Biometric Homomorphic Matching

Vincent Luke Ventura

Cyber Security, EPFL / ETHZ

Semester Project

January 9, 2026

Responsible
Prof. Serge Vaudenay
EPFL / LASEC

Abstract

Finger-vein biometrics offer strong security properties due to the internal, difficult-to-observe nature of vein patterns. However, deploying biometric matching on untrusted servers raises significant privacy concerns, as both the stored model and the authentication probe constitute sensitive personal data. This project investigates the feasibility of performing the computationally intensive matching step of the Miura finger-vein pipeline under homomorphic encryption, allowing an untrusted server to evaluate the match without ever observing the biometric data in the clear.

We design and implement two homomorphic variants of the frequency-domain matching block using the CKKS scheme in OpenFHE: a baseline approach using generic matrix–vector operations, and an optimized design exploiting the structure of the Fast Fourier Transform. Both variants compute a cross-correlation surface from encrypted frequency-domain representations and optionally produce an L^{16}-type approximation of the maximum correlation score.

Our experiments show that the FFT-based implementation reduces runtime from approximately 240 seconds (baseline) to around 40–55 seconds for the non-L^{16} pipeline at 10 threads when key caching is enabled (about 53 seconds without caching). However, the L^{16} scoring introduces a fundamental limitation: the normalization required to prevent modular wrap-around is input-dependent, making the full encrypted scoring unreliable in practice. The non-L^{16} variant, which returns the encrypted correlation matrix to the client, remains numerically stable but still incurs a roughly 80-fold slowdown compared to plaintext matching (hundreds of milliseconds versus $\sim$40–50 seconds in our best configuration). We conclude that while the approach is technically feasible, the performance overhead and the limitations of encrypted maximum approximation make it impractical for real-world deployment without significant further advances in homomorphic encryption efficiency or alternative algorithmic designs.

Contents

1 Introduction

Biometric authentication aims to identify or verify users based on physiological or behavioural characteristics. Finger-vein biometrics are especially attractive in this context because the vein pattern lies inside the finger, is difficult to observe covertly, and can be captured using near-infrared imaging [9]. At LASEC, a series of projects has led to a practical finger-vein recognition pipeline based on Miura et al.'s methods [11, 10], applied within the BioID and BioLocker systems for privacy-friendly identity documents and desktop authentication [2, 6].

In the current plaintext implementation, finger images are first preprocessed and aligned, then transformed into a vein map using a maximum-curvature method [10, 9]. Matching a probe against a stored model is performed in the frequency domain: both images are mapped through a two-dimensional Fourier transform, a matrix multiplication implements correlation over translations, an inverse transform brings the result back to the spatial domain, and finally the maximum value in the resulting array is extracted and used inside a distance measure, which is compared to a threshold in order to determine whether the image is a match or not [9]. This pipeline has been optimised and evaluated in a previous semester project [9], but all computations occur in the clear and therefore require a fully trusted matching server.

The goal of this project is to build a proof of concept (PoC) to assess whether it is technically and practically sensible to move the computationally heavy matching step to an *untrusted* server using homomorphic encryption. To keep the scope focused, we restrict attention to the core frequency-domain matching block: once both the model and the probe have been transformed into the Fourier domain, we move the subsequent matrix multiplication, inverse Fourier transform and the maximum calculation into the encrypted domain using a CKKS-style homomorphic encryption scheme [7, 4]. To approximate the maximum, we use an L_{16}-type score. Ideally, given a vector $x = (x_1, \dots, x_N)$, the L_{16} norm would be

$$\|x\|_{16} = \left(\sum_{i=1}^{N} |x_i|^{16} \right)^{1/16}. \tag{1}$$

In practice, we homomorphically compute only the sum

$$S = \sum_{i=1}^{N} |x_i|^{16} \tag{2}$$

in the encrypted domain, and defer the 16th root to the client side after decryption. Concretely, the Fourier-transformed model is encrypted, the Fourier-transformed probe is encoded as plaintext, the server evaluates the matching circuit homomorphically to obtain an encryption of S, and returns this ciphertext to the client. The client then decrypts this value, computes $S^{1/16}$ to obtain an approximation of the original matching score, and continues the remaining steps of the Miura/BioID/BioLocker logic in plaintext. Deferring the 16th root to after decryption avoids additional multiplicative depth and thus keeps the homomorphic computation lighter, while the cost of taking the root in plaintext is negligible.

This design allows long-term cryptographic keys to remain under the user's control (for example, on a smartcard or secure element, as in the BioID and BioLocker architectures [2, 6]), while delegating the expensive linear-algebra operations to a powerful server that never sees the model, the probe, or the resulting score in clear. However, it is not obvious a priori whether such an approach is numerically stable and whether the performance overhead introduced by homomorphic encryption is acceptable in practice. A central aim of this project is therefore to answer these questions experimentally, even if the outcome is that the current approach is not yet competitive.

Contributions and main findings. The contributions of this report are:

- a precise specification of the frequency-domain matching block used in the existing finger-vein pipeline, and of its role within the overall architecture;
- a homomorphic encryption design that moves the matrix multiplication, inverse transform and L_{16}-type score computation of this block into the encrypted domain, with keys held by a client-side trusted component;
- an implementation in OpenFHE together with an empirical evaluation of correctness, runtime and memory usage.

Our experiments show that the homomorphic implementation is functionally correct and can approximate the plaintext scores, but that it introduces a very substantial performance overhead: in our parameters, the FFT-optimized non-L_{16} variant that returns the matrix after the inverse transform completes in roughly 40–55 seconds at 10 threads when keys are cached (about 53 seconds without caching), while the full L_{16} variant requires about 80–106 seconds. Given that the plaintext pipeline completes the same work in well under a second, this corresponds to an overhead of roughly two orders of magnitude ($\sim$80$\times$) even in the best configuration. As a PoC, this demonstrates both that the approach is feasible and that significant further optimisation or architectural changes would be required before it becomes practical in a real system.

2 Scope and Context

2.1 Threat Model

We assume an *honest-but-curious* (also called *passive* or *semi-honest*) adversary model for the server. Under this model, the server follows the prescribed protocol exactly—it performs the correct homomorphic operations, does not modify ciphertexts maliciously, and returns the correct results. However, the server is assumed to be curious: it may attempt to learn information about the encrypted data by analyzing the ciphertexts it receives, the intermediate values it computes, or any observable access patterns.

The security of our construction relies on the semantic security of the CKKS scheme, which ensures that ciphertexts reveal no information about the underlying plaintexts to a computationally bounded adversary (under standard lattice hardness assumptions). The server therefore learns nothing beyond what can be inferred from the ciphertext sizes and the number of operations performed.

We explicitly *do not* consider *malicious* (active) adversaries in this work. A malicious server could deviate from the protocol—for example, by injecting carefully crafted noise into ciphertexts or by returning incorrect results designed to leak information about the secret key. Recent work has shown that certain active attacks on approximate HE schemes (including CKKS) can potentially recover secret keys if the adversary can observe decrypted results of maliciously constructed ciphertexts. Defending against such attacks requires additional countermeasures (e.g., ciphertext verification, noise flooding, or protocol-level checks) that are outside the scope of this proof-of-concept implementation. For a practical deployment, these considerations would need to be addressed.

2.2 Project Focus

This project focuses on the implementation and empirical evaluation of homomorphic encryption for a specific biometric matching pipeline. The primary research effort was directed toward understanding how to efficiently realize the 2D inverse FFT and related linear operations within the CKKS scheme, rather than conducting a comprehensive survey of prior work on privacy-preserving biometrics.

Homomorphic encryption for biometric applications is an active research area, with existing work on face recognition, fingerprint matching, and related modalities. A systematic review of this literature was outside the scope of the present semester project, but would be a valuable direction for future work—particularly to identify alternative algorithmic approaches that may be better suited to the encrypted domain.

The cryptographic foundations of our implementation (the CKKS scheme and its properties) are well established in the literature [4, 7], and we build directly on the OpenFHE library [1] without modification to the underlying cryptographic primitives. The biometric pipeline itself follows the Miura methods as implemented in prior LASEC projects [9, 2, 6].

3 Mathematical background

To motivate the homomorphic design choices, we isolate the exact plaintext subroutine of the existing Miura-style matching pipeline that we aim to move to an untrusted server. In the plaintext implementation, once the model and probe have been mapped to the Fourier domain, the pipeline computes a translation-invariant cross-correlation surface and then extracts a max-like score from it.

