Ghostshell: Secure Biometric Authentication using Integrity-based Homomorphic Evaluations

Jung Hee Cheon[1], HeeWon Chung[1], Myungsun Kim[2], and Kang-Won Lee[3]

[1] Department of Mathematical Sciences, Seoul National University
{jhcheon,runghyun}@snu.ac.kr
[2] Department of Information Security, The University of Suwon
msunkim@suwon.ac.kr
[3] Network IT Convergence, Corporate R&D Center, SK telecom
kangwon@sk.com

Abstract. Biometric authentication methods are gaining popularity due to their convenience. For an authentication without relying on trusted hardwares, biometrics or their hashed values should be stored in the server. Storing biometrics in the clear or in an encrypted form, however, raises a grave concern about biometric theft through hacking or man-in-the middle attack. Unlike ID and password, once lost biometrics cannot practically be replaced. Encryption can be a tool for protecting them from theft, but encrypted biometrics should be recovered for comparison.
In this work, we propose a secure biometric authentication scheme, named Ghostshell, in which an encrypted template is stored in the server and then compared with an encrypted attempt *without* decryption. The decryption key is stored only in a user's device and so biometrics can be kept secret even against a compromised server. Our solution relies on a somewhat homomorphic encryption (SHE) and a message authentication code (MAC). Because known techniques for SHE is computationally expensive, we develop a more practical scheme by devising a significantly efficient matching function exploiting SIMD operations and a one-time MAC chosen for efficient homomorphic evaluations (of multiplication depth 2). When applied to Hamming distance matching on 2400-bit irises, our implementation shows that the computation time is approximately 0.47 and 0.1 seconds for the server and the user, respectively.

Keywords: Biometric authentication, Homomorphic encryption, MAC

Table of Contents

1 Introduction

Biometric authentications are seeing greater industrial deployment, including mobile payment systems such as Apple Pay and Alipay. Compared to other types of authentication (e.g., passwords and secure tokens), biometrics cannot be lost or forgotten and, in particular, users being authenticated should be present at the time and place of authentication. On the other hand, privacy loss in biometric authentication systems is substantially more serious than in other authentication systems because biometrics are difficult to be replaced once stolen. Most recently, hackers have stolen a total of 5.6 million fingerprint records from the U.S. government [2]. The stolen biometric databases could be used to fool certain systems. Thus, it is imperative to develop a solution with a far stronger protection of such data.

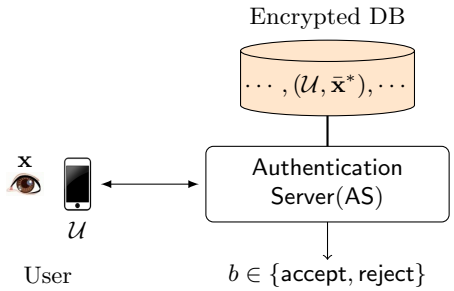

Fig. 1. Our authentication framework.

Cryptographic protection. We investigate a secure biometric authentication method without relying on trusted hardware. Our approach is to encrypt and store a users biometric template in a server as in Figure 1. During authentication, the user sends an encrypted biometric attempt to the server, which authenticates by comparing two ciphertexts without decryption. Comparing ciphertexts without decryption is the main security property that our proposed scheme provides.

One may consider a trivial approach to store only hashed templates in the server through a one-way hash function such as SHA3 [31]. However, biometric inputs are not exactly the same every time they are captured due to scanning noise and so cannot have the same hashed values. Indeed, we have no hash function to map two slightly different inputs to the same value.

Our solution. We employ a somewhat homomorphic encryption (SHE) that evaluates a polynomial of certain degree on encrypted data without decryption [18, 35]. Upon an encrypted template and an encrypted attempt, the server performs a matching algorithm without decryption, which outputs an encrypted matching result. Hence, the server comes to learn the matching result by asking its decryption to the user.

It is crucial to prevent an arbitrary manipulation of a result during the decryption process of a user corrupted by an attacker. To achieve this, we attach a tag to the encrypted result, which is decrypted to the message authentication code (MAC) of the result. It is possible by virtue of the homomorphic properties of SHE, but which is hard for ordinary encryption functions.

This approach provides a sort of decryption oracle, which clearly could result in a security weakness, allowing a malicious server to recover some biometrics. Note that SHE is secure only against chosen plaintext attacks (CPAs), not against chosen ciphertext attacks (CCAs). In order to prevent such an undesirable situation of obtaining decryption oracle, we further introduce a method to enclose the tag using a one-way function with discrete logarithm settings so that the sever can learn only the enclosed value of the tag. It prevents a misuse of the decryption key through receiving a decryption query of its ciphertext.

Optimization. Computing Hamming distance between two biometrics of n bits in length requires at least n multiplications. Considering slow multiplication time of two ciphertexts, it easily exceeds a tolerance range of practical systems, e.g., it takes at least 30 seconds for a comparison of two irises of 2400 bits when the multiplication takes 14 milliseconds per bit as in our implementation.

We utilize the packing techniques proposed in [38] to implement a homomorphic matching algorithm performing N multiplications of ciphertexts at a time where an N-bit plaintext is packed into one ciphertext. We mean by "homomorphic" that the matching algorithm runs on ciphertexts, and a multiplication of two ciphertexts indicates an encryption of an N-bit output which is a bitwise multiplication of two plaintexts. Further, we use single-instruction multiple-data (SIMD) techniques on SHE ciphertexts [38]. This allows us to compute the sum of the N-bit output only using $\log N$ additions. Recall that addition are far efficient than multiplication. To sum up, these techniques help the homomorphic matching algorithm to run approximately N times faster than a naïve one. To this end, we find an SHE system parameter with which an SHE scheme instantiated encrypts $N = 630$ bits into a ciphertext at a time.

When we combine our matching algorithm with MAC, however, we meet with another obstacle from the performance's point of view. An ordinary MAC function has a large multiplication degree, and so is considerably time-consuming to evaluate the MAC function homomorphically. However carefully looking into our authentication process, the MAC tag is verified immediately after generated and the server is the generator and the verifier of the MAC tag. This avoids the necessity of the server sharing a long-term MAC secret key. Thus, we observed that it is enough to use a lightweight one-time MAC scheme. We employ a variant of one-time MAC (OTM) proposed by Simmons [37], requiring only one multiplication and one addition. In turn, we again modify our homomorphic matching algorithm so as to accommodate the OTM scheme.

Despite the speedups of computation, the large size of SHE ciphertexts still causes long delays in transmission. This problem is overcome by applying a ciphertext-compression technique by Coron et al. [15]. Thus, we reduce the size of SHE ciphertext by half. By integrating those three techniques, we obtain a practical biometric authentication scheme, named *Ghostshell*.

When applying our optimizations over Brakerski et al.'s scheme [7], our implementation shows that encrypting a 2400-bit iris code only requires 16.2 milliseconds. The ciphertext size of our SHE scheme for a 2400-bit plaintext iris is approximately 40.1 KB while the bit-by-bit encryption requires 98.3 MB for the same iris. The server's matching process runs in approximately 0.46 second. Majority of the time is spent in computing the Hamming distance (0.45 seconds).

Organization. Our system architecture and participants are described in Section 2. We first attempt to construct a simple, but insecure protocol in Section 3. In Section 4, then we renovate our basic protocol to provide privacy and integrity of biometrics. Our authentication system is provided in Section 5. Section 6 shows our optimizations for practical deployment to the real-world applications. Section 7 provides our implementation and experimental results. Finally, we discuss closely related works in Section 8.

2 Models and Settings

In this section, we present the system architecture for this work. Throughout this paper, our proposal works in this framework.

2.1 System Model and Participants

Our system is designed for an authentication server, or server for short, to authenticate each user using the $1:1$ method. Our system consists of two participants, namely, a user and an authentication server (AS).

The user, denoted by $\mathcal{U}$, has a binary feature $\mathbf{x}$ properly extracted from his biometric source. The server, denoted by $\mathcal{S}$, has rich computational resources and storage; thus, it can evaluate arbitrary functions on SHE ciphertexts but cannot decrypt ciphertexts evaluated by itself as well as ciphertexts given by the user. To enrich the applicability of our system, we may introduce a third entity, called a service provider (SP), that communicates with the two other participants. For example, telecoms could use our authentication platform for bridging between mobile users and service providers (e.g., Amazon and Bank of America).

2.2 Threat Model

Our security goals are two-fold. The first is that no AS (or SP) should be able to learn anything about the biometric data contributed by users beyond what is revealed by the final result of the execution. In the case where the AS and some users collude, they should not learn anything about the biometrics from other honest users beyond the final result and its implications. The second goal is that an impostor should not be able to fool the AS into believing that he is authentic; see reference [34] for the detailed threats.

In this work, we focus on the following attacks to ensure security for authentication and privacy for the user's biometrics.

- Threat #1. Participants on the user side of our system are well motivated to be malicious and thus make an honest AS output an accept at the conclusion of the user authentication. Further, they may collude with arbitrary other users to attempt to break the security of the system.
- Threat #2. Some participants on the server side in our system are also motivated to behave maliciously and thus learn information about private biometrics from honest users because the data can potentially be sold to an attacker, e.g., passport forgers. A typical example is a biometric database administrator corrupted by an attacker.

However, our system does not consider additional tools to defend against the following attack: we assume that an AS does not collude with an SP. In other words, for an application where the SP need to use an authentication result from the AS, we assume that there exists an authenticated channel between the two participants.

2.3 Security Model

Although a corresponding formal security definition will be given in Section 4, we give an outline of the security and privacy properties exhibited by our authentication system.

Correctness: When all participants execute a given protocol faithfully, the verification result at the end of an executed protocol is equal to the result of applying each honest participant to a biometric database in the clear.

Privacy: The servers can only learn the verification result and its implications at the conclusion of a protocol.

Security: A dishonest user cannot persuade an honest AS into accepting himself as an authentic user.

3 A Strawman Proposal

We assume that a biometric is represented by a binary sequence and that a matching algorithm uses the Hamming distance (HD) between two input biometrics. Because the HD of two bit sequences counts the number of different coordinates, we encrypt each bit separately, and thus, the number of ciphertexts is equal to the length of the bit sequence. Herein, we use bold lowercase letters (e.g., $\mathbf{x}$) to denote a biometric, where $x_i \in \{0,1\}$ means the i-th component of $\mathbf{x}$. In addition, a bar over an integer means that the integer is encrypted by an SHE encryption function HEnc (i.e., $\bar{x} = \mathsf{HEnc}(x)$ for $x \in \mathbb{Z}$).

Because any boolean circuit can be constructed via sequences of additions and multiplications, we can consider an HD circuit constructed from Eq. (1) on two encrypted biometrics. Indeed, we compute an HD value by evaluating the HD circuit. Specifically, for $\bar{\mathbf{x}} = (\bar{x}_0, \bar{x}_1, \ldots, \bar{x}_{n-1})$ and $\bar{\mathbf{x}}^* = (\bar{x}_0^*, \bar{x}_1^*, \ldots, \bar{x}_{n-1}^*)$,

$$\mathsf{HD}(\bar{\mathbf{x}}, \bar{\mathbf{x}}^*) = \sum_{i=0}^{n-1} (\bar{x}_i - \bar{x}_i^*)^2 \tag{1}$$

As readers may see, it is clear that we do not simply take $\mathbb{Z}_2$ as the plaintext space of our baseline SHE scheme because an HD value is a sum of subtraction and multiplication. Therefore, we need to consider $\mathbb{Z}_M$ as the plaintext space for an integer $M \gg 2$.

