.83±5.62	96.30±1.82	99.17±0.73
	5	85.47±4.40	96.73±1.39	99.13±0.81
44	4	85.90±4.89	96.80±1.69	99.00±0.80
	5	87.13±4.16	96.90±1.58	99.13±0.78

Table 6.10: Matching performance in encrypted domain as a function of quantization bit-width (BW). For PCA-reduced embeddings at Dim = 40 and 44, this table reports TPR at FPR = 0.01 %, 0.1 %, and 1 % under the pipeline: PCA→per-feature min–max normalization→global quantization. Values are shown in percentages with standard deviations.

Dim	BW	TPR@0.01%	TPR@0.1%	TPR@1%
40	4	80.8±4.21	92.10±2.47	97.87±1.31
	5	79.5±4.05	93.80±2.13	98.3±1.15
44	4	85.0±2.46	93.93±2.11	97.63±1.52
	5	84.63±4.04	94.73±1.84	97.93±1.24

Table 6.11: Matching performance in encrypted domain as a function of quantization bit-width (BW). For PCA-reduced embeddings at Dim = 40 and 44, this table reports TPR at FPR = 0.01 %, 0.1 %, and 1 % under the pipeline: PCA→per-feature min–max normalization→per-dimension quantization. Values are shown in percentages with standard deviations.

Comparing Table 6.10 and Table 6.11, we see that global quantization outperforms per-dimension quantization applied after per-feature min-max normalization in all performance metrics. This makes global quantization better choice regardless of the min-max strategy.

We now compare the effect of min-max strategy by comparing Table 6.8 and Table 6.10. We can observe that the performance metrics are close, slightly favoring the global min-max normalization. Which aligns with our expectations but the slight gap in encrypted domain suggests that they could be interchanged without a significant drop in performance metrics.

We therefore reach the conclusion that after using PCA to reduce the dimensionality, combining global min-max normalization with global quantization yields the best overall pipeline performance. In the following experimental results, we employ this pipeline as the best overall performing setup.

6.2.2 Circuit Runtime: Quantization Bit-Width and LUT Impact

When compiling FHE circuits with the `Concrete` framework, one of the key complexity indicators is the MBW of the FHE circuit. MBW represents the largest number of bits needed to represent intermediate or output values during computation and directly influences both the size of LUTs and the overall computational cost.

Since our circuit operates on quantized vectors, the MBW is largely determined by the bit-width used during quantization. Higher quantization precision allows for more accurate computations but increases the MBW and consequently the encrypted evaluation time.

Table 6.12 reports the computation times for one-to-one comparisons in the encrypted domain. Results are provided for two representative embedding dimensions (40 and 44), additional dimensions (41 to 43) are reported in Table A.11. Each configuration is evaluated across two quantization bit-widths. As expected, the MBW increases with higher quantization precision. Computation time scales with both dimensionality and MBW, and we observe that

increasing the bit-width from 4 to 5 bits increases the execution time approximately more than a factor of five in some settings. These results highlight the importance of minimizing quantization bit-width, as even small increases can substantially impact performance.

Dim	BW	MBW	Computation Time (s)
40	4	12	0.173 ± 0.066
	5	14	0.920 ± 0.119
44	4	12	0.174 ± 0.025
	5	14	0.915 ± 0.081

Table 6.12: Encrypted computation time (mean ± standard deviation) and corresponding MBW required for different combinations of embedding dimensionality and quantization bit-width (BW). Results are reported for a preprocessing pipeline consisting of PCA → global min–max normalization → global quantization, and evaluated on an AMD EPYC 7763 using 16 cores out of 64.

6.2.3 Effect of Removing Min-Max Normalization

Given that the global min-max normalization followed by global quantization provided the best overall performance (see Table 6.8), we investigate the impact of removing the normalization step entirely. Table 6.13 presents matching performance metrics for a simplified pipeline consisting of PCA directly followed by global quantization (detailed results for intermediate dimensions (41–43) are provided in Table A.12). This configuration operates on signed integers.

Dim	BW	TPR@0.01%	TPR@0.1%	TPR@1%
40	4	63.63±32.62	96.03±2.64	99.4±0.53
	5	44.5±36.35	94.63±4.05	99.33±0.54
44	4	59.7±32.29	95.9±3.00	99.3±0.71
	5	26.87±35.72	93.37±4.18	99.37±0.57

Table 6.13: Matching performance in encrypted domain of the pipeline without min–max normalization for different quantization bit-widths (BW): PCA→global quantization. Values are shown in percentages with standard deviations.

Signed integers inherently reserve one bit for the sign, thereby reducing the precision available for representing magnitude. Consequently, the signed integer pipeline (Table 6.13) demonstrates a notable decrease in matching performance compared to the unsigned integer pipeline (Table 6.8). This performance drop is particularly pronounced for TPR@0.01%, whereas it is relatively minor for higher FPR thresholds (TPR@0.1% and TPR@1%).

Dim	BW	MBW	Computation Time (s)
40	4	11	0.165 ± 0.055
	5	14	0.928 ± 0.102
44	4	12	0.183 ± 0.024
	5	14	0.954 ± 0.098

Table 6.14: Encrypted computation time (mean ± standard deviation) and corresponding MBW required for different combinations of embedding dimensionality and quantization bit-width (BW). Results are reported for a preprocessing pipeline consisting of PCA → global quantization, and evaluated on an AMD EPYC 7763 using 16 cores out of 64.

Considering that computational complexity is strongly influenced by MBW, we further compare the signed and unsigned integer pipelines using Table 6.12 (unsigned) and Table 6.14 (signed), for intermediate dimensions refer to Table A.11 and Table A.13 respectively. Results show minimal variation in MBW and computation time between the two representations. Specifically, computation times are marginally higher for signed integers when MBWs are the same, confirming a slight overhead from signed arithmetic. Nonetheless, this overhead appears negligible for squared Euclidean distance calculations.

Therefore, our findings recommend using unsigned integers to optimize performance in encrypted biometric matching applications.

6.2.4 Memory Cost

We further evaluate memory requirements using the optimal preprocessing pipeline—global min-max normalization combined with global quantization—retaining 44 principal components quantized at a fixed bit-width of 4 bits.

Without compression, the ciphertext size per encrypted embedding is 704.58 KB, and the size of the evaluation keys for the FHE circuit is 204.65 MB. The corresponding computation time is approximately 0.174 seconds for one-to-one encrypted matching, as reported in Table 6.12.

Enabling compression for both ciphertexts and evaluation keys significantly reduces the memory footprint: the ciphertext size drops to 1.27 KB, and the evaluation key size is reduced to 51.27 MB. This configuration increases the computation time very slightly (at most by $\approx 5\%$), meaning compression can be enabled without performance degradation.

To contextualize these savings, we assess the communication overhead of the client-server scenario outlined in Chapter 4. The results, summarized in Table 6.15, illustrate communication requirements for each protocol phase, comparing compressed and uncompressed configurations. During enrollment, the client transmits five encrypted embeddings along with evaluation keys. Similarly, during authentication, the client sends five embeddings for comparison with previously stored encrypted embeddings on the server.

These findings highlight significant storage requirements inherent in homomorphic encryption-based biometric systems, notably larger than plaintext alternatives, which avoid evaluation keys and utilize compact floating-point vectors. Consequently, deploying this encryption-based biometric matching system at scale would require substantial storage and bandwidth resources.

