MONCHI: Multi-scheme Optimization For Collaborative Homomorphic Identification

Alberto Ibarrondo[*]
Copper.co
Sophia Antipolis, France
ibarrond@eurecom.fr

Ismet Kerenciler[†]
Télécom Paris
Paris, France
ismet.kerenciler@telecom-paris.fr

Hervé Chabanne
IDEMIA & Télécom Paris
Paris, France
herve.chabanne@idemia.com

Vincent Despiegel
IDEMIA
Paris, France
vincent.despiegel@idemia.com

Melek Önen
EURECOM
Sophia Antipolis, France
melek.onen@eurecom.fr

ABSTRACT

This paper introduces a novel protocol for privacy-preserving biometric identification, named MONCHI, that combines the use of homomorphic encryption for the computation of the identification score with function secret sharing to obliviously compare this score with a given threshold and finally output the binary result. Given the cost of homomorphic encryption, BFV in this solution, we study and evaluate the integration of two packing solutions that enable the regrouping of multiple templates in one ciphertext to improve efficiency meaningfully. We propose an end-to-end protocol, prove it secure and implement it. Our experimental results attest to MONCHI's applicability to the real-life use case of an airplane boarding scenario with 1000 passengers,taking less than one second to authorize/deny access to the plane to each passenger via biometric identification while maintaining the privacy of all passengers.

CCS CONCEPTS

• **Security and privacy → Privacy-preserving protocols**.

KEYWORDS

Multiparty Homomorphic Encryption, Function Secret Sharing, Secure Two Party Computation, Masking, Privacy Preserving Technologies, Scalar Product

1 INTRODUCTION

Biometric identification and authentication are being increasingly adopted in a wide range of applications including law enforcement, banking[1], personal hardware, and airport security[2]. Users are identified based on their unique biological traits (e.g., fingerprint, face, iris), making systems more secure. Biometric systems usually compute the so-called identification scores through scalar products between a fresh biometric template and templates that were collected in a prior enrollment phase, then stored in a reference database. This score is further compared to a pre-defined threshold and when exceeded, identification is considered successful.

The collection and processing of biometric data raises significant privacy concerns because such data uniquely identifies the individual and cannot be cancelled or re-issued. Therefore, the use of privacy enhancing technologies in such a context is crucial. Privacy-preserving identification systems call for advanced cryptographic techniques such as fully homomorphic encryption (FHE)[13] or multi-party computation[20], that enable computation while data being protected. While this technology ensures that biometric data is never revealed, it still incurs significant overhead and refrains their wide adoption.

In this paper, we study the suitability of FHE for biometric identification in the context of airport applications where passengers are authorized to board the plane only when identification is successful[3]. As opposed to the use of MPC, FHE does not consume any pre-processing and therefore results in a lighter and one-shot offline phase. To efficiently perform the final comparison operation, similar to [20], we propose to combines the use of FHE, BFV[7, 14] in this application, with function secret sharing[5, 6]. Hence, once the scalar product is computed, masks and splits this score to be distributed among two parties who further collaboratively decrypt the masked score and compare it to the threshold. To improve efficiency even more, we focus on the computation of scalar products and propose to study and evaluate the integration of two packing methods that enable the encoding of multiple templates in one ciphertext and hence increase performance in terms of communication and computation.

By leveraging FHE, MONCHI perfectly fits the airport access control use-case: given that linear operations are carried out without the need for a continuous supply of pre-processing material, and contrary to pure MPC solutions, we reduce the interactions in the offline phase of these linear operations to a one-time setup. This combination yields the best of both worlds, since it removes the otherwise endless supply of pre-processing material required to execute pure MPC protocols while leveraging MPC for non-linear operations that are impractical via FHE.

Our main contributions can be summarized as follows:

- MONCHI is the first protocol that combines the use of FHE with function secret sharing in order not to disclose the identification score but the access decision;

[*]This work was carried out while at Idemia
[†]This work was carried out while working at IDEMIA and studying at EURECOM
[1]https://www.mastercard.com/news/perspectives/2024/biometrics-will-soon-replace-passwords-once-and-for-all/
[2]https://www.fraport.com/en/newsroom/press-releases/2023/q4/using-facial-recognition–sita-and-fraport-enable-a-contactless-.html
[3]https://www.theguardian.com/world/2024/jan/01/facial-recognition-could-replace-passports-at-uk-airport-e-gates

- To improve performance even more, we propose to integrate two packing solutions and study their performance according to different metrics;
- Monchi is implemented and evaluated through real-life use case scenarios and more specifically to the airport application and demonstrated its feasibility: in an airplane boarding scenario with 1000 passengers, Monchi takes less than one second to authorize/deny access to the plane.

2 RELATED WORK

There have been multiple solutions that have investigated the use of biometrics (see [36] for a review). These solutions like [2, 23, 32], usually compute the score while the templates are homomorphically encrypted (either using BFV or CKKS). Other solutions like [4, 13] also investigated the use of packing in this context. As opposed to Monchi, all these solutions decrypt the actual identification score and further compare it to a given threshold. As shown in [15], revealing this identification score is harmful and can help attackers infer and finally disclose biometric templates. In Monchi, decryption occurs over the masked score and the decryption key is never revealed.

Another noteworthy FHE-based work, Colmade [19], which also employs FHE for distance metric computation in biometrics, addresses this issue by performing vector-matrix multiplication followed by comparison to a threshold in the encrypted domain, and subsequent masking of the result, revealing only its least significant bit. However, performing the comparison in the encrypted domain in Colmade is orders of magnitude more computationally expensive than Monchi.

Finally, as mentioned in the introduction, an alternative to the use of FHE for scalar products in biometric identification is the use of arithmetic secret sharing in a secure two-party computation (2PC) setting. AriaNN [31] employs arithmetic secret sharing to perform privacy-preserving distance metric computations, followed by a comparison with a threshold with FSS, requiring two rounds of online communication. Subsequently, Funshade [20] improves the communication overhead by employing a novel method for secret-sharing during 2PC called **Π**-Secret-Sharing (**Π**-SS). The new **Π**-SS protocol allows local additions and multiplications with only one round of communication in the online phase, thus outperforming AriaNN. In Monchi, we chose to replace the 2PC Π-SS building block of Funshade with a distributed implementation of the Brakerski-Fan-Vercauteren (BFV) FHE scheme (together with dedicated packing solutions), and improve its performance keeping the FSS block for subsequent comparison with a fixed threshold. Leveraging BFV for linear operations, we effectively get rid of the associated pre-processing material (typically beaver triples), meaningfully simplifying the continuous operation of a biometric identification system built with Monchi at the expense of an additional masking to adapt the numerical range of BFV to that of FSS while maintaining the privacy guarantees.

3 PRELIMINARIES

3.1 Notation

We use bold letters to denote matrices/vector (Eg. $\boldsymbol{x}$) and non-bold letters for scalars. R_v expresses a polynomial ring with integer coefficients modulo v. $a^{(i)}$ denotes the i-th feature/element/coefficient of a template/vector/polynomial. We use $\langle b \rangle_j$ to refer to share j in an arithmetic modular secret sharing of $b = \sum_{j=1}^{K} \langle b \rangle_j$. We denote $[\cdot]_q$ the reduction modulo q, and $\lfloor \cdot \rceil$, $\lceil \cdot \rceil$ the rounding up and rounding down, respectively, to the nearest integer. When applied to polynomials, these reductions are performed coefficient-wise. We use $U(X)$ to denote a uniformly random distribution in X, and $\mathcal{N}(\mu, \sigma)$ to denote a Gaussian distribution with mean μ and standard deviation σ. $\mathbf{a} \cdot \mathbf{b} = \mathbf{c}$ to denote the element-wise/coefficient-wise multiplication of two vectors/polynomials where $c^{(i)} = a^{(i)} b^{(i)}$. $\mathbb{Z}^*_{n^+}$ represents the range $0 \leq x \leq 2^{n-1} - 1$.

3.2 BFV

Scheme 1 BFV(t, q, N, σ, B)

```
Input: $t, q, N, \sigma, B$: Security Parameters
BFV.SecKeyGen():
    $s \leftarrow S_{R_q}$
    Output: the secret key $sk = s$
BFV.PubKeyGen($sk$):
    $p_1 \leftarrow U_{[R_q]}$ and $e \leftarrow \chi_{[R_q]}$
    Output: a public key $pk = (p_0, p_1) = (-sk \cdot p_1 + e, p_1)$
BFV.Encrypt($pk, m$):
    Let $pk = (p_0, p_1)$
    $u \leftarrow S_{[R_q]}$ and $e_0, e_1 \leftarrow \chi_{[R_q]}$
    $(c_{m_0}, c_{m_1}) = (\Delta m + u \cdot p_0 + e_0, u \cdot p_1 + e_1)$
    Output: a ciphertext $c_m = (c_{m_0}, c_{m_1})$
BFV.Add($c_a, c_b$):
    Let $c_a = (c_{a_0}, c_{a_1})$, $(c_{b_0}, c_{b_1})$
    $c_{add_0} = [c_{a_0} + c_{b_0}]_q$ and $c_{add1} = [c_{a_1} + c_{b_1}]_q$
    Output: the ciphertext $c_{add} = (c_{add0}, c_{add1})$
BFV.Mul($c_a, c_b$):
    Let $c_a = (c_{a_0}, c_{a_1})$, $(c_{b_0}, c_{b_1})$
    $c_{mul_0} = [c_{a_0} \cdot c_{b_0}]_q$ and $c_{mul_1} = [c_{a_1} \cdot c_{b_1}]_q$
    Output: the ciphertext $c_{mul} = (c_{mul_0}, c_{mul_1})$
BFV.Decrypt($sk, c_m$):
    Let $c_m = (c_{m_0}, c_{m_1})$
    $m_{res} = \left[ \left\lceil \frac{t}{q} \left[ c_{m_0} + s \cdot c_{m_1} \right]_q \right\rfloor \right]_t$
    Output: the decrypted message $m_{res}$
```

The Brakerski-Fan-Vercauteren (BFV) [1, 14, 17] scheme is a privacy-preserving, ring-Learning with Errors (RLWE) homomorphic encryption scheme. The plaintext space is defined by a parameter t. Messages are encoded into the plaintext ring $R_t = \mathbb{Z}_t[X]/(X^N + 1)$, which defines polynomials of degree at most $N - 1$ with coefficients in $\mathbb{Z}_t$. Specifically, the coefficients lie in the range $[0, t)$ for non-negative integer and in the symmetric range $[-t/2, t/2)$ for signed integers. The encrypted message will be an element of

$R_q = \mathbb{Z}_q[X]/(X^N + 1)$ with q, a security parameter defining the ciphertext modulus. The coefficients of the encrypted space reside in $\mathbb{Z}_q$. Typically, $t \ll q$, N is a power of two, and $\Delta = \lceil q/t \rceil$ defines the maximum number of homomorphic operations that can be performed on the ciphertext without affecting the correctness of the decrypted result. q can be decomposed with the Chinese Remainder Theorem (CRT) in small co-prime moduli $q_1, \ldots, q_v$ and establishing a ring isomorphism between the spaces $\mathbb{Z}_q$ and $\mathbb{Z}_{q_1} \times \cdots \times \mathbb{Z}_{q_v}$ [17]. Thanks to CRT, arithmetic modulo q can be replaced by v independent arithmetics in the smaller rings $\mathbb{R}_{q_i}$ enabling Residue Number System (RNS) to accelerate arithmetic in R_q over large integers.