3.1 A Simple, but Insecure Design

For simplicity, we assume that there are two participants, a user $\mathcal{U}$ with his biometric template $\mathbf{x}^*$ and a server $\mathcal{S}$. The point of departure of our construction is to protect $\mathbf{x}^*$ using an SHE scheme $\mathsf{SHE} = (\mathsf{HKg}, \mathsf{HEnc}, \mathsf{HDec})$; see Section 4.1 (and 6) for the details of our SHE scheme.

Let the user generate the key pair (pk, sk) by running the key generation function HKg, say, *Setup*. Then, the user sends the encrypted biometric template to the server, $\bar{\mathbf{x}}^* = \mathsf{HEnc}(pk, \mathbf{x}^*)$, say, *Enrollment*. We can depict the enrollment as shown in Figure 2.

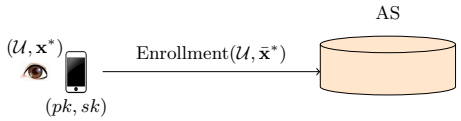

Fig. 2. Enrollment

If the user wants to be authenticated using his present biometric $\mathbf{x}$, the user simply sends $(\mathcal{U}, \bar{\mathbf{x}})$ to the server, where $\bar{\mathbf{x}} = \mathsf{HEnc}(pk, \mathbf{x})$. Then, the server retrieves $\bar{\mathbf{x}}^*$ indexed by $\mathcal{U}$ and runs the HD circuit, denoted by HD, on the two ciphertexts $\bar{\mathbf{x}}^*$ and $\bar{\mathbf{x}}$. We write this as $\bar{d} = \mathsf{HD}(\bar{\mathbf{x}}^*, \bar{\mathbf{x}})$, where $\bar{d}$ denotes a matching result. The server computes b by requesting the decryption of $\bar{d}$. We call it *Match* and depict the phase in Figure 3.

3.2 Challenges and Solution Sketch

In the construction shown in Figure 3, the server cannot detect whether d^* is the real decryption of $\bar{d}$ or a fake decryption. This is our first challenge: decryption integrity problem. We solve this problem by our woodenman protocol shown in Protocol 1 of Section 4.2. Conceptually, our idea is to send the result value $\bar{d}$ along with a tag. Later, the server can check the integrity of a received message d^* by verifying the tag.

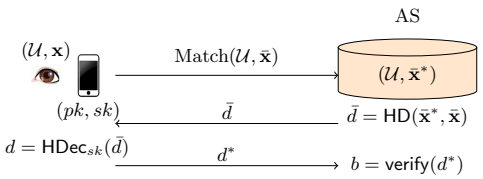

Fig. 3. Match

Unfortunately, this countermeasure is not able to prevent the server from learning personal biometrics. A trivial attack is for the server to send an $\bar{x}_i$ of its choice in place of $\bar{d}$, together with a valid tag for $\bar{x}_i$. SHE's indistinguishability ensures that the user cannot distinguish $\bar{x}_i$ from $\bar{d}$. Even an honest user with the decryption key still cannot tell x_i and d because d can be either 0 or 1. If the server repeatedly invokes the user as a decryption oracle, the user's biometric could be revealed to the server.

In order to tackle the second challenge, we present a privacy-enhanced protocol in Protocol 2 of Section 4.3. Our approach is to enable the server to execute a verification algorithm, denote by $\mathsf{verify}(\cdot)$, without knowing the decryption and the corresponding tag. Roughly speaking we enclose the decryption of a matching result by the discrete logarithm setting, but we provide an extra algorithm for the server to be able to verify the matching result.

In the following sections, we specify and evaluate the two protocols in a step-by-step manner. We are attempting to a protocol for ensuring integrity of decryptions, we build a protocol for privacy on this protocol. Thus, naturally the latter inherits much of the former. Then we plug it into our authentication protocol.

4 Upgrading the Strawman Proposal

The goals of this section are two-fold: 1) design an efficient mechanism for the integrity (but not necessarily privacy) of homomorphic evaluations for a server and 2) expand it into supporting privacy for a user. These building blocks will plug in our authentication protocol presented in Section 5. To this end, we utilize several known cryptographic tools. Our full solution will be designed and analyzed in an incremental way. We start by recalling SHE and MAC and their security concepts.

Notation. Throughout the paper, for a set A, we use $a \xleftarrow{\$} A$ to denote sampling from the uniform distribution on A. For $i \in \mathbb{N}$, we let $[i]$ be the set $\{0, 1, \ldots, i\}$. A function $\nu : \mathbb{N} \to \mathbb{R}$ is negligible in λ if for all positive polynomials $p(\cdot)$ and sufficiently large λ, $\nu(\lambda) \leq \frac{1}{p(\lambda)}$. We use $\mathsf{poly}(\lambda)$ and $\mathsf{negl}(\lambda)$ to represent unspecified polynomials and negligible functions in λ, respectively. A probabilistic polynomial-time (PPT) algorithm is a randomized algorithm that runs in time $\mathsf{poly}(\lambda)$.

We write $\mathsf{Prt}\langle P_1(a), P_2(b)\rangle \to (x, y)$ to denote a protocol Prt between P_1 and P_2, where a is P_1's input, b is P_2's input, x is P_1's output, and y is P_2's output. When describing protocols, $x \leftarrow y$ denotes that the value y is assigned to x; however, for two participants P_1 and P_2, $[P_1 \leftarrow P_2]$ denotes transmission from P_2 to P_1.

4.1 Cryptographic Background

Homomorphic encryption. A homomorphic encryption scheme is a quadruple of PPT algorithms as follows:

– $(pk, sk) \leftarrow \mathsf{HKg}(1^\lambda)$. This algorithm takes the security parameter λ and outputs a public encryption key pk and a secret decryption key sk. We assume that the public key specifies both the plaintext space $\mathcal{P}$ and the ciphertext space $\mathcal{C}$.
– $\bar{x} \leftarrow \mathsf{HEnc}(pk, x)$. The algorithm takes the public key pk and a message $x \in \mathcal{P}$ and outputs a ciphertext $\bar{x} \in \mathcal{C}$. For notational convenience, the randomizer will sometimes be omitted, and $\mathsf{HEnc}(pk, x)$ is interchangeable with $\mathsf{HEnc}_{pk}(x)$.
– $x \leftarrow \mathsf{HDec}(sk, \bar{x})$. The algorithm takes the secret key sk and a ciphertext $\bar{x}$ and outputs a message $x \in \mathcal{P}$. In addition, we use $\mathsf{HDec}(sk, \bar{x})$ together with $\mathsf{HDec}_{sk}(\bar{x})$.
– $\bar{x}_f \leftarrow \mathsf{HEv}(pk, f, \bar{x}_1, \ldots, \bar{x}_n)$. The homomorphic evaluation algorithm takes the evaluation key ek, a function $f : (\{0,1\}^*)^n \rightarrow \{0,1\}^*$, and a set of n ciphertexts $\bar{x}_1, \ldots, \bar{x}_n$ and outputs a ciphertext $\bar{x}_f$.

If a homomorphic encryption scheme is designed to allow a small number of homomorphic multiplications, it is called a somewhat homomorphic encryption (SHE) scheme. SHE offers a significant performance gain at the cost of the range of possible applications. For that reason, it is preferable to use SHE.

An SHE scheme is said to be IND-CPA secure if it achieves the indistinguishability against chosen plaintext adversaries. We use a widely known formulation of the IND-CPA security, defined as follows.

Definition 1 (IND-CPA Security) *An SHE scheme is IND-CPA secure if for any PPT adversary $\mathcal{A}$, it holds that*

$$|\mathbb{P}[\mathcal{A}(pk, ek, \mathsf{HEnc}(pk, m_0)) = 1] - \mathbb{P}[\mathcal{A}(pk, ek, \mathsf{HEnc}(pk, m_1)) = 1]| \leq \mathsf{negl}(\lambda)$$

where $(pk, ek, sk) \leftarrow \mathsf{HKg}(1^\lambda)$ and $m_0, m_1 \in \mathcal{P}$ are chosen by the adversary $\mathcal{A}$.

Specific SHE Instantiation. Fully homomorphic encryption (FHE) schemes (e.g., [14, 19]) that follow Gentry's idea [18] exhibit a fairly poor performance. A later series of results were proposed to address this concern. In particular, Brakerski and Vaikuntanathan introduced the concept of leveled FHE, which allows the evaluation of arbitrary circuits of polynomial depth [9]. Following this proposal, Brakerski, Gentry, and Vaikuntanathan (BGV) in [7] further presented a leveled FHE scheme with significantly improved performance.

Our implementation is built on the BGV scheme because it not only supports the SIMD operations but is also stably supported by HElib [21], which is a widely used software library of the BGV scheme. Although the BGV scheme supports a polynomial number of homomorphic multiplications, we instantiate the BGV scheme with a small multiplication depth.

Message authentication code (MAC). MAC is a cryptographic primitive used to provide data integrity. A MAC scheme, denoted by $\mathsf{MAC} = (\mathsf{MKg}, \mathsf{Tag}, \mathsf{Vrf})$, is a triple of PPT algorithms as follows:

– $(mk) \leftarrow \mathsf{MKg}(1^\lambda)$. The key generation algorithm takes the security parameter λ and outputs a secret key $mk \in \{0,1\}^\lambda$ for generating a tag.
– $(x, \tau) \leftarrow \mathsf{Tag}(mk, x)$. The algorithm takes as input a secret key mk and a message $x \in \mathcal{M}$ and outputs a tag τ along with the message x, where $\mathcal{M}$ is a message space.
– $(b) \leftarrow \mathsf{Vrf}(mk, x, \tau)$. The verification algorithm takes as input the secret key mk, a message x, and a tag τ and outputs $b \in \{0,1\}$. If τ is a valid tag for x, then $b = 1$; otherwise, $b = 0$.

If a secret key computed by the MAC key generation function can be used only once (otherwise, the MAC scheme can be forged), we call it a *one-time* MAC (OTM) scheme. An OTM is secure if an OTM withstands a chosen message attack. A formal definition for security of OTM is as follows.

Definition 2 (OTM Security Expermiment) *Consider the following experiment, denoted by* $\mathbf{Ex}_{\mathcal{A}}^{\mathsf{OTM}}(\lambda)$*, between an adversary* $\mathcal{A}$ *and a challenger* $\mathcal{C}$*:*

1. *The challenger* $\mathcal{C}$ *runs* $\mathsf{MKg}(1^{\lambda})$ *and obtains a secret key* mk *for a MAC.*
2. *The adversary* $\mathcal{A}$ *adaptively determines a message* x*. In* Tag *queries, the adversary* $\mathcal{A}$ *sends* x *and receives a MAC tag* $\tau \leftarrow \mathsf{Tag}(x)$*.*
3. *Once the adversary* $\mathcal{A}$ *decides that the query is over,* $\mathcal{A}$ *outputs* (x^*, τ^*)*.*
4. *The game outputs* 1 *if and only if* $\mathsf{Vrf}(mk, x^*, \tau^*) = 1$ *and* $x \neq x^*$*.*

Definition 3 *An OTM scheme* $\mathsf{OTM} = (\mathsf{MKg}, \mathsf{Tag}, \mathsf{Vrf})$ *is secure if for all PPT adversaries* $\mathcal{A}$

$$\mathbb{P}\left[\mathbf{Ex}_{\mathcal{A}}^{\mathsf{OTM}}(\lambda) = 1\right] \leq \mathsf{negl}(\lambda).$$

4.2 A Woodenman Protocol

In this section, we design a protocol Π between a user $\mathcal{U}$ and a server $\mathcal{S}$, defined by

$$\Pi\langle \mathcal{U}(sk), \mathcal{S}(\bar{d})\rangle \rightarrow (\perp, (d, b))$$

where $\perp$ indicates no output being generated, $b \in \{0, 1\}$, and sk is the decryption key for an SHE ciphertext $\bar{d}$ (i.e., $\bar{d} = \mathsf{HEnc}(pk, d)$ under a public key pk corresponding to sk). Roughly speaking, at the end of running the protocol Π only the server learns the decryption of $\bar{d}$ via interactions with the user holding the decryption key while *ensuring integrity of the decryption.* Later, the value $\bar{d}$ will enclose an encrypted result of a matching function in our authentication protocol.