3.1 Frequency-domain correlation block

Let $H \times W$ denote the image dimensions and let $\widehat{M}, \widehat{P} \in \mathbb{C}^{H\times W}$ be the 2D discrete Fourier transforms of the (preprocessed) model M and probe P. The core block computes

$$C = \mathcal{F}^{-1}\big(\widehat{M} \odot \overline{\widehat{P}}\big) \in \mathbb{C}^{H\times W}, \tag{3}$$

where $\odot$ denotes element-wise multiplication and the overline denotes complex conjugation. Concretely, the OpenCV [3] call `mulSpectrums` with `conjB=true` implements the element-wise product

$$\big(\widehat{M} \odot \overline{\widehat{P}}\big)_{i,j} = \widehat{M}_{i,j} \cdot \overline{\widehat{P}_{i,j}} \qquad (0 \le i < H,\ 0 \le j < W), \tag{4}$$

and `idft` computes the inverse transform $\mathcal{F}^{-1}(\cdot)$. In the plaintext pipeline, a maximum over the entries of C is then extracted and used as a matching score (or as an ingredient in a distance measure and threshold test). Figure 1 illustrates a typical correlation surface, showing a distinct peak at the translation offset where the model and probe align best.

In this project we focus on homomorphically evaluating the mapping

$$(\widehat{M}, \widehat{P}) \longmapsto C = \mathcal{F}^{-1}\big(\widehat{M} \odot \overline{\widehat{P}}\big), \tag{5}$$

and on producing an encrypted max-like summary statistic of C (Section 1 discusses our L_{16}-type surrogate).

3.2 The choice of encryption scheme

The operations in (3) are dominated by arithmetic over *complex-valued* Fourier coefficients and by linear transforms (i.e., additions, scalar multiplications, and structured permutations/rotations). Moreover, the intermediate values are inherently *approximate*: the FFT/IFFT pipeline itself is

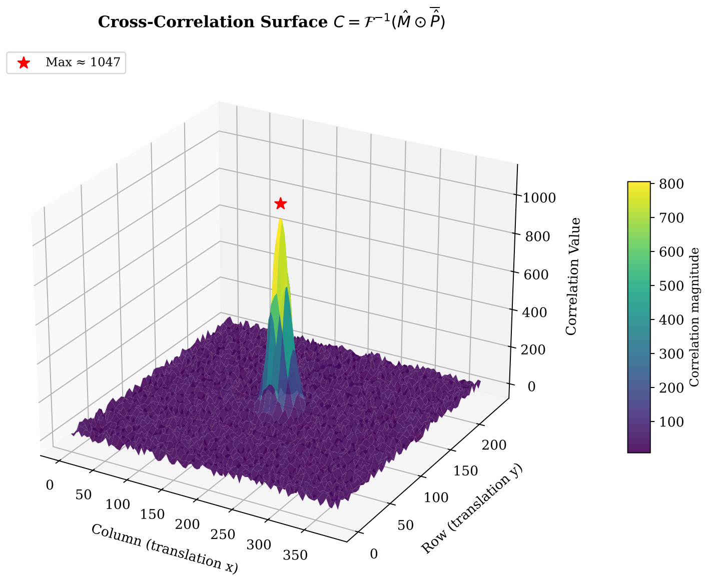

Figure 1: Cross-correlation surface $C = \mathcal{F}^{-1}(\hat{M} \odot \overline{\hat{P}})$ showing a peak at the best-matching translation offset. The height of this peak is extracted as the matching score.

numerical, and small floating-point errors do not affect the biometric decision as long as they remain well below the matching threshold margin.

These properties align well with the CKKS scheme, which natively supports: (i) approximate arithmetic on real/complex numbers, (ii) SIMD-style packing of many coefficients into one ciphertext, and (iii) efficient evaluation of linear operations (additions and slot rotations) that appear in FFT-style circuits. By contrast, exact-integer schemes such as BFV/BGV would require fixed-point encodings with careful scaling management for complex FFT arithmetic, typically increasing circuit complexity for this specific workload.

For these reasons, we adopt CKKS as the primary scheme for our proof of concept. The remainder of the report makes the above statement concrete: Section 4 starts from a generic linear-algebraic formulation of the transform, while Section 5 presents an optimized FFT-based construction and evaluates its accuracy and runtime.

3.3 Choice of coding environment

As this project builds upon the prior, mainly the optimized pipeline from [9], which is implemented in C++, the choice of programming language was quickly decided to be C++, for easier integration with the existing pipeline and its speed advantage at execution. In order to quickly create an initial POC, it made more sense to build on existing FHE libraries, instead of creating new standalone code, which could still be the subject of future iterations on this project. First tries were made with the Microsoft SEAL library [12], using the default polynomial modulus degree of 2^{16} (65,536). SEAL itself supports larger ring dimensions when configured appropriately and when enough memory is available, but with this prototype configuration we could only store 32 768 slots per ciphertext. Given that the matrices are of dimensions 240 rows of 376 values, that are padded in the previous timeline to 384, in order to allow better transformations since 384 has nicer prime factors than 376, we have a total of 92160 values, so we would have to segment the matrix into at least 3 ciphertexts. After some fruitless attempts, it became clear that being able to place the whole matrix into a single ciphertext would be advantageous (This is looked at from a pure efficiency standpoint in 6. These attempts are not looked into further as they brought no results and have nothing to do with the final implementation. Hence, a switch to the OpenFHE library [1], which we configured with a ring dimension of 262144, thus allowing a ciphertext to hold 131072 values, was made.

4 Baseline Homomorphic Matching without FFT

Before developing an optimized FFT-based implementation, we designed and experimented with a baseline homomorphic matching pipeline that directly implements the relevant linear transforms as generic matrix–vector operations. The aim of this baseline was twofold: (i) to verify that the Miura frequency-domain matching block can be realised under CKKS in a conceptually straightforward way, and (ii) to obtain a concrete understanding of the computational bottlenecks and parameter constraints that arise in practice.

4.1 Target computation

We recall from Section 1 that the plaintext matching pipeline computes the cross-correlation between a model M and a probe P in the Fourier domain. Let $M, P \in \mathbb{R}^{H\times W}$ denote the feature matrices obtained from the vein-extraction stage (for example, $H = 240$ and $W = 384$ in our setting [9]). The cross-correlation surface $C \in \mathbb{R}^{H\times W}$ is given by

$$C(u,v) = \sum_{x,y} M(x,y) \cdot P(x-u, y-v), \tag{6}$$

and can be computed via the Convolution Theorem as

$$C = \mathcal{F}^{-1}\big(\widehat{M} \odot \overline{\widehat{P}}\big), \tag{7}$$

where $\widehat{M} = \mathcal{F}(M)$ and $\widehat{P} = \mathcal{F}(P)$ are the 2D Discrete Fourier Transforms (DFTs) of M and P, respectively, $\mathcal{F}^{-1}$ denotes the inverse 2D-DFT, $\odot$ denotes element-wise multiplication and the overline denotes complex conjugation. The plaintext implementation computes $\widehat{M}$ and $\widehat{P}$ using efficient FFT routines, performs the element-wise product, and then applies an inverse FFT to obtain C.

In the homomorphic setting, we are only interested in the block that starts from the frequency-domain representations and ends with the correlation surface. In the baseline design, we therefore assume that the client provides $\widehat{P}$, and that the server holds $\widehat{M}$ in encrypted form. The homomorphic task of the baseline is to compute

$$C = \mathcal{F}^{-1}\big(\widehat{M} \odot \overline{\widehat{P}}\big) \tag{8}$$

using generic linear transforms implemented as matrix–vector products.

4.2 Linear-algebraic view of the DFT

The standard 1D-DFT of a vector $x \in \mathbb{C}^N$ can be written as

$$X = W_N x, \tag{9}$$

where $W_N \in \mathbb{C}^{N\times N}$ is the DFT matrix with entries

$$(W_N)_{j,k} = \omega^{jk} \quad \text{for} \quad \omega = e^{-2\pi i/N}, \qquad 0 \leq j,k < N. \tag{10}$$

Similarly, the inverse transform is given by

$$x = W_N^{-1} X, \tag{11}$$

where W_N^{-1} has the same structure (up to scaling and complex conjugation of ω).

A 2D-DFT on an image $M \in \mathbb{C}^{H\times W}$ can be realised separably by first transforming each row and then each column. Equivalently, if we flatten M into a vector $\operatorname{vec}(M)$ using row-major order, the 2D-DFT is a matrix–vector product with a sparse but structured matrix $T \in \mathbb{C}^{HW\times HW}$:

$$\operatorname{vec}(\widehat{M}) = T \cdot \operatorname{vec}(M), \tag{12}$$

and the same holds for the inverse transform with a matrix T^{-1}.