	Client sends		Server sends	
	C	–	C	–
Introduce	0.2 KB	0.2 KB	2.6 KB	2.6 KB
Schnorr	0.4 KB	0.4 KB	0.2 KB	0.2 KB
Enroll	68.3 MB	277.5 MB	38 B	38 B
Authenticate	8.7 KB	4.6 MB	541.5 KB	541.5 KB

Table 6.15: Communication overhead per protocol phase (with compression (C) and without compression (–)), corresponding to the communication scenario described in Chapter 4. Client sends five encrypted preprocessed embedding in enrollment and authentication.

In terms of computational performance, based on the data in Table 6.12, each encrypted comparison requires about 0.174 seconds. Given that the server performs 25 comparisons (five embeddings from enrollment against five new embeddings during authentication), the total computation time amounts to approximately $25 \times 0.174 = 4.35$ seconds. While network latency could further contribute to delays, such a performance might remain acceptable in security-critical scenarios (e.g., border control or facility access), though impractical for real-time, high-throughput applications.

6.3 Optimizations in the `Concrete` Framework

In this section, we analyze how varying the LUT error probability parameter, denoted as p_{error}, affects the compiled circuit's complexity and performance. If not specified, the default value for this parameter is set at 10^{-5}. All experiments are conducted using 44-dimensional embeddings with global min-max normalization and global quantization strategies with quantization bit-width of 4 bits.

p_{error}	Complexity	TPR@0.01%	TPR@0.1%	TPR@1%	Time (s)
10^{-5}	7.08e9	85.10±5.91	97.33±1.34	99.13±0.81	0.174
10^{-4}	6.33e9	85.10±5.91	97.30±1.35	99.13±0.81	0.161
10^{-3}	6.19e9	85.07±5.90	97.33±1.34	99.13±0.81	0.144
10^{-2}	4.90e9	85.13±5.29	97.17±1.36	99.17±0.78	0.142
10^{-1}	4.36e9	83.43±5.11	95.87±1.65	99.20±0.86	0.129

Table 6.16: Effect of LUT p_{error} on circuit performance metrics, complexity, and computation speed at 44 dimensions. The default value for error probability is 10^{-5}. Pipeline strategies are global min-max normalization with global quantization of 4 bits after PCA. TPR at FPR = 0.01 %, 0.1 %, and 1 % are reported in percentages with standard deviation, the execution time is given as the average in seconds, and complexity is the internal metric of the compiled circuit in `Concrete`.

In Table 6.16, raising the LUT error probability target from the default value of 10^{-5} to 10^{-1} decreases the circuits reported complexity metric by $\approx 38\%$ (7.08e9 $\rightarrow$ 4.36e9) while shortening the average evaluation time by $\approx 25\%$ from 0.174 s to 0.129 s.

This gain in efficiency is accompanied by a slight drop in some performance metrics. This drop shows that there is a trade-off between computational efficiency and verification performance when tuning this parameter, but since the drop is not a significant one, it indicates that squared Euclidean distance computation in encrypted domain is robust against higher LUT error probabilities. A good trade-off point seems to be increasing the error probability to 0.01, where we can gain in speed while keeping similar performance metrics.

6.4 Comparison with Prior Work

To position our approach relative to the state-of-the-art, we compare against the two closest systems: Boddeti [32] and Jindal et al. [34]. These two prior works focus on one-to-one matching in encrypted domain, and both utilize certain preprocessing techniques with different FHE schemes of choice, BFV and CKKS respectively. From these two prior studies we get the results for 128-dimensional embeddings to compare with our system with 128-dimensional embeddings; we additionally report our best 44-dimensional configuration to highlight the benefit of more aggressive dimensionality reduction without significant decrease in matching performance.

Method	Dim.	Security	Scheme	Comp. Time (ms) †	Ciphertext Size (KB)
Boddeti	128	128 bits	BFV	4	4.0
Jindal et al.	128	≥ 80 bits	CKKS	2.8	32.8
Ours	128	128 bits	TFHE	302	3.24
Ours	44	128 bits	TFHE	142	1.27

Table 6.17: Cryptographic characteristics and encrypted comparison costs for prior and proposed methods. †Prior work reports amortized batched timings; our times are single-pair evaluations on an AMD EPYC 7763 using 16 cores out of 64. Our configuration: PCA (44 or 128 components), global min-max normalization, global quantization (4-bit), compression enabled, and $p_{\text{error}} = 10^{-2}$.

Our TFHE-based solution achieves smaller ciphertext sizes (3.24 KB at 128 dimensions and 1.27 KB at 44 dimensions), which is advantageous for storage-constrained scenarios. However, this compact representation comes at the cost of higher computation time relative to BFV and CKKS schemes. Nevertheless, reducing the embedding to 44 components more than halves the runtime while preserving 128-bit security.

Method	TPR@0.01%	TPR@0.1%	TPR@1%
Boddeti [32]	84.06	94.56	98.65
Jindal et al. [34]	86.58	94.28	96.10
Ours (128 dims)	84.67±6.05	96.70±1.41	99.23±0.67
Ours (44 dims)	85.13±5.29	97.17±1.36	99.17±0.78

Table 6.18: Matching performance across TPR at FPR = 0.01 %, 0.1 %, and 1 %. The prior studies are reported without standard deviation, our results are reported on `pairs.txt` protocol using the 10-folds. Our configuration: PCA (44 or 128 components), global min-max normalization, global quantization (4-bit), compression enabled, and $p_{\text{error}} = 10^{-2}$.

Despite operating on integer-quantized ciphertexts, our method performs well on performance metrics. Notably, reducing dimensionality from 128 to 44 components not only significantly reduces ciphertext size and runtime but also results in slightly improved matching performance across TPR@0.01% and TPR@0.1% in Table 6.18. This could be attributed to the reduced computational complexity in the encrypted domain. Fewer dimensions lead to fewer homomorphic operations—particularly expensive non-linear ones—thereby lowering the chance of error.

Due to the absence of standard deviation information in prior work, we compare mean values of our results in Table 6.18. Our approach outperforms Boddeti's method across all metrics. Compared to Jindal et al., we achieve higher scores on all metrics except TPR@0.01%.

Even though the prior work and this work implement similar systems, the reported computation times for one-to-one matching should be interpreted with caution since direct comparison

across studies is challenging due to differences in experimental setups, particularly the hardware platforms and batching optimizations used by prior methods.

6.5 Comparison of Results in the Plaintext and Encrypted Domains

To conclude this analysis, Table 6.19 compares matching performance across three settings: (i) the original 512-dimensional FaceNet embeddings without preprocessing (baseline), (ii) the optimized pipeline evaluated in the clear domain, and (iii) the same pipeline evaluated under encryption.

The results show that the optimized pipeline yields identical performance in both clear and encrypted domains across all metrics. Specifically, the TPR values at FPR thresholds of 0.01%, 0.1%, and 1% match precisely, including their standard deviations. This confirms that the pipeline preserves matching behavior under homomorphic computation.

Compared to the plaintext baseline, the optimized pipeline—despite dimensionality reduction and quantization—incurs only a marginal drop in accuracy. These findings validate the effectiveness of the preprocessing strategies: PCA followed by global min–max normalization and 4-bit global quantization. This optimized pipeline not only reduces computational and memory costs but also enables encrypted computation while maintaining verification performance comparable to that of the baseline.