The concept. Our starting point is MAC which is a cryptographic tool for data integrity, and our main observation is that using the homomorphic property of SHE, it is possible to run a MAC generation algorithm over encryptions. More precisely, given an SHE scheme $\mathsf{SHE} = (\mathsf{HKg}, \mathsf{HEnc}, \mathsf{HDec}, \mathsf{HEv})$, we can consider a function Tag^* such that for a MAC generation function Tag,

$$\mathsf{Tag}^* \circ \mathsf{HEnc}(d) = \mathsf{HEnc} \circ \mathsf{Tag}(d).$$

Indeed, we take Tag^* as $\mathsf{HEv}_{ek}(\mathsf{Tag}, mk, \bar{d})$, and this leads us to a more concrete realization as shown in Figure 4. This possibility results from a property of our settings in which the server can be viewed as the originator and, simultaneously, as the recipient.

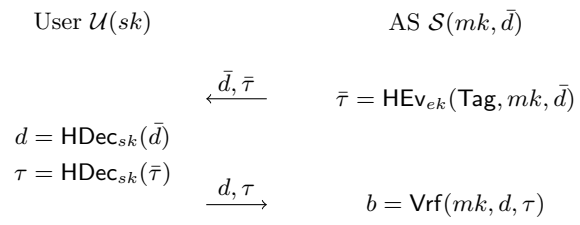

Fig. 4. Protocol-level activities of our basic idea.

However, this design strategy makes our choice of underlying MAC schemes determine the whole performance of a higher layer protocol using it. Therefore, we need to use a highly efficient MAC scheme. Besides, the server will communicate a number of users and then should provide a large storage for storing all secret keys. For these reasons, we apply a one-time variant of

MAC to our protocol. Specifically, we devise a simple variant of the Simmons OTM [37], which requires only one modular multiplication. However, our variant does not include the process of modular reduction because modular arithmetics over encryptions are significantly expensive.

Description. We assume that the server finished a computation of distance between an encrypted template and an encrypted attempt, and $\bar{d}$ denotes the computation's encrypted result. This work concentrates on the Hamming distance (HD) between two biometrics, but The details of HD computation will be given when discussing our full-fledged authentication protocol.

Let R_ℓ be a set of $\ell = \ell(\lambda)$-bit integers. Given an encrypted HD value $\bar{d}$, the server chooses r_0 and r_1 uniformly at random in R_ℓ and then computes $\bar{\tau} = r_0 \cdot \bar{d} + r_1$. The server sets $mk = (r_0, r_1)$. The user can obtain $\tau = r_0 \cdot d + r_1$ by decrypting $\bar{\tau}$ but cannot recover d because it does not know the secret key mk. The server recovers d using its secret key mk and outputs $b = \mathsf{Vrf}(mk, \tau, d)$ by checking if $d = \frac{(\tau - r_1)}{r_0}$. We present the woodenman protocol in Protocol 1.

protocol *Ensuring integrity of a matching result*	
syntax: $\langle \mathcal{U}(sk), \mathcal{S}(\bar{d}) \rangle \to (\perp, (d, b))$ where $b \in \{0, 1\}$	
1. $[\mathcal{S}]$	$(r_0, r_1) \xleftarrow{\$} (R_\ell)^2$, $\bar{\tau} \leftarrow r_0 \cdot \bar{d} + r_1$
2. $[\mathcal{U} \leftarrow \mathcal{S}]$	$(\bar{d}, \bar{\tau})$
3. $[\mathcal{U}]$	$(d, \tau) \leftarrow (\mathsf{HDec}(\bar{d}), \mathsf{HDec}(\bar{\tau}))$
4. $[\mathcal{U} \to \mathcal{S}]$	(d, τ)
5. $[\mathcal{S}]$	Check if $d \stackrel{?}{=} \frac{\tau - r_1}{r_0}$

Protocol 1. Our Woodenman Scheme Π_{T1}

Performance. Once after the server obtains $\bar{d}$, it only has to perform one addition and multiplication by constants. These operations are fairly cheaper than other homomorphic operations (see Table 4). On server's side the heaviest computation is to compute $\bar{d}$ by evaluating a matching function at encrypted biometrics. As before, let N be the number of plaintext slots, and let n be the bit length of plaintexts. For a plaintext, $\kappa = \lceil n/N \rceil$ SHE ciphertexts are generated. Then the evaluation requires κ homomorphic squarings, $\kappa + \lceil \log N \rceil$ additions (exactly $\log N$ additions and κ subtractions), and $\lceil \log N \rceil$ rotations over plaintext slots.

Security against Threat #1. As described above, the server generates an encrypted tag $\bar{\tau}$ and receives the decryption of $\bar{\tau}$ with the help of the user. From the MAC security point of view, the user (i.e., decryptor) is modeled as an attacker. Thus we develop a new security model in which the user is allowed to produce any output that it intends to return. We define security against one-time decryptor forgery attacks. This concept requires that, even if the adversary possesses the decryption key for the challenge input, the adversary should not be able to return an incorrect output that passes verification.

Definition 4 (Decryptor Forgery Experiment) *Let* $\mathsf{SHE} = (\mathsf{HKg}, \mathsf{HEnc}, \mathsf{HDec}, \mathsf{HEv})$ *be an SHE scheme and* $\mathsf{OTM} = (\mathsf{MKg}, \mathsf{Tag}, \mathsf{Vrf})$ *be an OTM scheme, and let* Π *be a protocol associated with* SHE *and* OTM*. We consider the following experiment between a PPT adversary* $\mathcal{A}$ *and the server* $\mathcal{S}$*,*

Experiment $\mathbf{Ex}^{dfa}_{\mathcal{A},\Pi}(\mathsf{SHE}, \mathsf{OTM}, \lambda)$
$(pk, ek) \leftarrow \mathcal{A}^{\mathsf{HKg}(\cdot)}$
Run the protocol $\Pi\langle \mathcal{A}(sk), \mathcal{S}(\bar{d}) \rangle$
$\mathcal{A}$ *outputs* (d^*, τ^*) *such that* $d^* \neq \mathsf{HDec}(sk, \bar{d})$
If $\mathsf{Vrf}(mk, d^*, \tau^*) = 1$*, output* 1*; else, output* 0

For correctness, we require that, for every mk and for every d such that $\mathsf{Vrf}(mk, d, \mathsf{Tag}(mk, d)) = b \in \{0, 1\}$, we have $\Pi\langle\mathcal{U}(sk), \mathcal{S}(mk, \bar{d})\rangle \to (\perp, (d, b))$. Next, we require that an adversary controlling a user succeeds in the experiment **Ex** with negligible probability.

Definition 5 *A protocol Π is* secure *against decryptor-forgery attacks if it is correct and if there exists a negligible function $\mathsf{negl}(\cdot)$ for every PPT algorithm $\mathcal{A}$ such that*

$$\mathbb{P}\left[\mathbf{Ex}^{dfa}_{\mathcal{A},\Pi}(\mathsf{SHE}, \mathsf{OTM}, \lambda) = 1\right] \leq \mathsf{negl}(\lambda).$$

We argue that the woodenman protocol Π_{T1} is secure against a corrupted user. In Lemma 1, we first show that our OTM variant is secure against one-time chosen-message forgery. Then, we prove Theorem 1 using the lemma.

Lemma 1 *Let R_ℓ be a set of ℓ-bit integers. Given d and τ such that $\tau = r_0 \cdot d + r_1$ for two integers r_0, r_1 uniformly chosen from R_ℓ,*

$$\mathop{\mathbb{P}}_{r_0,r_1}[\tau = r_0 \cdot d + r_1] < 2^{2-\ell}$$

where the probability is taken over the random choice of r_0 and r_1 from R_ℓ.

Proof. Because d and $\tau = r_0 \cdot d + r_1$ are given, we can obtain a set of candidate points r_0 and r_1. More specifically, by the extended Euclidean algorithm for d and τ, there exist unique $\tilde{r}_0$ and $\tilde{r}_1$ such that $\tau = \tilde{r}_0 d + \tilde{r}_1$ with $\tilde{r}_1 < d$. Because $\tilde{r}_0 \geq 2^{\ell-1}$, this can be transformed into

$$\begin{aligned}\tau &= \tilde{r}_0 d + \tilde{r}_1 \\ &= (\tilde{r}_0 - 1)d + (d + \tilde{r}_1) = (\tilde{r}_0 - 2)d + (2d + \tilde{r}_1) = \cdots \\ &= 2^{\ell-1}d + (\tilde{r}_0 - 2^\ell)d + \tilde{r}_1.\end{aligned}$$

We see that these are all candidates for r_0 and r_1, and thus, the number of candidates is $\tilde{r}_0 - 2^\ell + 1$. Further, $\tilde{r}_0$ is a quotient of τ and d, $\tilde{r}_0 = \lfloor \tau/d \rfloor$. Therefore, the probability of choosing the correct pair of r_0 and r_1 is $\frac{1}{\tilde{r}_0 - 2^{\ell-1}+1} = \frac{1}{\lfloor \tau/d \rfloor - 2^{\ell-1}+1}$. Moreover, because r_0 is randomly chosen in R_ℓ, we can assume that $r_0 \approx 3/2 \cdot 2^{\ell-1} = 3 \cdot 2^{\ell-2}$, and thus, the probability of this assumption becomes

$$\mathop{\mathbb{P}}_{r_0,r_1}[\tau = r_0 \cdot d + r_1] \approx \frac{1}{2^{\ell-2}+1} < \frac{1}{2^{\ell-2}}.$$

Thus, we may conclude the lemma. □

Before proceeding to the next step, we need to define two random variables dirty and verify. Intuitively, $\mathsf{dirty} = 1$ if a user modifies the decryptions of a pair $(\bar{d}, \bar{\tau})$ and $\mathsf{verify} = 1$ if the protocol outputs an accept.

Definition 6 *We define a random variable* dirty *such that* $\mathsf{dirty} = 1$ *if and only if* $d^* \neq \mathsf{HDec}(\bar{d})$ *or* $\tau^* \neq \mathsf{HDec}(\bar{\tau})$. *In addition, we define* verify *such that* $\mathsf{verify} = 1$ *if and only if the honest server outputs an* accept *(i.e., $b = 1$) at the end of the protocol Π_{T1}.*

Theorem 1 states that our basic protocol Π_{T1} is secure against Threat #1 except for a negligible probability in the security parameter λ.