The baseline homomorphic design uses this linear-algebraic view directly. Rather than exploiting the recursive structure of an FFT, we treat the forward and inverse transforms as generic linear operators and implement them via homomorphic matrix–vector multiplication.

4.3 Homomorphic matrix–vector multiplication via diagonals

CKKS provides native support for component-wise addition and multiplication of encrypted vectors, as well as slot rotations. A full matrix–vector multiplication is not a primitive, but can be implemented using the *diagonal method* [8]. Let $A \in \mathbb{C}^{N \times N}$ and let x be a vector encrypted in a single ciphertext. We can decompose A into its diagonals and write:

$$Ax = \sum_{k=0}^{N-1} \operatorname{diag}_k(A) \odot \operatorname{rot}(x, k), \tag{13}$$

where $\operatorname{rot}(x, k)$ denotes a cyclic rotation of the ciphertext slots by k positions, and $\operatorname{diag}_k(A)$ is the k-th diagonal of A encoded as a plaintext vector. Each term in the sum is realised homomorphically by:

1. a rotation of the input ciphertext by k slots;
2. a component-wise multiplication with the plaintext encoding of $\operatorname{diag}_k(A)$;
3. an accumulation of the results over all k.

For a 1D-DFT of length N, Equation (13) requires N rotations and N ciphertext–plaintext multiplications, plus $N - 1$ additions. In our 2D setting, we apply this method twice per transform (once horizontally and once vertically), and we need forward and inverse transforms for both model and probe, leading to a large number of homomorphic operations.

4.4 Ciphertext layout and packing choices

The image size used in the Miura pipeline is on the order of $H \times W = 240 \times 384$, which corresponds to 92160 entries. With a ring dimension of $n = 262144$ (i.e. 2^{18}) in CKKS, we can in principle pack the entire image into a single ciphertext, leaving some slots unused. In the baseline implementation, we adopt a compromise between simplicity and packing density:

- **Flattening.** We flatten the 2D image into a 1D vector in row-major order, and store it in one ciphertext, embedding each real-valued entry as the real part of a complex slot (the imaginary part is initially zero).
- **Row- and column-wise transforms.** For the separable 2D transforms, we conceptually partition the vector into contiguous blocks corresponding to rows (for the row-wise DFT) and to columns (for the column-wise DFT). Each of these operations can still be expressed in the diagonal form (13), but the diagonals now reflect the 2D structure.

- **Rotation keys.** To support all required rotations, we must generate and store Galois keys for a large set of rotation offsets. This contributes significantly to the memory footprint and key-generation time in this baseline design.

The element-wise product $\widehat{M} \odot \overline{\widehat{P}}$ is implemented as a single ciphertext–plaintext multiplication, since the model is encrypted whereas the probe is only encoded as plaintext. This part is comparatively inexpensive; the dominant cost lies in the homomorphic forward and inverse transforms.

4.5 Empirical behaviour and limitations

The baseline construction is conceptually simple and aligns closely with the linear-algebraic formulation of the DFT. Small-scale experiments on reduced image sizes confirmed that the method correctly reproduces the plaintext transforms and that the numerical error introduced by CKKS remains manageable as long as the number of composed transforms and multiplications is limited.

However, several practical limitations became apparent when scaling up towards realistic image sizes and full 2D transforms:

- **Asymptotic cost.** Even ignoring constants, the matrix–vector formulation incurs $O(N^2)$ work for a transform of length N, whereas an FFT would only require $O(N \log N)$ operations. For N in the hundreds, this difference translates into hundreds of rotations and multiplications per transform, which is prohibitive in the homomorphic setting.

- **Rotation-intensive.** The diagonal method requires a distinct rotation for each diagonal. For our parameter sizes, the number of required rotations dominates the runtime, and the storage of the associated Galois keys dominates the memory usage on the server.

- **Depth consumption.** Each transform consumes part of the modulus chain and contributes noise to the ciphertext. Chaining multiple forward and inverse transforms, followed by the element-wise multiplication and later the L^{16} scoring, leaves little flexibility in parameter selection unless very large moduli are used.

In summary, this baseline approach demonstrates that the Miura matching block can in principle be implemented homomorphically using generic linear transforms, but it is clearly not efficient enough for our target image sizes. These observations motivate the search for a more structured design that reduces the number of rotations and multiplications by leveraging the recursive structure of the FFT. The resulting optimized FFT-based implementation is presented in Section 5. In 7, we showcase the actual results and compare the two approaches.

5 Optimized FFT-based Implementation in OpenFHE

The baseline design in Section 4 confirms that the Miura frequency-domain matching block can be implemented in a homomorphic setting, but it also shows that a naive matrix-based realisation of the DFT is far too costly in terms of rotations, evaluation keys and multiplicative depth. To address these limitations, we designed an improved implementation that exploits the structure of the Fast Fourier Transform (FFT) and is implemented using the CKKS scheme in OpenFHE.

The main idea is to retain the same overall data flow as in the plaintext pipeline—correlation in the Fourier domain followed by an inverse transform and a max-like score—but to implement the inverse transform as a homomorphic 2D-IFFT using a sequence of structured, low-cost linear operations. This section describes the resulting design and its empirical behaviour.

5.1 Encrypted data flow

We keep the division of labour described in Section 1: all preprocessing and forward Fourier transforms are performed on the client side, while the heavy inverse transform and scoring are outsourced to the server. Figure 2 summarizes this division.

- **Client (Trusted)**: The model M is transformed to the frequency domain using a plaintext FFT to obtain $\widehat{M}$. The client then encrypts $\widehat{M}$ using CKKS under a public key, producing a ciphertext ct_M. The corresponding secret key is assumed to be held securely by a client-side trusted component (e.g. a smartcard or secure element). For each authentication attempt, the probe P is processed similarly to obtain $\widehat{P}$, which is *encoded* as a CKKS plaintext polynomial pt_P (not encrypted). The client sends ct_M and pt_P to the server.

- **Server (Untrusted)**: The server receives the ciphertext and plaintext, then performs the following homomorphic operations:

  1. Computes the element-wise spectral multiplication: $\mathsf{ct}_Z = \mathsf{ct}_M \otimes \overline{\mathsf{pt}_P}$
  2. Evaluates the homomorphic 2D inverse FFT on ct_Z to obtain ct_C
  3. (Optionally) Computes the L^{16}-type score $S = \sum |x_i|^{16}$ homomorphically

  The server returns either ct_C (non-L^{16} variant) or ct_S (L^{16} variant) to the client.

- **Client (Decryption)**: The client decrypts the returned ciphertext using the secret key. For the non-L^{16} variant, the client extracts the maximum from the decrypted correlation surface. For the L^{16} variant, the client computes $S^{1/16} \cdot$ norm to recover the approximate maximum score. In both cases, the client continues with the remaining steps of the distance calculation.

Precomputed constants. The homomorphic IFFT requires a set of precomputed plaintext vectors: *twiddle factors* (the complex roots of unity used in FFT butterfly operations) and *masking vectors* (used to select and combine slots during rotations). These constants depend only on the image dimensions and the CKKS parameters, not on the input data. In a deployment scenario, they would be generated once during system setup and stored on the server, avoiding the need to recompute or transmit them for each matching operation.

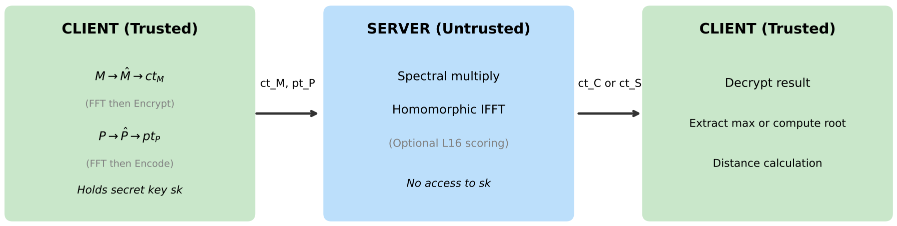

Figure 2: Data flow in the homomorphic matching pipeline: the client performs FFT and encryption, the server computes the homomorphic operations, and the client decrypts and completes the score calculation.

The main technical components are thus (i) an efficient homomorphic 2D-IFFT, and (ii) an implementation of the L^{16}-type score that remains numerically meaningful within the available modulus budget and multiplicative depth.

5.2 Packing strategy and ciphertext layout

We pack the frequency-domain representation $\widehat{M}$ into a single CKKS ciphertext. Each complex Fourier coefficient is embedded directly into one complex slot, so that the real and imaginary parts are represented natively by CKKS.