Domain	**TPR@0.01%**	**TPR@0.1%**	**TPR@1%**
Baseline	85.63±4.82	97.77±1.33	99.50±0.48
Clear	85.10±5.91	97.33±1.34	99.13±0.81
Encrypted	85.10±5.91	97.33±1.34	99.13±0.81

Table 6.19: Comparison of matching performance across baseline, clear, and encrypted domains. The baseline uses the original 512-dimensional FaceNet embeddings without preprocessing. The clear and encrypted configurations apply PCA (to 44 components), global min–max normalization, and 4-bit global quantization. In encrypted computation, the default error probability of 10^{-5} is used. Results are reported in percentages for TPR at FPR thresholds of 0.01%, 0.1%, and 1% with standard deviations based on 10-fold evaluation using the `pairs.txt` protocol.

6.6 Summary of Findings

In this chapter, we evaluated preprocessing and quantization strategies for privacy-preserving face verification using FHE, starting from 512-dimensional FaceNet embeddings.

We first applied PCA to reduce dimensionality to 40–44 components, retaining 90–95% of variance. This significantly compressed the data with negligible impact on performance, confirming known redundancy in FaceNet embeddings.

Next, we compared normalization and quantization methods. Global min–max normalization consistently outperformed per-feature normalization by better preserving relative embedding rankings. Similarly, global quantization yielded higher matching performance than per-dimension quantization, regardless of normalization.

The best overall pipeline—PCA(44 dimensions), global min–max normalization, and 4-bit global quantization—achieved near-baseline performance in the plaintext domain and identical performance under encryption. This validates its effectiveness for homomorphic computation.

We further analyzed circuit runtime and memory costs. A 4-bit quantization provided the best balance of speed and accuracy, with one-to-one encrypted comparisons averaging ≈ 0.174 s. Enabling built-in compression drastically reduced memory usage without affecting runtime.

Adjusting the LUT error probability parameter (p_{error}) revealed further optimization opportunities, improving the computation time to ≈ 0.142 s with minimal matching performance loss.

Compared to prior work, our TFHE-based approach achieved competitive—or superior—TPR at FPR thresholds of 0.01%, 0.1%, and 1%, while also producing smaller ciphertext sizes. Although this comes at the cost of higher latency, the trade-off may be acceptable in high-security, low-throughput settings.

Overall, our preprocessing pipeline offers a practical balance of performance, efficiency, and deployability for encrypted face matching on the LFW dataset.

Chapter 7

Conclusion and Future Work

Contents

7.1 Conclusion

This thesis investigated the application of FHE to one-to-one face verification, motivated by the sensitivity of biometric data and the need for privacy-by-design solutions under regulations such as GDPR and CCPA. FHE fulfills this requirement by enabling computation on encrypted data, making it a promising tool for secure biometric authentication. However, its widespread adoption remains limited due to significant computational overhead, even with recent advances such as Zama's `Concrete` framework.

To mitigate these constraints, we proposed an optimized preprocessing pipeline for FHE-based face verification and evaluated it using the LFW dataset. The pipeline begins with face alignment using MTCNN and embedding extraction via FaceNet, followed by PCA-based dimensionality reduction, min–max normalization, and quantization. Both per-feature (per-dimension) and global strategies for normalization and quantization were assessed, with global approaches consistently yielding superior matching performance.

In addition to preprocessing, we developed a client-server prototype that integrates FHE with other cryptographic tools, including ECDH, Schnorr zero-knowledge proofs, and HMAC. While the system lacks a formal security proof, it demonstrates the practical feasibility of privacy-preserving face verification where the server processes only encrypted data.

Moreover, compared to prior work, our method achieved competitive or improved performance metrics with reduced ciphertext sizes. It also provided an empirical analysis of preprocessing strategies and their influence on encrypted computation, an aspect often overlooked in earlier studies. Nonetheless, computation time remains a major bottleneck: encrypted comparisons using TFHE are considerably slower than those using BFV or CKKS, despite code-level optimizations.

Overall, this work introduces a robust, empirically validated preprocessing pipeline and a proof-of-concept client-server communication scenario for FHE-based face verification. The results show the central role of preprocessing in balancing performance metrics, runtime, and ciphertext size.

Limitations and Broader Implications

Adopting FHE introduces a fundamental shift in the conventional client–server paradigm. Traditional infrastructures rely heavily on backend-centric architectures, where servers process plaintext data and users must trust service providers with sensitive information. In contrast, FHE confines the server's role to performing computations on encrypted data, thereby eliminating the need to trust server operators with the actual content.

However, this paradigm shift brings several architectural and usability challenges. Existing identity management solutions often depend on centralized, federated services (e.g., Google Sign-In), which presuppose access to user credentials and profile data. In an FHE-enabled context, such assumptions are no longer valid. Instead, identity verification must be reimagined using privacy-preserving mechanisms—for example, using Schnorr zero-knowledge proofs, as illustrated in the client-server scenario described in Chapter 4.

Furthermore, FHE redistributes computational responsibilities toward the client. Unlike traditional systems where the server performs most operations, FHE-based architectures require the client to handle key generation, encryption, and decryption, as well as to perform the final decision-making based on encrypted computation results. This "client-heavy" architecture poses challenges in terms of hardware heterogeneity, limited compute capabilities on mobile devices, battery consumption, and increased local storage requirements. As a result, user adoption may be hindered by the latency and complexity introduced by encrypted computation—particularly in applications where real-time responsiveness is critical, such as web-based services.

Promoting the adoption of FHE-based systems thus requires not only technical optimization but also careful attention to interface and system design. Users should be made aware of the long-term privacy benefits that FHE provides, while the complexity of the underlying cryptographic operations should be abstracted away through intuitive, user-friendly interfaces to preserve usability.

Despite these limitations, promising developments are underway. Hardware acceleration efforts—such as Zama's GPU backends, FPGA implementations, and dedicated FHE co-processors—offer tangible paths to improving the practicality of encrypted computation. These technological advances, discussed further in the following section, could significantly reduce latency and enable practical deployment.

More broadly, the transition to FHE-based systems necessitates a rethinking of client–server architectures. Addressing performance bottlenecks is only one part of the equation; equally important is re-evaluating the role and design of the server within this new paradigm. In FHE-enabled systems, servers are no longer trusted entities with privileged access to data, but rather blind compute agents operating solely on encrypted inputs. This redefinition calls for the development and deployment of new secure protocols—such as the biometric authentication protocol proposed by Pradel et al. [38], discussed in Chapter 2—that are designed from the ground up with encrypted computation in mind. These findings contribute to advancing the practical adoption of FHE in privacy-sensitive biometric applications and lay the groundwork for future deployments that are both secure and user-friendly.

7.2 Future Work

The main challenge revealed in Chapter 6 is evaluation latency: a single encrypted one-to-one comparison takes around $\approx 0.142\,\mathrm{s}$ for our best pipeline. This latency originates from TFHE's binary gate-level arithmetic—highly efficient for Boolean logic yet comparatively slow for integer operations—whereas ring-based schemes such as BFV and CKKS support packed, vectorized integer and fixed-point arithmetic.

Recent hardware advances offer three complementary paths to reduce this overhead. First,

GPU back-ends: Zama's `Concrete` framework includes a GPU backend that parallelizes key switching and PBS across GPU cores [51]. Second, Homomorphic Processing Units (HPUs): prototype chips integrate number-theoretic transform and residue-arithmetic engines directly in hardware, reducing bootstrapping latency by an order of magnitude [52]. Early HPU support for TFHE operations was added as a mockup implementation in Zama's `tfhe-rs` v1.2.0 [53] and requires further investigation. Third, FPGA accelerators: the Belfort Labs FPGA core for TFHE, targeting `tfhe-rs`, claims more than a 300× speed-up [54], and an open demo integration[1] could be ported to cloud FPGAs for scaling. As the `Concrete` framework uses `tfhe-rs` for the TFHE implementation, systematic benchmarking of these accelerators is a natural continuation of this thesis.