The BFV scheme defines two uniform distributions: the secret key distribution $S_{R_q} = \mathbb{Z}_{\{-1,0,1\}}[X]/(X^N+1)$, defining ring polynomials of degree at most N with coefficients sampled from a uniform distribution $U(\{-1, 0, 1\})$. The error distribution χ_{err} is defined over the ring R_q where coefficients are drawn from a distribution statistically close to a Gaussian with standard deviation σ truncated into $[-B, B]$, where σ and B are two cryptographic parameters. χ_{err} is related to the hardness of the Ring Learning with Errors (RLWE) problem introduced in [24] as a version of the Learning-With-Error (LWE) problem [30]. In BFV, noise management is critical for maintaining correctness throughout the decryption process. BFV supports bootstrapping [12], which involves re-encrypting a ciphertext to get a fresh ciphertext of the same message, effectively resetting the noise growth. However, bootstrapping is practically slow, and despite suggested improvements in noise management [8][7], BFV is commonly instantiated with parameters sufficiently large to accommodate noise growth. Scheme 1 outlines several algorithms of the BFV scheme pertinent to our research.
Specifically, the algorithm BFV.Decrypt, which we aim to adapt and use in a distributed setting, involves using the secret key to compute an upscaled plaintext, followed by two rounding processes (first by q, then by t), with a downscaling step between the two rounding operations.

3.2.1 Ring Learning with Errors (RLWE) problem. The RLWE problem is defined as follows: given uniformly random a from $U(R_q)$, s from S_{R_q}, and e from χ_{err}, it is computationally challenging for an adversary to distinguish between the distributions of $(sa + e, a)$ and (b, a), where b is uniformly sampled from $U(R_q)$. Introduced in [24], the RLWE problem serves as a fundamental challenge in lattice-based cryptography.

3.3 Multiparty BFV Scheme

Multiparty Homomorphic Encryption (MHE) extends FHE to multiparty settings, facilitating collaborative computation while maintaining the privacy of each party's data. The Distributed Brakerski-Fan-Vercauteren (DBFV) scheme [26], derived from the BFV scheme, permits a pool of M parties to locally secret-share a global secret key and to collaboratively generate a public key using these local shares, as detailed in Scheme 2. The process of computing a common public key (cpk) introduces M additional noise terms, resulting in $\sum_{i=1}^{M} e_i = ||e_{cpk}|| < B \cdot M$, where B is the bound on the worst-case norm for an error term drawn from the distribution χ_{err} (see appendix A of [26] for more details). Thus, the worst-case fresh ciphertext noise is linear in the number M of parties. Furthermore, in the decryption phase, each party computes their local decrypted share and then adds a freshly generated noise. This introduction of noise differs from the decryption algorithm used in the single-party BFV scheme and is crucial for addressing the RLWE problem within a distributed setting.

Scheme 2 **DBFV**$(t, q, N, w, \sigma, B, M)$

```
**Input:** $t, q, N, w, \sigma, B$: Security Parameters; $M$: Number of parties
**DBFV.SecKeyGen**() :
  Each party $P_i$ :
      $s_i \leftarrow S_{R_q}$
      **Output:** $\langle sk \rangle_i = s_i$. Note $sk = \left[\sum_i^M \langle sk \rangle_i\right]_q$
**DBFV.ColPubKeyGen**$(\langle sk \rangle_1, \ldots, \langle sk \rangle_i, \ldots, \langle sk \rangle_M)$:
  Any : $p_1 \leftarrow U(R_q)$. Disclose to all parties.
  Each party $P_i$ :
      $e_i \leftarrow \chi_{[R_q]}$
      $\langle p_0 \rangle_i = -p_1 \cdot \langle sk_i \rangle + e_i$
  Any: **Output:** $cpk = (p_0, p_1) = \left(\sum_i \langle p_0 \rangle_i, p_1\right)$
**DBFV.Encrypt**$(cpk, m)$:
  Any: **Output:** $c_m = BFV.Encrypt(cpk, m)$.
**DBFV.ColDecrypt**$(c_m, \langle sk \rangle_1, \ldots, \langle sk \rangle_i, \ldots, \langle sk \rangle_M)$:
  Each party $P_i$ :
      Let $c_m = (c_{m_0}, c_{m_1})$.
      $e_i \leftarrow \chi_{R_q}$
      $\left\langle c_{m_{1s}} \right\rangle_i = \langle sk \rangle_i \cdot c_{m_1} + e_i$
  Any: **Output:** $m_{res} = \left[\left\lfloor \frac{t}{q} \left[c_{m_0} + \sum_i^M \left\langle c_{m_{1s}} \right\rangle_i\right]_q \right\rceil\right]_t$
```

3.4 Plaintext encoding and packing

Despite advances on the efficiency of homomorphic encryption, homomorphic operations remain slow because of the large size of ciphertexts (which is due to the security requirements of the HE scheme). To address this, SIMD (Single Instruction, Multiple Data) instructions were proposed: they consist of packing the encryption of multiple values in one ciphertext and further customizing the actual operation, accordingly. For the inner product computation, the literature counts two encoding techniques [3]: (i) packed-matrix encoding, where multiple vectors are encoded within a single polynomial, and (ii) packed-integer encoding, where the same value is cloned to all slots. This latter method is notably employed in [35]. Table 1 shows the overall complexity comparison between the two encoding schemes.

We opt to use these two techniques and study their performance in the context of biometric identification, namely, for matching a freshly encrypted live template against an encrypted database of K biometric templates. More specifically, a biometric template is denoted as $\boldsymbol{Y_i}$ and is represented by a vector of dimension l (for l features) as follows:

$$\mathbf{Y}_i = \begin{bmatrix} Y_i^{(1)} & Y_i^{(2)} & \cdots & Y_i^{(l)} \end{bmatrix}$$

We denote the live template as $\boldsymbol{x}$:

$$\boldsymbol{x} = \begin{bmatrix} x^{(1)} & x^{(2)} & \cdots & x^{(l)} \end{bmatrix}$$

Table 1: Computational Complexity of Matrix-Vector Multiplication in Inner Product Computations (Database of size K, l-dimensional vectors, and Ring Polynomials of Degree N)

Encoding Scheme	Packed-Matrix Encoding	Packed-Integer Encoding	Ratio
Ciphertexts	$\lceil (K \cdot l)/N \rceil$	$\lceil (K/N) \rceil \times l$	1
Multiplications	$\lceil ((K \cdot l)/N \rceil$	$\lceil (K/N) \rceil \times l$	1
Additions	$\lceil (K \cdot l)/N \rceil \cdot \log_2(l)$	$\lceil (K/N) \rceil \times l$	$\log_2(l)$
Rotations	$\lceil (K \cdot l)/N \rceil \cdot \log_2(l)$	0	N.A.
Decryptions	$\lceil (K \cdot l)/N \rceil$	$\lceil (K)/N \rceil$	1

Packed-Matrix Encoding. This encoding scheme consists of packing multiple plaintext vectors in one ciphertext. Multiple template vectors from the database are packed in a single vector until reaching the size of the ciphertext, i.e., N. Thus, each ciphertext can encapsulate up to N/l templates. For the sake of clarity, we assume that K is a multiple of N/l and we obtain:

$$Y^{'} = \begin{bmatrix} Y_1 & Y_2 & \cdots & Y_{N/l} \\ Y_{(N/l)+1} & Y_{(N/l)+2} & \cdots & Y_{2N/l} \\ \vdots & \vdots & \ddots & \vdots \\ Y_{K-(N/l)+1} & Y_{K-(N/l)+2} & \cdots & Y_K \end{bmatrix} \in \mathbb{Z}^{\frac{K \cdot l}{N} \times N}$$

On the other hand, the fresh template x is cloned N/l times

$$\boldsymbol{X} = \begin{bmatrix} \boldsymbol{x} & \boldsymbol{x} & \cdots & \boldsymbol{x} \end{bmatrix} \in \mathbb{Z}^N$$

To encode the entire database, a total of $(K \cdot l)/N$ polynomials would be needed and the operations would be performed by encoding each row of the $Y^{(')}$ at a time. If Kl is not a multiple of N, the last encoded polynomial will be padded with zero values until the degree of the polynomial is reached. To compute the inner product of each template $\boldsymbol{Y_i}$ from the database with the live template $\boldsymbol{x}$ in the encrypted domain, $(K \cdot l)/N$ multiplications followed by cumulative additions ($log_2(l)$ rotations and additions per ciphertexts of the encrypted database) will be performed (see appendice A for more details).