Given an image of size $H \times W$, we flatten $\widehat{M}$ into a vector of length $N = H \cdot W$, using row-major order:

$$\widehat{M}_{\text{flat}} = \big(\widehat{M}(0,0), \widehat{M}(0,1), \ldots, \widehat{M}(0,W-1), \widehat{M}(1,0), \ldots, \widehat{M}(H-1,W-1)\big).$$

This vector is directly encoded and encrypted into a single ciphertext ct_M. The same layout is used for the probe, so that the component-wise product $\widehat{M} \odot \overline{\widehat{P}}$ corresponds to a single ciphertext–plaintext multiplication in this flattened representation.

The choice of ring dimension $n = 2^{18}$ provides $n/2 = 131072$ complex slots, which is sufficient to host the $H \cdot W$ coefficients in our setting (for example $H \cdot W = 92160$). The remaining slots are left unused. This design keeps the ciphertext count minimal and avoids the overhead of managing multiple ciphertexts for a single image.

5.3 Homomorphic 2D inverse FFT

The 2D inverse FFT is implemented as a sequence of homomorphic 1D inverse FFTs applied first to rows and then to columns. We adopt a radix-2 Cooley–Tukey structure [5], which decomposes a length-N transform into $\log_2 N$ stages of butterfly operations and twiddle-factor multiplications.

For a 1D inverse FFT of length W, the algorithm has the following structure:

1. At each stage $s = 0, 1, \ldots, \log_2 W - 1$, we group the data into blocks of size 2^{s+1}. Within each block, we apply butterflies of the form
$$(a, b) \mapsto (a + \omega b, a - \omega b),$$
where ω is an appropriate twiddle factor.

2. In the homomorphic setting, each stage is realised using a fixed pattern of slot rotations and component-wise additions and multiplications by plaintext twiddle-factor vectors.

Because the data are packed row-wise, each row IFFT can be applied to disjoint segments of the same ciphertext using a single set of rotation and mask patterns. We precompute the necessary twiddle factors and masks as CKKS plaintexts and reuse them across all rows. The column-wise IFFT is implemented analogously, using a different pattern of rotations and masks that reflect the column stride in the flattened layout.

The overall complexity of a length-N FFT is $O(N \log N)$ operations. In our homomorphic implementation, this translates into $O(\log W + \log H)$ stages of structured rotations and multiplications, which is asymptotically much more efficient than the $O(N^2)$ cost of the matrix-based baseline. In practice, this reduces the number of required rotations and Galois keys by more than an order of magnitude for our parameter sizes.

5.4 Encrypted L^{16}-type scoring

Once the inverse FFT has been applied, the ciphertext ct_C encrypts the correlation surface C in the same flattened ordering as above. The plaintext algorithm extracts the maximum value of C and uses it within a distance measure. In the encrypted setting, exact argmax and max are expensive to implement, so we consider an L^{16}-type score as a differentiable approximation: for a vector $x = (x_1, \ldots, x_N)$, the ideal L^{16} norm is

$$\|x\|_{16} = \left(\sum_{i=1}^{N} |x_i|^{16} \right)^{1/16}. \tag{14}$$

We homomorphically compute only the sum

$$S = \sum_{i=1}^{N} |x_i|^{16} \tag{15}$$

on encrypted data, and defer the 16th root to the client after decryption.

5.4.1 Scaling window and robustness limitations

A key difficulty is that the L^{16} surrogate is extremely sensitive to dynamic range. In CKKS, plaintext values are represented approximately, but internally the scheme operates over a residue ring modulo a ciphertext modulus $Q = \prod_{\ell=0}^{L} q_\ell$. In our parameter sets, the modulus chain consists of primes q_ℓ of about 50 bits. This implies that intermediate encoded values must remain safely within the range supported by Q throughout evaluation: if magnitudes become too large relative to Q, the computation may effectively *wrap around* modulo Q, and the decoded result no longer corresponds to the intended real quantity.

This issue is particularly acute for the L^{16} score because it involves repeated multiplications (e.g. forming $|x_i|^{16}$). Preventing wrap-around therefore requires careful scale management (rescaling and, if needed, explicit normalization factors). However, overly aggressive downscaling pushes values toward 0, at which point they become dominated by CKKS approximation error and accumulated noise. Consequently, there is a narrow operating window:

- **Insufficient downscaling:** intermediate values grow too large $\Rightarrow$ wrap-around modulo Q and loss of meaning of the score;
- **Excessive downscaling:** intermediate values become too small $\Rightarrow$ domination by noise and loss of discriminative power.

Ideally, scale selection would be based on a provable upper bound on the peak correlation responses produced by the pipeline. Deriving such a bound is difficult: a naive worst-case bound (e.g. assuming all entries are maximal, such as an all-1 image) is overly pessimistic and does not reflect realistic finger-vein inputs. We therefore estimate a *probabilistic* upper bound empirically. Specifically, we ran 20,100 match simulations using 201 images (out of 780) and recorded the observed maximum values to calibrate a normalization range. This provides a practical working point, but it cannot guarantee correctness in all cases: rare outliers may still exceed the estimated bound and trigger wrap-around.

Importantly, using a single conservative bound introduces a second failure mode. If the chosen bound (and corresponding normalization) is set high enough to safely accommodate unusually large peaks, it may be *too conservative* for instances where the true maximum response is small. In such cases, the magnitude of the correlation values after normalization can approach the noise floor, and the L^{16} score becomes noise-dominated even though no wrap-around occurs. In principle, one would like to adapt the normalization to the input instance, e.g. by dividing by an estimate of $\max_i |x_i|$ prior to exponentiation. However, obtaining such a normalization factor in the encrypted domain is itself a variant of the encrypted maximum problem that the L^{16} surrogate was introduced to bypass.

As a result, our implementation is *conditionally* successful: it produces numerically meaningful L^{16} scores for instances where the chosen normalization keeps intermediate magnitudes within the modulus budget while remaining sufficiently above the noise floor, but it may fail outside this window.

5.4.2 Illustrative example

To make this dependency concrete, consider two pairs of images from our tests. When using `0_left_index_3_cam2.png` as the model and `6_left_index_5_cam1.png` as the probe, the plaintext correlation maximum is approximately 379. For the pair `0_left_index_1_cam1.png` (model) and `0_left_index_2_cam1.png` (probe), the plaintext maximum is approximately 1180. With a normalization factor of 5000, the encrypted L^{16} computation yields stable outputs for both pairs. However, when increasing the normalization to 7000, the first pair (with the smaller peak) becomes noise-dominated and the resulting score is no longer reliable, while the second pair remains within a usable range. This example illustrates the fundamental tension described above: a normalization chosen to be "safely large" for potential outliers can degrade accuracy on low-peak instances.

Concretely, starting from $\mathtt{ct}_C$, we proceed as follows:

1. **Absolute value approximation.** We treat the real and imaginary parts of C as independent; for each slot value c, we approximate $|c|$ by $\sqrt{(\Re c)^2 + (\Im c)^2}$. In practice, this is done using polynomial approximations and by neglecting cross terms when justified by the structure of the data.

2. **Exponentiation.** To obtain $|x_i|^{16}$, we use repeated squaring:

$$|x_i|^2, \quad |x_i|^4, \quad |x_i|^8, \quad |x_i|^{16},$$

each step being a homomorphic component-wise multiplication. This consumes several levels of the modulus chain and therefore constrains our parameter choices.

3. **Slot-wise summation.** We sum all slot values using a binary-tree pattern of rotations and additions: starting from a ciphertext encrypting $(|x_1|^{16}, \ldots, |x_N|^{16})$, we rotate by powers of two and add, until every slot contains the same sum S.

The resulting ciphertext ct_S encrypts the scalar S in all slots. The server returns ct_S to the client, who decrypts it using the secret key and computes $S^{1/16}$ in plaintext. This final step is computationally cheap and avoids further increasing the homomorphic depth.

5.5 Performance and comparison to plaintext

We implemented the FFT-based design described above in OpenFHE using CKKS with ring dimension $n = 2^{18}$ and a modulus chain chosen to support the inverse FFT and the L^{16}-type scoring. The model and probe were taken from the existing plaintext pipeline [9], which is "interrupted" after the calculation of the FFT's of both. These transformed images are then given to the pipeline.

We evaluated two variants of the encrypted pipeline:

- a *non-*L^{16} variant that stops after the homomorphic inverse FFT and returns the decrypted correlation surface C to the client; and
- a full L^{16} variant that computes the encrypted sum S as described in Section 5.4 and returns only the scalar ciphertext ct_S.

For our parameter choices and implementation, the non-L^{16} variant completes in approximately 53 seconds at 10 threads without caching and about 40 seconds when keys are loaded from cache; the full L^{16} variant takes roughly 106 seconds without caching and about 80 seconds with caching. These measurements include the homomorphic inverse FFT, the element-wise product in the spectral domain, and the necessary rotations and rescalings. In contrast, the fully plaintext Miura pipeline as implemented in [9] completes the same matching step in well under a second on the same hardware. This means that our homomorphic proof of concept is roughly two orders of magnitude slower (about $80\times$) than the optimized plaintext implementation.