Theorem 1 *Let R_ℓ be a set of ℓ-bit integers. Assume that the underlying OTM scheme is secure against one-time existential forgery attacks. Then, for every PPT adversary $\mathcal{A}$, there exists a negligible function $\mathsf{negl}(\cdot)$ such that*

$$\mathbb{P}[\mathbf{Ex}^{dfa}_{\mathcal{A},\Pi_{T1}}(\mathsf{SHE}, \mathsf{OTM}, \lambda) = 1] \leq 2^{3-\ell} + \mathsf{negl}(\lambda).$$

Proof. To prove the theorem, it suffices to show that

$$\mathbb{P}\left[\mathsf{dirty} = 1 \wedge \mathsf{verify} = 1\right] \leq \mathsf{negl}(\lambda).$$

To do this, we prove that we can construct an efficient algorithm that can forge our OTM scheme with non-negligible probability, assuming that there is an adversary that can succeed in passing the protocol Π_{T1} with non-negligible probability, using (d^*, τ^*) such that $d^* \neq \mathsf{HDec}(\bar{d})$ or $\tau^* \neq \mathsf{HDec}(\bar{\tau})$. The remainder of the proof follows directly by a standard reduction argument; however, the calculation of the success probabilities can be quite tedious.

We proceed to construct an OTM adversary $\mathcal{A}_{\text{otm}}$ that works as follows. Let $\mathcal{A}$ be an adversary for the protocol Π_{T1} such that $\mathbb{P}\left[\mathsf{dirty} = \mathsf{verify} = 1\right] = \epsilon(\lambda)$.

The adversary $\mathcal{A}_{\text{otm}}$. On input 1^λ and the values (d, τ) from the challenger,

1. $\mathcal{A}_{\text{otm}}$ invokes $\mathcal{A}$ on input 1^λ and outputs (pk, ek).
2. $\mathcal{A}_{\text{otm}}$ interacts with $\mathcal{A}$, playing the honest server in the protocol as follows:
   (a) $\mathcal{A}_{\text{otm}}$ sets up the OTM key mk of the honest server as in the protocol and sets $\bar{d} = \mathsf{HEnc}_{pk}(d)$ but computes $\bar{\tau} = \mathsf{HEnc}_{pk}(\tau)$ without the key mk, where (d, τ) is given by the challenger.
   (b) $\mathcal{A}_{\text{otm}}$ sends the pair $(\bar{d}, \bar{\tau})$ to the adversary $\mathcal{A}$.
3. On receiving a pair of decryptions (d^*, τ^*), $\mathcal{A}_{\text{otm}}$ checks if $d = d^*$. If $d = d^*$, then $\mathcal{A}_{\text{otm}}$ outputs a pair of random values $(r_0, r_1) \in (R_\ell)^2$. Otherwise, if $d \neq d^*$ and $\tau \neq \tau^*$, it outputs $\left(r_0 = \frac{\tau - \tau^*}{d - d^*}, \tau - d \cdot r_0\right)$; however, if $d \neq d^*$ and $\tau = \tau^*$, it outputs $(0, \tau)$.

The intuition behind the attack strategy by $\mathcal{A}_{\text{otm}}$ is straightforward. Next, we proceed to prove that $\mathcal{A}_{\text{otm}}$ outputs the correct pair of values (r_0, r_1) with probability $\epsilon(\lambda)(1 - \mathsf{negl}(\lambda))$, which is non-negligible if $\epsilon(\lambda)$ is non-negligible. To do this, define Fail to be the event where $\mathcal{A}_{\text{otm}}$ outputs a pair of random values in this attack. We have

$$\begin{aligned}\mathbb{P}[\mathbf{Ex}^{\text{otm}}_{\mathcal{A}_{\text{otm}}}(1^\lambda) = 1] = \mathbb{P}[\mathbf{Ex}^{\text{otm}}_{\mathcal{A}_{\text{otm}}}(1^\lambda) = 1 | \neg\mathsf{Fail}] \cdot \mathbb{P}[\neg\mathsf{Fail}] + \\ \mathbb{P}[\mathbf{Ex}^{\text{otm}}_{\mathcal{A}_{\text{otm}}}(1^\lambda) = 1 | \mathsf{Fail}] \cdot \mathbb{P}[\mathsf{Fail}] \qquad (2)\end{aligned}$$

We see that $\mathbb{P}[\mathbf{Ex}^{\text{otm}}_{\mathcal{A}_{\text{otm}}, \Pi_{T1}}(1^\lambda) = 1 | \mathsf{Fail}] = 2^{2-\ell}$ by the definition of Fail, and the probability that $\mathcal{A}_{\text{otm}}$ outputs an incorrect pair of values, conditioned on Fail not occurring, is at most negligible. Thus we have

$$\mathbb{P}[\mathbf{Ex}^{\text{otm}}_{\mathcal{A}_{\text{otm}}}(1^\lambda) = 1 | \neg\mathsf{Fail}] \geq 1 - \mathsf{negl}(\lambda)$$

for a negligible function $\mathsf{negl}(\cdot)$.

We now only have to compute $\mathbb{P}[\mathsf{Fail}]$ and $\mathbb{P}[\neg\mathsf{Fail}]$ to evaluate Eq. (2). We compute $\mathbb{P}[\neg\mathsf{Fail}]$ by

$$\begin{aligned}\mathbb{P}[\neg\mathsf{Fail}] = \mathbb{P}[\neg\mathsf{Fail} | \mathsf{dirty} = \mathsf{verify} = 1] \cdot \mathbb{P}[\mathsf{dirty} = \mathsf{verify} = 1] + \\ \mathbb{P}[\neg\mathsf{Fail} | \mathsf{dirty} = 0 \vee \mathsf{verify} = 0] \cdot \mathbb{P}[\mathsf{dirty} = 0 \vee \mathsf{verify} = 0].\end{aligned}$$

By our assumption regarding $\mathcal{A}$, we have that $\mathbb{P}[\mathsf{dirty} = \mathsf{verify} = 1] = \epsilon(\lambda)$. Hence, it follows that $\mathbb{P}[\mathsf{dirty} = 0 \vee \mathsf{verify} = 0] = 1 - \epsilon(\lambda)$. Next, if $\mathsf{dirty} = \mathsf{verify} = 1$, then $\mathcal{A}_{\text{otm}}$ outputs a pair of random values only when $d = d^* \wedge \tau \neq \tau^*$. Thus

$$\mathbb{P}[\neg\mathsf{Fail} | \mathsf{dirty} = \mathsf{verify} = 1] = 1 - 2^{2-\ell}.$$

In contrast, if $\mathsf{dirty} = 0$ or $\mathsf{verify} = 0$, then $\mathcal{A}_{\mathrm{otm}}$ always outputs a pair of random values. Thus, $\mathbb{P}[\neg\mathsf{Fail}|\mathsf{dirty} = 0 \vee \mathsf{verify} = 0] = 0$. Combining these, we have that

$$\mathbb{P}[\neg\mathsf{Fail}] = (1 - 2^{2-\ell})\epsilon(\lambda) \text{ and } \mathbb{P}[\mathsf{Fail}] = 2^{2-\ell}\epsilon(\lambda).$$

Merging this into Eq (2), we have that

$$\begin{aligned}\mathbb{P}[\mathbf{Ex}^{\mathrm{otm}}_{\mathcal{A}_{\mathrm{otm}}}(1^\lambda) = 1] &= (1 - \mathsf{negl}(\lambda))(1 - 2^{2-\ell})\epsilon(\lambda) + 2^{4-2\ell}\epsilon(\lambda) \\ &= \epsilon(\lambda)\left(1 - \mathsf{negl}(\lambda) - 2^{2-\ell}(1 - \mathsf{negl}(\lambda) - 2^{2-\ell})\right) \\ &= \epsilon(\lambda)(1 - \mathsf{negl}(\lambda)) - \mathsf{negl}^\star(\lambda)\end{aligned}$$

for a negligible function $\mathsf{negl}^\star(\cdot)$. Thus, if $\epsilon(\lambda)$ is non-negligible, then $\mathcal{A}_{\mathrm{otm}}$ succeeds in $\mathbf{Ex}^{\mathrm{otm}}_{\mathcal{A}_{\mathrm{otm}}}(1^\lambda)$ with probability $\epsilon(\lambda)(1-\mathsf{negl}(\lambda))$, which is also non-negligible. However, this contradicts Lemma 1.

We have proven that $\mathbb{P}[\mathsf{dirty} = \mathsf{verify} = 1]$ is negligible. For notational convenience, we assume that the experiment $\mathbf{Ex}^{\mathrm{dfa}}_{\mathcal{A},\Pi_{T1}}$ takes $(\mathsf{SHE}, \mathsf{OTM}, \lambda)$ as input. To show the remainder of the proof, we observe that

$$\begin{aligned}\mathbb{P}[\mathbf{Ex}^{\mathrm{dfa}}_{\mathcal{A},\Pi_{T1}} = 1] &= \mathbb{P}[\mathbf{Ex}^{\mathrm{dfa}}_{\mathcal{A},\Pi_{T1}} = 1 \wedge \mathsf{dirty} = 1] + \mathbb{P}[\mathbf{Ex}^{\mathrm{dfa}}_{\mathcal{A},\Pi_{T1}} = 1 \wedge \mathsf{dirty} = 0] \\ &= \mathbb{P}[\mathbf{Ex}^{\mathrm{dfa}}_{\mathcal{A},\Pi_{T1}} = 1 \wedge \mathsf{dirty} = 1 \wedge \mathsf{verify} = 1] \\ &\quad + \mathbb{P}[\mathbf{Ex}^{\mathrm{dfa}}_{\mathcal{A},\Pi_{T1}} = 1 \wedge \mathsf{dirty} = 1 \wedge \mathsf{verify} = 0] + \mathbb{P}[\mathbf{Ex}^{\mathrm{dfa}}_{\mathcal{A},\Pi_{T1}} = 1 \wedge \mathsf{dirty} = 0] \\ &\leq \mathbb{P}[\mathsf{dirty} = \mathsf{verify} = 1] + \mathbb{P}[\mathbf{Ex}^{\mathrm{dfa}}_{\mathcal{A},\Pi_{T1}} = 1 \wedge \mathsf{verify} = 0] \\ &\quad + \mathbb{P}[\mathbf{Ex}^{\mathrm{dfa}}_{\mathcal{A},\Pi_{T1}} = 1 \wedge \mathsf{dirty} = 0] \\ &\leq \mathbb{P}[\mathbf{Ex}^{\mathrm{dfa}}_{\mathcal{A},\Pi_{T1}} = 1 \wedge \mathsf{verify} = 0] + \mathbb{P}[\mathbf{Ex}^{\mathrm{dfa}}_{\mathcal{A},\Pi_{T1}} = 1 \wedge \mathsf{dirty} = 0] + \mathsf{negl}(\lambda).\end{aligned}$$

Because we see that $\mathbb{P}[\mathbf{Ex}^{\mathrm{dfa}}_{\mathcal{A},\Pi_{T1}} = 1 \wedge \mathsf{verify} = 0] = 2^{2-\ell}$ and $\mathbb{P}[\mathbf{Ex}^{\mathrm{dfa}}_{\mathcal{A},\Pi_{T1}} = 1 \wedge \mathsf{dirty} = 0] = 2^{2-\ell}$, we conclude that

$$\mathbb{P}[\mathbf{Ex}^{\mathrm{dfa}}_{\mathcal{A},\Pi_{T1}} = 1] \leq 2^{3-\ell} + \mathsf{negl}(\lambda).$$

This completes the proof of the theorem. □

Discussion. We argue that our choice of OTM implies optimality in the multiplicative depth. This is because its Boolean circuit construction requires only one multiplication by a random value r_0; however, if we omit either multiplication by r_0 or addition of r_1, we lose its security.