Packed-Integer Encoding. This packing scheme corresponds to a feature-wise encoding of the database. Instead of concatenating each template vector, we consider a matrix $Y^{''}$ where each row encapsulates the j^{th} feature from every template in the database:

$$\boldsymbol{Y}^{''} = \begin{bmatrix} \boldsymbol{Y}_1^T & \boldsymbol{Y}_2^T & \cdots & \boldsymbol{Y}_K^T \end{bmatrix} \in \mathbb{Z}^{l \times K}$$

i.e.,

$$\boldsymbol{Y}^{''} = \begin{bmatrix} Y_1^{(1)} & Y_2^{(1)} & \cdots & Y_K^{(1)} \\ Y_1^{(2)} & Y_2^{(2)} & \cdots & Y_K^{(2)} \\ \vdots & \vdots & \ddots & \vdots \\ Y_1^{(l)} & Y_2^{(l)} & \cdots & Y_K^{(l)} \end{bmatrix} \in \mathbb{Z}^{l \times K}$$

On the other hand, the live template will be encoded in a matrix $X^{'}$ by cloning each feature $x^{(j)}$ N times :

$$\boldsymbol{X}^{'} = \begin{bmatrix} \boldsymbol{x}^T & \boldsymbol{x}^T & \cdots & \boldsymbol{x}^T \end{bmatrix}^T \in \mathbb{Z}^{N \times l}$$

i.e.,

$$\boldsymbol{X}^{'} = \begin{bmatrix} x^{(1)} & x^{(2)} & \cdots & x^{(l)} \\ x^{(1)} & x^{(2)} & \cdots & x^{(l)} \\ \vdots & \vdots & \ddots & \vdots \\ x^{(1)} & x^{(2)} & \cdots & x^{(l)} \end{bmatrix} \in \mathbb{Z}^{N \times l}$$

Each feature from N enrolled templates at a time are encoded across all slots of a single polynomial, thus resulting in l polynomials. If $K > N$ this step will be repeated until all the database is encoded. If, on the other hand, the database size K is less than N, each polynomial is padded with zero values. Regarding the live template, each column of the matrix $\boldsymbol{X}^{'}$ is encoded in a single polynomial resulting in l encodings, each holding N repetitions of the same features of $\boldsymbol{x}$. To compute the inner product of a subset of N templates $\boldsymbol{Y_i}$ from the database with the live template x in the encrypted domain, l multiplications followed by cumulative additions will be performed (see appendice B for more details). This encoding scheme is not suitable to small databases since the complexity of matrix-vector multiplications in the encrypted domain remains constant (and high).

3.5 FSS

Function Secret Sharing (FSS) was introduced by Boyle et al. in [6]. It is defined for a family of efficiently computable and succinctly described functions $f : \{0,1\}^n \to \mathbb{G}$, with $\mathbb{G}$ representing a finite Abelian group. In a semi-honest, two-party setting, FSS enables the splitting of a function $f \in \mathcal{F}$ into two additive shares, denoted as (f_0, f_1); Each share, f_j, conceals the secret function f, revealing no information about it. For every input x, the output of the secret function $f(x)$ can be reconstructed by separately computing $f_0(x)$ and $f_1(x)$ by each distinct party, resulting in $f_0(x) + f_1(x) = f(x)$. FSS aims to obtain succinct descriptions of f_0 and f_1 using short keys, $\boldsymbol{k_0}$ and $\boldsymbol{k_1}$, called function keys. This approach leads to a fast online phase with only one round of communication. Besides secret sharing a function f, FSS can also be applied with a secret input, concealed from both parties performing the evaluation. This concealment is achieved by adding a random mask r, with $\hat{x} = x + r$. To maintain correctness during evaluation, the mask r is used in generating the function keys $\boldsymbol{k_0}$ and $\boldsymbol{k_1}$.

A central building block of FSS is the Distributed Comparison Function (DCF)[16], defining a family of comparison functions $f^{<}_{\alpha,\beta}$ which output β if $x > \alpha$ and 0 otherwise. Interval Containment (IC) gates in an FSS scheme enable to determine whether a secret value x falls within an interval $[p, q]$. Initially built upon two DCF blocks, Boyle et al. introduced a method to construct IC using a single DCF block (as shown in Fig. 3 of Section 4.1 [5]). Setting $p = 0$ and $q = 2^{n-1}$ allows FSS to secretly share a unit step function for n-bit signed integers, denoted as $1_{0 \leq x \leq 2^{n-1}}$.

In our work, we use IC gates in FSS in the context of biometric identification for comparing a secret identification score derived from the scalar product of two templates, with a fixed threshold. We therefore define two algorithms: the key generation process (FSS.Gen^{IC} - Algorithm 1) and the IC gate evaluation in FSS (FSS.Eval^{IC} - Scheme 3). For the sake of clarity, we do not elaborate on the generation and evaluation of DCF calls and refer to the original protocols in Fig. 1 of Section 3 in [5]. FSS.Gen^{IC} generates a pair of function keys containing descriptions of the shares f_0, f_1 of a given secret function f. The evaluation algorithm, given a function key and a masked secret input, evaluates the interval containment gate and outputs an additive share $f_j(\hat{x})$ of the output $f(x) = f_0(\hat{x}) + f_1(\hat{x})$.

Algorithm 1 **FSS.Gen**$^{IC}(\lambda, n, r) \rightarrow (\boldsymbol{k_0}, \boldsymbol{k_1})$

```
Input: $\lambda, n$: Security parameters; r input mask
Output: $\boldsymbol{k_0}, \boldsymbol{k_1}$: preprocessing keys
1. Define the interval $[p, q]$ for sign extraction:
    $p \leftarrow 0; q \leftarrow 2^{n-1} - 1$
2. Generate DCF for $\gamma$, an arbitrary value above the interval limit:
    $\gamma \leftarrow (2^n - 1) + r$
    $(k_{\gamma 0}, k_{\gamma 1}) \leftarrow Gen_n^{<}(\lambda, \gamma, 1, \mathbb{U}_{[\mathbb{Z}_{2^n}]})$
3. Generate the correction terms to fix overflows:
    $c \leftarrow -1_{p+r>q} + r + 1_{q+r+1>p} + r + 2^n - 1_{1+p+r>p} + 1_{p+r=2^n-1}$
    $c_0 \leftarrow \mathbb{U}_{[\mathbb{Z}_{2^n}]}; c_1 \leftarrow c - c_0$
4. Compose the function keys:
    $\boldsymbol{k_0} \leftarrow (\boldsymbol{k_{\gamma 0}}, c_0); \boldsymbol{k_1} \leftarrow (\boldsymbol{k_{\gamma 1}}, c_1)$
```

Scheme 3 **FSS.Eval**$^{IC}(j, \boldsymbol{k_j}, \hat{x}) \rightarrow o_j$

```
Input: $j$: Party number, $j \in \{0, 1\}$; $\boldsymbol{k_j}$: Function key for $P_j$; $\hat{x}$: Masked public input $(x + r)$.
Output: $o_j$: Additive secret share of $1_{x \in [0, 2^{n-1}-1]}$.
1. Define the interval $[p, q]$ for sign extraction:
    $p \leftarrow 0; q \leftarrow 2^{n-1} - 1$
2. Parse the function key and obtain local overflow term $\eta$:
    $(\boldsymbol{k_{\gamma j}}, c_j) \leftarrow \boldsymbol{k_j}$
    $\eta \leftarrow 1_{\hat{x}>p} - 1_{\hat{x}>q+1}$
3. Evaluate the DCF with two inputs and compute result:
    $o_j^L \leftarrow Eval_n^{<}(j, \boldsymbol{k_{\gamma j}}, 1, \hat{x} - 1)$
    $o_j^R \leftarrow Eval_n^{<}(j, \boldsymbol{k_{\gamma j}}, 1, \hat{x} - q)$
    $o_j \leftarrow j \cdot \eta - o_j^L + o_j^R + c_j$
```

4 OUR SOLUTION

4.1 Idea

We consider an airport biometric access control system use case which aims at authorizing access to the plane to passengers if biometric verification succeeds. Three main entities are involved in the process: a Biometric Identity Provider (BIP), a Gatekeeper (Gate), and a group of passengers (users) seeking plane access. The BIP maintains a database of enrolled users' templates and is responsible for performing biometric identification operations against a live template. The Gate is in charge of extracting a live template from a user seeking access, sending it to the BIP, and making decisions regarding authorizing or denying user access. Additionally, we can include one more entity in our scenario to carry out the prior acquisition of users' templates upon enrollment, which we name Enroller. In this context, storing the database in an unencrypted form poses significant privacy risks.

Motivated by this use case of biometric identification in airports introduced in [19], we combine BFV with FSS to enable third parties to perform biometric identification between a freshly encrypted biometric template and a previously encrypted database of biometric templates. The aim is to make this privacy-preserving identification more efficient and robust thanks to the use of dedicated, preliminary encoding operations before the actual encryption operation. We propose and study the two packing encoding techniques introduced in section E of [3]: the packed-matrix encoding technique packing several templates in a single polynomial and the packed-integer encoding technique encoding one feature element from enrolled templates in a single polynomial. Further, inspired by [19], the ultimate phase of the protocol, involves two parties collaboratively masking and further decrypting the scores in order to securely transit from BFV to FSS to proceed with an efficient comparison operation.

4.2 Monchi participants

In line with the scenario presented below, we formalize the four main participants in our protocol based on their roles:

- *Enroller*, responsible for enrollment, i.e., acquiring and encrypting the reference biometric templates of users seeking registration during the setup phase. These encrypted reference templates are then forwarded to the BIP for secure storage and subsequent phases.
- *BIP*, holding the encrypted database of reference templates and responsible for evaluating the scores by computing the scalar product between a freshly encrypted live template and each encrypted template of the database with BFV.
- *Gate*, in charge of capturing the live biometric template of the users requesting access, encrypting the live template, and forwarding them to the BIP during the identification phase. Later on *Gate* receives the final decision on this actual identification and allows or not the user to access the plane.
- P_0&P_1, two collaborating parties who help evaluate the last comparison operation of biometric identification in an efficient manner (through FSS).

Additionally, Monchi involves a trusted key server responsible for generating the keying and other secret material. This server only plays a role during the setup phase and is considered offline during the actual execution of biometric identification.

