Secure Biometric Verification in the Presence of Malicious Adversaries

Kamela Al-Mannai [a], Elmahdi Bentafat [b], Spiridon Bakiras [c,*] and Jens Schneider [a]
[a] *Division of Information and Computing Technology, College of Science and Engineering, Hamad Bin Khalifa University, Doha, Qatar*
E-mails: kaalmannai@hbku.edu.qa, jeschneider@hbku.edu.qa
[b] *Information Systems Department, Ahmed Bin Mohammed College, Doha, Qatar*
E-mail: mahdi@abmmc.edu.qa
[c] *Infocomm Technology Cluster, Singapore Institute of Technology, Singapore*
E-mail: spiridon.bakiras@singaporetech.edu.sg

Abstract. In a secure biometric verification system, users authenticate themselves by submitting their encrypted biometric data (i.e., feature vectors) to the application server. Such systems must be able to defend against (i) malicious clients that try to gain unauthorized access to the system; and (ii) malicious servers that aim to identify the users' plaintext biometric data. To this end, our work introduces an efficient biometric verification protocol that is provably secure against both a malicious client and a malicious server. The protocol is based on a two-level homomorphic encryption scheme that is constructed over bilinear groups of prime order. We formally prove the security of our scheme in the random oracle model and also present results from a proof-of-concept implementation using Barreto-Naehrig elliptic curves. Our results demonstrate that the protocol is very efficient in terms of both computation and communication costs.

Keywords: Biometric verification, Biometric data privacy, Secure computation, Pairing-based crypto

1. Introduction

Traditional password-based authentication systems have several drawbacks in terms of security. For example, even if implemented correctly using protocols such as `bcrypt` [1] (which slow down brute-force attacks significantly), most users tend to select rather weak passwords that are also similar across different platforms, because they are easier to memorize. Furthermore, two-factor authentication (2FA) methods using short OTP strings have also been shown to be vulnerable [2]. As a result, researchers have been investigating alternative authentication methods that are more secure and easier to use on behalf of the client.

Biometric verification is the process of authenticating users based on their biometric characteristics, which are unique for every individual. As such, it is an excellent candidate to replace passwords either in a standalone setting or as part of a 2FA protocol. Specifically, in a biometric verification system, clients first enroll their biometric credentials (e.g., facial characteristics) to a remote server. Typically, biometric data consist of a feature vector of length N that is obtained via a variety of AI tools, such as deep learning. The feature vector stored at the server is referred to as the *template*. During the verification

*Corresponding author. E-mail: spiridon.bakiras@singaporetech.edu.sg.

phase, the client prepares a fresh feature vector (called *probe*) that is sent to the remote server, along with the client's ID. The server then computes the similarity between the template and the probe (e.g., using the Euclidean distance) and, if the similarity is above a certain threshold, the client is successfully authenticated.

However, to protect the privacy of the biometric data (template and probe), the computation of the similarity score must be performed in a secure manner. Most existing protocols employ a public-key homomorphic cryptosystem that allows the server to blindly compute the similarity score (typically, the squared Euclidean distance) in the encrypted domain. In particular, the client first initializes a public-key cryptosystem and generates the corresponding key pair. During enrollment, the client uses its public key to encrypt every element of the user's feature vector, and the resulting ciphertexts are sent to the remote server. (The server does not possess the client's secret key and is, thus, unable to decrypt the stored template.) During the verification phase, the client and the server engage in a two-party protocol, where the client's input is the encrypted probe and the server's input is the user's encrypted template. When the protocol terminates, the server outputs the plaintext similarity score between the two vectors that reveals the result of the verification session. Note that, such protocols essentially implement a 2FA functionality, based on (i) the user's biometric data; and (ii) the user's possession of a device that stores the secret key(s).

Most existing protocols for secure biometric verification are designed for either honest-but-curious adversaries [3, 4] or for the case of malicious clients [5, 6]. For example, a malicious client may attempt to successfully authenticate as a legitimate user, by actively manipulating the exchanged messages. However, a malicious server also poses a significant threat to user privacy, because it may lead to an adversary gaining access to a user's plaintext biometric data. Besides being a violation of numerous privacy laws around the world (e.g., the European GDPR legislation), plaintext biometric data can be used to gain unauthorized access to other systems that the user is subscribed to.

To this end, there exist a few protocols in the literature that are secure against malicious adversaries (both client and server). For example, Barni et al. [7] introduce a scheme based on secure multiparty computation (MPC) protocols. While the online stage of their protocol is very efficient, their method necessitates an expensive offline stage that must be executed periodically between every client and the server. On the other hand, Bassit et al. [8] propose a protocol based on threshold homomorphic encryption that does not require any offline computations. Nevertheless, to achieve security against malicious servers, the authors introduce a trusted third-party that is involved in the enrollment phase of their protocol, by digitally signing the users' encrypted templates.

To address such limitations, we introduce a novel biometric verification protocol that is secure against malicious adversaries. More importantly, our protocol is very efficient in terms of both computation and communication costs, and does not depend on a trusted third-party. Specifically, our construction leverages the two-level homomorphic encryption scheme by Attrapadung et al. [9] that allows the server to blindly compute the squared Euclidean distance between two encrypted feature vectors. The protocol is provably secure in the malicious setting, and we outline a formal security proof in the random oracle model.

Furthermore, to illustrate the practicality of our scheme in a real-life application, we built a proof-of-concept system that employs face recognition as the authentication factor. In particular, we utilized the Π-nets [10] platform at the client-side to perform the facial recognition operations, and implemented our secure protocol as a client-server application. The implementation leverages the original pairing-based crypto library developed by Attrapadung et al. [9], which employs computationally efficient pairings over Barreto-Naehrig curves. Our experimental results demonstrate that a verification session incurs just

520ms (resp. 360ms) of compute time at the server (resp. client). Additionally, the overall communication cost between the client and server is just 99KB. To summarize, the contributions of our work are as follows:

(1) We introduce a fast and efficient biometric verification protocol that is secure against malicious adversaries (both client and server).
(2) We formally prove the security of our protocol in the random oracle model.
(3) We present experimental results from a proof-of-concept implementation of a secure face verification system.

The remainder of the paper is organized as follows. Section 2 presents a literature review on secure biometric verification protocols. Section 3 introduces some concepts and tools that we incorporated in our system. Section 4 describes in detail the proposed biometric verification protocol and Section 5 outlines the security proof. Section 6 discusses the proof-of-concept implementation and Section 7 presents the experimental results. Finally, Section 8 concludes our work.

2. Related Work

Previous work on secure biometric verification includes two-party protocols that are secure when either (i) both parties are honest-but-curious (HBC); or (ii) the server is HBC but the client is malicious. Specifically, under the HBC model, the adversary follows the protocol correctly, but is actively trying to learn more information about the other party's input by analyzing the received messages. On the other hand, a malicious adversary may deviate from the protocol specification at any time, e.g., by manipulating the exchanged messages. Typically, secure two-party protocols employ application-specific solutions based on homomorphic encryption or leverage generic protocols, such as garbled circuits [11].

In the HBC adversarial setting, Im et al. [3] modify Paillier's cryptosystem [12] (which is an additive homomorphic encryption scheme) into a multiplicative one, in order to support the computation of the Euclidean distance. They detect the user's palm print with a guided setting, using a smartphone camera, and leverage a random projection technique for feature extraction [13]. The execution time of the proposed protocol is 24.16s with an equal error rate (EER) of 15.20%.