The FFT-based design is therefore a substantial improvement over the generic matrix-based baseline in terms of asymptotic and practical efficiency, but it still remains far from practical for real-world deployment. Nevertheless, it demonstrates that the matching block can be implemented homomorphically in a way that preserves the structure of the Miura pipeline and that yields numerically meaningful results when the scoring normalization is appropriately chosen. Further architectural changes, parameter tuning and possibly hardware acceleration would be required to close the performance gap. An alternative row-wise architecture, which we investigated but ultimately found less promising, is discussed in Section 6.

6 Alternative Architecture: Row-wise Ciphertext Encapsulation

In the development of our primary architecture, we investigated an alternative design strategy where each row of the input image is encapsulated into a distinct ciphertext. The primary motivation for this approach was to reduce the required ring dimension (N) of the cryptographic parameters. By storing only a single row ($W = 384$) per ciphertext rather than the full image grid, we hypothesized that the scheme could operate with significantly smaller keys and faster basic operations compared to the monolithic packing strategy described in Section 5.

However, theoretical analysis and performance profiling demonstrated that this architecture is fundamentally ill-suited for the vein matching problem due to catastrophic underutilization of the SIMD (Single Instruction, Multiple Data) slots inherent to the CKKS scheme.

6.1 SIMD Underutilization and Throughput Waste

The defining characteristic of this alternative architecture is the mapping of an image $M \in \mathbb{C}^{H \times W}$ to a vector of H distinct ciphertexts, $\mathbf{c} = \{c_0, c_1, \ldots, c_{H-1}\}$. While this allows for a reduction in the polynomial modulus degree N, ensuring 128-bit security still requires a minimum dimension (typically $N \geq 32,768$ for the required depth), providing $N/2 = 16,384$ available slots per ciphertext.

This creates a severe imbalance between the available capacity and the actual data payload:

- **Efficiency Drop:** By storing only one image row ($W = 384$ pixels) per ciphertext, the slot utilization rate (η) becomes negligible:

$$\eta_{row} = \frac{384}{16,384} \approx 2.3\% \tag{16}$$

  Consequently, over 97% of the computational throughput is wasted processing padding zeros. Even if the individual homomorphic operations are faster due to a smaller N, the system must perform these operations on $H = 240$ separate ciphertexts.

- **Comparison to Unified Packing:** In contrast, our optimized implementation packs the entire image ($240 \times 384 = 92,160$ elements) into a single, larger ciphertext. Despite requiring a larger ring dimension, the unified approach achieves a utilization efficiency of approximately 70%, amortizing the cryptographic overhead across the entire dataset simultaneously.

6.2 Execution Overhead

Beyond storage inefficiency, the row-wise encapsulation prevents the application of global linear transformations. In the unified approach, a single homomorphic rotation shifts the entire image grid. In the row-wise architecture, operations such as the row-wise IFFT must be applied individually to each of the 240 ciphertexts. This serialization leads to an explosion in the total operation count, increasing the execution time by orders of magnitude compared to the fully packed approach. Consequently, despite the potential for smaller cryptographic parameters, the architectural overheads renders this approach infeasible for latency-constrained biometric matching.

A concrete test run exposed flaws in the mathematical assumptions and took around 21 minutes, so we did not pursue this design further.

7 Experimental Evaluation

This section presents a systematic evaluation of both the baseline and FFT-optimized homomorphic implementations. We measure runtime performance across different configurations and analyze the impact of key generation, caching, and parallelization on overall execution time.

7.1 Experimental Setup

All experiments were conducted on a 2021 14-inch MacBook Pro equipped with an Apple M1 Pro chip (8 performance cores, 2 efficiency cores), 16 GB of unified memory, and 1 TB of storage. The implementations use OpenFHE version 1.1.2 [1] compiled with OpenMP support for parallelization.

OpenFHE internally manages thread-level parallelism for homomorphic operations, and the library documentation advises against wrapping OpenFHE calls in additional parallel constructs. However, our pipeline includes preprocessing stages (e.g., twiddle factor computation, plaintext encoding) that do not involve OpenFHE and can therefore benefit from OpenMP parallelization. We report results for 1, 5, and 10 threads to illustrate the impact of parallelism on different pipeline stages.

Each reported timing is the median of three independent runs to reduce measurement noise. We distinguish between two pipeline variants:

- **Non-L^{16}**: The pipeline stops after the homomorphic inverse FFT and returns the encrypted correlation surface to the client.
- L^{16}: The pipeline additionally computes the L^{16}-type score homomorphically before returning a single scalar ciphertext.

7.2 End-to-End Runtime

Table 1 presents the total execution time for the baseline (non-FFT) implementation, while Table 2 shows the corresponding results for the FFT-optimized version.

Threads	**Non-L^{16} [s]**	L^{16} **[s]**
1	240.10	432.31
5	155.78	266.71
10	145.32	291.84

Table 1: Total runtime for the baseline (matrix-based) implementation.

Threads	**Non-L^{16} [s]**	L^{16} **[s]**
1	195.50	324.32
5	63.39	113.61
10	52.85	105.72

Table 2: Total runtime for the FFT-optimized implementation.

The FFT-based design achieves a significant speedup over the baseline: at 10 threads, the non-L^{16} variant runs in approximately 53 seconds compared to 145 seconds for the baseline—a reduction of roughly 2.7×. The improvement is even more pronounced at lower thread counts, where the baseline's $O(N^2)$ rotation overhead dominates. Figure 3 visualizes this comparison for the non-L^{16} variants.

7.3 Runtime Breakdown by Pipeline Stage

To understand where time is spent, we instrument the pipeline to measure three distinct phases:

1. **Key generation**: Creation of public/secret keys, relinearization keys, and rotation (Galois) keys. The time depends on the multiplicative depth and the number of rotation indices required.
2. **Precomputation**: Encoding of the plaintext vectors used during homomorphic evaluation—specifically, the masking vectors and twiddle

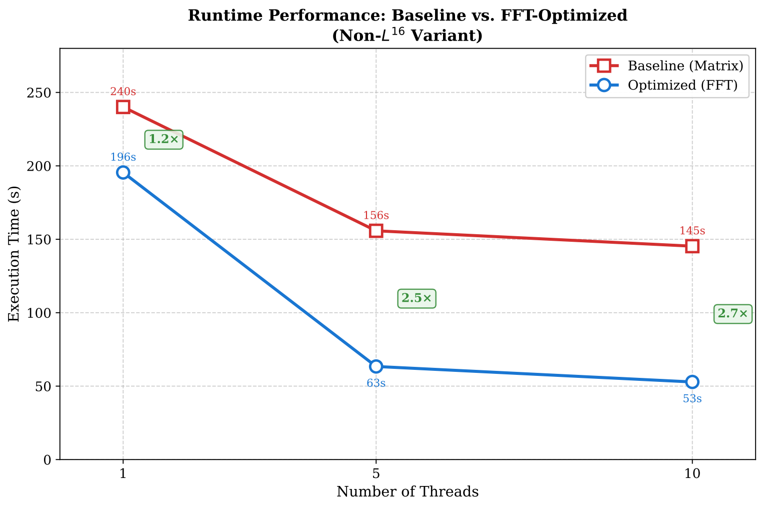

Figure 3: Runtime comparison of the baseline (matrix-based) and FFT-optimized implementations as a function of thread count (non-L^{16} variants). The FFT-optimized version achieves a 1.2–2.7× speedup depending on the degree of parallelism.

factors described in Section 5.3. This includes mask building (for slot selection) and row/column IFFT table generation. Unlike key generation, this phase involves no cryptographic operations and the results can be reused across all inputs.

3. **Homomorphic evaluation**: The actual encrypted computation: spectral multiplication, 2D-IFFT via the staged butterfly operations, and (optionally) L^{16} scoring as described in Sections 5.1–5.4.

Tables 3 and 4 present these breakdowns. Each cell shows the three timings separated by slashes: *Key generation / Precomputation / Evaluation*.

Threads	**Non-L^{16} [s]**	L^{16} [s]
1	37.55 / 36.35 / 164.89	64.02 / 59.05 / 308.80
5	9.61 / 8.37 / 136.55	17.35 / 15.59 / 233.32
10	8.18 / 7.67 / 128.07	14.52 / 13.67 / 263.19

Table 3: Runtime breakdown for the baseline implementation (Key generation / Precomputation / Evaluation).