We may interpret the security as the false positive rate (FPR). Specifically, If we choose $r_0, r_1 \in R_{83}$, we see that the FPR is approximately 2^{-80}. Theoretically, SHE can handle homomorphic operations over such large integers. However, as a tradeoff, an SHE instance with this capability cannot avoid serious performance degradation. One solution to address the efficiency issue is to split into multiple tags consisting of random numbers of small bits.

4.3 An Ironman Protocol

Allowing decryption queries results in an attack to compromise biometric privacy. We have discussed how to mount the attack in Section 3.2. Our idea for circumventing this attack is that a user does not send d directly but rather a hint enclosing d, by which the server can verify 1) whether the d in the hint is authentic and 2) whether the d hidden in the hint is less than a matching threshold T.

The concept and description. We borrow this idea from a standard technique appending a non-interactive proof of well-formedness (satisfying certain criteria) to a message. To realize

our idea, we require that all decrypted values by the user be raised to the power of a generator in the discrete logarithm (DL) setting.

Let $\mathbb{G}$ be a cyclic group of a large prime order p in which the DL assumption holds, and let h be a generator of $\mathbb{G}$. Let $H_1 : \{0,1\}^* \to R_\ell$ and $H_2 : \mathbb{G} \to \{0,1\}^{\mathsf{poly}(\lambda)}$ be random oracles. Then, a user chooses a random generator $h \in \mathbb{G}$, computes $v = h^\tau$ with $\tau = \mathsf{HDec}(sk, \bar{\tau})$. From the DL assumption, the server cannot construct an efficient algorithm to compute τ from v. The only remaining problem is how to enable the server to run the verification using the value v. Our idea is as follows: given a threshold T, we require the user to build a set G of hashed values by $G = \{H_2(h^j) | j \in [T]\}$, and send the value v along with the set G to the server.

The MAC verification $\mathsf{Vrf}(r_0, r_1, u, v)$ is straightforward: compute $h^* = (vu^{-r_1})^{\frac{1}{r_0}}$ and check if $H_2(h^*) \in G$, where $u = h$. Notice that neither d nor h^d is given to the server. The full description of the protocol is shown in Protocol 2.

protocol *Ensuring privacy of biometrics*

syntax: $\langle \mathcal{U}(sk), \mathcal{S}(\bar{d}) \rangle \to (\perp, b)$

1. $[\mathcal{S}]$ $(r_0^*, r_1^*) \xleftarrow{\$} (\{0,1\}^*)^2$,
   $(r_0, r_1) \leftarrow (H_1(r_0^*), H_1(r_1^*))$,
   $\bar{\tau} \leftarrow r_0 \cdot \bar{d} + r_1$
2. $[\mathcal{U} \leftarrow \mathcal{S}]$ $(\bar{d}, \bar{\tau})$
3. $[\mathcal{U}]$ $(d, \tau) \leftarrow \left(\mathsf{HDec}(\bar{d}), \mathsf{HDec}(\bar{\tau})\right)$,
   $h \xleftarrow{\$} \mathbb{G}$ such that $\langle h \rangle = \mathbb{G}$
4. $[\mathcal{U} \to \mathcal{S}]$ $(u, v, G) \leftarrow \left(h, h^\tau, \{H_2(h^j) | j \in [T]\}\right)$
5. $[\mathcal{S}]$ $h^* \leftarrow \left(vu^{-r_1}\right)^{\frac{1}{r_0}}$,
   check if $H_2(h^*) \stackrel{?}{\in} G$

Protocol 2. Our Ironman Scheme Π_{T2}

Performance. An additional overhead in computation arises on the user side. The user performs T exponentiations in modulo p along with T hash operations. Because a biometric is encrypted into κ ciphertexts, the total cost amounts to κT exponentiations and κT hashings. However, each exponent is very small (in our case, at most 8 bits), and hash operations are negligible in computation overhead. User's decryption costs are the same as before, i.e., 2κ times HDec operations.

Security against Threat #2. We define a precise security model for analyzing the security of Protocol 2. We would like to note that we do not need to allow the adversary to have full access to a decryption oracle, as observed above. Nevertheless, the protocol seems to be useful for the attacker to extract some biometric information because it reveals something at Step 5 of the protocol Π_{T2}.

To characterize privacy, we need to modify the protocol Π given in Section 4.2, and newly define an ideal functionality $\mathcal{F}$ as

$$\widetilde{\Pi}\langle \mathcal{U}(sk), \mathcal{S}(\bar{d}) \rangle \to (\perp, b)$$

where $b \in \{0,1\}$. We notice that d does not appear at the output of the server.

Our privacy requirement captures the notion of protecting users' biometric templates during protocol executions. In the secure computation model, participants have their own private input and are willing to evaluate a desired functionality $\mathcal{F}$ on their inputs without revealing any information except the outputs and their unavoidable implications [20].

Intuitively, the following two scenarios should be indistinguishable in a computational sense: 1) securely computing $\mathcal{F}$ by executing a protocol, and 2) privately sending their private inputs to a trusted party, who then computes $\mathcal{F}$ and privately returns the outputs to each participant. This formalization of securely computing is referred to as the simulation-based approach. The idea of the standard simulation-based privacy proof technique is that, given a well-defined privacy-leakage, a simulator running in polynomial time can generate a transcript that is indistinguishable from the output of the real protocol. If such an efficient simulator exists, then an adversary cannot learn any additional information beyond the defined leakage. The simulator must perform its task without knowing the private information of the participant who wants to proves his authenticity.

We now argue that our ironman protocol Π_{T2} is secure against Threat #2. In this proof, we show that given a participant is corrupted (either user or server), there exists a simulator, denoted by $\mathcal{A}$, that can produce a view which is statistically indistinguishable from the view of that participant interacting with the honest counterpart. Assuming that one participant is corrupted, we build an efficient simulator that has access to the public input and private materials (e.g., secret key and biometrics) of the corrupted participant. In addition, the simulator knows the public output.

It is worth mentioning that the proposed protocol gives computational privacy to both the user and the server because the underlying SHE scheme provides the IND-CPA security. Furthermore, these simulations produce views which are statistically indistinguishable from the views in the real protocol executions.

Theorem 2 *Assuming that the SHE scheme provides the IND-CPA security, the protocol Π_{T2} in Protocol 2 is secure in the presence of semi-honest adversaries.*

We first sketch an idea showing why we can construct an efficient simulator $\mathcal{A}$. Because the attacker owns the decryption key of the SHE scheme and the corrupted user's biometric, the simulator can output an indistinguishable view from that in the real protocol executions using the ciphertext indistinguishability of SHE. The simulator knows the secret key of a MAC scheme and, further, all decryptions of matching results are enclosed in the discrete logarithm setting. For that reason, the adversary against the real protocol cannot distinguish from the output of the simulator.

Proof. We show that given a participant is corrupted, there exists a simulator that can produce a view to the adversary that is statistically indistinguishable from the view in the real protocol execution based on its secret inputs as well as public information.

Case I: User is corrupted. In this case, we prove that our protocol is secure when a user is corrupted. The simulator $\mathcal{A}$ has the decryption key sk of the user and knows the user's biometrics including other public information generated by the protocol.

Now the simulator constructs a view for the user which is statistically indistinguishable to the one that the user observes during interacting with the honest server over these values. The simulator works as follows:

1. The simulator obtains a pair of attempt and template biometrics $(\mathbf{x}, \mathbf{x}^*)$ and the identity $\mathcal{U}$.
2. The simulator computes $\bar{\mathbf{x}} = \mathsf{HEnc}(pk, \mathbf{x})$ and $\bar{\mathbf{x}}^* = \mathsf{HEnc}(pk, \mathbf{x}^*)$.
3. The simulator computes $\delta = s_0 \cdot d + s_1$ for randomizers $s_0, s_1 \in R_\ell$ and a HD value d between $\mathbf{x}$ and $\mathbf{x}^*$.
4. The simulator encrypts the value δ into $\bar{\delta} = \mathsf{HEnc}(pk, \delta)$ and the encryption $\bar{\delta}$ is the simulated output.

Each step of the proposed authentication protocol for the simulator is simulated and this completes the simulation for the compromised user. The transcript is consistent and statistically indistinguishable from the users view when interacting with the honest server.

Case II: Server is corrupted. In this case, we prove that our protocol is secure when a server is compromised by an attacker in the real-world protocol. The simulator $\mathcal{A}$ has the secret ket $mk = (r_0, r_1)$ of the server and user's encrypted biometric attempt $\bar{\mathbf{x}}$ and template $\bar{\mathbf{x}}^*$ from the protocol. Now we can construct the simulator that runs as follows:

1. The simulator outputs an encryption $\bar{\delta} = r_0 \cdot \bar{\mathbf{x}} + r_1$.
2. Upon receiving (u, v, G) from the honest user, the simulator would run the remaining steps the protocol.
3. The simulator outputs $\beta \in \{0, 1\}$ of his choice.

Consequently, each step of the proposed authentication protocol for the simulator is simulated and this completes the simulation for the corrupted server. The transcript is consistent and statistically indistinguishable from the users view because u, v are random elements in the DL group which is uniformly distributed. Moreover, the set G also contains only elements in the DL group $\mathbb{G}_p$. □

Discussion. We need to take care of choosing the group $\mathbb{G}$. The inverse of r_0 modulo $p - 1$ should exist for correctness on the server side. For security's sake, it is desirable for the set R_ℓ to be large. Thus, we suggest the use of Sophie Germain primes since they allow the use of larger exponents. Then, the server only has to choose an odd number r_0, randomly.

If the size of τ is relatively small, the server can find logarithms modulo p. This reveals the number of different coordinates in two encrypted biometrics. However, the adversary cannot see the precise positions of different coordinates.

One may worry about privacy breach by device theft. An adversary could steal a smartphone of a target victim, and try to authenticate himself by playing the victim. More specifically, if an attacker can intercept a communication transcript between a user and a server during the user's enrollment with $\bar{\mathbf{x}}^*$, it may later play the user. One approach to averting such an attack depends on forward secrecy, and thus, we require that all protocol messages be encrypted under a session key k induced by an forward-secure key agreement (FKA) protocol. In our implementation, we used the MTI/A0 variant of the Diffie-Hellman key agreement [30, §12.6].

5 Our Biometric Authentication

In this section, we demonstrate our biometric authentication protocol by putting together enhancements developed in Sections 4. We argue that our suggestion meets the security requirements, and analyze the computation and communication overheads.

5.1 How Ghostshell Works

Ghostshell consists of three phases: *Setup*, *Enrollment*, and *Match*. In the Setup phase, Ghostshell fixes the system settings by running the key generation algorithm of each underlying cryptographic scheme (see Section 4.1). Ghostshell then lets a user extract its biometric template, encodes the template as a binary string, and stores the encrypted template to the server. In subsequent uses, the user's attempt biometric is compared with the encrypted template.

In what follows, we provide the detailed descriptions for each phase.