On the other hand, Boddeti [4] proposes using a fully homomorphic encryption (FHE) scheme [14] to support biometric privacy. His method also supports revocability of the biometric templates, by simply changing the encryption-decryption keys. In general, FHE is a rather expensive cryptographic primitive, so the author utilizes the more efficient Fan-Vercauteren scheme [15], which reduces the communication cost from 48.7MB to 16.5MB and the template matching time from 12.5s to 0.6s. The facial features are extracted with both the FaceNet [16] and SphereFace [17] neural networks. The author also proposes a batching technique, based on the Chinese Remainder Theorem, which further reduces the computational cost. With this optimization, the overall time for executing the protocol is under 10ms.

In the malicious client setting, Shahandashti et al. [5] develop a profile matching function over the encrypted domain. The function decides whether a fresh feature value belongs to a distribution of pre-registered values. As the function generates new feature-level scores, a weighted sum approach is used to compute the final verification score. All features are encrypted with an additive homomorphic encryption scheme, such as Paillier. The protocol is also designed to mitigate arbitrary input attacks, by requiring the client to prove that the encryption of the fresh input is well formed. Furthermore, during the decryption request at the client, the server injects additional fake ciphertexts of known values. If there are any inconsistencies on the fake values returned by the client, the verification operation fails.

Šeděnka et al. [6] modify Yao's garbled circuit protocol to be secure against malicious clients. Then, they implement two secure distance computation algorithms on the modified garbled circuit, namely scaled Euclidean and scaled Manhattan. They also propose a protocol for HBC adversaries that is based on additive homomorphic encryption. Their results for the malicious setting indicate that scaled Manhattan outperforms scaled Euclidean (9s vs. 290s of CPU time, and 0.09MB vs. 1.66MB of communication cost). On the other hand, the homomorphic encryption protocol needs only 36ms of CPU time, but incurs a communication cost of 47MB.

Gunasinghe and Bertino [18] suggest to perform the enrollment phase with an identity provider once, and then use the certified ID for authentication with all other service providers. Specifically, during verification, the client proves to the server (in zero knowledge) that he is the owner of the identity, by solving a challenge using the secret parameters. The protocol requires 120MB of resources on the mobile device and is completed in 26s. The authors believe that the one-time enrollment should occur in person, in order to prevent attackers from impersonating other users in this early stage. In the verification phase, they also integrate a key-agreement protocol to prevent known attacks, such as man-in-the-middle and hijacking attacks.

Cheon et al. [19] employ a somewhat homomorphic encryption scheme (SHE) with single-instruction multiple-data (SIMD) operations [20] to optimize the distance computation. They also utilize a lightweight message authentication code (MAC) and apply a ciphertext compression method to further reduce the cost. The optimization process shortens the matching algorithm execution time from 30s to 0.45s. When a Hamming distance algorithm is used, the overall execution time of the protocol is 0.6s. The authors introduce two security measures to defend against (i) incorrect client computations via integrating a MAC verification phase; and (ii) server chosen ciphertext attacks by sending back a randomized plaintext that is still valid for computing the verification result.

Im et al. [21] employ the Catalano-Fiore technique [22] to transform a linear (additive) homomorphic encryption scheme into a scheme that is able to evaluate quadratic functions. Then, they utilize this cryptosystem to compute the squared Euclidean distance between two encrypted feature vectors. Feature extraction is performed with ResNet, a deep neural network architecture [23]. The protocol is quite efficient, with an execution time of 1.3s (on a mobile device) and an EER of 3.04%. Security against malicious clients is achieved by randomizing the computed similarity score, in order to keep the plaintext score hidden from a malicious client after the decryption process.

For sake of completeness, we should also mention that there exist several protocols in the literature that introduce a third-party in the biometric verification process. This entity is typically a cloud server, and the motivation is to improve the protocol's execution time by outsourcing expensive operations (running on client's smartphone) to the cloud. To reduce the impact on security, researchers propose different solutions to secure their system against a malicious cloud server [24–28]. However, we believe that such protocols introduce an additional attack surface for adversaries, and are not essential for today's smartphones that are powerful enough to compute complex cryptographic operations.

Finally, in the malicious setting (both client and server), Barni et al. [7] employ the SPDZ MPC protocol [29] to compute the verification result in a secure manner. The online phase of their protocol is very efficient, however, it involves a very expensive offline phase where a large number of secret values are jointly computed and stored at the two parties. More importantly, the client and the server must invoke this offline phase periodically, i.e., when all the precomputed values have been used by the online verification protocol. On the other hand, Bassit et al. [8] introduce an efficient protocol based on threshold homomorphic encryption. Unfortunately, to defend against malicious servers, the protocol necessitates a trusted third-party that is involved in the enrollment phase. In particular, the role of the

third-party is to digitally sign the users' encrypted templates, so that the server cannot submit malicious templates to an honest client.

3. Preliminaries

3.1. Two-level Homomorphic Cryptosystem

In our work, we leverage the two-level homomorphic encryption scheme of Attrapadung et al. [9] that is based on bilinear groups of prime order. Such cryptosystems allow for the evaluation of a single multiplication operation (and unlimited additions) directly on encrypted data. The cryptosystem is constructed as follows:

(1) Let $(\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T)$ be an asymmetric pairing group, where a bilinear map $e : \mathbb{G}_1 \times \mathbb{G}_2 \rightarrow \mathbb{G}_T$ is defined. Also, let g_1 and g_2 be generators of the groups $\mathbb{G}_1$ and $\mathbb{G}_2$, respectively, and let $z = e(g_1, g_2)$ be a generator of group $\mathbb{G}_T$. All three groups are of prime order q. The bilinear property states that, given $u \in \mathbb{G}_1$, $v \in \mathbb{G}_2$, and $a, b \in \mathbb{Z}_q$,

$$e(u^a, v^b) = e(u, v)^{ab}$$

(2) Assume two messages m_1 and m_2 that are encrypted in the two groups via a lifted-ElGamal scheme:

$$\left[m_1\right]_1 = (g_1^{r_1}, g_1^{m_1 + r_1 s_1}) = (g_1^{r_1}, h_1^{r_1} g_1^{m_1})$$

$$\left[m_2\right]_2 = (g_2^{r_2}, g_2^{m_2 + r_2 s_2}) = (g_2^{r_2}, h_2^{r_2} g_2^{m_2})$$

We use the notation $\left[\cdot\right]_i$ to represent encryption under group $\mathbb{G}_i$. In the equations above, $s_1, s_2 \in \mathbb{Z}_q^*$ are the secret encryption keys in the two groups, respectively, and h_1, h_2 are the corresponding public keys. The encryption randomizers r_1, r_2 are uniformly random in $\mathbb{Z}_q^*$. Note that, the lifted-ElGamal cryptosystem is additively homomorphic (within the same group) via a pairwise multiplication of the underlying ciphertexts.