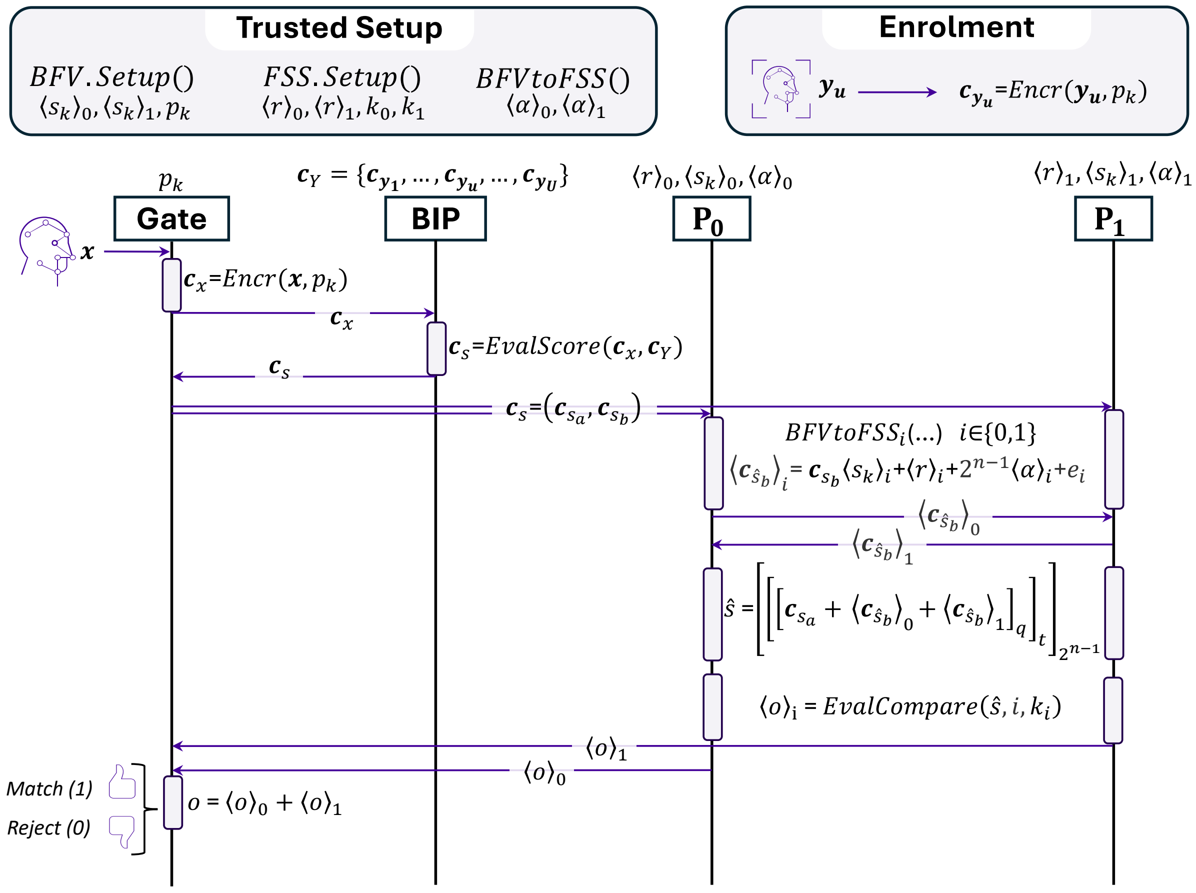
Trusted Setup
$BFV.Setup()$
$\langle s_k\rangle_0, \langle s_k\rangle_1, p_k$
$FSS.Setup()$
$\langle r\rangle_0, \langle r\rangle_1, k_0, k_1$
$BFVtoFSS()$
$\langle \alpha\rangle_0, \langle \alpha\rangle_1$
Enrolment
$\boldsymbol{y_u} \longrightarrow \boldsymbol{c}_{\boldsymbol{y_u}}=Encr(\boldsymbol{y_u}, p_k)$
p_k
$\boldsymbol{c}_Y = \{\boldsymbol{c}_{y_1}, \dots, \boldsymbol{c}_{y_u}, \dots, \boldsymbol{c}_{y_U}\}$
$\langle r\rangle_0, \langle s_k\rangle_0, \langle \alpha\rangle_0$
$\langle r\rangle_1, \langle s_k\rangle_1, \langle \alpha\rangle_1$
Gate
BIP
$\mathbf{P_0}$
$\mathbf{P_1}$
$\boldsymbol{x}$
$\boldsymbol{c}_x=Encr(\boldsymbol{x}, p_k)$
$\boldsymbol{c}_x$
$\boldsymbol{c}_s=EvalScore(\boldsymbol{c}_x, \boldsymbol{c}_Y)$
$\boldsymbol{c}_s$
$\boldsymbol{c}_s=(\boldsymbol{c}_{s_a}, \boldsymbol{c}_{s_b})$
$BFVtoFSS_i(\dots) \quad i\in\{0,1\}$
$\langle \boldsymbol{c}_{\hat{s}_b}\rangle_i = \boldsymbol{c}_{s_b}\langle s_k\rangle_i + \langle r\rangle_i + 2^{n-1}\langle \alpha\rangle_i + e_i$
$\langle \boldsymbol{c}_{\hat{s}_b}\rangle_0$
$\langle \boldsymbol{c}_{\hat{s}_b}\rangle_1$
$\hat{s} = \left[\left[\left[\boldsymbol{c}_{s_a} + \langle \boldsymbol{c}_{\hat{s}_b}\rangle_0 + \langle \boldsymbol{c}_{\hat{s}_b}\rangle_1\right]_q\right]_t\right]_{2^{n-1}}$
$\langle o\rangle_i = EvalCompare(\hat{s}, i, k_i)$
$\langle o\rangle_1$
$\langle o\rangle_0$
Match (1)
Reject (0)
$o = \langle o\rangle_0 + \langle o\rangle_1$

Figure 1: System Diagram of a biometric access control using Monchi's algorithm.

4.3 Threat Model

In this scenario, we consider all parties to be semi-honest, namely an adversary follows the protocol correctly but aims at extracting maximum information possible about the input templates and the resulting score. The simplicity and linear nature of distance metric functions yield substantial input leakage. An adversary may leverage this leakage to extract information about the inputs by inverting the used distance metric function [28]. Hence the score also needs strong protection. The outputs of a private protocol can also leak information about the inputs, as evidenced by model extraction attacks [34] and membership inference attacks [33] in privacy-preserving machine learning inference. Our approach minimizes the output information to the least amount of information necessary, yielding a one-bit output that indicates a match or no match against the encrypted database. By employing this technique, we achieve an optimally minimal input leakage.

4.4 Monchi Overview

Monchi is defined in two phases and six algorithms, as illustrated in Figure 1. The setup phase consists of the generation and distribution of the keying and security material (Algorithm 1) and the appropriate encryption and storage of the database of biometric templates (Algorithm 2). The identification phase starts with Gate extracting the live template, encrypting, and sending it to BIP. Upon receiving the encrypted live template, BIP computes the scalar product of this template together with each template of the database (Algorithm 3). The resulting encrypted scores are further sent to P_1 and P_0 who transform these scores through Algorithm 4 in order to apply FSS to perform the secure comparison (Algorithm 5) with the threshold and return the result to Gate. Finally, with Algorithm 6 Gate reconstructs this result.

4.5 Monchi Algorithms

Algorithm 1: Monchi.Setup. This algorithm consists of the generation and the distribution of the relevant security and keying materials for BFV, FSS, and the secure transition from BFV to FSS. The key server first generates the encryption key for BFV as well as two shares of the corresponding secret key; For a 1:K biometric identification, K pairs of FSS keys (k_0, k_1) are generated together with K masks r which are secret-shared according to threshold θ. r is used to mask the score before the evaluation of FSS and represent n-bit signed integers uniformly drawn. Finally, to securely transit from BFV to FSS, the key server also generates K random α from $U(\mathbb{Z}_{2^m})$ and secret-shares it. α is uniformly drawn from the

range -2^{m-1} to 2^{m-1}, where m is defined as the maximum bit size that ensures both the correctness of the decrypted results and the minimal leakage of private inputs to FSS.

```
Algorithm 2 MONCHI.Setup($K, pp_{(BFV)}, pp_{(FSS)}, m$)
    Players: Key server
    Input: $K$: Number of enrolled users; $pp_{(BFV)}$ $(t, q, N, \sigma, B)$: BFV public parameters; $pp_{(FSS)}$ $(\lambda, n, \theta)$: FSS parameters; $m$ Security parameter for BFV to FSS.
    Output: $\langle sk\rangle_0, \langle sk\rangle_1$ $pk$: DBFV Secret key shares and public key
    $\langle \boldsymbol{r}\rangle_0, \langle \boldsymbol{r}\rangle_1, \boldsymbol{k}_0, \boldsymbol{k}_1$: Random mask shares and FSS function keys
    $\langle \boldsymbol{\alpha}\rangle_0, \langle \boldsymbol{\alpha}\rangle_0$: Additional random shares to prevent from leakage when switching from BFV to FSS.
    1. Run BFV Setup:
        BFV.SecKeyGen() $\rightarrow \langle sk\rangle_1, \langle sk\rangle_1$
        BFV.PubKeyGen($\langle sk\rangle_1 + \langle sk\rangle_0$) $\rightarrow pk$
    2. Run FSS gate setup. For each template in the database, do:
        $r \leftarrow \mathbb{U}_{[\mathbb{Z}_{2(n)}]}$
        FSS.Gen$^{IC}$($\lambda, n, r$) $\rightarrow k_0, k_1$
        $\langle r\rangle_0, \langle r\rangle_1 \leftarrow r$ and $\langle r\rangle_1 \leftarrow \langle r\rangle_1 - \theta$
    3. Run BFVtoFSS setup. For each template in the database, do:
        $\boldsymbol{\alpha} \leftarrow \mathbb{U}_{[\mathbb{Z}^K_{2m}]}$
        $(\langle \boldsymbol{\alpha}\rangle_0, \langle \boldsymbol{\alpha}\rangle_1) \leftarrow \boldsymbol{\alpha}$
```

Algorithm 2: MONCHI.*Encr.* In this algorithm, Enroller first encodes the templates of the database and encrypts this encoded information with BFV using pk. The encrypted database is then transmitted to BIP. Gate also uses the same algorithm to encode and encrypt the live template.

```
Algorithm 3 MONCHI.Encr($pk, V$)
    Players: Enroller or Gate
    Input: $pk$: BFV public key; $V$: Input vector(s) (either the database of K templates or the freshly extracted live template)
    Output: $C_V$: Matrix of encrypted template(s).
        $P_V = BFV.PackEncode(V)$
        $C_V = BFV.Encrypt(pk, P_V)$
```