Setup. For the security parameter λ, a user determines a pair of keys (pk, sk) by invoking $\mathsf{HKg}(1^\lambda)$ of our SHE scheme together with other system parameters.

Our authentication system is designed to run on a symmetric variant of SHE. Namely, the server cannot generate an encryption of its choice because it does not know the secret key sk, but it still can perform homomorphic evaluations using the public key pk. This approach enables us to obtain several benefits: On the security side, the server is not allowed to encrypt arbitrary matching result. On the practical side, this approach reduces the computational cost.

Enrollment. The user first extracts his biometric feature $\mathbf{x}^*$, computes its SHE ciphertext $\bar{\mathbf{x}}^* = \mathsf{HEnc}(sk, \mathbf{x}^*)$, and sends $(\mathcal{U}, \bar{\mathbf{x}}^*)$ to the server. On receiving it, the server stores $(\mathcal{U}, \bar{\mathbf{x}}^*)$ on its database.

For better security against device-theft attack discussed in Section 4.3, we may encrypt all interactions between two participants using a session key deduced by an FKA protocol. In this case, the user encrypts $(\mathcal{U}, \bar{\mathbf{x}}^*)$ by a symmetric encryption scheme; the server stores $(\mathcal{U}, \bar{\mathbf{x}}^*)$ after decryption under the session key.

Match. The first step of this phase is that the user encrypts his attempt biometric into $\bar{\mathbf{x}} = \mathsf{HEnc}(sk, \mathbf{x})$ and sends $(\mathcal{U}, \bar{\mathbf{x}})$ to the server.

– Upon receiving it, the server retrieves $\bar{\mathbf{x}}^*$ indicated by $\mathcal{U}$ and computes $\bar{d} = \mathsf{HD}(\bar{\mathbf{x}}, \bar{\mathbf{x}}^*)$. Next, the server and the user jointly execute the protocol Π_{T2} given in Protocol 2, i.e.,

$$\Pi_{T2}\langle \mathcal{U}(sk), \mathcal{S}(\bar{d})\rangle \rightarrow (\perp, b = \{\mathsf{accept}, \mathsf{reject}\})$$

5.2 Analysis

Performance. We evaluate the performance of Ghostshell by counting cryptographic operations and measuring the transmission amount by the bit lengths of biometrics. This analysis is based on our performance tuning techniques in Section 6.

Let n be the size of $\mathbf{x}$ and $\mathbf{x}^*$, and let γ denote the SHE ciphertext size. Assuming the SHE instantiation with N plaintext slots, we only need to keep $\kappa = \lceil n/N \rceil$ SHE ciphertexts for an n-bit biometric. By contrast, a naïve construction using the SHE yields n SHE ciphertexts. For the Enrollment phase, the user invokes HEnc κ times. The performance of the Match phase is determined by the performance of Π_{T2} and the performance of the circuit HD which computes a distance on encrypted biometrics. The heavier computation is to evaluate the HD circuit. On the other hand, user's most expensive computation is the SHE decryption.

Security. We examine the security of our protocol. In the Enrollment phase, biometric privacy is protected by our SHE encryption. For the Match phase, a user corrupted by an attacker cannot fool the AS by Theorem 1. Theorem 2 states that an attacker mounted on the server side cannot learn any information about the user's biometrics except for the verification result and its unavoidable implications. In conclusion, Ghostshell provides security against Threats #1 and #2.[4]

6 Optimizations

In this section, we demonstrate our optimization techniques; our solution demonstrates the technique's preliminary feasibility for practical deployment to real-world systems. Our optimizations have two different objectives: speeding-up computation times and reducing transmission costs.

[4] Using an FKA protocol, our authentication protocol ensures security against device-theft attack.

6.1 Speeding-up Computations

In Ghostshell, computing the Hamming distance between two ciphertexts for n-bit biometrics is the most expensive computation. A naïve computation of the Hamming distance on ciphertexts requires n homomorphic multiplications and subtractions and $n-1$ homomorphic additions; see Eq. (1). Among them, we focus on improving the computation efficiency of homomorphic multiplication because this operation is about $280\times$ slower than homomorphic addition.

Our technique builds on SIMD techniques, which were introduced by Smart and Vercauteren [39] (Journal version of [38]). Their idea is that a plaintext space can be treated as a partition of plaintext spaces of small size, called slots, and that a ciphertext carries a vector of plaintexts instead of a plaintext. By adding (resp., multiplying) such packed ciphertexts, one can perform component-wise addition (resp., multiplication) of two vectors of plaintexts.

Obviously, ciphertext packing and SIMD operations allow to efficiently perform homomorphic evaluations on multiple data at once. Multiplication and subtraction of the HD circuit benefit from two techniques. The problem with that we are faced is that the HD circuit requires to sum all the components additionally. However, there is no intrinsic operation that supports addition and multiplication across different positions (or slots). Accordingly, the question is how to efficiently compute the sum over plaintext slots.

We resolve this problem by using automorphisms as a technique to move values between the different slots in a given plaintext vector. More specifically, performing automorphism $X \mapsto X^i$ means that a vector of messages (cyclically) rotates to the right i times, respectively. We denote the automorphism mapping X to X^i by σ^i and a vector of N bits by $\mathbf{w} = (w_0, \ldots, w_{N-1})$.

A simple way of computing the Hamming distance for $\mathbf{w}$ is to add all vectors $\sigma^i(\mathbf{w})$ for $i \in [N-1]$. Because this naïve method requires $N-1$ homomorphic additions and automorphisms, the complexity remains $O(N)$. Our idea for reducing this computational cost is to use a tree structure and to proceed recursively with each of 2^i elements. This leads to a binary tree of depth $\lceil \log N \rceil$, where log is the binary logarithm. Applying this idea, we can compute the HD value of two independent vectors with one homomorphic multiplication, one homomorphic subtraction, and $O(\log N)$ homomorphic additions and automorphisms. Recall that the homomorphic automorphism does not change the noise estimate.

Example. Let $N = 8$ and $\mathbf{w}_0 = (w_0, w_1, \ldots, w_7)$. Letting $\mathbf{w}_1 := \sigma^1(\mathbf{w}_0) = (w_7, w_0, \ldots, w_6)$, we have $\mathbf{w}_0 + \mathbf{w}_1 = (w_0+w_7, w_0+w_1, \ldots, w_6+w_7)$. Similarly, we obtain $\mathbf{w}_2 = (w_5+w_6, \ldots, w_4+w_5)$ with $\sigma^2(\mathbf{w}_0 + \mathbf{w}_1)$ and $\mathbf{w}_3 = (w_1 + w_2 + w_3 + w_4, \ldots, w_0 + w_1 + w_2 + w_3)$ with $\sigma^4(\mathbf{w}_2)$. Then, we can obtain the HD value $\sum w_i$ by $\sum_{i\in[3]} \mathbf{w}_i$.

A Practical Variant of the Ironman Protocol Theoretically, this idea holds for arbitrary N. However, the larger the number of slots N, the bigger the other parameters, which could result in slowdowns of our system. Thus, we have to choose a modest N as a trade-off between the size of associated parameters and the BGV instantiation's performance using the HElib library [21]. On balance, we take $N = 630$ and $m = 8191$, where m is a system parameter and the m-th cyclotomic polynomial $\Phi_m(X)$ is used to determine polynomial rings on which the BGV scheme runs.

Because $n \geq 2048 > N$ for some standard biometric features, when we implement the ironman protocol Π_{T2}, we need to maintain one or more ciphertexts at a time. For example, we have four ciphertexts for 2400-bit iris codes. As a consequence, the server should evaluate the HD circuit as many as the number of required ciphertexts and, thus, the result HD value is split and stored into each ciphertext in order. This is the reason why we revise the ironman protocol.

As before, let use $\kappa = \lceil \frac{n}{N} \rceil$ to denote the minimum number of ciphertexts for carrying an HD value. We describe only the differences from the original protocol as shown in Protocol 3. We use $\bar{d} = (\bar{d}[0] \parallel \cdots \parallel \bar{d}[\kappa - 1])$ to denote splitting a resutling HD value $\bar{d}$ into κ ciphertexts $\bar{d}[0], \ldots, \bar{d}[\kappa - 1]$, and we set $u_j = h_j, v_j = {h_j}^{\tau[j]}$, and $G_j = \left\{H_2({h_j}^l)\right\}$ for all $l \in [T_\kappa]$ likewise (see below for T_κ).

protocol Practical Variant of the Ironman Protocol	
syntax: $\langle \mathcal{U}(sk), \mathcal{S}(\bar{x}) \rangle \rightarrow (\perp, b)$ where $b \in \{0, 1\}$	
1. $[\mathcal{S}]$	Compute $(r_0[j], r_1[j])_{j \in [\kappa - 1]}$ likewise $\forall j \in [\kappa - 1], \bar{\tau}[j] \leftarrow r_0[j] \cdot \bar{d}[j] + r_1[j]$
2. $[\mathcal{U} \leftarrow \mathcal{S}]$	$\left\{(\bar{d}[i], \bar{\tau}[i])\right\}_{i \in [\kappa]}$
3. $[\mathcal{U}]$	For all $j \in [\kappa - 1]$, obtain $(d[j], \tau[j])$, $h_j \xleftarrow{\$} \mathbb{G}$ such that $\langle h_j \rangle = \mathbb{G}$
4. $[\mathcal{U} \rightarrow \mathcal{S}]$	$\{(u_j, v_j, G_j)\}_{j \in [\kappa - 1]}$
5. $[\mathcal{S}]$	$\forall j \in [\kappa - 1], h_j^* \leftarrow \left(v_j {u_j}^{-r_1[j]}\right)^{\frac{1}{r_0[j]}}$, check if $H_2(h_j^*) \stackrel{?}{\in} G_j$

Protocol 3. Adaptation of Π_{T2} to implementation

Discussion. Following reference [13], we use a matching threshold $T = 600$ (approximately 30% of the 2048-bit iris code) on the Hamming distance. However, since we are restricted to carrying a κ-th of an iris on an SHE ciphertext; we use a scaled threshold of T by κ, denoted by T_κ. As a result, we need to consider some practical issues as follows:

- We compute a partial HD value per ciphertext and then compare the resulting HD value with the scaled threshold T_κ. Thus, we set the scaled threshold $T_\kappa = 150$ for $\kappa = 4$. Consider that an iris is separately encrypted and sent in four small pieces. There are $\binom{604}{5} \approx 2^{40}$ ways that the sum of all HD values from two small pieces of encryptions is less than or equal to $T = 600$. During a session, a dishonest user is far from correctly manipulating all pieces of HD values at its disposal.
- Next, if we use $\mathbb{Z}_{2^t}$ as the plaintext space, then the FPR is $\left(\frac{T_\kappa}{2^t}\right)^\kappa$. For example, for the plaintext space $\mathbb{Z}_{2^{20}}$ and $\kappa = 4$, the FPR comes to $\left(\frac{150}{2^{15}}\right)^4 \approx 2^{-31}$.

6.2 Ciphertext Compression

Ghostshell requires that each time the user attempts the authentication process, it send a set of encrypted biometrics to the server. Biometrics are usually represented by a lengthy sequence of bits (e.g., 2048 bits for standard iris codes [41]). Thus, encrypting biometrics in a bit-by-bit manner leads to a long transmission delay. Two directions to avoid long delays are considered: first is to pack numerous plaintext bits into a ciphertext and second is to compress the ciphertext in a cryptographic manner. The former technique has been studied in Section 6.1; here, we only discuss how to cryptographically compress ciphertexts.