(3) We can homomorphically multiply m_1 and m_2 in the ciphertext domain, by constructing the following ciphertext $\left[m_1 m_2\right]_T = (c_1, c_2, c_3, c_4)$ in $\mathbb{G}_T$:

$$\left[m_1 m_2\right]_T = \left(e(g_1^{r_1}, g_2^{r_2}), e(g_1^{r_1}, g_2^{m_2 + r_2 s_2}), e(g_1^{m_1 + r_1 s_1}, g_2^{r_2}), e(g_1^{m_1 + r_1 s_1}, g_2^{m_2 + r_2 s_2})\right)$$

Using the bilinear property, this can be written as:

$$\left[m_1 m_2\right]_T = \left(z^{r_1 r_2}, z^{r_1 (m_2 + r_2 s_2)}, z^{(m_1 + r_1 s_1) r_2}, z^{(m_1 + r_1 s_1)(m_2 + r_2 s_2)}\right)$$

Note that, the encrypted ciphertexts in $\mathbb{G}_T$ are additively homomorphic via a pairwise multiplication of the underlying ciphertexts. However, no additional multiplications are possible in $\mathbb{G}_T$.

(4) Decryption in $\mathbb{G}_T$ is performed by computing $z^{m_1 m_2}$ as

$$z^{m_1 m_2} = c_1^{s_1 s_2} c_2^{-s_1} c_3^{-s_2} c_4$$

and solving the discrete log problem to obtain $m_1 m_2$. As such, for efficient decryption, the encrypted plaintexts must be of polynomial size.

3.2. Π*-nets*

To demonstrate the efficiency of our protocol in a real-world setting, we opted to build a proof-of-concept biometric verification system. More specifically, we chose face recognition as the verification factor, due to the overwhelming availability of high-resolution cameras in most laptops, monitors, and mobile devices today. Our approach is to leverage an existing face recognition platform to compute the facial feature vector, and then pass this vector to our cryptographic engine for authenticating the user to a remote server.

To this end, we selected Π-nets [10] as the underlying face recognition system. Π-nets is a recent family of neural networks based on polynomial neural networks where the output is a high-order polynomial of the input. This can be used to perform both generative and discriminative tasks; the Π-nets architecture can be employed in different applications, including image generation, image and audio classification, 3D Mesh representation learning, and face recognition and identification.

Specifically, Π-nets maps face images to a compact Euclidean space of dimensionality 512. It was trained on the publicly-available MS1M-RetinaFace dataset [30, 31] which includes 5.1M images of 93K identities. The architecture demonstrates its competitiveness over existing state-of-the-art face recognition methods, as it achieves an accuracy of 99.833% on the benchmark Labeled Faces in the Wild (LFW) dataset [32] that contains 13,233 web-collected images from 5,749 different identities. It even outperforms the accuracy of models trained on larger private datasets, such as FaceNet [16].

Under Π-nets, the feature vectors consist of 512 floating point values, and the similarity between two vectors is measured by their squared Euclidean distance. Nevertheless, in our system, we have to make several adjustments in the computed feature vectors and the similarity threshold, because most public key homomorphic encryption schemes typically work with integer values (with few exceptions). We will discuss all these issues, and how they affect the accuracy of Π-nets, in Section 6.

4. Secure Biometric Verification Protocol

The protocol consists of two phases, namely, enrollment and verification. We discuss them in detail in the following sections.

4.1. Client Enrollment

During system initialization, the server instantiates an asymmetric pairing group $(\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T)$, as described in Section 3.1. When a new client enrolls into the biometric verification system, the server shares the group parameters $(\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, e, g_1, g_2, q)$ with the client. The client then chooses its secret (private) keys s_1, s_2 uniformly at random in $\mathbb{Z}_q^*$, for groups $\mathbb{G}_1$ and $\mathbb{G}_2$, respectively. It also computes the underlying public keys $h_1 = g_1^{s_1}$ and $h_2 = g_2^{s_2}$. The client's input in this phase is a biometric feature vector $\mathbf{x} = (x_1, x_2, \ldots, x_N)$, which must be encrypted before it is shared with the verification

Protocol 1: Client enrollment

Client		Server
Input: $\mathbf{x}$		
	$\xleftarrow{(\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, e, g_1, g_2, q)}$	
$s_1 \leftarrow\$ \mathbb{Z}_q^*$		
$s_2 \leftarrow\$ \mathbb{Z}_q^*$		
$h_1 = g_1^{s_1}$		
$h_2 = g_2^{s_2}$		
$\mathbf{r} \leftarrow\$ \mathbb{Z}_q^N$		
$\mathbf{x} = \mathbf{x} + \mathbf{r}$		
	$\xrightarrow{ID, h_1, h_2, \left(\left[\hat{x}_i\right]_1, \left[\hat{x}_i\right]_2\right)_{i=1}^N}$	
Delete $\mathbf{x}, \mathbf{x}$		Store data

Fig. 1. Enrollment protocol

server. However, prior to encryption, the client chooses a randomization vector $\mathbf{r} = (r_1, r_2, \ldots, r_N)$ that is uniformly random in $\mathbb{Z}_q^N$. Subsequently, it obfuscates the plaintext feature vector as follows

$$\mathbf{x} = (\mathbf{x} + \mathbf{r}) \bmod q$$

Then, for each element $\hat{x}_i$, $i \in \{1, 2, \ldots, N\}$, in the obfuscated vector, the client computes its lifted-ElGamal ciphertexts in groups $\mathbb{G}_1$ and $\mathbb{G}_2$, denoted as $\left[\hat{x}_i\right]_1$ and $\left[\hat{x}_i\right]_2$, respectively. All the ciphertexts are then sent to the server, along with the client's ID and its public keys. Finally, the client securely deletes $\mathbf{x}$ and $\mathbf{x}$ from its local memory. In other words, after enrollment, the client's memory only stores the two public keys (h_1 and h_2), the client's secret decryption keys (s_1 and s_2), and the randomization vector $\mathbf{r}$. The complete enrollment protocol is shown in Fig. 1

At the server-side, during the system's initialization, the server computes $z = e(g_1, g_2)$ and proceeds to pre-compute all possible encodings z^d, where d is the squared Euclidean distance between any two vectors. This is done in order to optimize the final computation of d (discrete log), via the use of a look-up table.

4.2. Client Verification

During an verification session, the client first generates a fresh feature vector $\mathbf{y}$ and obfuscates it with the randomization vector $\mathbf{r}$ stored on the user's device:

$$\mathbf{y} = (\mathbf{y} + \mathbf{r}) \bmod q$$

Then, it encrypts the values $-\hat{y}_i$, $i \in \{1, 2, \ldots, N\}$, under groups $\mathbb{G}_1$ and $\mathbb{G}_2$ and sends the ciphertexts to the server, along with its *ID*. The server then accesses the user's stored biometric feature vector and proceeds to compute the ciphertexts of $(x_i - y_i)$ under groups $\mathbb{G}_1$ and $\mathbb{G}_2$. For instance, the ciphertext under $\mathbb{G}_1$ is derived as

$$\left[x_i - y_i\right]_1 = \left[\hat{x}_i\right]_1 \odot \left[-\hat{y}_i\right]_1$$

where $\odot$ denotes the pairwise multiplication of the two lifted-ElGamal ciphertexts. Notice that the randomizer r_i is canceled out via the subtraction operation.