A notable difference between the two designs is the relative cost of key generation. In the baseline implementation, key generation accounts for approximately 5–15% of the total runtime (depending on thread count and variant). In the FFT-optimized version, this fraction rises to 29–45%. This

Threads	**Non-L^{16} [s]**	L^{16} [s]
1	94.68 / 7.03 / 92.78	140.80 / 9.43 / 172.45
5	23.57 / 1.97 / 36.85	38.16 / 2.58 / 71.37
10	17.75 / 1.58 / 32.54	28.99 / 2.20 / 73.06

Table 4: Runtime breakdown for the FFT-optimized implementation (Key generation / Precomputation / Evaluation).

is explained by the deeper multiplicative circuits required by the FFT design: the optimized variants use depths of 8 (non-L^{16}) and 12 (L^{16}), compared to depths of 4 and 8 for the baseline. Additionally, the FFT design requires 120 rotation keys versus 85 for the baseline, further increasing key generation time.

There is a minor discrepancy (approximately 1 second) between the sum of the breakdown timings and the end-to-end measurements in Tables 1 and 2; this is attributable to test harness overhead.

7.4 Key Sizes and Memory Footprint

Table 5 reports the serialized sizes of the cryptographic keys for different multiplicative depths. Both implementations use one public key, one secret key, one relinearization key, one conjugation key, and the rotation keys mentioned above.

Key Type	**Depth 4 [MB]**	**Depth 8 [MB]**	**Depth 12 [MB]**
Public	28.00	48.00	72.00
Secret	10.00	18.00	26.00
Relinearization	84.00	144.00	216.01
Rotation (each)	84.00	144.00	216.01
Conjugation	84.00	144.00	216.01
Rotation total (120 keys) [GB]	9.84	16.88	25.33

Table 5: Serialized key sizes as a function of multiplicative depth.

For the FFT-optimized L^{16} variant (depth 12, 120 rotation keys), the total key material exceeds 26 GB. The rotation-total row makes the otherwise implicit 120×216 MB requirement explicit (approximately 25 GB just for rotation keys), which motivates the key caching optimization discussed below.

7.5 Memory Profiling

Beyond key sizes, we measured the peak memory usage of the complete pipeline using the system profiler (`/usr/bin/time -l` on macOS). Table 6 reports the peak memory footprint for each implementation variant.

Implementation	**Variant**	**Depth**	**Peak Memory [GB]**
Baseline (non-FFT)	Non-L^{16}	4	23.5
Baseline (non-FFT)	L^{16}	8	39.5
FFT-optimized	Non-L^{16}	8	22.3
FFT-optimized	L^{16}	12	33.0

Table 6: Peak memory footprint for each implementation variant at 10 threads.

The memory footprint is dominated by the cryptographic keys and the ciphertext buffers. The L^{16} variants require substantially more memory due to their deeper modulus chains, which increase the size of each ciphertext element. Notably, the FFT-optimized variants use less memory than their baseline counterparts at equivalent functionality: the FFT non-L^{16} variant (22.3 GB) uses slightly less than the baseline non-L^{16} (23.5 GB), and the FFT L^{16} variant (33.0 GB) uses significantly less than the baseline L^{16} (39.5 GB). This is despite the FFT variants having deeper circuits, and is likely because the baseline requires more intermediate ciphertext copies during the matrix-based IFFT computation.

These memory requirements (22–40 GB) are substantial and require a well-provisioned server or workstation. They would be prohibitive for resource-constrained devices, strongly reinforcing the client-server architecture where the heavy computation is offloaded to a powerful server while the client handles only encryption, decryption, and the final score calculation.

7.6 Impact of Key Caching

Regenerating keys for every matching operation is wasteful in scenarios where the same cryptographic context is reused across multiple authentications. We implemented a caching mechanism that serializes the keys to disk after the first run and loads them on subsequent runs. This optimization was applied only to the FFT-optimized implementation, where key generation constitutes a larger fraction of the total time.

The disk storage required for the cached keys is substantial: 18.64 GB for the non-L^{16} variant (depth 8) and 27.96 GB for the L^{16} variant (depth 12). This reflects the size of the public key, secret key, relinearization key, conjugation key, and 120 rotation keys at the respective depths.

Key caching yields substantial improvements. With caching enabled, the total runtime for the non-L^{16} variant at 10 threads drops from 52.85 s

Threads	**Non-L^{16} [s]**	**L^{16} [s]**
1	11.75	21.09
5	11.34	20.11
10	11.82	19.66

Table 7: Time to load keys from cache (FFT-optimized implementation).

Threads	**Non-L^{16} [s]**	**L^{16} [s]**
1	11.75 / 7.59 / 87.89 (107.80)	21.09 / 10.95 / 163.44 (196.33)
5	11.34 / 2.19 / 31.55 (45.71)	20.11 / 3.51 / 72.10 (96.19)
10	11.82 / 1.77 / 25.75 (39.88)	19.66 / 2.91 / 56.40 (79.80)

Table 8: Runtime breakdown with key caching (Load / Precomputation / Evaluation; total in parentheses).

to 39.88 s—a 25% reduction. Notably, loading keys from disk takes roughly the same time regardless of thread count (approximately 11–21 s), whereas key generation scales with parallelism. This makes caching particularly attractive for deployments with limited CPU resources but sufficient storage.

The small difference in the homomorphic evaluation phase between cached and freshly generated keys falls within run-to-run noise. We did not find an algorithmic reason for this and attribute it to allocator or OS caching effects rather than a systematic speedup. Figure 4 visualizes the impact of key caching on total runtime.

7.7 Comparison to Plaintext Baseline

For context, the fully plaintext Miura matching pipeline as implemented in the prior semester project [9] completes the same matching operation (FFT, element-wise multiplication, IFFT, and maximum extraction) in the range of hundreds of milliseconds on comparable hardware. Our homomorphic implementation, even in its fastest configuration (FFT-optimized, cached keys, 10 threads), requires tens of seconds per match.

While we did not conduct a rigorous head-to-head comparison on identical hardware and workloads, the qualitative conclusion is clear: homomorphic encryption introduces a performance penalty of roughly two orders of magnitude for this workload. Whether this overhead is acceptable depends on the deployment context. For offline or batch processing scenarios where latency is not critical, the privacy guarantees may justify the cost. For interactive authentication with sub-second response time requirements, the current implementation would not be suitable without significant further optimization or hardware acceleration.

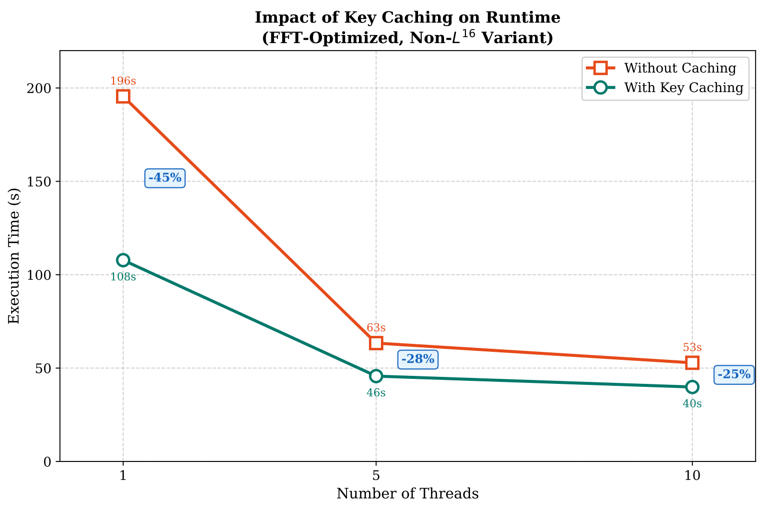

Figure 4: Impact of key caching on runtime for the FFT-optimized non-L^{16} variant. Caching reduces total execution time by 25–45% depending on thread count, with the largest relative improvement at single-threaded execution.

7.8 Cryptographic Parameters

Table 9 summarizes the CKKS parameters used in our OpenFHE implementation. These parameters were chosen to provide 128-bit security while supporting the multiplicative depth required by our circuits.

The ring dimension $N = 2^{18}$ was chosen to provide sufficient slots to pack an entire 240×384 image (92,160 values) into a single ciphertext while maintaining 128-bit security for the required modulus sizes. The scaling technique FLEXIBLEAUTO allows OpenFHE to automatically manage rescaling operations, simplifying the implementation at the cost of some efficiency compared to manual scale management.

7.9 Numerical Accuracy and Biometric Impact

A critical question for any approximate computation scheme is whether the introduced errors affect the correctness of the final application. In our case, we must verify that the numerical differences between plaintext and homomorphic L^{16} scores do not alter biometric matching decisions.