Further gains may arise from replacing PCA with non-linear dimensionality-reduction techniques such as DeepMDS++, which could capture intrinsic structure more compactly and shorten ciphertexts. Exploring larger and more diverse face datasets would also test the robustness and reproducibility of the conclusions drawn here.

Finally, a formal security analysis of the client–server communication scenario from Chapter 4, along with improvements to storage-intensive key management strategies, will be essential for real-world deployment. Taken together, these future directions can advance the practical viability of FHE-based biometric systems.

[1] `https://github.com/belfortlabs/hello-fpga`

Bibliography

[1] European Parliament and Council of the European Union. General Data Protection Regulation (GDPR). `https://gdpr-info.eu/`, 2018.

[2] California Department of Justice, Office of the Attorney General. California Consumer Privacy Act (CCPA). `https://oag.ca.gov/privacy/ccpa`, 2018. Accessed: 2025-05-27.

[3] Zama. Concrete: TFHE Compiler that converts python programs into FHE equivalent, 2022. `https://github.com/zama-ai/concrete`.

[4] Zama. TFHE-rs: A Pure Rust Implementation of the TFHE Scheme for Boolean and Integer Arithmetics Over Encrypted Data, 2022. `https://github.com/zama-ai/tfhe-rs`.

[5] Matthew Turk and Alex Pentland. Eigenfaces for recognition. *Journal of cognitive neuroscience*, 3(1):71–86, 1991.

[6] Peter N. Belhumeur, Joao P Hespanha, and David J. Kriegman. Eigenfaces vs. fisherfaces: Recognition using class specific linear projection. *IEEE Transactions on pattern analysis and machine intelligence*, 19(7):711–720, 1997.

[7] Yaniv Taigman, Ming Yang, Marc'Aurelio Ranzato, and Lior Wolf. Deepface: Closing the gap to human-level performance in face verification. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1701–1708, 2014.

[8] Gary B Huang, Marwan Mattar, Tamara Berg, and Eric Learned-Miller. Labeled faces in the wild: A database for studying face recognition in unconstrained environments. In *Workshop on faces in'Real-Life'Images: detection, alignment, and recognition*, 2008.

[9] Florian Schroff, Dmitry Kalenichenko, and James Philbin. Facenet: A unified embedding for face recognition and clustering. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 815–823, 2015.

[10] Samadhi P K. Wickrama Arachchilage and Ebroul Izquierdo. Deep-learned faces: a survey. *EURASIP Journal on Image and Video Processing*, 2020(1):25, 2020.

[11] Weiyang Liu, Yandong Wen, Zhiding Yu, Ming Li, Bhiksha Raj, and Le Song. Sphereface: Deep hypersphere embedding for face recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 212–220, 2017.

[12] Hao Wang, Yitong Wang, Zheng Zhou, Xing Ji, Dihong Gong, Jingchao Zhou, Zhifeng Li, and Wei Liu. Cosface: Large margin cosine loss for deep face recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 5265–5274, 2018.

[13] Jiankang Deng, Jia Guo, Niannan Xue, and Stefanos Zafeiriou. Arcface: Additive angular margin loss for deep face recognition. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 4690–4699, 2019.

[14] Gang Qian, Shamik Sural, Yuelong Gu, and Sakti Pramanik. Similarity between euclidean and cosine angle distance for nearest neighbor queries. In *Proceedings of the 2004 ACM symposium on Applied computing*, pages 1232–1237, 2004.

[15] Anil K. Jain, Karthik Nandakumar, and Arun Ross. 50 years of biometric research: Accomplishments, challenges, and opportunities. *Pattern Recognition Letters*, 79:80–105, 2016.

[16] Yinpeng Dong, Hang Su, Baoyuan Wu, Zhifeng Li, Wei Liu, Tong Zhang, and Jun Zhu. Efficient decision-based black-box adversarial attacks on face recognition. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 7714–7722, 2019.

[17] Bharat Yalavarthi, Arjun Ramesh Kaushik, Arun Ross, Vishnu Boddeti, and Nalini Ratha. Enhancing privacy in face analytics using fully homomorphic encryption. In *2024 IEEE 18th International Conference on Automatic Face and Gesture Recognition (FG)*, pages 1–9. IEEE, 2024.

[18] Zhang Rui and Zheng Yan. A survey on biometric authentication: Toward secure and privacy-preserving identification. *IEEE access*, 7:5994–6009, 2018.

[19] Zekeriya Erkin, Martin Franz, Jorge Guajardo, Stefan Katzenbeisser, Inald Lagendijk, and Tomas Toft. Privacy-preserving face recognition. In *Privacy Enhancing Technologies: 9th International Symposium, PETS 2009, Seattle, WA, USA, August 5-7, 2009. Proceedings 9*, pages 235–253. Springer, 2009.

[20] Reginald L Lagendijk, Zekeriya Erkin, and Mauro Barni. Encrypted signal processing for privacy protection: Conveying the utility of homomorphic encryption and multiparty computation. *IEEE Signal Processing Magazine*, 30(1):82–105, 2012.

[21] Mauro Barni, Giulia Droandi, and Riccardo Lazzeretti. Privacy protection in biometric-based recognition systems: A marriage between cryptography and signal processing. *IEEE Signal Processing Magazine*, 32(5):66–76, 2015.

[22] Maneesh Upmanyu, Anoop M Namboodiri, Kannan Srinathan, and CV Jawahar. Blind authentication: a secure crypto-biometric verification protocol. *IEEE transactions on information forensics and security*, 5(2):255–268, 2010.

[23] Wencheng Yang, Song Wang, Hui Cui, Zhaohui Tang, and Yan Li. A review of homomorphic encryption for privacy-preserving biometrics. *Sensors*, 23(7):3566, 2023.

[24] Craig Gentry. Fully homomorphic encryption using ideal lattices. In *Proceedings of the Forty-First Annual ACM Symposium on Theory of Computing*, STOC '09, page 169–178, New York, NY, USA, 2009. Association for Computing Machinery.

[25] Ilaria Chillotti, Nicolas Gama, Mariya Georgieva, and Malika Izabachène. TFHE: Fast fully homomorphic encryption over the torus. Cryptology ePrint Archive, Paper 2018/421, 2018.

[26] Léo Ducas and Daniele Micciancio. Fhew: bootstrapping homomorphic encryption in less than a second. In *Annual international conference on the theory and applications of cryptographic techniques*, pages 617–640. Springer, 2015.

[27] Zvika Brakerski, Craig Gentry, and Vinod Vaikuntanathan. (leveled) fully homomorphic encryption without bootstrapping. *ACM Transactions on Computation Theory (TOCT)*, 6(3):1–36, 2014.

[28] Junfeng Fan and Frederik Vercauteren. Somewhat practical fully homomorphic encryption. Cryptology ePrint Archive, Paper 2012/144, 2012.

[29] Jung Hee Cheon, Andrey Kim, Miran Kim, and Yongsoo Song. Homomorphic encryption for arithmetic of approximate numbers. Cryptology ePrint Archive, Paper 2016/421, 2016.

[30] Juan Ramón Troncoso-Pastoriza, Daniel González-Jiménez, and Fernando Pérez-González. Fully private noninteractive face verification. *IEEE Transactions on Information Forensics and Security*, 8(7):1101–1114, 2013.

