Secure and Efficient Protocols for Iris and Fingerprint Identification

Marina Blanton
Department of Computer Science and Engineering
University of Notre Dame
mblanton@cse.nd.edu

Paolo Gasti
Department of Information and Computer Science
University of California, Irvine
pgasti@ics.uci.edu

Abstract

Recent advances in biometric recognition and the increasing use of biometric data prompt significant privacy challenges associated with the possible misuse, loss or theft, of biometric data. Biometric matching is often performed by two mutually suspicious parties, one of which holds one biometric image while the other owns a possibly large biometric collection. Due to privacy and liability considerations, neither party is willing to share its data. This gives rise to the need to develop secure computation techniques over biometric data where no information is revealed to the parties except the outcome of the comparison or search. To address the problem, in this work we develop and implement the first privacy-preserving identification protocol for iris codes. We also design and implement a secure protocol for fingerprint identification based on FingerCodes with a substantial improvement in the performance compared to existing solutions. We show that new techniques and optimizations employed in this work allow us to achieve particularly efficient protocols suitable for large data sets and obtain notable performance gain compared to the state-of-the-art prior work.

1 Introduction

Recent advances in biometric recognition make the use of biometric data more prevalent for authentication and other purposes. Today large-scale collections of biometric data include face, fingerprint, and iris images collected by the US Department of Homeland Security (DHS) from visitors through its US-VISIT program [24], iris images collected by the United Arab Emirates (UAE) Ministry of Interior from all foreigners and also fingerprints and photographs from certain types of travelers [26], and several others. While biometry serves as an excellent mechanism for authentication and identification of individuals, such data is undeniably extremely sensitive and must be well protected. Furthermore, once leaked biometric data cannot be revoked or replaced. For these reasons, biometric data cannot be easily shared between organizations or agencies. However, there could be legitimate reasons to carry out computations on biometric data belonging to different entities. For example, a non-government agency may need to know whether a biometric it possesses appears on the government watch-list. In this case the agency would like to maintain the privacy of the individual if no matches are found, and the government also does not want to release its database to third parties.

The above requires carrying out computation over biometric data in a way that keeps the data private and reveals only the outcome of the computation. In particular, we study the problem of *biometric identification*, where a client C is in a possession of a biometric X and a server S possesses a biometric database D. The client would like to know whether X appears in the database D by comparing its biometric to the records in D. The computation amounts to comparing X to each $Y \in D$ in a privacy-preserving manner. This formulation is general enough to apply to a number of other scenarios, ranging from a single comparison of X and Y to the case where two parties need to compute the intersection of their respective databases. We assume that the result of comparing biometrics X and Y is a bit, and no additional information about X or Y should be learned by the parties as a result of secure computation. With our secure protocols, the outcome can be made available to either party or both of them; for concreteness in our description, we have the client learn the outcome of each comparison.

In this work we assume that both the client's and the server's biometric images have been processed and have representations suitable for biometric matching, i.e., each raw biometric image has been processed by a feature extraction algorithm. For the types of biometric considered in this work, this can be performed for each image independently and we do not discuss this further.

Algorithms for privacy-preserving two-party face recognition have recently appeared in the literature [16, 41, 37], and therefore this work concentrates on iris and fingerprint identification, which are popular types of biometric data with good distinguishing capability. We design and implement secure and efficient two-party protocols for iris identification and two types of fingerprint identification: matching based on FingerCodes [28] which are particularly well suited for privacy-preserving computation and traditional minutiae-based matching.

Prior work. Literature on secure multi-party computation is extensive. Starting from the seminal work on garbled circuit evaluation [44, 20], it has been known that any function can be securely evaluated by representing it as a boolean circuit. Similar results are also known for securely evaluating any function using secret sharing techniques (e.g., [40]) or homomorphic encryption (e.g., [10]). In the last several years a number of tools have been developed for automatically creating a secure protocol from a function description written in a high-level language. Examples include Fairplay [34], VIFF [13], TASTY [21], and others. It is, however, well-known that custom optimized protocols are often constructed for specific applications due to the inefficiency of generation solution. Such custom solutions are known for a wide range of application (e.g., set operations [18, 30, 19, 15], DNA matching [42], k-means clustering [8], etc.), and this work focuses on secure biometric identification using iris codes and fingerprints. Furthermore, some of the optimizations employed in this work can find their uses in protocol design for other applications, as well as general compilers and tools such as TASTY [21].