Now the server has all the information it needs to compute the squared Euclidean distance, which is defined as

$$d = \sum_{i=1}^{N} (x_i - y_i)^2$$

To this end, the server first computes the ciphertexts (in $\mathbb{G}_T$) of $(x_i - y_i)^2$, using $\left[x_i - y_i\right]_1$ and $\left[x_i - y_i\right]_2$:

$$\left[(x_i - y_i)^2\right]_T = \mathbf{e}\left(\left[x_i - y_i\right]_1, \left[x_i - y_i\right]_2\right)$$

Here, we use a boldface font for the pairing function $\mathbf{e}$ to denote the vector of four distinct pairings, as described in Section 3.1. Recall that ciphertexts in $\mathbb{G}_T$ are still additively homomorphic, so the server eventually computes the encrypted squared Euclidean distance d as follows:

$$\left[d\right]_T = \bigodot_{i=1}^{N} \left[(x_i - y_i)^2\right]_T$$

Let (c_1, c_2, c_3, c_4) denote ciphertext $\left[d\right]_T$, i.e., the four elements in $\mathbb{G}_T$. At this point, the server has to engage the client in the decryption process of d. As discussed in Section 3.1, decryption in $\mathbb{G}_T$ necessitates three exponentiations (one for each of c_1,c_2, and c_3) with three secret values that are stored on the user's device. Therefore, the server sends c_1, c_2, and c_3 to the client, in order for the client to compute the following values:

$$c_1' = c_1^{s_1 s_2}, c_2' = c_2^{-s_1}, c_3' = c_3^{-s_2}$$

Note that it is infeasible for the client to retrieve the plaintext similarity score, because c_4 is only known to the server.

The client then sends back to the server (i) c_1', c_2', and c_3'; and (ii) three *non-interactive* zero knowledge proofs (NIZKPs) that prove to the server that the client knows the three secret exponents. Each proof is essentially an instance of Schnorr's protocol [33] with the Fiat-Shamir heuristic [34]. The server then verifies the NIZKPs and computes the encoding of the squared Euclidean distance as

$$z^d = c_1' c_2' c_3' c_4$$

The final step involves a query at the stored look-up table to retrieve the actual value of d that determines the verification output. In particular, if $d \leqslant \tau$, where τ is the upper bound of the squared Euclidean

distance that signifies a positive match, the client's verification is considered successful. The complete verification protocol is depicted in Fig. 2.

Protocol 2: Client verification

Client		**Server**
Input: $\mathbf{y}$		
$\mathbf{x} = \mathbf{y} + \mathbf{r}$		
	$ID, \left(\left[-\hat{y}_i\right]_1, \left[-\hat{y}_i\right]_2\right)_{i=1}^{N} \longrightarrow$	
		$\left[x_i - y_i\right]_1 = \left[\hat{x}_i\right]_1 \odot \left[-\hat{y}_i\right]_1, 1 \leqslant i \leqslant N$
		$\left[x_i - y_i\right]_2 = \left[\hat{x}_i\right]_2 \odot \left[-\hat{y}_i\right]_2, 1 \leqslant i \leqslant N$
		$\left[(x_i - y_i)^2\right]_T = \mathbf{e}\left(\left[x_i - y_i\right]_1, \left[x_i - y_i\right]_2\right), 1 \leqslant i \leqslant N$
		$\left[d\right]_T = (c_1, c_2, c_3, c_4) = \bigodot_{i=1}^{N} \left[(x_i - y_i)^2\right]_T$
	$\longleftarrow c_1, c_2, c_3$	
$c_1' = c_1^{s_1 s_2}$		
$c_2' = c_2^{-s_1}$		
$c_3' = c_3^{-s_2}$		
Construct NIZKPs		
	$\longrightarrow c_1', c_2', c_3', \text{NIZKPs}$	
		Verify all NIZKPs (abort if failure)
		$z^d = c_1' c_2' c_3' c_4$
		Output d from look-up table
		if $(d \notin$ look-up table$)$ $\{auth = \textbf{invalid}\}$
		else if $(d \leqslant \tau)$ $\{auth = \textbf{yes}\}$ **else** $\{auth = \textbf{no}\}$
	$\longleftarrow auth$	
Output *auth*		

Fig. 2. Verification protocol

Note that, if any of the NIZKP verifications fails, the server instantly aborts the protocol. Similarly, if z^d does not exist in the look-up table, the server labels the verification session as invalid (an indication that the client has cheated somewhere in the previous steps). Indeed, the look-up table is constructed for all possible outcomes of d, i.e., if the bit-length of the feature vector elements is k, the max value of d stored in the look-up table is $d_{max} = N \cdot (2^k - 1)^2$.

For completeness, we describe next the NIZKP protocol that we employ, which has been proven secure in the random oracle model. The protocol assumes that the prover (the client, in our case) has knowledge of a secret s, such that $c' = c^s$. (For example, in the case of c_1 and c'_1, the client's secret is $s_1 s_2$.) The protocol is executed as follows:

(1) The prover selects $t \leftarrow\$ \mathbb{Z}_q^*$ and generates a commitment $a = c^t$.
(2) The prover computes $v = H(a, c, c')$, using a cryptographically secure hash function H, such as SHA-256.
(3) The prover computes $b = (t + v \cdot s) \bmod q$ and sends to the verifier (a, b, v).
(4) The verifier (server) accepts the proof iff $c^b = ac'^v$.

5. Security

In multiparty computation protocols, security is defined by comparing what an adversary can do when the protocol is executed in the *real* world to what the adversary can do when the protocol is executed in an *ideal* model that is secure by definition [35]. Specifically, during a protocol execution in the ideal model, all parties send their inputs to a trusted third party who computes the output. On the other hand, an execution in the real world involves an adversary who sends all messages on behalf of all corrupted parties and may deviate arbitrarily from the protocol specification. Under this definition of the real and ideal models, a protocol is considered secure if an adversary in the ideal model is able to simulate a real world execution of the protocol. We should emphasize that, in two-party protocols, there is no honest majority, so it is impossible to provide fairness or guaranteed output, i.e., the adversary may prevent the honest party from receiving their output.

Recall that our protocol employs a secure NIZKP functionality during its execution. As such, in our security proof, we will consider the *hybrid* model (instead of the real model), in which the two parties interact with each other as usual, but also use a trusted party to compute the NIZKP functionality (i.e., similar to the ideal model). In other words, the real world protocol will run as normal, until an *ideal call* is made to the trusted party to compute the NIZKP functionality. At this point, the two parties send their inputs to the trusted party, which computes and sends back their respective outputs.

Let us now give a formal definition of the two functionalities that protocols π_{NIZKP} and π_{AUTH} (our protocol) compute. First, protocol π_{NIZKP} assumes that the two parties share the description of a certain group $\mathbb{G}_T$ of prime order q and two elements $c, c' \in \mathbb{G}_T$, such that $c' = c^s$. The client's input is the secret value $s \in \mathbb{Z}_q^*$, while the server does not have an input. The server's output is a triplet (a, b, v), such that $c^b = ac'^v$. Element $a \in \mathbb{G}_T$, element $b \in Z_q$, and v is a random string. We use the following notation to describe this functionality $\mathcal{F}_{\mathsf{NIZKP}}$, where λ is the empty string:

$$(s, \lambda) \mapsto (\lambda, (a, b, v))$$

In protocol π_{AUTH}, the client's input is an encrypted feature vector $\mathbf{y}$ (denoted as $[\mathbf{y}]$) and secret keys s_1, s_2, while the server's input is an encrypted feature vector $\mathbf{x}$. (We can ignore the randomizers as they

do not affect the protocol's security proof. Also, the ciphertext notation captures the encryptions under both groups.) The server's output is the squared Euclidean norm of $\mathbf{x}$ and $\mathbf{y}$, while the client's output is the verification result. Formally, functionality $\mathcal{F}_{\mathsf{AUTH}}$ is denoted as follows:

$$(([\mathbf{y}], s_1, s_2), [\mathbf{x}]) \mapsto (auth, \|\mathbf{x}-\mathbf{y}\|_2^2)$$

For simplicity, let us denote as C and S the client's and server's input, respectively. Also, let $\mathcal{A}$ be a non-uniform probabilistic polynomial-time (PPT) adversary, and let n be the security parameter. Then, the ideal execution of $\mathcal{F}_{\mathsf{AUTH}}$, denoted as $\text{IDEAL}_{\mathcal{F}_{\mathsf{AUTH}}}(C,S,n)$, is defined as the output pair of the two parties (the honest party and $\mathcal{A}$) from the execution in the ideal model involving a trusted party. Similarly, the real execution of π_{AUTH} in the hybrid model (where the NIZKP functionality is computed by a trusted party), denoted as $\text{HYBRID}_{\pi_{\mathsf{AUTH}}}(C,S,n)$, is the output pair of the honest party and $\mathcal{A}$ from the real execution of π_{AUTH}.