Algorithm 3: MONCHI.*EvalScore.* BIP executing this algorithm performs the scalar product between the encrypted live template and each ciphertext in the encrypted database. If the templates are encoded using packed-matrix encoding, the encrypted live template consists of a single ciphertext, encrypting N/l instances of the live template and the encrypted database matrix contains $(K \times l)/N$ ciphertexts. The evaluation involves SIMD multiplications followed by cumulative additions, ($log_2(l)$ rotations and additions per ciphertext in the encrypted database matrix C_Y). Consequently, the resulting matrix of encrypted scores includes $(K \times l)/N$ ciphertexts, each storing N/l encrypted scores. Conversely, if the templates are packed with packed-integer encoding, the matrix encapsulating the encryption of the live template will be composed l ciphertexts. Each of these ciphertexts contains N slots, with each slot holding the encryption of the same feature of the live template, having N encryptions of each feature in each ciphertext. The encrypted database matrix will be composed of $(K \times l)/N$ ciphertexts, each containing N features from N templates. Computing the scalar product requires only l multiplications followed by additions for each ciphertext in the encrypted database C_Y. Rotations are no longer required as the coefficients of the ciphertexts are already aligned. As a result, the resulting matrix of encrypted scores contains K/N ciphertexts, with each ciphertext holding N encrypted scores.

```
Algorithm 4 MONCHI.EvalScore($C_Y, C_x$)
    Players: BIP
    Input: $C_Y$: Encrypted Database matrix; $C_x$ Encrypted live template.
    Output: $C_s$: Encrypted scores matrix.
        $C_s = BFV.MatMul(C_Y, C_x)$
```

Algorithm 4: MONCHI.*BFVtoFSS.* P_0 and P_1 who have received the encrypted shares of scores first masks them before their decryption. This step is needed for the subsequent comparison step which is executed using FSS. The masking of each score with r is performed via addition. To keep consistency with the FSS-based works [5], the local shares additions needs to be performed modulo 2^n. Due to the high computational cost of performing this modulus operation in the encrypted domain, we decide to perform the modulus reduction post-decryption. However, this approach introduces a security vulnerability by potentially exposing information about the results. To mitigate this risk while still performing modulus reduction after decryption, an additional mask α is introduced and secret-shared among the two parties. These shares are multiplied with 2^n to shift them with n bits to the left before being added concurrently with the local shares used for masking during the intermediate decryption. This approach enables the removal of the additional share α via a modulus operation in $\mathbb{Z}_2^n$ post-decryption, while decreasing the probability of leakage associated with modulus reduction performed in cleartext (see Section 5 for more details).

Algorithm 5: MONCHI.*EvalCompare.* Once the scores are collaboratively decrypted and transformed to become inputs to FSS, the two parties evaluate the interval containment gate to determine whether the scores are below or above the fixed threshold θ.

Algorithm 6: MONCHI.*Result.* Gate reconstructs the results of the identification from the arithmetic shares, which are derived from the evaluation of the interval containment gate in the FSS scheme, using the decrypted randomized and masked scores.

5 SECURITY ANALYSIS

5.1 Overview

We begin by succinctly analyzing the security of MONCHI.

Firstly, the security of MONCHI directly derives from the security of BFV and FSS. Nevertheless, since BFV and FSS are not executed in the same rings, during $BFVtoFSS$ algorithm, the extra term α_i is added. In $BFVtoFSS$, similar to Funshade, the score is first masked with r before its decryption. At some point during decryption, both

Algorithm 5 MONCHI.**BFVtoFSS**($c_s, \langle sk\rangle_j, \langle r\rangle_j, \langle\alpha\rangle_j$)

```
Players: $P_j$, $j \in \{0,1\}$ computing parties
Input:
  $\langle sk\rangle_j$: Secret key share for BFV
  $\langle \boldsymbol{r}\rangle_j$: Vector of secret mask shares for FSS
  $\langle \boldsymbol{\alpha}\rangle_j$: Vector of secret mask shares for BFV to FSS
  $c_s$: Matrix of ciphertext of the encrypted scores.
Output: Masked and randomized decrypted scores $\hat{s}$.
For each ciphertext $c_s$ of the matrix $C_s$:
  Let $c_s = (c_{s_a}, c_{s_b})$
  $\langle \boldsymbol{r}\rangle_j \leftarrow BFV.PackEncode(\langle \boldsymbol{r}\rangle_j)$
  $\langle \boldsymbol{\alpha}\rangle_j \leftarrow BFV.PackEncode(\langle \boldsymbol{\alpha}\rangle_j)$
  $e_i \leftarrow \chi_{R_q}$
      $\langle c_{s_b}\rangle_i = \langle sk\rangle_j c_{s_b} + \langle \boldsymbol{r}\rangle_j + \langle \boldsymbol{\alpha}\rangle_j * 2^n + e_i$
      $\hat{s} = \left[\left[\frac{t}{q}\left[(c_{s_a} + \langle c_{s_b}\rangle_0 + \langle c_{s_b}\rangle_1)\right]_q\right]_t\right]_{2^{n-1}}$
```

Algorithm 6 MONCHI.**EvalCompare**($\hat{s}, j, k_j$)

```
Players: $P_j$, $j \in \{0,1\}$ computing parties
Input: $\hat{s}$:masked scores; $k_j$: FSS key.
Output: $\boldsymbol{o_j}$: Arithmetic shares.
For each masked score $\hat{s}_i$ of $\hat{s}$:
  $\langle o\rangle_i \leftarrow FSS.Eval^{IC}(j, k_{vecj}[i], \hat{s}_i)$
  $\boldsymbol{o_j} \leftarrow \boldsymbol{o_j}.append(\langle o\rangle_i)$
```

Algorithm 7 MONCHI.**Result**($\boldsymbol{o_0}, \boldsymbol{o_1}$)

```
Players: Gate
Input: $(\boldsymbol{o_0}, \boldsymbol{o_1})$: arithmetic shares of the results
Output: $\boldsymbol{o}$: Results of the comparison $f^{<}_{\theta,1}(\hat{s})$
  $\boldsymbol{o} = \boldsymbol{o_0} + \boldsymbol{o_1}$
```

P_0 and P_1 obtain $s + r$ without any modular reduction, and, hence, some leakage may appear. Such leakage is mitigated by the introduction of the new α. Considering that s (resp. α) is drawn uniformly at random with n (resp. m) bits, given the sum $s + r + 2^n\alpha$, all values of s can be expected – and an attacker cannot get any information on the score – with an overwhelming probability of $1 - \frac{2^n-1}{2^{(n+m)}}$. This term α cancels out during the modular reduction $\mod 2^n$ which precedes the beginning of the MONCHI.Eval algorithm.

A more detailed formal analysis can be found below.

5.2 Security Proof

We consider security against a Honest-but-Curious adversary $\mathcal{A}$ that corrupts up to one of the parties $\{Enrollment, BIP, Gate, P_0, P_1\}$. We consider a static corruption model where the adversary must choose which participant to corrupt before the execution of the computations. This is a standard security model in previous MPC frameworks [5, 9, 11, 25, 29]. Under this threat model, we define and later prove the privacy and correctness of our constructions. Later, we extend our security to certain combinations of corruptions.

We employ the standard real world - ideal world paradigm, providing a simulator for each party corrupted by the ideal adversary $\mathcal{A}'$ such that the adversary cannot distinguish the simulator-led interactions of $\mathcal{A}'$ from the real-world view of $\mathcal{A}$ with the party/ies it corrupted. The ideal world simulation contains an additional trusted party that receives all the inputs from all parties, computes the ideal functionality correctly and sends the corresponding results back to the corresponding parties. Conversely, the MONCHI protocol is executed in real world in the presence of $\mathcal{A}$.

Our security proof works in the ($\mathcal{F}_{\text{MONCHI.setup}}$)-hybrid model, that is, grounded on the faithful execution of MONCHI.setup by a trusted party that generates and distributes each piece of setup material to its designated recipient.

Ideal Functionality $\mathcal{F}_{identif}$
$\mathcal{F}_{identif}$ interacts with $Enrollment, BIP, Gate, P_0, P_1$ and the simulator $\mathcal{S}$, and is parameterized by the inputs to MONCHI.setup and the template length l.

- ***Inputs***: $\mathcal{F}_{identif}$ receives $\boldsymbol{x}$ from *Gate* and, $\boldsymbol{Y}$ from *Enrollment*.
- ***Computation***: $\mathcal{F}_{identif}$ sets $\boldsymbol{s} = \{Y_i \boldsymbol{x}^T\}_{\forall i \in \{1...K\}}$ and obtains $\boldsymbol{o} = \{\boldsymbol{s_i} \geqslant \theta\}_{\forall i \in \{1...K\}}$
- ***Output***: Sends $\boldsymbol{o}$ to *Gate*.