Because ciphertext sizes are designed to be very large to avoid lattice attacks, they can be a big burden in communications. Coron et al. [15] observed that the size of ciphertexts in their integer-based scheme can be reduced by introducing a pseudorandom number generator (PRG) to the encryption function. Further, the authors sketched an extension to Brakerski and Vaikuntanathan's scheme [8, 9].

We borrow their idea for the BGV-type scheme. Recall that, for a plaintext x, the BGV ciphertext defined on a polynomial ring $\mathbb{A}_q := \mathbb{Z}_q[X]/\langle \Phi_m(X) \rangle$ is of the form $(a(X), \langle a(X), s(X) \rangle +$

$x + 2e(X)$), where $a(X)$ is a random polynomial, $s(X)$ is a secret key, and $e(X)$ is a small error, and all components are in $\mathbb{A}_q$. Now, let F be a q-bit PRG taking as input a public random seed ω. We keep only ω rather than $a(X)$, where $H(\omega \parallel i)$ corresponds to each coefficient of X^i for all $i \in [\varphi(m) - 1]$ and a random oracle $H : \{0,1\}^* \to \mathbb{Z}_q$. As a result, the resulting ciphertext size is $\lceil \log q \rceil \varphi(m)$, while the original ciphertext size is $2\lceil \log q \rceil \varphi(m)$.

A side-effect of ciphertext compression is that we cannot perform homomorphic evaluations on compressed ciphertexts because the PRG F is not homomorphic. That is, homomorphically evaluating the ciphertexts requires to recover the original ciphertext.

7 Evaluation

7.1 Implementation

To validate the impact of our optimizations and to evaluate the practicability of our biometric authentication system, we implemented the prototype in the setting below.

Test environment. A prototype of Ghostshell was implemented in C using the NTL library [36] over GMP. In addition, our implementation utilized the HElib library for the BGV cryptosystem. Our code was compiled using g++ 4.9.2 on Ubuntu 14.04.2 LTS. We ran all timing experiments on a server with 56 2.60 GHz Intel Xenon E5-2697 processors and 264 GB RAM. Our implementation includes a symmetric cryptosystem and random oracles. We used AES-CBC and SHA1 for symmetric encryption and hash functions to instantiate the random oracles. For public confidence, we utilized the AES and SHA1 modules provided by OpenSSL.

Parameter selection. To instantiate the BGV scheme via `HElib`, we should first determine basic parameters such as the security parameter λ, the multiplicative depth L, and the plaintext space $\mathbb{Z}_M$. In our implementation, we set $\lambda = 80, L = 5$, and $M = 2^{15}$. Moreover, as stated in Section 6.1, we tested the four different plaintext slot numbers and corresponding degree of cyclotomic polynomial. Our candidate degree m and the number of plaintext slots N is as follows: $(N, m) = (330, 10261), (630, 8191), (682, 15709), (1285, 43691)$.

We should take care when choosing an extension degree t of plaintext space $\mathbb{Z}_M$. In general, it is preferable to use a large plaintext space, such $\mathbb{Z}_{2^{50}}$; however, this raises many performance issues, especially in SHE settings. Such an example is whereby if $t \geq 20$, then one homomorphic multiplication consumes two or more levels. Worse yet, performing automorphism operations is no longer achieved for free in the noise estimate; our choice is that $t = 15$.

As a side-effect, if the size of $\mathbb{Z}_M$ gets small, the MAC key (r_0, r_1) should be small in length. In our case, $\lceil \log r_0 \rceil = 7$. Then, the server is faced with a problem in ensuring the integrity of the decryption d because a user has a high FPR . To fix this problem, we require that the server sends a set of s tags $(\bar{\tau}_0, \ldots, \bar{\tau}_{s-1})$ of $\bar{d}$, where $\bar{\tau}_i = r_{i0} \cdot \bar{d} + r_{i1}$ for uniformly random values r_{i0}, r_{i1} of small size and for each $i \in [s]$. By Lemma 1, it suffices to choose s such that $\ell \leq s \cdot \lceil \log r_{i0} \rceil$.

Biometric specification. In our implementation, we used the input biometric of irises. Among the various biometric modalities, there is no universally best selection. However, the iris has been shown to be a superior biometric due to the relatively higher accuracy obtainable (e.g., see [27, 42]). We represented each iris code as a binary string of 300 bytes. Because any wavelet can be adjusted to the output length of the bit sequence, participants may choose a proper iris bit size n depending on the server's environment. In contrast to the iris matching algorithms found in commercial software, our matching procedure is performed only once. Although the matching process must compensate for misalignment errors resulting from small rotations, the total performance eventually depends on computing the Hamming distance, and this can be executed in parallel.

Table 4. Performances over 630 bits biometrics

Operations	CPU Time (msec)
Encryption	16.16
Decryption	163.72
Addition	0.05
Multiplication	14.32
Multiplication by constant	0.196

7.2 Micro Experiments

Test datasets. There are various public iris databases, such as CASIA-Iris [32], for use in research. CASIA-IrisV4 is composed of six subsets: CASIA-Iris-Interval, CASIA-Iris-Lamp, CASIA-Iris-Twins, CASIA-Iris-Distance, CASIA-Iris-Thousand, and CASIA-Iris-Syn. Among them, we used CASIA-Iris-Inverval, which consists of 2639 iris images from 249 subjects. We used a public MATLAB code [29] for extracting a binary iris template. The original code takes as input an image of the human eye and outputs a binary template of 9600 bits. We slightly modified the code to extract iris templates of 2400 bits.

Basic operations. As shown in Section 5.1, the simplicity of Ghostshell results from our exclusive use of SHE. This causes Ghostshell's performance to heavily depend on SHE's performance. Thus, our optimizations extensively use SIMD and automorphims to reduce the number of basic operations such as homomorphic addition and multiplication. In Table 4, we report on the running times for computing a ciphertext, deciphering the ciphertext, and adding and multiplying two ciphertexts for each N. Accordingly, one can predict that the total encryption time for an iris code is about four-times encryption time for each N. Indeed, this was confirmed by our experiments. We observed the same result for the other operations and parameters.

HD circuit evaluation. The heaviest computational operation is to evaluate the HD circuit. We measured the running time of computing the Hamming distance using the circuit optimized in Section 6.1.

According to parameter selection (§7.1), an SHE ciphertext carries N plaintext bits at a time. Hence, we only have to retain $\lceil 2400/N \rceil$ ciphertexts for a 2400-bit iris code. For example, if $N = 630$, the only four ciphertexts is used to represent 2400-bit iris code. For $N = 630$, the evaluation requires four multiplications and additions, together with at most ten additions and automorphisms. In our experiments, on average, 0.37 seconds were spent on computing an HD value between two encrypted iris codes.

We have tested for various N and corresponding m and the number of retaining ciphertexts is determined by N. Experiment results for these N and m selections are reported on Figure 5. Since each of the multiple ciphertexts corresponding to an iris code can be independently processed, this computation was executed using thread-level parallelism. However, this result is substantially worse than our expectations from performance measures reported in Table 4. The primary reason for the computation delay is that the HElib library spends much of this time in the noise-control mechanism each time homomorphic operations are performed.

Overall performance. In the Setup phase, the user (1) runs the key generation algorithm with the parameters (§7.1) and (2) sends the public parameters for running AES-CBC, SHA1, and our FKA scheme to the server. In this paragraph, we only describe in case of $N = 630$ because this parameter shows the best performance among chosen N. As a result, the size of SHE ciphertexts is $327,600$ bits with q of 40 bits, the block length of AES is 128 bits, and the number of SHE ciphertext per iris code $\kappa = 4$.

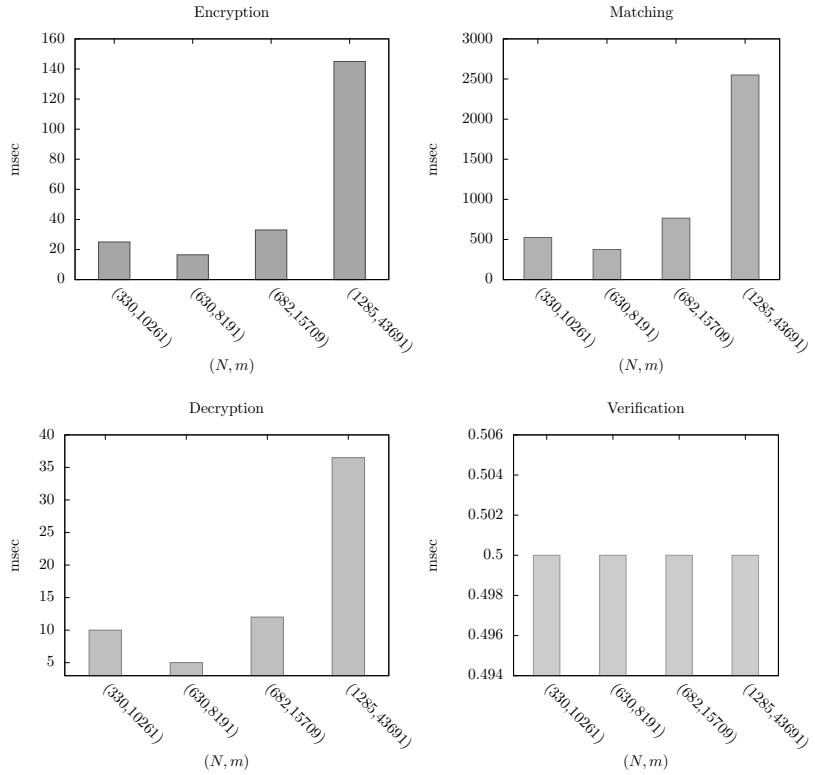

Fig. 5. Time evaluations for different selections of (N, m)

After the user takes an image and computes its feature vectors, it (1) encrypts the iris feature code $\mathbf{x}^*$ into $\bar{\mathbf{x}}^*$ and (2) sends it to the server. This computation is performed only once and would continue to be used until re-enrollment. The delay from encryption is approximately 65 milliseconds.

After receiving the user's authentication request with $\bar{\mathbf{x}}$, the server (1) computes the HD distance $\bar{d} = \mathsf{HD}(\bar{\mathbf{x}}, \bar{\mathbf{x}}^*)$, (2) generates a tag $\bar{\tau}$ using a pair of randomly chosen keys, and (3) sends the pair of resulting values.

Because the Hamming distance is divided into four chunks, $\bar{d}$ consists of four SHE ciphertexts, whose total size is 327600 bits and whose computation requires 0.37 seconds. Note that the server has to perform the homomorphic evaluations after decompression. With the local parameter $s = 10$ and $\bar{d} := (\bar{d}[0] \parallel \cdots \parallel \bar{d}[3])$, the server randomly chooses $(r_{i0}[j], r_{i1}[j]) \in (\mathbb{Z}_7)^2$ for each $i \in [9], j \in [3]$. Then, the server computes $\bar{\tau}_i[j] = r_{i0}[j] \cdot \bar{d}[j] + r_{i1}[j]$. The total computation time is approximately 0.01 seconds and the total bandwidth requirement is approximately 6.6 MB. The remaining computation time is significantly smaller than this computation time. Indeed, the total computation time on the user's side is approximately 5 milliseconds, and the server's computation time for the last verification is about 0.5 milliseconds.