Relative error measurement. We computed the relative error between the plaintext L^{16} score and the homomorphically computed score across a range of normalization factors. Within the valid operating window (scale factors approximately 3.2× to 9.7× the maximum spatial correlation value), the relative error remains small:

Parameter	**Value**
Polynomial Modulus Degree (N)	2^{18} (262,144)
Number of Slots	$N/2 = 131,072$
Security Level	128-bit (HE Standard)
Scaling Technique	FLEXIBLEAUTO
First Modulus Size	60 bits
Scaling Modulus Size	50 bits
Multiplicative Depth	8 (non-L^{16}) / 12 (L^{16})
Total Modulus ($\log_2 Q$)	≈ 460 bits (depth 8) / ≈ 660 bits (depth 12)
Secret Key Distribution	Ternary (uniform $\{-1, 0, 1\}$)
Error Distribution	Discrete Gaussian ($\sigma \approx 3.2$)

Table 9: Cryptographic parameters used in the OpenFHE/CKKS implementation.

- Best case (optimal scale factor): relative error $< 10^{-6}$ (specifically $\approx 9.9 \times 10^{-7}$)
- Typical case (mid-range scale factor): relative error $< 8 \times 10^{-5}$
- Edge of working range: relative error $\approx 2 \times 10^{-3}$

For our test images, where maximum spatial correlation values ranged from approximately 260 to 2000, this corresponds to a working scale factor range of roughly 3800–11500 for a reference image with max value $\sim$1180.

Error propagation analysis. We analyzed how errors in the L^{16}-derived maximum propagate to the final distance measure. The distance formula $d = 1 - \max/(\text{count}_M + \text{count}_P)$ implies that the distance error is:

$$\Delta d \approx \frac{\max}{\text{count}_M + \text{count}_P} \times \varepsilon_{\max}$$

where $\varepsilon_{\max}$ is the relative error in the maximum estimate.

For our test pairs, the ratio $\max/\text{denom}$ ranged from approximately 0.07 (impostor pairs) to 0.21 (genuine pairs). This means:

- With 0.01% (10^{-4}) error in max: distance error $\approx 10^{-5}$ to 2×10^{-5}
- With 1% error in max: distance error $\approx 7 \times 10^{-4}$ to 1.5×10^{-3}

Biometric decision impact. On the plaintext pipeline, the L^{16}-derived maximum coincides with the direct maximum to within machine precision (both implement $\max |x_i|$ through equivalent formulations), so any difference is below the numerical noise floor. The genuine/impostor separation in our test set showed distance values of approximately 0.73–0.89 for genuine pairs and 0.93–0.97 for impostor pairs, providing a margin of roughly 0.04–0.20.

Since the FHE-induced errors (on the order of 10^{-5} to 10^{-3} in the distance) are two to four orders of magnitude smaller than the decision margin, we conclude that the homomorphic implementation preserves the accuracy of the biometric matching decisions within the valid normalization window.

Normalization window constraints. The key limitation is that the valid normalization window depends on the (unknown) maximum correlation value of the input. If the normalization constant is set too conservatively (to accommodate potential high-peak inputs), low-peak inputs may fall below the noise floor. If set too aggressively, high-peak inputs may cause modular overflow. For a fixed normalization constant, only inputs whose maximum correlation falls within a factor of approximately 3× of each other can be reliably handled (derived from the ratio of the upper to lower bounds of the working window: $9.7/3.2 \approx 3$). Figure 5 illustrates this trade-off schematically.

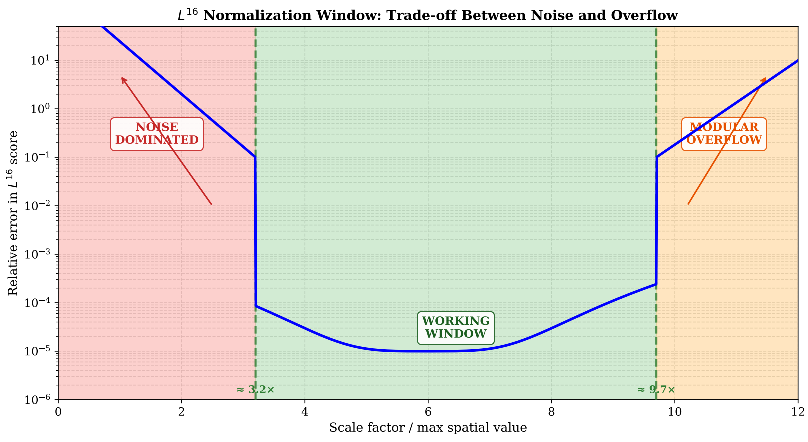

Figure 5: The L^{16} normalization trade-off: scale factors must fall within a narrow window to avoid noise-dominated computation (too low) and modular overflow (too high). The bounds shown (3.2× to 9.7×) are empirically determined and may not be the tightest possible, as exhaustive testing was infeasible due to runtime constraints.

Note on bounds: The working window bounds reported here (scale factors 3800–11500 for a reference max value of 1180) were determined through targeted testing rather than exhaustive search. Due to the long runtime of each homomorphic evaluation (tens of seconds to minutes), we could not sweep all possible scale factors. The reported bounds are therefore sufficient but not necessarily tight—the true working range may be slightly wider.

Critically, this problem cannot be resolved by simply choosing a "safe" high normalization value. Our dataset analysis revealed that genuine pairs (same finger) exhibit maximum correlation values ranging from approximately 300 to over 2000—a factor of nearly $7\times$. While genuine pairs have higher average correlation (approximately 786) compared to impostor pairs (approximately 460), there is substantial overlap: some genuine pairs produce correlation peaks as low as 300, while some impostor pairs reach values above 800. This means that a normalization constant chosen to safely accommodate high-correlation genuine pairs would push low-correlation genuine pairs below the noise floor, potentially causing *false rejections* of legitimate users. Conversely, a lower normalization constant that works well for typical inputs would cause overflow for high-correlation cases.

This fundamental constraint—arising from the natural variability in biometric data—limits the practical applicability of the L^{16} variant to scenarios where the input distribution is tightly controlled or where per-input normalization can somehow be estimated.

8 Discussion

8.1 Interpretation of Results

The experimental results in Section 7 demonstrate that homomorphic evaluation of the Miura frequency-domain matching block is technically feasible. The FFT-optimized implementation represents a substantial improvement over the naive matrix-based baseline, reducing both asymptotic complexity and practical runtime by exploiting the recursive structure of the Fourier transform. With key caching and moderate parallelism, the non-L^{16} variant achieves matching times on the order of 40 seconds—a significant reduction from the baseline's 145+ seconds.

However, these results must be interpreted in the context of what the homomorphic pipeline actually computes. The non-L^{16} variant offloads only the element-wise spectral multiplication and the inverse FFT to the server; the client still receives the full encrypted correlation matrix and must decrypt it before extracting the maximum. This means the server learns nothing about the matching score, but the computational savings compared to performing these operations in plaintext are modest: the plaintext pipeline completes the same operations in under 0.5 seconds. The roughly $80\times$ slowdown may be acceptable in scenarios where privacy is paramount and latency requirements are relaxed (e.g., offline enrollment verification), but it is unlikely to be practical for interactive authentication.

8.2 The Normalization Problem and L^{16} Limitations

The L^{16} scoring mechanism was introduced to approximate the maximum correlation value entirely in the encrypted domain, which would allow the server to return only a single scalar rather than the full matrix. In principle, this is desirable because it further reduces information leakage and bandwidth.

In practice, however, the L^{16} approach suffers from a fundamental limitation tied to the CKKS scheme's approximate arithmetic. Computing $|x|^{16}$ via repeated squaring amplifies the dynamic range of the input values dramatically. To prevent the intermediate results from exceeding the ciphertext modulus (which would cause wrap-around and produce meaningless outputs), we must normalize the correlation values by a conservative upper bound before exponentiation.

The problem is that this bound is inherently *input-dependent.* A normalization factor chosen to accommodate inputs with high correlation peaks will be overly conservative for inputs with lower peaks, pushing the normalized values toward the CKKS noise floor and destroying the signal. Conversely, a factor tuned for typical inputs will fail catastrophically on outliers. As demonstrated in Section 5.4, changing the normalization from 5000 to 7000 can cause certain input pairs to become noise-dominated while leaving others unaffected.

This is not merely an implementation artifact but a structural limitation of using high-degree polynomial surrogates for non-polynomial functions (such as max) in CKKS. Without a reliable way to estimate or bound the input magnitude *before* the homomorphic computation—which would itself require solving a variant of the encrypted maximum problem—the L^{16} scoring cannot be deployed robustly.