THEOREM 5.1 (SECURITY OF MONCHI). *For each party in {BIP, Enrolment, Gate, P_0, P_1}, there exists a PPT algorithm $\mathcal{S}$ (simulator) such that $\forall \theta \in \mathbb{Z}^*_{n^+}$, $\forall \boldsymbol{x} \in \mathbb{Z}^l_n$, $\forall \boldsymbol{Y} \in \mathbb{Z}^{l\times K}_n$, $\mathcal{S}$ realizes the ideal functionality $\mathcal{F}_{identif}$, such that its behavior is computationally or statistically indistinguishable from a real world execution of the* MONCHI *protocol (consisting of the sequential execution of algorithms 4, 5, 6 by the designated players) in the presence of a static semi-honest adversary $\mathcal{A}$ corrupting said party.*

PROOF. We define a simulator $\mathcal{S}$ for each possible corruption during the execution of each sequential step in MONCHI protocol:

- For corrupt *Enrollment*: Since it does not receive any messages from any other party, it suffices for $\mathcal{S}$ to run $\boldsymbol{C_Y} = \{\text{MONCHI.Encr}(pk, \boldsymbol{Y_i})\}_{\forall i \in \{1...K\}}$ under BFV using pk with the chosen packing technique and give it to $\mathcal{A}'$, then yield $\boldsymbol{Y}$ to $\mathcal{F}_{identif}$. The adversary cannot distinguish anything, as the valid $\boldsymbol{C_Y}$ is enough to perfectly simulate his view of the protocol.
- For corrupt *BIP*: $\mathcal{S}$ artificially sets $\boldsymbol{Y'} = \boldsymbol{0}_{\mathbb{Z}^{l\times K}_n}$ and $\boldsymbol{x'} = \boldsymbol{0}_{\mathbb{Z}^l_n}$, encrypts them into $C_{Y'}$ & $C_{x'}$ under BFV using pk with the chosen packing technique and gives both to $\mathcal{A}'$. It also gives $C_{s'} = \{\text{MONCHI.EvalScore}(C_{Y'}, C_{x'})$ to $\mathcal{A}'$. For all these ciphertexts, any non-negligible advantage to distinguish $C_{s'}$ from C_s, $C_{x'}$ from C_x or $C_{Y'}$ from C_Y would immediately provide a non-negligible advantage to solve the decisional RLWE problem of BFV [14].
- For corrupt *Gate*: $\mathcal{S}$ runs MONCHI.Encr$(pk, \boldsymbol{x})$ and gives it to $\mathcal{A}'$, then it artificially sets $\boldsymbol{s'} = \boldsymbol{0}_{\mathbb{Z}^K_n}$, encrypts it under BFV using pk with the chosen packing technique and give it to $\mathcal{A}'$. Again, any non-negligible advantage to distinguish $C_{s'}$ from C_s would immediately provide a non-negligible advantage to solve the decisional RLWE problem of BFV [14].

- For corrupt P_j with $j \in \{0,1\}$: $\mathcal{S}$ artificially samples $\boldsymbol{s'}$ from $\mathcal{U}[\mathbb{Z}_{2^n}^K]$, encrypts it into $C_{s'}$ under BFV using pk and gives it to $\mathcal{A}'$. $\mathcal{S}$ runs Monchi.BFVtoFSS to generate share j of $\hat{s}'$ and gives this share to $\mathcal{A}'$. Then, $\mathcal{S}$ computes the other share $1-j$ so that their reconstruction yields $\hat{s}'$ and yields it to $\mathcal{A}'$. Finally, $\mathcal{S}$ runs Monchi.EvalCompare($\hat{s}'$, and gives the output share o_j to $\mathcal{A}'$.
  Crucially, in the absence of the masking term α, the real-world adversary would receive $\hat{s} = s + r$ without any modular reduction, thus he would have a meaningful advantage at distinguishing between the distributions of $\hat{s}$ and $\hat{s}'$ and thus breaking the simulation. By adding α drawn uniformly at random with m bits, the distributions $s + r + 2^n\alpha$ and $s' + r + 2^n\alpha$ are indistinguishable with probability $1 - \frac{2^n-1}{2^{(n+m)}}$, based on the probability of obtaining a sampling a value of α that falls under the interval of $[1, 2^n - 1]$. This term α cancels out during the 2^n modular reduction, hence having no impact on the FSS gate evaluation.
  For the FSS IC gate, we resort to the simulation-based security of [5, Definition 2] to argue computational indistinguishability of the keys from random strings, hiding the information of r contained in $\boldsymbol{k}_0$ and $\boldsymbol{k}_1$ from $\mathcal{A}/\mathcal{A}'$, and thus allowing the addition of this n-bit uniformly random mask r to perfectly hide the input value s in the signed n-bits interval.

□

Theorem 5.2 (Correctness of Monchi). *For every threshold $\theta \in \mathbb{Z}_{n+}^*$, every pair of input vectors $\boldsymbol{x} \in \mathbb{Z}_n^l$, $\boldsymbol{Y} \in \mathbb{Z}_n^{l\times K}$, and for suitable choices of BFV & FSS parameters, the execution of the* Monchi *protocol fulfills:*

$$\Pr[\text{Monchi}(\boldsymbol{x}, \boldsymbol{Y}, \dots) \neq \mathcal{F}_{identif}(\boldsymbol{x}, \boldsymbol{Y})] \leq 1/negl(pp(BFV))$$

where $negl(pp(BFV))$ is a negligible function for sufficiently large BFV parameters in Monchi.*Setup.*

Proof. To argue the correctness of the BFV-based matrix multiplication we resort to [14, Theorem 1], stating that for an appropriately chosen BFV parameters B, t and q, the BFV scheme supports the evaluation of a circuit of depth L_{BFV}, where $L_{BFV} = 1$ in our case for the one single multiplication present in the scalar product.

Adding the mask α has no impact on correctness, as this mask is cancelled out when performing reduction modulo 2^n.

Lastly, we focus on the correctness of the FSS gate. Based on [5, Theorem 3], (FSS.Gen$^{IC}(\lambda, n, r)$ and FSS.Eval$^{IC}(j, \boldsymbol{k}_j^{IC}, \hat{z}_\theta)$) constitute an FSS gate realizing $f(s,\theta) = (s \geqslant \theta)$. Then, following the Correctness definition in [5, Definition 2], we argue that $\Pr[\text{FSS.Eval}^{IC}(0, \boldsymbol{k}_0^{IC}, \hat{s}_\theta) + \text{FSS.Eval}^{IC}(1, \boldsymbol{k}_1^{IC}, \hat{s}_\theta) = (s \geqslant \theta)] = 1$. □

Beyond this security definition, and due to the lack of direct interactivity between *BIP* and $\{P_0, P_1\}$, Monchi also maintains its privacy guarantees if $\mathcal{A}$ corrupts both *BIP* and one of $\{P_0, P_1\}$. This can be proven straightforwardly combining the simulator for corrupted *BIP* with that of one corrupted party out of $\{P_0, P_1\}$. A similar argument can be made for the case where $\mathcal{A}$ corrupts both *Enrollment* and one of $\{P_0, P_1\}$. Finally, *Enrollment* and *BIP* may be collapsed into a single party while preserving the privacy of the fresh template and the biometric operations, yet relinquishing the privacy of the reference template database, which is why we presented them as separate entities.

6 PERFORMANCE EVALUATION

In this section, we describe the results of our experimental evaluation of Monchi.

6.1 Environment setting

We implemented Monchi with the two packing methods in Golang with efficient C blocks and using several cryptography libraries including: the (2, 2) threshold variant of the BFV implementation of Lattigo [27] for Monchi.Encrypt and Monchi.BFVtoFSS partially, and Funshade's FSS algorithm for Monchi.EvalCompare. Monchi was implemented and executed on an Intel Core i7-7700 CPU, 3600 MHz with 4 physical cores available; To fully leverage the CPU capabilities, multi-threading was used, running the 4 cores in parallel.

Regarding security parameters, collaborative decryption requires the use of an adequately large *smudging noise*[10] (the e_i term). Accordingly, the BFV parameters were set as follows: the polynomials degree is set to $N = 8192$, and, the size of the ciphertext modulus to 162 bits. Based on this choice of security parameters, we chose the maximum available plaintext modulus, and set the size of t to ~ 32 bits[4]. For FSS we work with $n = 16$-bit modular arithmetic. Finally, in order to avoid errors while adding α be random m signed bits integers where $m = t - n = 16$. The reason behind the choice of a large plaintext modulus for the BFV scheme is primarily driven by the need to achieve the best performance results while keeping a high security degree and avoid errors during decryption.

To study and evaluate the performance of Monchi according to the airport use case, we have used the Labeled Faces in the Wild (LFW) dataset [18], the publicly available dataset for face verification. The dataset includes 13,233 facial images of 5,749 individuals. To process these images and extract templates, we used 4 pre-trained neural network models – Swin Transformer [22] – and, obtain templates size of 64, 128, 256, 512 floats. We employ the False Acceptance Rate (FAR) and False Rejection Rate (FRR) as key metrics, as defined in [21], to assess the impact of quantization on our facial recognition model's accuracy.

6.2 Quantization and Accuracy

In this section, we study the quantization to minimize the accuracy degradation. The extracted features are quantized to fit BFV integer inputs. Given that templates are normalized so that scores range between -1 and 1, the quantization factor defines the range of possible scores. Hence, the quantization factor has a direct impact on the accuracy of biometric identification and the security of Monchi. Given that n is the security parameter for FSS $n = 16$, the scores' range should not exceed $[-2^{n-1}, 2^{n-1} - 1]$. For each template size, the quantization factor has been found to guarantee a degradation lower than 10% rate of FRR at a FAR of 10^{-6}. For simplicity, we pick a fixed quantization factor (qf) of 180.

[4]Note that the plaintext modulus needs to be a prime with special properties to allow ciphertext rotations [7]

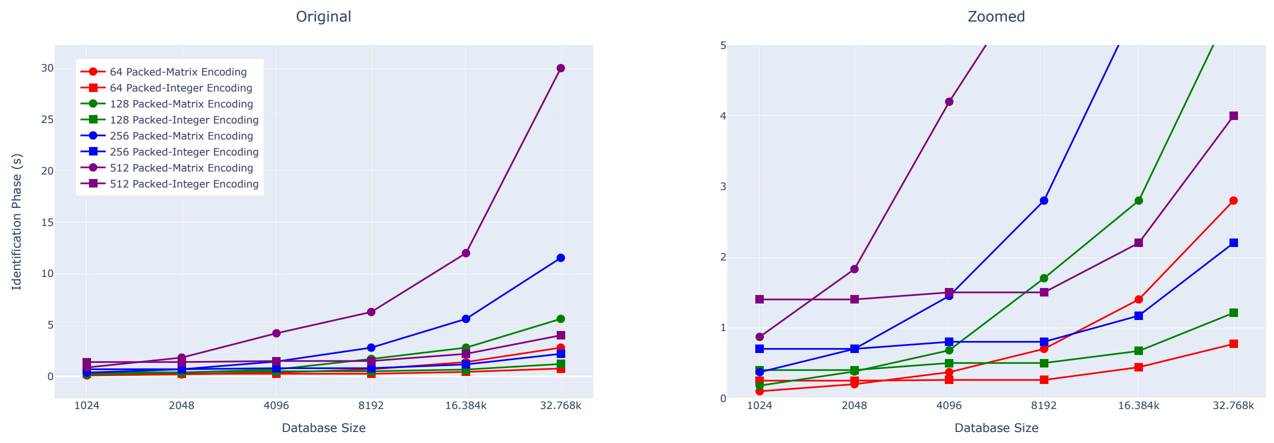

Figure 2: Execution time of 1:K identification with Monchi for l-dimensional templates using a ring with $N = 8192$

Table 2: Computation overhead of Monchi (online phase, $N = 8192$, $qf = 180$) running the 4 cores in parallel.