Definition 1. *Protocol π_{AUTH} securely computes $\mathcal{F}_{\mathsf{AUTH}}$ in the presence of malicious adversaries if, for every non-uniform PPT adversary $\mathcal{A}$ in the hybrid model, there exists a non-uniform PPT adversary $\mathcal{S}$ (the simulator) in the ideal model such that, for the case of either malicious party, it holds that*

$$\text{IDEAL}_{\mathcal{F}_{\mathsf{AUTH}}}(C,S,n) \overset{c}{\equiv} \text{HYBRID}_{\pi_{\mathsf{AUTH}}}(C,S,n)$$

In the above definition, symbol $\overset{c}{\equiv}$ denotes that the two distributions are computationally indistinguishable. What the definition essentially implies, is that the protocol is secure if an adversary $\mathcal{S}$ in the ideal model is able to simulate a protocol execution in the hybrid model.

Theorem 1. *Assume that π_{NIZKP} securely computes the NIZKP functionality in the presence of malicious adversaries. Then, π_{AUTH} securely computes $\mathcal{F}_{\mathsf{AUTH}}$ in the presence of malicious adversaries.*

Proof. We consider two separate cases in our proof, that is, the case where the client is malicious and the case where the server is malicious.

Malicious client. In this case, the adversary $\mathcal{A}$ is controlling the client, and $\mathcal{S}$ has access to $\mathcal{A}$'s input. The simulation of the real protocol is performed as follows, where $\mathcal{S}$ is playing the role of the server interacting with the adversary.

$\mathcal{S}$ sends $\mathcal{A}$'s input to the trusted party and receives back its output *auth*. Because $\mathcal{S}$ does not have the real server's input $[\mathbf{x}]$, it constructs one from $\mathcal{A}$'s input $[\mathbf{y}]$ and the verification output *auth*. Specifically, based on the value of *auth*, $\mathcal{S}$ constructs an encrypted feature vector $[\mathbf{y}+\delta]$ (leveraging the homomorphic properties of the lifted-ElGamal cryptosystem), such that

(1) $\|\delta\|_2^2 \leqslant \tau$, if the verification is successful.
(2) $\tau < \|\delta\|_2^2 < d_{max}$, if the verification is unsuccessful.
(3) $\|\delta\|_2^2 > d_{max}$ if the verification is invalid.

Initially, $\mathcal{S}$ receives $[\mathbf{y}]$ from $\mathcal{A}$ and proceeds to compute $[d]_T$ according to the protocol specification. It then sends c_1, c_2, and c_3 to $\mathcal{A}$, and receives back c_1', c_2', c_3'. Finally, $\mathcal{S}$ decrypts d and sends back to $\mathcal{A}$ the verification result *auth*, concluding the simulation.

Let us now look at the outputs from the real and ideal executions. First, for the adversary $\mathcal{A}$, the output *auth* is identical in both executions, because $\mathcal{S}$ constructed its input according to the verification result from the ideal execution. For the server, the distribution of the output d follows the distribution from real executions, assuming $\mathcal{S}$ knows that distribution from multiple protocol executions and chooses δ accordingly.

Malicious server. In this case, the adversary is controlling the server, and $\mathcal{S}$ has access to $\mathcal{A}$'s input. The simulation of the real protocol is performed as follows, where $\mathcal{S}$ is playing the role of the client interacting with the adversary.

$\mathcal{S}$ sends $\mathcal{A}$'s input to the trusted party and receives back its output d. Note that $\mathcal{S}$ does not have the client's real input $[\mathbf{y}]$, so it has to construct one from $\mathcal{A}$'s input $[\mathbf{x}]$ and output d. Similar to the case of the corrupted client, $\mathcal{S}$ constructs an encrypted feature vector $[\mathbf{x}+\delta]$, such that $\|\delta\|_2^2 = d$. (If $d > d_{max}$, i.e., z^d does not exist in the look-up table, $\mathcal{S}$ selects a large random value instead.) $\mathcal{S}$ then sends $[\mathbf{x}+\delta]$ to the adversary $\mathcal{A}$. $\mathcal{A}$ computes $[d]_T$ and sends to the simulator the first three elliptic curve points, i.e., c_1, c_2, and c_3. It is important to note that $\mathcal{S}$ also has knowledge of c_4, because it has access to $\mathcal{A}$'s input vector.

Next, $\mathcal{S}$ selects two random elements in $\mathbb{G}_T$, namely c_1' and c_2', and computes another element c_3', such that

$$c_1' c_2' c_3' c_4 = z^d$$

i.e., $c_3' = z^d (c_1')^{-1} (c_2')^{-1} (c_4)^{-1}$. $\mathcal{S}$ sends all three elements back to $\mathcal{A}$. Finally, for each element c_i', $i \in \{1,2,3\}$, $\mathcal{S}$ does the following:

(1) Chooses two random elements $v_i, b_i \in \mathbb{Z}_q$.
(2) Sets $a_i = c_i^{b_i} (c_i')^{-v_i}$.
(3) Enters (a_i, v_i) in the random oracle's *History*.
(4) Sends (a_i, b_i, v_i) to $\mathcal{A}$.

Finally, $\mathcal{A}$ verifies the NIZKP's (with the help of the random oracle), decrypts d, and sends the verification result to $\mathcal{S}$. Note that, each NIZKP is verified successfully, because of the way that a_i is computed in Step 2 above.

Clearly, $\mathcal{A}$'s output is identical in the ideal and hybrid executions (because of how vector δ is selected), which implies that the client's output is also identical. Therefore, we have shown that, under any malicious party, it holds that

$$\text{IDEAL}_{\mathcal{F}_{\mathsf{AUTH}}}(C,S,n) \stackrel{c}{\equiv} \text{HYBRID}_{\pi_{\mathsf{AUTH}}}(C,S,n)$$

which concludes our proof. □

5.1. Discussion

We now discuss the types of attacks that a malicious party can launch in real-life, and how our protocol is able to defend against them.

Malicious client. First, the adversary may launch an attack without having access to the client's device, i.e., without knowledge of the randomization vector $\mathbf{r}$ and the two secret keys s_1 and s_2. In this case, the adversary chooses these values randomly, and will fail the verification process (via an invalid outcome) with an overwhelming probability. Specifically, given that each value is an element of $\mathbb{Z}_q$, the probability that the adversary succeeds in guessing all of them correctly is $2^{-q(N+2)}$.