[31] Craig Gentry and Shai Halevi. Implementing gentry's fully-homomorphic encryption scheme. In *Annual international conference on the theory and applications of cryptographic techniques*, pages 129–148. Springer, 2011.

[32] Vishnu Naresh Boddeti. Secure face matching using fully homomorphic encryption. In *2018 IEEE 9th international conference on biometrics theory, applications and systems (BTAS)*, pages 1–10. IEEE, 2018.

[33] Weiyang Liu, Yandong Wen, Zhiding Yu, Ming Li, Bhiksha Raj, and Le Song. Sphereface: Deep hypersphere embedding for face recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 212–220, 2017.

[34] Arun Kumar Jindal, Imtiyazuddin Shaik, Vasudha Vasudha, Srinivasa Rao Chalamala, Rajan Ma, and Sachin Lodha. Secure and privacy preserving method for biometric template protection using fully homomorphic encryption. In *2020 IEEE 19th international conference on trust, security and privacy in computing and communications (TrustCom)*, pages 1127–1134. IEEE, 2020.

[35] Pawel Drozdowski, Nicolas Buchmann, Christian Rathgeb, Marian Margraf, and Christoph Busch. On the application of homomorphic encryption to face identification. In *2019 international conference of the biometrics special interest group (biosig)*, pages 1–5. IEEE, 2019.

[36] Joshua J Engelsma, Anil K Jain, and Vishnu Naresh Boddeti. Hers: Homomorphically encrypted representation search. *IEEE Transactions on Biometrics, Behavior, and Identity Science*, 4(3):349–360, 2022.

[37] Sixue Gong, Vishnu Naresh Boddeti, and Anil K Jain. On the intrinsic dimensionality of image representations. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 3987–3996, 2019.

[38] Gaëtan Pradel and Chris Mitchell. Privacy-preserving biometric matching using homomorphic encryption. In *2021 IEEE 20th International Conference on Trust, Security and Privacy in Computing and Communications (TrustCom)*, pages 494–505. IEEE, 2021.

[39] Tim Esler. facenet-pytorch. `https://github.com/timesler/facenet-pytorch`, 2024. Accessed: 2025.

[40] Qiong Cao, Li Shen, Weidi Xie, Omkar M Parkhi, and Andrew Zisserman. Vggface2: A dataset for recognising faces across pose and age. In *2018 13th IEEE international conference on automatic face & gesture recognition (FG 2018)*, pages 67–74. IEEE, 2018.

[41] Kaipeng Zhang, Zhanpeng Zhang, Zhifeng Li, and Yu Qiao. Joint face detection and alignment using multitask cascaded convolutional networks. *IEEE signal processing letters*, 23(10):1499–1503, 2016.

[42] Jonathon Shlens. A tutorial on principal component analysis. *arXiv preprint arXiv:1404.1100*, 2014.

[43] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[44] Marc Joye. Guide to fully homomorphic encryption over the [discretized] torus. *Cryptology ePrint Archive*, 2021.

[45] Jakub Klemsa. Hitchhiker's guide to the tfhe scheme. *Journal of Cryptographic Engineering*, 2023.

[46] Zama AI. Fhetotfhescalar.cpp - conversion from fhe to tfhe scalar operations in concrete compiler. `https://github.com/zama-ai/concrete/blob/b6a43cfc5c43c66eca52ef516dcf42fc2df62421/compilers/concrete-compiler/compiler/lib/Conversion/FHEToTFHEScalar/FHEToTFHEScalar.cpp`, 2025. Accessed: 2025-05-22.

[47] Hugo Krawczyk. Cryptographic extraction and key derivation: The HKDF scheme. Cryptology ePrint Archive, Paper 2010/264, 2010.

[48] Hugo Krawczyk and Pasi Eronen. HMAC-based Extract-and-Expand Key Derivation Function (HKDF). RFC 5869, May 2010.

[49] Olivier Pereira and Thomas Peters. Lmat2450 – cryptography. `https://sites.uclouvain.be/archives-portail/cdc2023/cours-2023-lmat2450`, 2023. Université catholique de Louvain, Louvain-la-Neuve, Belgium. 5 ECTS credits.

[50] Kerem Okyay. epl_fhe_thesis: Code for master's thesis on fully homomorphic encryption for face verification. `https://github.com/BayKeremm/fhe_face_verification`, 2025.

[51] Zama Team. Gpu acceleration | concrete. `https://docs.zama.ai/concrete/execution-analysis/gpu_acceleration`. Accessed: 2025-05-21.

[52] Vikas Gopal and Oguz Ozturk. Homomorphic processing unit (hpu) for accelerating secure computations under homomorphic encryption, 2019. Issued May 21, 2019.

[53] Zama Team. Tfhe-rs v1.2.0. `https://github.com/zama-ai/tfhe-rs/releases/tag/tfhe-rs-1.2.0`, 2024. Release date: 2024-05-20. Includes HPU backend and various improvements. Accessed: 2025-05-21.

[54] Michiel Van Beirendonck, Jan-Pieter D'Anvers, Furkan Turan, and Ingrid Verbauwhede. FPT: a fixed-point accelerator for torus fully homomorphic encryption. Cryptology ePrint Archive, Paper 2022/1635, 2022.

Appendix A

Full Evaluation Tables

Here we provide complete tables referenced in Chapter 6.

A.1 Matching Performance in the Plaintext Domain

A.1.1 Principal Component Analysis

dim	TPR@0.01%	TPR@0.1%	TPR@1%	Variance
10	52.20±4.89	64.70±4.40	87.60±2.54	34.47±4.73
11	46.70±4.81	62.03±3.72	87.57±2.81	37.19±4.82
12	54.60±5.44	65.63±3.54	90.13±2.47	39.77±4.78
13	55.00±6.73	68.73±6.42	91.93±1.58	42.26±4.30
14	54.60±5.46	71.47±3.92	92.37±1.95	44.71±4.29
15	58.33±5.24	72.40±4.17	93.40±1.58	47.09±4.84
16	61.0±4.43	72.10±5.11	95.30±1.15	49.38±4.81
17	62.70±5.26	77.20±3.46	95.97±1.07	51.63±5.36
18	61.23±6.01	78.77±3.92	96.83±1.44	53.81±5.62
19	72.97±4.48	84.60±4.21	97.47±1.09	55.91±5.49
20	78.77±5.97	88.40±3.07	97.67±1.15	57.97±5.00
21	82.70±4.97	89.73±2.40	97.90±1.01	60.0±5.40
22	84.13±4.57	90.13±2.49	98.20±0.87	62.01±4.93
23	85.87±2.96	91.27±1.95	98.23±0.79	63.98±4.76
24	85.10±3.25	91.40±2.03	98.3±0.69	65.89±4.83
25	84.97±2.88	91.47±1.73	98.50±0.78	67.70±4.53
26	83.90±3.08	92.40±1.98	98.60±0.76	69.47±4.47
27	83.20±3.15	94.40±2.28	98.73±0.77	71.22±4.33
28	83.23±3.42	94.77±1.71	98.83±0.79	72.90±4.26
29	87.90±2.62	94.97±1.62	98.97±0.75	74.54±3.96
30	88.60±2.48	95.90±1.45	99.13±0.85	76.13±3.82
31	88.47±3.01	96.80±0.93	99.17±0.72	77.70±2.94
32	88.80±2.94	96.90±0.79	99.23±0.63	79.24±2.77
33	88.83±2.74	96.87±0.93	99.43±0.65	80.74±2.56
34	88.87±2.50	96.93±1.34	99.37±0.66	82.23±2.40
35	88.73±3.41	96.97±1.32	99.43±0.60	83.69±2.63
36	88.97±3.74	97.23±1.53	99.43±0.60	85.11±2.44
37	88.33±3.98	97.53±1.22	99.47±0.52	86.51±2.26
38	87.57±4.40	97.87±1.22	99.50±0.50	87.85±2.43
39	87.23±4.60	97.70±1.25	99.50±0.50	89.16±2.11
40	86.77±4.81	97.67±1.3	99.40±0.57	90.45±1.68
41	88.13±4.42	97.87±1.32	99.37±0.6	91.7±1.18
42	87.8±4.29	97.87±1.36	99.4±0.55	92.89±1.12
43	87.3±4.39	98.20±1.07	99.4±0.59	94.05±0.71
44	86.87±4.34	98.27±1.02	99.37±0.57	95.17±0.62
45	86.93±4.3	98.23±1.05	99.37±0.6	96.22±0.94
46	86.9±4.29	98.1±1.17	99.4±0.59	97.25±1.04
47	86.73±4.21	97.97±1.28	99.47±0.52	98.24±1.18
48	85.97±4.43	97.87±1.36	99.53±0.45	99.18±0.87
49	85.63±4.82	97.77±1.33	99.5±0.48	100.0±0.0
50	85.63±4.82	97.77±1.33	99.5±0.48	100.0±0.0