With the growing prevalence of applications that use biometrics, the need for secure biometric identification was recognized in the research community. A number of recent publications address the problem of privacy-preserving face recognition [16, 41, 37]. This problem was first treated by Erkin et al. [16], where the authors designed a privacy-preserving face recognition protocol based on the Eigenfaces algorithm. The performance of that solution was consequently improved by Sadeghi et al. [41]. More recently, Osadchy et al. [37] designed a new face recognition algorithm together with its privacy-preserving realization called SCiFI. The design targeted to simultaneously address robustness to different viewing conditions and efficiency when used for secure computation. As a result, SCiFI is currently recognized as the best face identification algorithm with efficient privacy-preserving realization. SCiFI takes 0.31 sec (during the online phase) [37] to compare two biometrics, and therefore would take about 99 sec to compare a biometric to a database of 320 images.

Another very recent work by Barni et al. [3] designs a privacy-preserving protocol for fingerprint identification using FingerCodes [28]. FingerCodes use texture information from a fingerprint to compare two biometrics. The algorithm is not as discriminative as fingerprint matching techniques based on location of minutiae points, but it was chosen by the authors as particularly suited for efficient realization in the privacy-preserving framework. As of the time of this writing, similar results for other types of biometrics or other fingerprint matching techniques are not available in the literature. We narrow this gap by providing a secure two-party protocol for widely used iris identification, as well as address fingerprint identification. Our protocols follow the standard algorithms for comparing two biometrics, yet they are very efficient and outperform the state-of-the-art protocols with a notable reduction in the overhead.

Bringer et al. [7] describe a biometric-based authentication mechanism with privacy protection of biometric, where the Hamming distance is used as the distance metric. The authentication server is composed of three entities that must not collude, and one of them, the matcher, learns the computed Hamming distance. In our work, however, no information beyond the outcome of the comparison is revealed, the computation itself is more complex and corresponds to the actual algorithm used for iris code comparisons, and there is no need for additional or third-party entities. Barbosa et al. [2] extend the framework with a classifier to improve authentication accuracy and propose an instantiation based on Support Vector Machine using homomorphic encryption.

Our contributions. In this work we treat the problem of privacy preserving biometric identification. We develop new secure protocols for two types of biometric, iris and fingerprints, and achieve security against semi-honest adversaries. While iris codes are normally represented as binary strings and use very similar matching algorithms, there is a variety of representations and comparison algorithms for fingerprints. For that reason, we study two types of matching algorithms for fingerprints: (i) FingerCodes that use fixed-size representations and a simple comparison algorithm and (ii) a traditional and most widely used method for pairing minutia points in one fingerprint with minutiae in another fingerprint. Our protocols were designed with efficiency in mind to permit their use on relatively large databases, and possibly in real time. While direct performance comparison of our protocols and the results available in the literature is possible only in the case of FingerCode, we can use complexity of the computation to draw certain conclusions. The results we achieve in this work are as follows:

1. Our secure FingerCode protocol is extremely fast and allows the parties to compare two fingerprints X and Y using a small fraction of a second. For a database of 320 elements, the online computation can be carried out in 0.45 sec with the communication of 277KB. This is an over 30-fold improvement in both communication and computation over the privacy-preserving solution of [3], as detailed in Section 5, and a significant improvement over an adaptation of [41] to this context.

2. Iris codes use significantly longer representations (thousands of bits) and require more complex transformation of the data. Despite the length and complexity, our solution allows two iris codes to be compared in 0.15 sec. With respect to the state-of-the-art face recognition protocol SCiFI, which also relies on Hamming distance computation, our protocol achieves lower overhead despite the fact that the computation involves an order of magnitude larger number of more complex operations.

3. Finally, we develop a secure protocol for traditional fingerprint matching based on pairing of minutiae points in one biometric to minutiae points within the tolerance in another biometric. This computation exhibits the largest complexity, but our secure protocol still allows us to compare two fingerprints in about one second.

2 Description of Computation

In what follows, we assume that client C holds a single biometric X and server S holds a database of biometrics D. The goal is to learn whether C's biometric appears in S's database without learning any additional information. This is accomplished by comparing X to each biometric $Y \in D$, and as a result of each comparison C learns a bit that indicates whether the comparison resulted in a match.

2.1 Iris

Let an iris biometric X be represented as an m-bit binary string. We use X_i to denote i-th bit of X. In iris-based recognition, after feature extraction, biometric matching is normally performed by computing the Hamming distance between two biometric representations. Furthermore, the feature extraction process is such that some bits of the extracted string X are unreliable and are ignored in the matching process. Information about such bits is stored in an additional m-bit string, called *mask*, where its i-th bit is set to 1 if the i-th bit of X should be used in the matching process and is set to 0 otherwise. For biometric X, we use $M(X)$ to denote the mask associated with X. Often, a predetermined number of bits (e.g., 25% in [23] and 35% in [4]) is considered unreliable in each biometric template. Thus, to compare two biometric representations X and Y, their Hamming distance takes into account the respective masks. That is, if the Hamming distance between two iris codes without masks is computed as:

$$HD(X,Y) = \frac{(||X \oplus Y||)}{m} = \frac{\sum_{i=1}^{m}(X_i \oplus Y_i)}{m}$$

the computation of the Hamming distance that uses masks becomes [14]:

$$HD(X, M(X), Y, M(Y)) = \frac{||(X \oplus Y) \cap M(X) \cap M(Y)||}{||M(X) \cap M(Y)||} \tag{1}$$