On the other hand, an adversary who has compromised the user's device (and has access to all its secret values) may launch two types of attacks. The simplest one is a brute-force attack on the feature vector, i.e., generating and trying different input vectors $\mathbf{y}$, until it succeeds. In this case, the probability of success (for each attempt) is equal to the false positive rate of the underlying biometric recognition protocol. Nevertheless, such attacks are mitigated by (i) limiting the number of successive unsuccessful verification attempts by a client; and (ii) utilizing more accurate biometric recognition protocols, such as face recognition with depth cameras.

The adversary may also attempt to manipulate one of the partially decrypted ciphertexts c'_1, c'_2, or c'_3, in order to lower the value of the underlying squared Euclidean norm. More specifically, this attack is performed by having the adversary replace one of the client's secret keys with a value w, such that the computed squared norm at the server becomes lower than the threshold τ. Assume, for example, that $c_1 = z^r$, where r is a random value unknown to the adversary (because of the randomizers used in the encryptions of the stored feature vector $\mathbf{x}$). The adversary's objective is to compute a value $w \in \mathbb{Z}_q$, such that

$$c'_1 = c_1^w = c_1^{s_1 s_2} z^{-\theta}$$

which essentially decreases the computed squared norm at the server by a value of θ. Substituting c_1 with z^r and solving for w, we get

$$z^{rw} = z^{r s_1 s_2 - \theta} \Rightarrow w = s_1 s_2 - \theta r^{-1}$$

Given that r^{-1} is a random element in $\mathbb{Z}_q$, the adversary will succeed with probability 2^{-q} in guessing a valid candidate key w.

Malicious server. A malicious server will attempt to decrypt the client's stored (or submitted) feature vector and retrieve the plaintext biometric data. (For example, by decrypting one vector element during each client verification session.) Notice, however, that the adversary in our protocol can only decrypt ciphertexts in $\mathbb{G}_T$ so, to recover an element x_i, the adversary has to produce $[x_i]_T$. But this is infeasible to do in our case, due to the presence of the randomizers in the encryptions of vectors $\mathbf{x}$ and $\mathbf{y}$. Indeed, the best an adversary can do is compute $[x_i + r_i]_T$, which is infeasible to decrypt under a discrete log based cryptosystem.

6. System Implementation

We implemented our system on a client/server architecture with two separate processes, one emulating the verification server and the other emulating the client device. We ran the experiments on a single Ubuntu laptop with Intel Core i7-6500U CPU 2.50GHz×4 and 16 GB of RAM (it is also equipped with a camera) and an SSD 860 EVO 1TB.

The face recognition operation employs the implementation of Π-nets[1]. The original implementation is built on Python version 3 and requires the `mxnet` and `opencv` libraries. The pre-trained model is also provided by the authors. On our hardware configuration, the face recognition and normalization process takes approximately 600ms for a 640×480 image resolution. We implemented the cryptographic layer in C, using the `mcl` library[2] which is a portable and fast pairing-based cryptographic library. Specifically, our results are obtained with `mcl`'s Barreto-Naehrig curve type BN254. We also used SWIG to connect C with Python (version 4.0.1). Each result is computed by running the experiment four times and reporting the average time. Finally, our implementation leverages the parallel computing abilities of the multi-core machine, especially at the server-side.

Before applying any cryptographic operations on the feature vectors generated by Π-nets, we needed to convert the floating point representations of elements in vectors $\mathbf{x}$ and $\mathbf{y}$ into integers. Specifically, for any element x_i in a feature vector, we employed the following transformation, based on an empirical evaluation: $\lfloor x_i \times 600 + 128 \rfloor$, where $x_i \in \mathbb{Q} : -0.213 < x_i < 0.213$. This operation maps Π-nets's floating point values into integers in the range $[0, 256)$. The aforementioned transformation invokes a negligible loss in recognition accuracy, as quantified in Table 1.

The table illustrates the accuracy of the modified Π-nets system (using normalized feature vectors) when compared to the original Π-nets implementation and other state-of-the-art face recognition models. The comparison was done on the Labeled Faces in the Wild (LFW) benchmark [36], which is one of the largest publicly-available datasets on the web. The best threshold value for the Π-nets system is 1.359, which translates into $\tau = 486000$ after being normalized and squared to fit our protocol specifications. In other words, any (squared) Euclidean distance less than the threshold value τ indicates a positive match between two feature vectors. At this specific threshold value, the false positive rate (FPR) of the system is only 0.0668%. FPR represents the percentage of false positives against positive predictions and is calculated as $FPR = \frac{FP}{FP+TN}$, where FP is the number of false positives and TN is the number of true negatives.

Table 1
Accuracy Results on the LFW Benchmark [37]

Model	**Accuracy**
Human-Individual	97.27%
Human-Fusion	99.85%
Center Loss [38]	98.75%
SphereFace [17]	99.27%
VGGFace2 [39]	99.43%
ArcFace [37]	99.82%
Π-nets	$99.833\% \pm 0.211$
Π-nets, normalized	**$99.83\% \pm 0.194$**

7. Experimental Results

In this section, we experimentally evaluate the overhead of our secure biometric verification system in terms of computation and communication/storage costs. First, Table 2 depicts the CPU time required

[1] https://github.com/grigorisg9gr/polynomial_nets
[2] https://github.com/herumi/mcl

at both the server and the client, for the two protocol phases: enrollment and verification. During enrollment, the client begins by extracting the feature vector from the image. This is the most time consuming operation, but it is not related to our cryptographic protocol.

(The cost of the operation is between 0.6s and 1.3s, depending on the quality of the image/frame.) The client then obfuscates and encrypts the feature vector, a process that incurs a cost of 298ms. The cost at the server does not involve any cryptographic operations related to our protocol, so we do not report any results. Indeed, the server's only task in the enrollment phase is to receive and store the encrypted feature vector and the client's information.

During the verification phase, the CPU cost at the client consists of (i) extracting the feature vector $\mathbf{y}$ from the image; (ii) obfuscating and encrypting the feature vector; and (iii) partially decrypting the server's similarity score and constructing the necessary NIZKPs. The CPU time for the cryptographic steps (ii) and (iii) is approximately 360ms. At the server, the CPU time is dominated by the computation of the encrypted similarity score $\left[d\right]_T$ (around 356ms and involving numerous pairing operations), while the final decryption step (including the verification of the client's NIZKPs) is very fast.

Table 2
Computation Time for Cryptographic Operations

Phase	**Server time (ms)**	**Client time (ms)**
Enrollment	—	298
Verification	520	360

In Table 3, we compare the performance of our system against other secure biometric verification techniques that are resilient against malicious adversaries. Note that the SEMBA [7] protocol uses a fusion approach that combines two modalities: iris and face. While the EERs of the modalities are 2.08% and 17.37%, respectively, the resulting fusion EER is 1.15%. The similarity metrics used by this technique are the weighted Hamming distance (HD) and the squared Euclidean distance (ED), respectively. The second approach listed in our table is HELR [8] which, similar to our protocol, employs only face recognition. As we can observe from this table, our method outperforms HELR in terms of computation time. SEMBA is significantly faster but, as we emphasized earlier, it necessitates a very expensive offline phase that must be invoked periodically. Furthermore, our system enjoys a better EER, due to the usage of the state-of-the-art Π-nets face verification protocol.