Table A.1: Performance metrics across reduced dimensions with ± standard deviations, expressed in percentages (%).

A.1.2 Min-max Normalization Strategies

Dim	Method	TPR@0.01%	TPR@0.1%	TPR@1%
40	G	86.77±4.81	97.67±1.30	99.40±0.57
	PF	84.80±4.26	96.67±1.42	99.17±0.70
41	G	88.13±4.42	97.87±1.32	99.37±0.60
	PF	88.33±4.24	97.03±1.52	99.17±0.79
42	G	87.80±4.29	97.87±1.36	99.40±0.55
	PF	87.57±3.99	97.20±1.31	99.10±0.80
43	G	87.30±4.39	98.20±1.07	99.40±0.59
	PF	87.20±4.03	96.87±1.35	99.07±0.79
44	G	86.87±4.34	98.27±1.02	99.37±0.57
	PF	86.27±4.32	97.17±1.34	99.03±0.77

Table A.2: Comparison of global (G) and per-feature (PF) min-max normalization methods after applying PCA. All values are reported in percentages (%) with standard deviations.

A.1.3 Quantization Strategies

Dim	BW	TPR@0.01%	TPR@0.1%	TPR@1%
40	2	45.73±13.06	64.27±8.42	83.30±3.90
	3	74.40±11.91	90.33±3.29	96.87±1.49
	4	84.70±5.55	96.97±1.41	99.20±0.62
	5	87.23±3.85	96.23±2.15	99.17±0.72
41	2	46.47±11.88	68.10±5.60	83.33±2.70
	3	76.57±10.86	90.10±2.94	97.17±1.33
	4	86.30±5.56	97.07±1.57	99.27±0.61
	5	89.03±3.53	96.17±2.44	99.17±0.75
42	2	46.40±12.64	68.10±5.28	84.07±4.48
	3	75.73±10.52	90.70±3.26	97.43±1.05
	4	85.87±5.72	97.13±1.58	99.27±0.73
	5	88.83±3.46	96.70±2.01	99.23±0.65
43	2	50.63±13.57	69.17±5.22	85.43±3.34
	3	77.00±9.23	90.93±3.15	97.03±1.25
	4	85.43±6.05	97.07±1.55	99.23±0.79
	5	88.43±4.03	96.67±1.90	99.23±0.73
44	2	54.07±15.12	70.27±6.42	85.57±2.25
	3	76.93±9.51	91.10±2.84	97.63±1.12
	4	85.10±5.91	97.33±1.34	99.13±0.81
	5	88.03±3.91	96.63±2.03	99.27±0.66

Table A.3: Matching performance in the plaintext domain as a function of quantization bit-width (BW). For PCA-reduced embeddings across different dimensions, this table reports TPR at FPR = 0.01 %, 0.1 %, and 1 % under the pipeline: PCA→global min–max normalization→global quantization. Values are shown in percentages with standard deviations.

Dim	BW	TPR@0.01%	TPR@0.1%	TPR@1%
40	2	35.60±10.33	46.27±5.40	69.70±4.65
	3	76.93±6.99	85.87±2.50	96.00±1.27
	4	83.87±3.57	94.83±1.59	98.70±0.90
	5	82.47±6.70	96.33±1.57	99.07±0.81
41	2	36.07±10.73	44.47±5.59	68.83±5.82
	3	80.07±5.86	87.40±1.93	95.83±1.67
	4	87.03±3.91	95.00±1.86	98.87±0.85
	5	86.47±5.60	96.73±1.51	99.13±0.73
42	2	37.00±8.58	46.60±6.45	68.73±4.84
	3	80.40±5.30	86.97±2.46	96.03±1.22
	4	86.10±3.85	95.03±2.00	98.77±0.83
	5	85.90±5.51	96.67±1.42	99.03±0.78
43	2	36.70±8.41	45.83±5.13	68.60±4.47
	3	79.57±5.24	87.53±2.42	96.10±1.16
	4	85.40±3.81	94.87±2.51	98.93±0.70
	5	85.70±5.58	96.37±1.43	99.03±0.78
44	2	36.57±9.10	45.83±3.47	70.77±3.01
	3	79.70±5.81	87.70±2.24	95.77±1.80
	4	84.57±4.54	95.20±2.28	98.90±0.73
	5	85.03±5.68	96.27±1.73	98.97±0.85

Table A.4: Matching performance in the plaintext domain as a function of quantization bit-width (BW). For PCA-reduced embeddings at Dim = 40 and 44, this table reports accuracy and TPR at FPR = 0.01 %, 0.1 %, and 1 % under the pipeline: PCA→global min–max normalization→per-dimension quantization. Values are shown in percentages with standard deviations.

Dim	BW	TPR@0.01%	TPR@0.1%	TPR@1%
40	2	53.47±6.40	60.63±5.96	85.63±3.34
	3	80.83±7.92	92.83±2.36	97.80±0.95
	4	84.83±5.62	96.30±1.82	99.17±0.73
	5	83.10±4.40	94.20±1.80	98.37±0.78
41	2	54.50±6.57	63.63±5.76	84.73±2.32
	3	84.27±6.32	93.37±1.88	98.27±0.84
	4	87.87±4.45	96.70±1.77	99.20±0.70
	5	87.13±4.33	94.60±2.29	98.47±0.81
42	2	55.73±6.42	65.90±6.13	86.60±2.56
	3	83.60±6.59	93.37±1.93	98.13±0.93
	4	87.13±4.49	96.90±1.55	99.13±0.70
	5	86.63±4.01	94.83±2.26	98.50±0.83
43	2	58.47±7.06	67.83±5.21	85.83±2.16
	3	84.27±6.29	93.63±1.63	98.03±1.08
	4	86.60±4.90	96.73±1.62	98.97±0.89
	5	86.37±4.07	94.77±2.47	98.60±0.76
44	2	60.37±7.54	68.20±6.50	86.93±3.34
	3	83.90±7.31	93.30±1.35	97.87±1.35
	4	85.90±4.89	96.80±1.69	99.00±0.80
	5	85.83±3.72	94.73±2.29	98.60±0.77

Table A.5: Matching performance in the plaintext domain as a function of quantization bit-width (BW). For PCA-reduced embeddings at Dim = 40 and 44, this table reports accuracy and TPR at FPR = 0.01 %, 0.1 %, and 1 % under the pipeline: PCA→per-feature min–max normalization→global quantization. Values are shown in percentages with standard deviations.