In other words, we have

$$HD(X, M(X), Y, M(Y)) = \frac{\sum_{i=1}^{m}((X_i \oplus Y_i) \wedge M(X_i) \wedge M(Y_i))}{\sum_{i=1}^{m}(M(X_i) \wedge M(Y_i))}.$$

Throughout this work, we assume that the latter formula is used and simplify the notation to $HD(X,Y)$. Then the computed Hamming distance is compared with a specific threshold T, and the biometrics X and Y are considered to be a match if the distance is below the threshold, and a mismatch otherwise. The threshold T is chosen based on the distributions of authentic and impostor data. (In the likely case of overlap of the two distributions, the threshold is set to achieve the desired levels of false accept and false reject rates based on the security goals.)

Two iris representations can be slightly misaligned. This problem is caused by head tilt during image acquisition. To account for this, the matching process attempts to compensate for the error and rotates the biometric representation by a fixed amount to determine the lowest distance. Each biometric is represented as a two-dimensional array, therefore a circular shift is applied to each row by shifting its representation by a small fixed number of times, which we denote by c. The minimum Hamming distance across all runs is then compared to the threshold. That is, if we let $\mathsf{LS}^j(\cdot)$ (resp., $\mathsf{RS}^j(\cdot)$) denote a circular left (resp., right) shift of the argument by a fixed number of bits (2 bits in experiments conducted by the biometrics group at our institution, where application of the Gabor filter during feature extraction results in a complex number, which is quantized into a 2-bit value), the matching process becomes:

$$\min(HD(X, \mathsf{LS}^c(Y)), \ldots, HD(X, \mathsf{LS}^1(Y)), HD(X, Y), \\ HD(X, \mathsf{RS}^1(Y)), \ldots, HD(X, \mathsf{RS}^c(Y))) \overset{?}{<} T \tag{2}$$

Throughout this work we assume that the algorithms for comparing two biometrics are public, as well as any constant thresholds T. Our protocols, however, maintain their security and performance guarantees if the (fixed) thresholds are known only to the server who owns the database.

2.2 Fingerprints

Work on fingerprint identification dates many years back with a number of different approaches currently available (see, e.g., [35] for an overview). The most popular and widely used techniques extract information about minutiae from a fingerprint and store that information as a set of points in the two-dimensional plane. Fingerprint matching in this case consists of finding a matching between two sets of points so that the number of minutiae pairings is maximized. In more detail, a biometric X is represented as a set of m_X points $X = \langle (x_1, y_1, \alpha_1), \ldots, (x_{m_X}, y_{m_X}, \alpha_{m_X}) \rangle$. A minutia $X_i = (x_i, y_i, \alpha_i)$ in X and minutia $Y_j = (x'_j, y'_j, \alpha'_j)$ in Y are considered matching if the spatial (i.e., Euclidean) distance between them is all smaller than a given threshold d_0 and the directional difference between them is smaller than a given threshold α_0, computed as:

$$\sqrt{(x'_j - x_i)^2 + (y'_j - y_i)^2} < d_0 \quad \text{and} \quad \min(|\alpha'_j - \alpha_i|, 360^\circ - |\alpha'_j - \alpha_i|) < \alpha_0. \tag{3}$$

These tolerance values are necessary to account for errors introduced by feature extraction algorithms (e.g., quantizing) and small skin distortions. Two points within a single fingerprint are also assumed to lie within at least distance d_0 of each other.

Before two fingerprints can be compared, they need to be pre-aligned, which maximizes the number of matching minutiae. The literature distinguishes two types of alignment: absolute and relative. With absolute alignment, each fingerprint is pre-aligned independently using core point of other information. With relative alignment, information contained in two biometrics is used to guide their alignment relative to each other. While relative pre-alignment can be more accurate that absolute pre-alignment, such techniques are not feasible to implement in a privacy-preserving protocol, and we assume that absolute pre-alignment is used. To increase the accuracy of the matching process, a single fingerprint can be stored using a small number of templates with slightly different alignment, and the result of the comparison is a match if at least one of them matches the biometric being queried. A more detailed treatment of this problem is outside of the scope of this work.

A simple way used for determining a pairing between minutiae of fingerprints X and Y consists of pairing a minutia X_i with the closest minutia Y_j in Y. Let $mm(X_i, Y_j)$ denote minutiae matching predicate in equation 3. Then the pairing function $P(\cdot)$ that determines the mapping of minutiae in X and Y can be defined as follows: for $i = 1, \ldots, m_X$, $P(i) = j$ if Y_j is the closest to X_i among all $Y_k \in Y$ such that $mm(X_i, Y_k) = 1$, and $P(i) = \perp$ if no such Y_j exists. Because each minutia Y_j can be paired with at most one minutia from X, the above algorithm needs to mark minutiae in Y to enforce this constraint.