Table 3
Computation Time Comparison With Other SOTA Methods

Proposed method	**Biometric modal**	**Dataset**	**Number of Features**	**EER (%)**	**Classification technique**	**Cryptographic technique**	**Computation time (ms)**
SEMBA [7]	Iris + face	IIT Delhi Iris[40] + ORL[41]	(6400,2)	1.15	HD + ED	SPDZ [42]	120
HELR [8]	Face	PUT [43], FRGC [44]	49, 94	0.27, 0.25	LLR	ELGamal[45] & ZK-prooofs	2500, 1220
Our work	Face	LFW[32]	512	0.17	Π-nets (ED)	Attrapadung et al.[9] & NIZKPs	880

Regarding the communication cost (shown in Table 4), the enrollment phase consists of the client sending its ID, public keys, and encrypted feature vector to the server, which incurs a cost of approximately 96KB of data. On the other hand, the verification phase involves 3 communication rounds. First, the client sends its ID and encrypted feature vector to the server (96KB). Then, the server sends back to the client 3 ciphertexts (1.12KB) that are part of the encrypted similarity score. In the last step, the

Table 4
Communication Cost

Phase	Cost (KB)
Enrollment	96.11
Verification	99.58

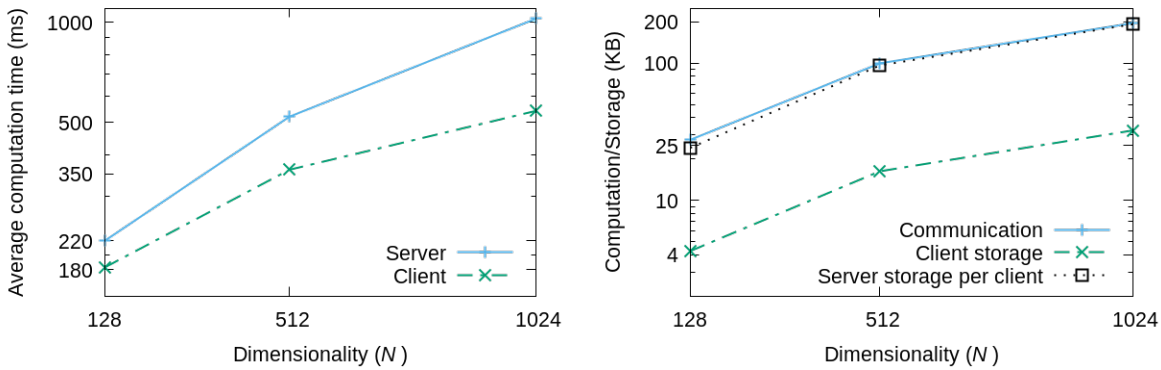
client sends to the server the modified ciphertexts and the corresponding NIZKPs (2.43KB). As such, the overall communication cost is approximately 99KB.

In terms of storage requirements, both parties need to store some information that is necessary for a successful verification session. As depicted in Table 5, the client needs around 4KB of storage space, which includes its ID, its public and secret (private) keys, the group descriptions, and the randomization vector $\mathbf{r}$. Similarly, the server needs to store the enrollment data of each client (96KB), which includes the client's ID, its public keys, and its encrypted feature vector $\hat{\mathbf{x}}$. Additionally, the server maintains a very large look-up table that is essential for efficient ciphertext decryption in $\mathbb{G}_T$. The size of that table in our specific implementation is 3.19GB.

Table 5
Storage Requirements

Party	Cost
Client	16.26 KB
Server (per client)	96.11 KB
Server (look-up table)	3.19 GB

It is worth mentioning that our proposed approach is independent of the underlying face recognition algorithm, as long as the similarity score is measured with the squared Euclidean distance. However, the computation times are affected by the dimensionality of the feature vectors that are generated by the chosen face recognition technique (which typically ranges between 128 and 1024). To this end, Fig. 3 depicts the protocol's overhead, as a function of the feature vector dimensionality. Clearly, our protocol is very efficient, incurring a very low overhead even at large dimensionalities.



(a) Computation time (b) Communication/Storage

Fig. 3. Verification cost vs. feature vector dimensionality

8. Conclusions

We have proposed a fast and efficient biometric verification protocol that is secure in the malicious setting, i.e., when either the server or the client are potentially malicious. The protocol employs an elaborate two-level homomorphic encryption scheme that allows us to compute the squared Euclidean norm between two encrypted vectors in a secure manner. We outlined a formal security proof of our protocol in the random oracle model, and implemented a proof-of-concept system for a secure face verification protocol. Our results show that the protocol incurs a low computational cost at the client and server, while maintaining a communication cost of just 99KB. In our future work, we plan to extend our implementation to mobile devices, and also improve its security by employing more accurate face recognition models that leverage depth cameras.

References

[1] N. Provos and D. Mazieres, Bcrypt algorithm, in: *USENIX*, 1999.

[2] Z. Lei, Y. Nan, Y. Fratantonio and A. Bianchi, On the Insecurity of SMS One-Time Password Messages against Local Attackers in Modern Mobile Devices, in: *Proc. Annual Network and Distributed System Security Symposium (NDSS)*, 2021.

[3] J.-H. Im, J. Choi, D. Nyang and M.-K. Lee, Privacy-preserving palm print authentication using homomorphic encryption, in: *Proc. International Conference on Big Data Intelligence and Computing (DataCom)*, 2016, pp. 878–881.

[4] V.N. Boddeti, Secure face matching using fully homomorphic encryption, in: *Proc. IEEE International Conference on Biometrics Theory, Applications and Systems (BTAS)*, 2018, pp. 1–10.

[5] S.F. Shahandashti, R. Safavi-Naini and N.A. Safa, Reconciling user privacy and implicit authentication for mobile devices, *Computers & Security* **53** (2015), 215–233.

[6] J. Šeděnka, S. Govindarajan, P. Gasti and K.S. Balagani, Secure outsourced biometric authentication with performance evaluation on smartphones, *IEEE Transactions on Information Forensics and Security* **10**(2) (2014), 384–396.

[7] M. Barni, G. Droandi, R. Lazzeretti and T. Pignata, SEMBA: secure multi-biometric authentication, *IET Biometrics* **8**(6) (2019), 411–421.

[8] A. Bassit, F. Hahn, J. Peeters, T. Kevenaar, R. Veldhuis and A. Peter, Fast and accurate likelihood ratio-based biometric verification secure against malicious adversaries, *IEEE transactions on information forensics and security* **16** (2021), 5045–5060.

[9] N. Attrapadung, G. Hanaoka, S. Mitsunari, Y. Sakai, K. Shimizu and T. Teruya, Efficient Two-level Homomorphic Encryption in Prime-order Bilinear Groups and a Fast Implementation in WebAssembly, in: *Proc. Asia Conference on Computer and Communications Security (AsiaCCS)*, 2018, pp. 685–697.

[10] G.G. Chrysos, S. Moschoglou, G. Bouritsas, Y. Panagakis, J. Deng and S. Zafeiriou, P-nets: Deep polynomial neural networks, in: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2020, pp. 7325–7335.

[11] A.C.-C. Yao, How to Generate and Exchange Secrets, in: *Proc. IEEE Annual Symposium on Foundations of Computer Science (FOCS)*, 1986, pp. 162–167.

[12] P. Paillier, Public-key cryptosystems based on composite degree residuosity classes, in: *Proc. International Conference on the Theory and Applications of Cryptographic Techniques (EUROCRYPT)*, 1999, pp. 223–238.