Dim	BW	TPR@0.01%	TPR@0.1%	TPR@1%
40	2	45.00±8.53	54.57±5.75	76.80±3.49
	3	74.37±8.10	86.93±3.05	96.63±1.66
	4	80.77±4.20	92.10±2.47	97.87±1.31
	5	78.67±5.07	91.70±2.73	97.60±1.09
41	2	46.00±6.45	56.00±7.44	78.80±2.10
	3	79.23±5.90	89.63±2.85	96.53±1.90
	4	86.57±2.72	92.97±2.50	97.93±1.33
	5	84.33±4.18	92.93±2.40	97.73±1.12
42	2	48.10±8.62	55.47±6.69	78.70±3.41
	3	80.00±5.58	89.53±2.27	96.67±1.72
	4	85.77±2.79	93.57±2.55	97.93±1.19
	5	83.77±4.23	92.47±2.44	97.53±1.01
43	2	47.27±6.64	60.53±6.18	80.43±2.08
	3	81.33±5.84	89.50±2.18	96.37±1.80
	4	85.43±2.67	93.43±2.63	97.73±1.24
	5	83.27±4.44	92.10±2.36	97.57±1.14
44	2	49.13±9.57	60.47±6.48	78.90±2.32
	3	80.47±5.15	89.10±2.22	96.40±1.87
	4	85.00±2.46	93.93±2.11	97.63±1.52
	5	83.27±4.32	92.23±1.72	97.33±1.22

Table A.6: Matching performance in the plaintext domain as a function of quantization bit-width (BW). For PCA-reduced embeddings at Dim = 40 and 44, this table reports accuracy and TPR at FPR = 0.01 %, 0.1 %, and 1 % under the pipeline: PCA→per-feature min–max normalization→per-dimension quantization. Values are shown in percentages with standard deviations.

A.2 Matching Performance in the Encrypted Domain

A.2.1 Pipeline Performance

Dim	BW	TPR@0.01%	TPR@0.1%	TPR@1%
40	4	84.70±5.55	96.97±1.41	99.2±0.62
	5	87.93±4.33	97.37±1.29	99.37±0.66
41	4	86.30±5.56	97.07±1.57	99.27±0.61
	5	89.43±3.93	97.77±1.43	99.40±0.59
42	4	85.87±5.72	97.13±1.58	99.27±0.73
	5	89.27±3.75	97.83±1.38	99.40±0.55
43	4	85.43±6.05	97.07±1.55	99.23±0.79
	5	88.60±4.09	97.90±1.10	99.4±0.51
44	4	85.10±5.91	97.33±1.34	99.13±0.81
	5	88.13±3.91	97.87±1.33	99.33±0.61

Table A.7: Matching performance in encrypted domain as a function of quantization bit-width (BW). For PCA-reduced embeddings at Dim = 40 and 44, this table reports TPR at FPR = 0.01 %, 0.1 %, and 1 % under the pipeline: PCA→global min–max normalization→global quantization. Values are shown in percentages with standard deviations.

Dim	BW	TPR@0.01%	TPR@0.1%	TPR@1%
40	4	83.87±3.57	94.83±1.59	98.7±0.90
	5	83.77±5.09	96.37±1.55	99.1±0.82
41	4	87.03±3.91	95.0±1.86	98.87±0.85
	5	87.17±4.94	96.83±1.56	99.13±0.73
42	4	86.1±3.85	95.03±2.00	98.77±0.83
	5	86.5±4.91	96.87±1.45	99.07±0.80
43	4	85.4±3.81	94.87±2.51	98.93±0.70
	5	86.2±5.17	96.47±1.43	99.07±0.79
44	4	84.57±4.54	95.2±2.28	98.9±0.73
	5	85.47±5.36	96.47±1.62	99.03±0.81

Table A.8: Matching performance in encrypted domain as a function of quantization bit-width (BW). For PCA-reduced embeddings at Dim = 40 and 44, this table reports accuracy and TPR at FPR = 0.01 %, 0.1 %, and 1 % under the pipeline: PCA→global min–max normalization→per-dimension quantization. Values are shown in percentages with standard deviations.

Dim	BW	TPR@0.01%	TPR@0.1%	TPR@1%
40	4	84.83±5.62	96.3±1.82	99.17±0.73
	5	85.47±4.40	96.73±1.39	99.13±0.81
41	4	87.87±4.45	96.7±1.77	99.2±0.70
	5	89.03±4.20	97.03±1.60	99.17±0.83
42	4	87.13±4.49	96.9±1.55	99.13±0.70
	5	88.43±4.24	96.83±1.34	99.13±0.81
43	4	86.6±4.90	96.73±1.62	98.97±0.89
	5	87.83±4.21	96.67±1.41	99.03±0.82
44	4	85.9±4.89	96.8±1.69	99.0±0.80
	5	87.13±4.16	96.9±1.58	99.13±0.78

Table A.9: Matching performance in encrypted domain as a function of quantization bit-width (BW). For PCA-reduced embeddings at Dim = 40 and 44, this table reports accuracy and TPR at FPR = 0.01 %, 0.1 %, and 1 % under the pipeline: PCA→per-feature min–max normalization→global quantization. Values are shown in percentages with standard deviations.

Dim	BW	TPR@0.01%	TPR@0.1%	TPR@1%
40	4	80.8±4.21	92.1±2.47	97.87±1.31
	5	79.5±4.05	93.8±2.13	98.3±1.15
41	4	86.6±2.75	92.97±2.50	97.97±1.29
	5	86.23±3.38	94.37±2.06	98.3±1.25
42	4	85.77±2.79	93.57±2.55	97.93±1.19
	5	85.17±3.58	94.47±1.64	98.17±1.01
43	4	85.43±2.67	93.5±2.63	97.73±1.24
	5	84.7±3.89	94.2±2.26	98.23±1.08
44	4	85.0±2.46	93.93±2.11	97.63±1.52
	5	84.63±4.04	94.73±1.84	97.93±1.24

Table A.10: Matching performance in encrypted domain as a function of quantization bit-width (BW). For PCA-reduced embeddings at Dim = 40 and 44, this table reports accuracy and TPR at FPR = 0.01 %, 0.1 %, and 1 % under the pipeline: PCA→per-feature min–max normalization→per-dimension quantization. Values are shown in percentages with standard deviations.

A.2.2 Circuit Runtime: Quantization Bit-Width and LUT Impact

Dim	BW	MBW	Computation Time (s)
40	4	12	0.173 ± 0.066
	5	14	0.920 ± 0.119
41	4	11	0.203 ± 0.037
	5	14	0.902 ± 0.062
42	4	12	0.173 ± 0.023
	5	14	0.937 ± 0.104
43	4	12	0.183 ± 0.024
	5	14	0.915 ± 0.075
44	4	12	0.174 ± 0.025
	5	14	0.915 ± 0.081

Table A.11: Encrypted computation time (mean ± standard deviation) and corresponding MBW required for different combinations of embedding dimensionality and quantization bit-width (BW). Results are reported for a preprocessing pipeline consisting of PCA → global min–max normalization → global quantization, and evaluated on an AMD EPYC 7763 using 16 cores out of 64.