DB Size	Dim	Packed-Matrix Encoding						Packed-Integer Encoding					
		Time (s)				Total	Total	Time(s)				Total	Total
(K)	(l)	Enc	EvalScore	BFVtoFSS	EvalCompare	Time (s)	Mem (MB)	Enc	EvalScore	BFVtoFSS	EvalCompare	Time (s)	Mem (MB)
1024	64	0.003	0.09	0.024	0.015	0.132	8	0.06	0.14	0.006	0.015	0.221	52
	128	0.003	0.15	0.055	0.015	0.223	14	0.09	0.27	0.006	0.015	0.381	101
	256	0.003	0.28	0.123	0.015	0.421	26	0.24	0.45	0.006	0.015	0.711	202
	512	0.003	0.67	0.226	0.015	0.914	52	0.34	0.90	0.006	0.015	1.261	403

Table 3: Impact of Quantization on Accuracy at $FAR = 10^{-6}$

(l)	FRR_{org}	FRR_{qnt}
64	0.0187	0.0189
128	0.0089	0.0086
256	0.0025	0.0025
512	0.0031	0.0031

Table 3 presents the FRR before (FRR_{org}) and after (FRR_{qnt}) quantization for a fixed $FAR = 10^{-6}$. In all cases, the impact of quantization is very low, and it is negligible above $l = 256$.

6.3 Performance evaluation

To evaluate the performance of Monchi and study the effectiveness of the two encoding schemes, we conducted experiments with different database sizes and across various template dimensions.

Table 2 depicts the computational cost of Monchi's identification phase. comparing the two encoding schemes for an encrypted database of 1024 templates. The packed-matrix encoding yields a quicker process for the encryption of freshly extracted templates (Monchi.Encr), around 30 milliseconds (ms) for 512-dimensional templates compared to 0.34 seconds (s) with packed-integer encoding. The computation of the encrypted scores (Monchi.Evalscore) is also faster, taking around 0.67 s versus 0.90 s with packed-integer encoding. On the other hand, the transformation of scores during the transition from BFV to FHE, involving collective decryption associated with masking and randomization, is faster with packed-integer encoding. Monchi.BFVtoFSS takes 0.006 s with packed-integer encoding (vs. 0.226s with packed-matrix encoding) thanks to the reduced number of ciphertexts holding the encrypted scores. The processing complexity of evaluating the Interval Containment gate (Monchi.EvalCompare) remains constant, regardless of the encoding scheme used. Overall, the identification phase is faster with packed-matrix encoding, requiring 0.914 s with packed-integer encoding for 512-dimensional templates instead of 1.261 s in the case of packed-matrix encoding and the memory required is also lower with packed-matrix encoding (403 MB vs. 52 MB for $l = 512$).

Figure 2 demonstrates the computational complexity of the online phase, covering a broader range of database sizes. We observe that, for small database size (typically less than 2048), packed-matrix encoding outperforms packed-integer encoding. However, as the database size exceeds 2048, there is a notable shift in efficiency. Additionally, the computational cost of the online computation remains constant with packed-integer encoding provided that the database size does not exceed $N = 8192$. The reduced efficiency of packed-integer encoding for small databases can be attributed to two factors: the requirement to encode each feature of the live template into a single polynomial, resulting in l ciphertexts encrypting

Table 4: Computation and Communication overhead of MONCHI's actors (online phase, $N = 8192$, $qf = 180$)

DB Size	Dim	Packed-Matrix Encoding						Packed-Integer Encoding					
		Time (s)			Bandwidth (MB)			Time(s)			Bandwidth (MB)		
(K)	(l)	BIP	Gate	P_j	BIP $\longleftrightarrow$ $Gate$	Gate $\longleftrightarrow$ P_j	P_{1-j} $\longleftrightarrow$ P_j	BIP	Gate	P_j	BIP $\longleftrightarrow$ Gate	Gate $\longleftrightarrow$ P_j	P_{1-j} $\longleftrightarrow$ P_j
1024	64	0.09	0.003	0.039	3.6	3.2	4e-4	0.14	0.06	0.021	26	0.41	5e-5
	128	0.15	0.003	0.070	6.7	6.3	8e-4	0.27	0.09	0.021	51	0.41	5e-5
	256	0.28	0.003	0.14	13	12	1.6e-3	0.45	0.24	0.021	101	0.41	5e-5
	512	0.67	0.003	0.24	26	25	3.2e-3	0.90	0.34	0.021	201	0.41	5e-5

one live template compared to one ciphertext with packed-integer encoding, and the need for padding of each polynomial with $N - K$ zero coefficients, considering that the ring polynomial degree in use is N, K is the database size, and templates are of l-dimension. Furthermore, the better efficiency of packed-integer encoding for larger databases (where $K \geq N$) can be explained with the varying computational demands of the two encoding methods. In packed-matrix encoding, the operations carried out in the encrypted space, involving $\log_2(l)$ cumulative rotations and additions per ciphertext of the encrypted database, become increasingly expensive. Conversely, with packed-integer encoding, the required operations consist of cheaper l cumulative additions for each batch of N templates.

Lastly, Table 4 shows the computational costs and the communication bandwidth of the different actors involved in the identification phase. The BIP and the Gate exchanging the encrypted live template (Gate $\rightarrow$ BIP) and the encrypted scores (BIP $\rightarrow$ Gate) results in a communication cost of 26 MB with packed-matrix encoding compared to 201 MB with packed-integer encoding for 512-dimensional templates. Conversely, the communication cost between the Gate and each party (Gate $\rightarrow P_j$) exchanging encrypted scores and arithmetic shares of the FSS IC gate ($P_j \rightarrow$ Gate) is higher with packed-matrix encoding (25 MB vs. 0.41 MB with $l = 512$).

7 CONCLUSION

MONCHI is a novel protocol for privacy-preserving biometric identification. We build our protocol upon BFV, and FSS, for full correctness for decryption and comparison with a fixed threshold, requiring only a single round of communication among the two parties in the identification phase. To ensure leakage resilience, we introduce local shares $\langle \alpha \rangle$ to securely transition from BFV to FSS. We propose and analyze two packing encoding techniques, packed-matrix encoding and packed-integer encoding. We observe that the primary advantage of packed-integer encoding is its elimination of costly rotational operations during inner product evaluation in the encrypted domain. Conversely, packed-matrix encoding is more suited for operations involving small databases. We analyze practical security aspects of the biometric identification, and implement these protocols on top of the Lattigo library with efficient C blocks.

Acknowledgements. We thank our colleague Michael Brauckmann for his support. This work was also partially supported by the 3IA Côte d'Azur program (ANR19-P3IA-0002).

REFERENCES

[1] Jean-Claude Bajard, Julien Eynard, M Anwar Hasan, and Vincent Zucca. 2016. A full RNS variant of FV like somewhat homomorphic encryption schemes. , 423–442 pages.

[2] Mauro Barni, Tiziano Bianchi, Dario Catalano, Mario Di Raimondo, Ruggero Donida Labati, Pierluigi Failla, Dario Fiore, Riccardo Lazzeretti, Vincenzo Piuri, Alessandro Piva, et al. 2010. A privacy-compliant fingerprint recognition system based on homomorphic encryption and fingercode templates. , 7 pages.

[3] Marcelo Blatt, Alexander Gusev, Yuriy Polyakov, Kurt Rohloff, and Vinod Vaikuntanathan. 2020. Optimized homomorphic encryption solution for secure genome-wide association studies. , 13 pages.

[4] Vishnu Naresh Boddeti. 2018. Secure face matching using fully homomorphic encryption. , 10 pages.

[5] Elette Boyle, Nishanth Chandran, Niv Gilboa, Divya Gupta, Yuval Ishai, Nishant Kumar, and Mayank Rathee. 2021. Function secret sharing for mixed-mode and fixed-point secure computation. , 871–900 pages.

[6] Elette Boyle, Niv Gilboa, and Yuval Ishai. 2015. Function secret sharing. , 337–367 pages.

[7] Zvika Brakerski. 2012. Fully homomorphic encryption without modulus switching from classical GapSVP. , 868–886 pages.

[8] Zvika Brakerski, Craig Gentry, and Vinod Vaikuntanathan. 2014. (Leveled) fully homomorphic encryption without bootstrapping. , 36 pages.

[9] Nishanth Chandran, Divya Gupta, Aseem Rastogi, Rahul Sharma, and Shardul Tripathi. 2019. EzPC: programmable and efficient secure two-party computation for machine learning. , 496–511 pages.

[10] Jung Hee Cheon, Hyeongmin Choe, Alain Passelègue, Damien Stehlé, and Elias Suvanto. 2024. Attacks Against the INDCPA-D Security of Exact FHE Schemes. *Cryptology ePrint Archive* (2024).

[11] Daniel Demmler, Thomas Schneider, and Michael Zohner. 2015. ABY-A framework for efficient mixed-protocol secure two-party computation. , 15 pages.

[12] Léo Ducas and Daniele Micciancio. 2015. FHEW: bootstrapping homomorphic encryption in less than a second. , 617–640 pages.

[13] Joshua J Engelsma, Anil K Jain, and Vishnu Naresh Boddeti. 2022. HERS: Homomorphically encrypted representation search. , 349–360 pages.

[14] Junfeng Fan and Frederik Vercauteren. 2012. Somewhat practical fully homomorphic encryption.

[15] Javier Galbally, Chris McCool, Julian Fierrez, Sebastien Marcel, and Javier Ortega-Garcia. 2010. On the vulnerability of face verification systems to hill-climbing attacks. , 1027–1038 pages.

[16] Niv Gilboa and Yuval Ishai. 2014. Distributed point functions and their applications. , 640–658 pages.

[17] Shai Halevi, Yuriy Polyakov, and Victor Shoup. 2019. An improved RNS variant of the BFV homomorphic encryption scheme. , 83–105 pages.