[13] E. Bingham and H. Mannila, Random projection in dimensionality reduction: applications to image and text data, in: *Proc. ACM International Conference on Knowledge Discovery and Data Mining (SIGKDD)*, 2001, pp. 245–250.

[14] C. Gentry, Fully homomorphic encryption using ideal lattices, in: *Proc. ACM Symposium on Theory of Computing (STOC)*, 2009, pp. 169–178.

[15] J. Fan and F. Vercauteren, Somewhat practical fully homomorphic encryption, *IACR Cryptology ePrint Archive* **2012** (2012), 144.

[16] F. Schroff, D. Kalenichenko and J. Philbin, FaceNet: A unified embedding for face recognition and clustering, in: *Proc. IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2015, pp. 815–823.

[17] W. Liu, Y. Wen, Z. Yu, M. Li, B. Raj and L. Song, SphereFace: Deep Hypersphere Embedding for Face Recognition, in: *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017, pp. 6738–6746.

[18] H. Gunasinghe and E. Bertino, PrivBioMTAuth: privacy preserving biometrics-based and user centric protocol for user authentication from mobile phones, *IEEE Transactions on Information Forensics and Security* **13**(4) (2017), 1042–1057.

[19] J.H. Cheon, H. Chung, M. Kim and K.-W. Lee, Ghostshell: Secure Biometric Authentication using Integrity-based Homomorphic Evaluations, *IACR Cryptology ePrint Archive* **2016** (2016), 484.
[20] N.P. Smart and F. Vercauteren, Fully Homomorphic SIMD Operations, *IACR Cryptology ePrint Archive* (2011), 133.
[21] J.-H. Im, S.-Y. Jeon and M.-K. Lee, Practical Privacy-Preserving Face Authentication for Smartphones Secure Against Malicious Clients, *IEEE Transactions on Information Forensics and Security* **15** (2020), 2386–2401.
[22] D. Catalano and D. Fiore, Boosting Linearly-Homomorphic Encryption to Evaluate Degree-2 Functions on Encrypted Data, *IACR Cryptology ePrint Archive* (2014), 813.
[23] K. He, X. Zhang, S. Ren and J. Sun, Deep residual learning for image recognition, in: *Proc. IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016, pp. 770–778.
[24] D. Lin, N. Hilbert, C. Storer, W. Jiang and J. Fan, UFace: Your universal password that no one can see, *Computers & Security* **77** (2018), 627–641.
[25] P. Gasti, J. Šeděnka, Q. Yang, G. Zhou and K.S. Balagani, Secure, fast, and energy-efficient outsourced authentication for smartphones, *IEEE Transactions on Information Forensics and Security* **11**(11) (2016), 2556–2571.
[26] A. Abidin, On privacy-preserving biometric authentication, in: *Proc. International Conference on Information Security and Cryptology*, Springer, 2016, pp. 169–186.
[27] M. Salem, S. Taheri and J.-S. Yuan, Utilizing transfer learning and homomorphic encryption in a privacy preserving and secure biometric recognition system, *Computers* **8**(1) (2019), 3.
[28] H. Higo, T. Isshiki, K. Mori and S. Obana, Privacy-preserving fingerprint authentication resistant to hill-climbing attacks, in: *Proc. International Conference on Selected Areas in Cryptography*, 2015, pp. 44–64.
[29] I. Damgård, V. Pastro, N. Smart and S. Zakarias, Multiparty Computation from Somewhat Homomorphic Encryption, in: *Proc. Advances in Cryptology (CRYPTO)*, R. Safavi-Naini and R. Canetti, eds, 2012, pp. 643–662.
[30] Y. Guo, L. Zhang, Y. Hu, X. He and J. Gao, Ms-celeb-1m: A dataset and benchmark for large-scale face recognition, in: *Computer Vision–ECCV 2016: 14th European Conference, Amsterdam, The Netherlands, October 11-14, 2016, Proceedings, Part III 14*, Springer, 2016, pp. 87–102.
[31] J. Deng, J. Guo, D. Zhang, Y. Deng, X. Lu and S. Shi, Lightweight face recognition challenge, in: *Proceedings of the IEEE/CVF International Conference on Computer Vision Workshops*, 2019, pp. 0–0.
[32] G.B. Huang, M. Ramesh, T. Berg and E. Learned-Miller, Labeled Faces in the Wild: A Database for Studying Face Recognition in Unconstrained Environments, Technical Report, 07-49, University of Massachusetts, Amherst, 2007.
[33] C.P. Schnorr, Efficient identification and signatures for smart cards, *Journal of Cryptology* (1991), 161–174.
[34] A. Fiat and A. Shamir, How to Prove Yourself: Practical Solutions to Identification and Signature Problems, in: *Proc. Advances in Cryptology (CRYPTO)*, 1986, pp. 186–194.
[35] Y. Lindell, How to Simulate It - A Tutorial on the Simulation Proof Technique, in: *Tutorials on the Foundations of Cryptography*, Springer International Publishing, 2017, pp. 277–346.
[36] G.B. Huang and E. Learned-Miller, Labeled Faces in the Wild: Updates and New Reporting Procedures, Technical Report, UM-CS-2014-003, University of Massachusetts, Amherst, 2014.
[37] J. Deng, J. Guo, N. Xue and S. Zafeiriou, Arcface: Additive angular margin loss for deep face recognition, in: *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, 2019, pp. 4690–4699.
[38] Y. Wen, K. Zhang, Z. Li and Y. Qiao, A discriminative feature learning approach for deep face recognition, in: *Computer Vision–ECCV 2016: 14th European Conference, Amsterdam, The Netherlands, October 11–14, 2016, Proceedings, Part VII 14*, Springer, 2016, pp. 499–515.
[39] Q. Cao, L. Shen, W. Xie, O.M. Parkhi and A. Zisserman, Vggface2: A dataset for recognising faces across pose and age, in: *2018 13th IEEE international conference on automatic face & gesture recognition (FG 2018)*, IEEE, 2018, pp. 67–74.
[40] A. Kumar and A. Passi, Comparison and combination of iris matchers for reliable personal authentication, *Pattern recognition* **43**(3) (2010), 1016–1026.
[41] F.S. Samaria and A.C. Harter, Parameterisation of a stochastic model for human face identification, in: *Proceedings of 1994 IEEE workshop on applications of computer vision*, IEEE, 1994, pp. 138–142.
[42] I. Damgård, M. Keller, E. Larraia, V. Pastro, P. Scholl and N.P. Smart, Practical covertly secure MPC for dishonest majority–or: breaking the SPDZ limits, in: *Computer Security–ESORICS 2013: 18th European Symposium on Research in Computer Security, Egham, UK, September 9-13, 2013. Proceedings 18*, Springer, 2013, pp. 1–18.
[43] A. Kasinski, A. Florek and A. Schmidt, The PUT face database, *Image Processing and Communications* **13**(3–4) (2008), 59–64.
[44] P.J. Phillips, P.J. Flynn, T. Scruggs, K.W. Bowyer, J. Chang, K. Hoffman, J. Marques, J. Min and W. Worek, Overview of the face recognition grand challenge, in: *2005 IEEE computer society conference on computer vision and pattern recognition (CVPR'05)*, Vol. 1, IEEE, 2005, pp. 947–954.
[45] R. Cramer, R. Gennaro and B. Schoenmakers, A secure and optimally efficient multi-authority election scheme, *European transactions on Telecommunications* **8**(5) (1997), 481–490.