8.3 Practical Implications

Given these limitations, we conclude that only the **non-L^{16} variant** of the homomorphic pipeline is reliable in practice. This variant returns the encrypted correlation matrix to the client, who decrypts it and extracts the maximum in plaintext.

This design choice has implications for the overall system architecture:

- **What is outsourced**: The server performs the spectral multiplication and inverse FFT—operations that are linear and well-suited to CKKS.
- **What remains on the client**: The client must still perform the final maximum extraction (and any subsequent distance/threshold computation) in plaintext.
- **Privacy guarantee**: The server never observes the model, the probe, or the correlation surface in the clear. However, it does observe encrypted ciphertexts and their sizes, which may leak metadata (e.g., that a matching operation occurred).

- **Performance cost**: The overhead of approximately 80× compared to plaintext matching is substantial. For a single authentication, this may be tolerable; for large-scale batch operations, it is prohibitive.

The question then becomes: *is replacing a simple matrix multiplication and IFFT with homomorphic equivalents worth an 80–100× increase in execution time?* The answer depends heavily on the threat model and deployment context. If the matching server is semi-trusted (e.g., operated by a regulated entity with audit obligations), the privacy benefits may not justify the performance penalty. If the server is fully untrusted and privacy is non-negotiable, the overhead may be acceptable—but alternative approaches (e.g., secure multi-party computation, trusted execution environments) should also be considered.

9 Conclusion and Future Work

9.1 Summary of Contributions

This project investigated the feasibility of performing the frequency-domain matching step of the Miura finger-vein pipeline under homomorphic encryption. The main contributions are:

1. **Problem formulation**: We identified the specific computational block (spectral multiplication, inverse FFT, and maximum extraction) that could be delegated to an untrusted server and formalized the requirements for a homomorphic implementation.
2. **Baseline implementation**: We developed a proof-of-concept using generic matrix–vector operations to implement the 2D inverse DFT, demonstrating feasibility but also revealing the prohibitive cost of the $O(N^2)$ approach.
3. **FFT-optimized implementation**: We designed and implemented a homomorphic 2D-IFFT using the Cooley–Tukey radix-2 structure, reducing the number of rotations and achieving runtime improvements of 2–3× over the baseline.
4. L^{16} **scoring analysis**: We implemented an L^{16}-type surrogate for the maximum function and characterized its fundamental limitation: the input-dependent normalization required to prevent modular overflow makes it unreliable for general inputs.
5. **Empirical evaluation**: We systematically measured runtime, memory usage, and the impact of parallelism and key caching, providing concrete performance data for future work.

9.2 Limitations

Several limitations of this work should be acknowledged:

- **Single-image scope**: We focused on matching a single probe against a single model. Realistic deployments may involve one-to-many matching against a database, which would require different algorithmic strategies.
- **No end-to-end integration**: The homomorphic pipeline was evaluated in isolation. Integration with the full BioID/BioLocker architecture, including key management on smartcards, was not attempted.
- **Hardware platform**: All experiments were conducted on a single consumer laptop. Server-class hardware with AVX-512 support or GPU acceleration could yield different performance characteristics.
- **Parameter tuning**: The CKKS parameters were chosen conservatively to ensure correctness. More aggressive parameter selection or noise-budget optimization might improve performance at the cost of robustness.

9.3 Future Directions

Several avenues for future work could address the limitations identified in this project:

- **Alternative maximum approximations**: The polynomial surrogate approach (L^p norms) is fundamentally limited by CKKS's approximate arithmetic. Comparison-based methods using other FHE schemes (e.g., TFHE) or hybrid protocols could provide exact maximum extraction at potentially lower cost for small output domains.
- **Hardware acceleration**: GPU implementations of homomorphic encryption could reduce the runtime significantly, potentially making the approach more practical.
- **Algorithmic redesign**: Rather than encrypting the existing plaintext pipeline, one could design a matching algorithm specifically for the encrypted domain—for example, using distance metrics that are naturally compatible with CKKS (e.g., Euclidean distance on feature embeddings).
- **Alternative privacy technologies**: Secure multi-party computation (MPC) or trusted execution environments (TEEs) offer different trade-offs between performance, trust assumptions, and security guarantees. A systematic comparison of these approaches for the biometric matching use case would be valuable.

9.4 Concluding Remarks

This project demonstrates that homomorphic encryption can, in principle, be applied to the Miura finger-vein matching pipeline. The FFT-optimized design successfully evaluates the inverse Fourier transform and spectral correlation in the encrypted domain, preserving privacy against an honest-but-curious server.

However, the practical utility of this approach is limited. The L^{16} scoring mechanism was designed to return an encrypted approximation of the maximum correlation value: the server computes the sum $S = \sum |x_i/\text{norm}|^{16}$ homomorphically (where norm is the normalization constant) and returns it to the client. The client decrypts S, computes $S^{1/16} \cdot \text{norm}$ to recover the properly scaled approximation, and continues with the remaining steps of the distance calculation. (Note that scaling can be applied after the root extraction rather than before, avoiding the need to compute norm^{16} on the client side.) Unfortunately, this approach is unreliable due to input-dependent normalization requirements, as discussed in Section 8.

The non-L^{16} variant, which returns the encrypted correlation matrix to the client for decryption and subsequent plaintext processing, is numerically robust. However, it incurs a performance overhead of approximately 50–100$\times$ compared to plaintext matching (depending on whether key caching is used), for outsourcing only a matrix multiplication and an inverse FFT. This trade-off is difficult to justify: the computational savings on the client side are modest, while the latency increase is substantial.

For the specific workload considered—a relatively simple linear transformation followed by a non-linear reduction—the cost-benefit analysis does not favor homomorphic encryption in its current form. The approach may become viable as HE libraries and hardware accelerators mature, or it may serve as a baseline against which alternative privacy-preserving techniques can be compared. Ultimately, the choice of privacy technology must be informed by the specific threat model, performance requirements, and deployment constraints of the target application.

References

[1] Ahmad Al Badawi et al. “OpenFHE: Open-Source Fully Homomorphic Encryption Library”. In: *Proceedings of the 10th Workshop on Encrypted Computing & Applied Homomorphic Cryptography (WAHC)*. ACM, 2022.

[2] Fatih Balli, F. Betül Durak, and Serge Vaudenay. “BioID: A Privacy-Friendly Identity Document?” In: *Security and Trust Management (STM 2019)*. Vol. 11738. Lecture Notes in Computer Science. Springer, 2019.

[3] Gary Bradski. “The OpenCV Library”. In: *Dr. Dobb’s Journal of Software Tools* (2000).

[4] Jung Hee Cheon et al. “Homomorphic Encryption for Arithmetic of Approximate Numbers”. In: *Advances in Cryptology – ASIACRYPT 2017*. Springer, 2017.

[5] James W. Cooley and John W. Tukey. “An Algorithm for the Machine Calculation of Complex Fourier Series”. In: *Mathematics of Computation* 19.90 (1965), pp. 297–301.

[6] F. Betül Durak, Loïs Huguenin-Dumittan, and Serge Vaudenay. “BioLocker: A Practical Biometric Authentication Mechanism based on 3D Fingervein”. In: *International Conference on Information Security and Cryptology (CISC)*. Full version manuscript. 2020.

[7] Craig Gentry. “Fully Homomorphic Encryption Using Ideal Lattices”. In: *Proceedings of the 41st Annual ACM Symposium on Theory of Computing (STOC)*. ACM, 2009.

[8] Shai Halevi and Victor Shoup. “Algorithms in HElib”. In: *Advances in Cryptology – CRYPTO 2014*. Vol. 8616. Lecture Notes in Computer Science. Introduces the diagonal method for homomorphic matrix–vector multiplication. Springer, 2014, pp. 554–571.

[9] Lara Sofie Lenz. *Finger Vein Biometric Matching.* Unpublished semester project report provided as reference material. Cyber Security, ETHZ / EPFL (Semester Project), 2024.

[10] Naoto Miura, Akio Nagasaka, and Takafumi Miyatake. “Extraction of Finger-Vein Patterns Using Maximum Curvature Points in Image Profiles”. In: *IEICE Transactions on Information and Systems* E90-D.8 (2007), pp. 1185–1194.

[11] Naoto Miura, Akio Nagasaka, and Takafumi Miyatake. “Feature Extraction of Finger Vein Patterns Based on Iterative Line Tracking and Its Application to Personal Identification”. In: *Systems and Computers in Japan* 35.7 (2004), pp. 61–71.

[12] Microsoft Research. *Microsoft SEAL (release 3.6)*. `https://github.com/Microsoft/SEAL`. Accessed 2026-01-05. 2020.