[18] Gary B Huang, Marwan Mattar, Tamara Berg, and Eric Learned-Miller. 2008. Labeled faces in the wild: A database forstudying face recognition in unconstrained environments.

[19] Alberto Ibarrondo, Hervé Chabanne, Vincent Despiegel, and Melek Önen. 2022. Colmade: Collaborative masking in auditable decryption for bfv-based homomorphic encryption. , 129–139 pages.

[20] Alberto Ibarrondo, Hervé Chabanne, and Melek Önen. 2022. Funshade: Functional secret sharing for two-party secure thresholded distance evaluation.

[21] Anil K Jain, Patrick Flynn, and Arun A Ross. 2007. *Handbook of biometrics*. Springer Science & Business Media, USA.

[22] Ze Liu, Yutong Lin, Yue Cao, Han Hu, Yixuan Wei, Zheng Zhang, Stephen Lin, and Baining Guo. 2021. Swin transformer: Hierarchical vision transformer using shifted windows. , 10012–10022 pages.

[23] Ying Luo, S Cheung Sen-ching, and Shuiming Ye. 2009. Anonymous biometric access control based on homomorphic encryption. , 1046–1049 pages.

[24] Vadim Lyubashevsky, Chris Peikert, and Oded Regev. 2010. On ideal lattices and learning with errors over rings. , 23 pages.

[25] Payman Mohassel and Peter Rindal. 2018. ABY3: A mixed protocol framework for machine learning. , 35–52 pages.

[26] Christian Mouchet, Juan Troncoso-Pastoriza, Jean-Philippe Bossuat, and Jean-Pierre Hubaux. 2021. Multiparty homomorphic encryption from ring-learning-with-errors. , 291–311 pages.

[27] Christian Vincent Mouchet, Jean-Philippe Bossuat, Juan Ramón Troncoso-Pastoriza, and Jean-Pierre Hubaux. 2020. Lattigo: A multiparty homomorphic encryption library in go. , 64–70 pages.

[28] Elena Pagnin, Christos Dimitrakakis, Aysajan Abidin, and Aikaterini Mitrokotsa. 2014. On the leakage of information in biometric authentication. , 265–280 pages.

[29] Arpita Patra, Thomas Schneider, Ajith Suresh, and Hossein Yalame. 2021. {ABY2. 0}: Improved {Mixed-Protocol} Secure {Two-Party} Computation. , 2165–2182 pages.

[30] Oded Regev. 2009. On lattices, learning with errors, random linear codes, and cryptography. , 40 pages.

[31] Théo Ryffel, Pierre Tholoniat, David Pointcheval, and Francis Bach. 2020. Ariann: Low-interaction privacy-preserving deep learning via function secret sharing.

[32] Christian Shahreza, Hatef Otroshiand Rathgeb, Dailé Osorio-Roig, Vedrana Krivokuca Hahn, Sébastien Marcel, and Christoph Busch. 2022. Hybrid Protection of Biometric Templates by Combining Homomorphic Encryption and Cancelable Biometrics.

[33] Reza Shokri, Marco Stronati, Congzheng Song, and Vitaly Shmatikov. 2017. Membership inference attacks against machine learning models. In *2017 IEEE symposium on security and privacy (SP)*. IEEE, 3–18.

[34] Florian Tramèr, Fan Zhang, Ari Juels, Michael K Reiter, and Thomas Ristenpart. 2016. Stealing machine learning models via prediction {APIs}. In *25th USENIX security symposium (USENIX Security 16)*. 601–618.

[35] Juan Ramón Troncoso-Pastoriza, Alberto Pedrouzo-Ulloa, and Fernando Pérez-González. 2017. Secure genomic susceptibility testing based on lattice encryption. In *2017 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE, 2067–2071.

[36] Yang, Wencheng and Wang, Song and Cui, Hui and Tang, Zhaohui and Li, Yan,. 2023. A review of homomorphic encryption for privacy-preserving biometrics. Issue Special Issue on Signal and Image Processing in Biometric Detection.

A PACKED-MATRIX ENCODING

This section details the computation of the inner product between a database of K biometric templates and a live template, for dimension l. We introduce matrices $\boldsymbol{Y'}$ and $\boldsymbol{x}$, representing the database and the live template, respectively, using packed-matrix encoding. We assume that $K = N/l$, obtaining:

$$\boldsymbol{Y_i} = \begin{bmatrix} \mathrm{Y}_1 & \mathrm{Y}_2 & \cdots & \mathrm{Y}_{N/l} \end{bmatrix} \in \mathbb{Z}^{1\times N},$$

$$\boldsymbol{X} = \begin{bmatrix} \boldsymbol{x} & \boldsymbol{x} & \cdots & \boldsymbol{x} \end{bmatrix} \in \mathbb{Z}^{N}.$$

Encrypting $\boldsymbol{Y'}$ and $\boldsymbol{X}$ yields:

$$C_{\boldsymbol{Y'}} = \begin{bmatrix} c_{Y_1^{(1)}} & \cdots & c_{Y_1^{(l)}} & \cdots & c_{Y_{N/l}^{(1)}} & \cdots & c_{Y_{N/l}^{l}} \end{bmatrix}$$

$$C_{\boldsymbol{X}} = \begin{bmatrix} c_{x^{(1)}} & c_{x^{(2)}} & \cdots & c_{x^{(l)}} & \cdots & c_{x^{(1)}} & \cdots & c_{x^{(l)}} \end{bmatrix}$$

First coefficient-wise multiplication (denoted as C_{Mul1}):

$$C_{Mul1} = \begin{bmatrix} c_{x^{(1)}} \cdot c_{Y_1^{(1)}} & \cdots & c_{x^{(1)}} \cdot c_{Y_{N/l}^{(1)}} & \cdots & c_{x^{(l)}} \cdot c_{Y_{N/l}^{(l)}} \end{bmatrix},$$

Left rotation of C_{Mul1} by 1 (denoted as C_{Rot1}):

$$C_{Rot1} = \begin{bmatrix} c_{x^{(2)}} \cdot c_{Y_1^{(2)}} & \cdots & c_{x^{(2)}} \cdot c_{Y_{N/l}^{(2)}} & \cdots & c_{x^{(1)}} \cdot c_{Y_1^{(1)}} \end{bmatrix}$$

Coefficient-wise addition of C_{Mul1} and C_{Rot1} (denoted as C_{Sum1}): $C_{Sum1} =$

$$\begin{bmatrix} c_{x^{(1)}} c_{Y_1^{(1)}} + c_{x^{(2)}} c_{Y_1^{(2)}} & \cdots & c_{x^{(1)}} c_{Y_{N/l}^{(1)}} + c_{x^{(2)}} c_{Y_{N/l}^{(2)}} & \cdots \end{bmatrix}$$

Subsequent left rotation by 3 (denoted as C_{Rot2}): $C_{Rot2} =$

$$\begin{bmatrix} c_{x^{(3)}} c_{Y_1^{(3)}} + c_{x^{(4)}} c_{Y_1^{(4)}} & \cdots & c_{x^{(3)}} \cdot c_{Y_{N/l}^{(3)}} + c_{x^{(4)}} c_{Y_{N/l}^{(4)}} & \cdots \end{bmatrix}$$

We recursively repeat the steps of coefficient-wise rotation, and addition $\log_2(l)$ times, culminating in the computation of C_z:

$$C_z = \begin{bmatrix} \sum_{i=1}^{l} c_{x^{(i)}} \cdot c_{Y_1^{(i)}} & \cdots & \sum_{i=1}^{l} c_{x^{(i)}} \cdot c_{Y_{N/l}^{(i)}} & \cdots \end{bmatrix}.$$

The coefficients at indices $\{0, N/l, \cdots, K - N/l\}$ in C_z contain the encrypted scores for the N/l templates, while other coefficients are disregarded post-decryption.

B PACKED-INTEGER ENCODING

This section introduces a packed-integer encoding scheme for feature-wise representation of a biometric database. Assuming $K = N$ for clarity, we define a matrix $\boldsymbol{Y''}$ where each row captures the j^{th} feature from every template in the database. The matrix $\boldsymbol{X'}$ comprises l rows, each row containing N replications of each feature of the live template.

$$\boldsymbol{Y''} = \begin{bmatrix} Y_1^{(1)} & Y_2^{(1)} & \cdots & Y_N^{(1)} \\ Y_1^{(2)} & Y_2^{(2)} & \cdots & Y_N^{(2)} \\ \vdots & \vdots & \ddots & \vdots \\ Y_1^{(l)} & Y_2^{(l)} & \cdots & Y_N^{(l)} \end{bmatrix} \in \mathbb{Z}^{l\times N},$$

$$\boldsymbol{X'} = \begin{bmatrix} x^{(1)} & x^{(2)} & \cdots & x^{(l)} \\ x^{(1)} & x^{(2)} & \cdots & x^{(l)} \\ \vdots & \vdots & \ddots & \vdots \\ x^{(1)} & x^{(2)} & \cdots & x^{(l)} \end{bmatrix} \in \mathbb{Z}^{N\times l}.$$

Each row of $\boldsymbol{Y''}$ and each column of $\boldsymbol{X'}$ are encoded into single polynomials, resulting after the encryption in l ciphertexts for the entire database and l ciphertexts for the live template:

For $i \in \{1, \cdots, l\}$, we have:

$$C_{\boldsymbol{Y}^i} = \begin{bmatrix} c_{Y_1^{(i)}} & c_{Y_2^{(i)}} & \cdots & c_{Y_N^{(i)}} \end{bmatrix},$$

$$C_{\boldsymbol{X}^i} = \begin{bmatrix} c_{x^{(i)}} & c_{x^{(i)}} & \cdots & c_{x^{(i)}} \end{bmatrix}.$$

The computation of the inner product involves l multiplications and cumulative additions.

For each $i \in \{1, \cdots, l\}$:

$$C_{mul^{(i)}} = BFV.Mul(C_{\boldsymbol{Y}^{(i)}}, C_{\boldsymbol{X}^{(i)}}),$$
$$C_z = BFV.Add(C_{mul}^{i}, C_z).$$

The coefficients in C_z represent encrypted scores, yielding a single ciphertext that encapsulates N encrypted scores.