Finally, we observed that a different selection of N and m does not have an effect on the performance of the verification step. The higher the degree m of the cyclotomic polynomial, the more computation required to perform on ciphertexts; however, since the verification step works after decryption, performance does not differentiate among different choices of (N, m).

8 Related Work

Possible Approaches. One solution for protecting biometric data during authentication is to only store them in a user's device and to use hardware-based security mechanisms, as in Fast Identity Online (FIDO) [1]. However, it yields another authentication problem between a user's device and a server, so not proper for some applications. Further, trusted execution environments (TEEs), such as ARM TrustZone [3], may not always be available and do not provide theoretical security guarantees, though they are not easy to compromise [28]. Personal computers, including desktops, notebooks, and basic mobiles, are all examples of devices widely used for online payments but that do not support TEE.

Alternatively, one may recall searching on encrypted data (a.k.a., searchable encryption) suggested by Song et al. [40]. However, this method is not satisfactory because we need to match two ciphertexts whose plaintexts slightly differ from each other. Some others may recommend using functional encryption [17] based on cryptographic multilinear maps, which allows us to evaluate a specific function for computing on encrypted data and output its result in the clear. This approach is conceptually simpler; however, we are currently unaware of any secure cryptographic multilinear maps and it is not clear whether we can obtain secure and efficient cryptographic multilinear maps in the near future.

Comparison with Previous Schemes. Recently, for the purpose of guaranteeing full privacy of biometric data, there have been studies using homomorphic encryption techniques for the secure computation of matching algorithms and on secure multiparty computation (SMC) techniques. SMC provides the same functionality with our approach based on SHE, but with interaction-intensive computations. Kerschbaum et al. [25] suggested an SMC-based protocol that is secure only when all participants are honest. Erkin et al. [16] proposed a protocol using Paillier's cryptosystem; however, their protocol also requires participants to be honest and is limited to using an eigen-face recognition algorithm.

There are some related works on securely computing Hamming distance using oblivious transfers (OTs) and garbled circuits (GCs), e.g., see [10, 12, 22, 23]. These results, however, restrict the scope of their works only to secure Hamming distance computation with storing biometrics in the clear. Moreover, their round complexity is in proportion to the number of running an OT protocol (i.e., $O(n)$ rounds for the input size n).

We can find more similar work in the privacy-preserving literature based on homomorphic encryption, more precisely, additive homomorphic encryption. Osadchy et al. [33] proposed a face identification protocol which is secure only when the participants are honest. They report that server's online computation requires about 0.3 second over 900-bit values; but 213 seconds are spent for offline computations. Blanton and Gasti [5] suggested an iris identification protocol in the semi-honest model. Their OT-based protocol also requires $O(n)$ interactions between the user and the server. Blanton and Aliasgari [4] proposed solutions for iris identification based on predicate encryption, but which is efficient only when the size of biometric templates is very small. Šeděnka et al.'s scheme [43] is also in this line.

Kulkarni and Namboodiri [26] presented an iris authentication scheme based on Boneh et al.'s SHE scheme [6]; however, the online execution time of the server is 58 seconds for 2048-bit irises. Their scheme requires only 3 rounds as our solution. More recently, Bringer et al. [11] showed that Oblivious RAM (ORAM) techniques can be used in an iris identification protocol; however, a service provider is given user's biometrics in the clear and then performs the identification process with a server. Thus their scheme should trust the service provider. Karabat et al. [24] proposed an authentication protocol solely using threshold homomorphic encryption; however, their scheme requires 6.1 seconds at the server side, 2.1 seconds at the user side, and 3 rounds as our solution.

9 Summary

The future of biometrics is quite promising not only in the financial services area, but in our social environment as well. The main drawback is that potential damages for privacy breaches are extensive compared to other authentication tools. In this work, we showed that those concerns can be significantly reduced by developing a secure authentication protocol. Our idea is to adapt SHE and OTM to ensure the integrity of homomorphically evaluated ciphertexts. To address the challenge of efficiency, we devised computation- and bandwidth-efficient SHE operations as well as applied an efficient OTM variant whose multiplicative depth is optimal. Our experimental results support our argument that Ghostshell works almost practically and is an evidence for practical applications of SHE.

References

1. FIDO alliance. https://fidoalliance.org. 8
2. http://www.reuters.com/article/2015/09/23. 1
3. ARM. Building a secure system using TrustZone technology. In *http://www.arm.com*, 2009. 8
4. M. Blanton and M. Aliasgari. Secure outsourced computation of iris matching. *Journal of Computer Security*, 2012. 8
5. M. Blanton and P. Gasti. Secure and efficient protocols for iris and fingerprint identification. In *ESORICS*, pages 190–209, 2011. 8
6. D. Boneh, E. Goh, and K. Nissim. Evaluating 2-DNF formulas on ciphertexts. In *Theory of Cryptography, Second Theory of Cryptography Conference, TCC 2005, Cambridge, MA, USA, February 10-12, 2005, Proceedings*, pages 325–341, 2005. 8
7. Z. Brakerski, C. Gentry, and V. Vaikuntanathan. (Leveled) fully homomorphic encryption without bootstrapping. In *ITCS*, pages 309–325, 2012. 1, 4.1
8. Z. Brakerski and V. Vaikuntanathan. Efficient fully homomorphic encryption from (standard) LWE. In *FOCS*, pages 97–106, 2011. 6.2
9. Z. Brakerski and V. Vaikuntanathan. Fully homomorphic encryption from ring-LWE and security for key dependent messages. In *Advances in Cryptology–Crypto*, pages 505–524, 2011. 4.1, 6.2
10. J. Bringer, H. Chabanne, M. Favre, A. Patey, T. Schneider, and M. Zohner. GSHADE: faster privacy-preserving distance computation and biometric identification. In *IH & MMSec*, pages 187–198, 2014. 8
11. J. Bringer, H. Chabanne, and A. Patey. Practical identification with encrypted biometric data using oblivious RAM. In *ICB*, pages 1–8, 2013. 8
12. J. Bringer, H. Chabanne, and A. Patey. SHADE: Secure HAmming DistancE computation from oblivious transfer. In *FC*, pages 164–176, 2013. 8
13. J. Bringer, M. Favre, H. Chabanne, and A. Patey. Faster secure computation for biometric identification using filtering. In *ICB*, pages 257–264, 2012. 6.1
14. J. Coron, A. Mandal, D. Naccache, and M. Tibouchi. Fully homomorphic encryption over the integers with shorter public keys. In *Advances in Cryptology–Crypto*, pages 487–504, 2011. 4.1
15. J. Coron, D. Naccache, and M. Tibouchi. Public key compression and modulus switching for fully homomorphic encryption over the integers. In *Advances in Cryptology–Eurocrypt*, pages 446–464, 2012. 1, 6.2
16. Z. Erkin, M. Franz, J. Guajardo, S. Katzenbeisser, I. Lagendijk, and T. Toft. Privacy-preserving face recognition. In *PETS*, pages 235–253, 2009. 8
17. S. Garg, C. Gentry, S. Halevi, M. Raykova, A. Sahai, and B. Waters. Candidate indistinguishability obfuscation and functional encryption for all circuits. In *FOCS*, pages 40–49, 2013. 8
18. C. Gentry. Fully homomorphic encryption using ideal lattices. In *STOC*, pages 169–178, 2009. 1, 4.1
19. C. Gentry and S. Halevi. Implementing Gentry's fully-homomorphic encryption scheme. In *Advances in Cryptology–Eurocrypt*, pages 129–148, 2011. 4.1
20. O. Goldreich. *The foundations of cryptography: Volume 2–Basic Applications*. Cambridge University Press, 2004. 4.3
21. S. Halevi and V. Shoup. `HElib`-An implementation of homomorphic encryption. 4.1, 6.1
22. Y. Huang, D. Evans, J. Katz, and L. Malka. Faster secure two-party computation using garbled circuits. In *USENIX Security Symposium*, 2011. 8
23. A. Jarrous and B. Pinkas. Secure Hamming distance based computation and its applications. In *ACNS*, pages 107–124, 2009. 8
24. C. Karabat, M. Kiraz, H. Erdogan, and E. Savas. THRIVE: threshold homomorphic encryption based secure and privacy preserving biometric verification system. *EURASIP J. Adv. Sig. Proc.*, 2015(71), 2015. 8

25. F. Kerschbaum, M. Atallah, D. M'Raïhi, and J. Rice. Private fingerprint verification without local storage. In *ICBA*, pages 387–394, 2004. 8
26. R. Kulkarni and A. Namboodiri. Secure Hamming distance based biometric authentication. In *ICB*, pages 1–6, 2013. 8
27. T. Mansfield. Biometric authentication in the real world. http://www.npl.co.uk/biometrics. 7.1
28. C. Marforio, N. Karapanos, C. Soriente, K. Kostiainen, and S. Capkun. Secure enrollment and practical migration for mobile trusted execution environments. In *SPSM*, pages 93–98, 2013. 8
29. L. Masek and P. Kovesi. MATLAB source code for a biometric identification system based on iris patterns. *The School of Computer Science and Software Engineering, The University of Western Australia. 2003.*, 2003. 7.2
30. A. Menezes, P. van Oorschot, and S. Vanstone. *Handbook of Applied Cryptography*. CRC Press, 1996. 4.3
31. NIST. SHA-3 standard: Permutation-based hash and extendable-output functions. In *FIPS 202*, 2015. 1
32. C. A. of Science-Institute of Automation. CASIA iris database. http://biometrics.idealtest.org. 7.2
33. M. Osadchy, B. Pinkas, A. Jarrous, and B. Moskovich. SCiFI: A system for secure face identification. In *IEEE Symposium on Security and Privacy*, pages 239–254, 2010. 8
34. N. Ratha, J. Connell, and R. Bolle. Enhancing security and privacy in biometrics-based authentication systems. *IBM Systems Journal*, 40(3):614–634, 2001. 2.2
35. R. L. Rivest, A. Shamir, and L. Adleman. A method for obtaining digital signatures and public-key cryptosystems. *Commun. ACM*, 21(2):120–126, 1978. 1
36. V. Shoup. `NTL`: A library for doing number theory. 7.1
37. G. Simmons. Authentication theory/coding theory. In *Advances in Cryptology–Crypto*, pages 411–431, 1984. 1, 4.2
38. N. Smart and F. Vercauteren. Fully homomorphic SIMD operations. *Cryptology ePrint Archive*, 133, 2011. 1, 6.1
39. N. Smart and F. Vercauteren. Fully homomorphic SIMD operations. *Designs, Codes and Cryptography*, 71(1):57–81, 2014. 6.1
40. D. X. Song, D. Wagner, and A. Perrig. Practical techniques for searches on encrypted data. In *IEEE Symposium on Security and Privacy*, pages 44–55, 2000. 8
41. E. Tabassi, P. Grother, and W. Salamon. *IREX II-IQCE Iris quality calibration and evaluation performance of iris image quality assessment algorithms*. NIST Interagency Report 7820. 2011. 6.2
42. S. Thavalengal, P. Bigioi, and P. Corcoran. Iris authentication in handheld devices - considerations for constraint-free acquisition. *IEEE Trans. Consumer Electronics*, 61(2):245–253, 2015. 7.1
43. J. Šeděnka, S. Govindarajan, P. Gasti, and K. Balagani. Secure outsourced biometric authentication with performance evaluation on smartphones. *IEEE Transactions on Information Forensics and Security*, 10(2):384–396, 2015. 8