A.2.3 Effect of Removing Min-Max Normalization

Dim	BW	TPR@0.01%	TPR@0.1%	TPR@1%
40	4	63.63±32.62	96.03±2.64	99.4±0.53
	5	44.5±36.35	94.63±4.05	99.33±0.54
41	4	67.47±31.44	96.1±3.10	99.37±0.59
	5	47.37±37.61	94.93±4.18	99.4±0.55
42	4	68.13±30.53	96.07±2.88	99.27±0.74
	5	25.1±34.92	94.0±3.97	99.33±0.58
43	4	59.57±32.23	95.87±2.94	99.3±0.71
	5	26.27±35.32	93.77±4.18	99.33±0.60
44	4	59.7±32.29	95.9±3.00	99.3±0.71
	5	26.87±35.72	93.37±4.18	99.37±0.57

Table A.12: Matching performance in encrypted domain of the pipeline without min–max normalization for different quantization bit-widths (BW): PCA→global quantization. Values are shown in percentages with standard deviations.

Dim	BW	MBW	Computation Time (s)
40	4	11	0.165 ± 0.055
	5	14	0.928 ± 0.102
41	4	12	0.174 ± 0.030
	5	14	0.956 ± 0.083
42	4	12	0.167 ± 0.021
	5	14	0.906 ± 0.066
43	4	12	0.188 ± 0.026
	5	14	0.950 ± 0.080
44	4	12	0.183 ± 0.024
	5	14	0.954 ± 0.098

Table A.13: Encrypted computation time (mean ± standard deviation) and corresponding MBW required for different combinations of embedding dimensionality and quantization bit-width (BW). Results are reported for a preprocessing pipeline consisting of PCA → global quantization, and evaluated on an AMD EPYC 7763 using 16 cores out of 64.

Appendix B

TPR @ FPR Definitions, ROC curve, and Cross-Validation Protocol

B.1 Definitions

Let $y_i \in \{0, 1\}$ denote the ground-truth label for pair i, where 1 indicates a same-person pair and 0 a different-person pair. Let s_i be the associated similarity score (or a distance score, with smaller values indicating higher similarity).

For any decision threshold t, the following metrics are defined:

$$\mathrm{TPR}(t) = \frac{\#\{\, i : y_i = 1 \ \wedge \ s_i \geq t\}}{\#\{\, i : y_i = 1\}},$$
$$\mathrm{FPR}(t) = \frac{\#\{\, i : y_i = 0 \ \wedge \ s_i \geq t\}}{\#\{\, i : y_i = 0\}}.$$

Here, **True Positive Rate (TPR)**—also known as *recall* or *sensitivity*—measures the proportion of actual positives correctly classified. The **False Positive Rate (FPR)** quantifies the proportion of negatives incorrectly classified as positives.

These can also be expressed in terms of confusion matrix components:

$$\mathrm{TPR} = \frac{\mathrm{TP}}{\mathrm{TP} + \mathrm{FN}}, \quad \mathrm{FPR} = \frac{\mathrm{FP}}{\mathrm{FP} + \mathrm{TN}}.$$

B.2 ROC Curve

By sweeping the threshold t over all possible score values, we obtain a parametric curve:

$$\Big(\mathrm{FPR}(t),\, \mathrm{TPR}(t)\Big),$$

known as the *Receiver Operating Characteristic (ROC)* curve. This curve captures the trade-off between TPR and FPR at varying decision thresholds.

B.2.1 Youden's J Statistic

Youden's J statistic is a summary metric to select an optimal threshold t by balancing sensitivity (TPR) and specificity (1 – FPR). It is defined as:

$$J(t) = \mathrm{TPR}(t) - \mathrm{FPR}(t).$$

The threshold t^* that maximizes $J(t)$ is considered optimal when a single operating point is needed:

$$t^* = \arg\max_t \left[\mathrm{TPR}(t) - \mathrm{FPR}(t)\right].$$

This criterion seeks the point on the ROC curve farthest from the diagonal (i.e., random guessing). It is especially useful when equal weight is given to false positives and false negatives.

B.3 Reading off TPR at a Target FPR

B.3.1 Cross-Validation Thresholding

To obtain an unbiased estimate—without using test labels—a threshold is selected on the training folds and evaluated on the test fold:

1. Collect all similarity scores for negative pairs from the training set:

$$\mathcal{S}_{\text{train}}^{\text{neg}} = \{s_i : y_i = 0\}.$$

2. Choose threshold t to satisfy the desired training FPR= $\alpha\%$:

$$t = \text{percentile}\left(\mathcal{S}_{\text{train}}^{\text{neg}},\, \alpha\%\right).$$

   This ensures that only $\alpha\%$ of training negatives fall below t, achieving target FPR on training data.

3. Apply this threshold t to the test fold, and compute the resulting TPR:

$$\mathrm{TPR}_{\text{test}} = \frac{\#\{\, i : y_i = 1,\, s_i \geq t\}}{\#\{\, i : y_i = 1\}}.$$

Interpretation

The threshold t is fixed using training negatives, simulating deployment conditions. On the test set:

- Predict "same person" if $s_i \geq t$ (or $s_i < t$, for distance-based scores).
- Measure how many test positives satisfy this criterion.

This method gives an unbiased estimate of TPR at FPR = $\alpha\%$ on the test set.

B.4 10-Fold Cross-Validation on LFW

The standard LFW verification protocol divides the 6000 face pairs into 10 folds of 600. For each fold $k \in \{1, \ldots, 10\}$:

1. Train PCA on folds $\{1, \ldots, 10\} \setminus \{k\}$, and select threshold t using the training negatives as described.
2. Project the held-out fold k using the learned PCA transform.
3. Evaluate TPR@$\alpha\%$ using the fixed threshold t and record it.

Final results are reported as the average and standard deviation across folds:

$$\overline{\mathrm{TPR}}@\alpha\% = \frac{1}{10}\sum_{k=1}^{10} \mathrm{TPR}_k@\alpha\%, \tag{B.1}$$

$$\sigma_{\mathrm{TPR}@\alpha\%} = \sqrt{\frac{1}{10}\sum_{k=1}^{10}\left(\mathrm{TPR}_k@\alpha\% - \overline{\mathrm{TPR}}@\alpha\%\right)^2}. \tag{B.2}$$

Appendix C

Acknowledgements and Use of AI tools

C.1 Acknowledgements

I would like to express my gratitude to my supervisors, Prof. François-Xavier Standaert and Prof. Thomas Peters, for their guidance, support, and insights throughout the course of this thesis. I also wish to thank the developers at Zama for their helpful responses and technical assistance related to the `Concrete` framework on the community Discord server.

Computational resources have been provided by the supercomputing facilities of the Université catholique de Louvain (CISM/UCL) and the Consortium des Équipements de Calcul Intensif en Fédération Wallonie Bruxelles (CÉCI) funded by the Fond de la Recherche Scientifique de Belgique (F.R.S.-FNRS) under convention 2.5020.11 and by the Walloon Region.

C.2 Use of AI tools

During the preparation of this thesis, AI-assisted tools—specifically OpenAI's ChatGPT and Google's Gemini—were utilized strictly for the purpose of enhancing the clarity, coherence, and linguistic quality of the text. The content, analysis, and conclusions were entirely developed by the author. No scientific ideas or original contributions were generated by AI. Its use was limited to language refinement, in a way comparable to conventional grammar or style correction tools.

UNIVERSITÉ CATHOLIQUE DE LOUVAIN
École polytechnique de Louvain
Rue Archimède, 1 bte L6.11.01, 1348 Louvain-la-Neuve, Belgique | **www.uclouvain.be/epl**