The above approach will not find the optimum assignment (i.e., the one that maximizes the number of mates) when a minutia X_i should be paired with another minutia Y_j which is not the closest to X_i. The optimum pairing can be achieved by formulating the problem as an instance of minimum-cost maximum flow, where fingerprints X and Y are used to create a flow network.

Then this problem can be solve using one of the known algorithms such as Ford-Fulkerson [17] and others. In particular, [29, 43] use a flow network representation of minutia pairing problem to find an optimal pairing, where there is an edge from a node corresponding to minutia $X_i \in X$ to $Y_j \in Y$ iff $mm(X_i, Y_j) = 1$. We refer the reader to [29, 43] for additional detail. For fingerprints consisting of m minutiae, the optimal pairing can be found in $O(m^2)$ time using Ford-Fulkerson algorithm because each minutia from X is connected to at most a constant number of minutiae from Y. In a privacy-preserving setting, however, when information about connections between minutiae in X and Y (and thus the structure of the graph) must remain private, the complexity of this approach based on Ford-Fulkenson algorithm increases to $O(m^3)$.[1] Such a solution would not be of a practical importance even for modest values of m. This calls for alternative algorithms, and we implement the pairing approach based on the minimum distance outlined above. The algorithm is not guaranteed to find the optimal pairing, but is feasible for privacy-preserving computation.

For the purposes of this work, we assume that during fingerprint identification the number of minutiae in a pairing is compared to a fixed threshold T. If in specific fingerprint matching algorithms this threshold is not constant, but rather is a function of the biometrics X and Y being compared (e.g., a number of points in each set), our solution can be easily extended to accommodate those variations as well.

Fingerprint matching can also be performed using different type of information extracted from a fingerprint image. One example is FingerCode [28] which uses texture information from a fingerprint scan to form fingerprint representation X. While FingerCodes are not as distinctive as minutiae-based representations and are best suited for use in combination with minutiae to improve the overall matching accuracy [35], FingerCode-based identification can be implemented very efficiently in a privacy-preserving protocol. In particular, each FingerCode consists of a fixed number m elements of ℓ bits each. Then FingerCodes $X = (x_1, \ldots, x_m)$ and $Y = (y_1, \ldots, y_m)$ are considered a match if the Euclidean distance between their elements is below the threshold T:

$$\sqrt{\sum_{i=1}^{m} (x_i - y_i)^2} \stackrel{?}{<} T \tag{4}$$

Barni et al. [3] was the first to provide a privacy-preserving protocol for FingerCode-based biometric identification. We first show that the techniques employed in this work improve both computation and communication of the protocol of [3] by a large factor. We then proceed with providing a secure protocol for superior (but less efficient) identification algorithm for minutia-based matching.

3 Preliminaries

Security model. We use the standard security model for secure two-party computation in presence of semi-honest participants (also known as honest-but-curious or passive). In particular, it means that the parties follow the prescribed behavior, but might try to compute additional information from the information obtained during protocol execution. Security in this setting is

[1]The complexity of Ford-Fulkerson algorithm in a regular setting is $O(Ef)$, where E is the number of edges in the graph, which for this application would be $O(m)$, and f is maximum flow in the graph which is also $O(m)$. It consists of repeatedly searching the graph for a new path (e.g., using depth-first search) which will allow the current flow to be increased by 1, resulting in $O(Ef)$ complexity. When the structure of the graph (which corresponds to the matching information) is to be protected, the graph needs to be padded with dummy edges that do not change the outcome of the algorithm. In that case the degree of each node in the graph becomes $O(m)$, and the the total number of edges increases to $O(m^2)$. Searching such a graph for a new path without disclosing any information about true and available edges, all paths of length 5 must be considered. For a bipartite graph used in this setting, this can be accomplished in $O(m^2)$ time, which means that constructing the entire flow after $O(m)$ searches will take $O(m^3)$ time.

defined using simulation argument: the protocol is secure if the view of protocol execution for each party is computationally indistinguishable from the view simulated using that party's input and output only. This means that the protocol execution does not reveal any additional information to the participants. The definition below formalizes the notion of security for semi-honest participants:

Definition 1 *Let parties P_1 and P_2 engage in a protocol π that computes function $f(\mathsf{in}_1, \mathsf{in}_2) = (\mathsf{out}_1, \mathsf{out}_2)$, where in_i and out_i denote input and output of party P_i, respectively. Let $\mathrm{VIEW}_\pi(P_i)$ denote the view of participant P_i during the execution of protocol π. More precisely, P_i's view is formed by its input, internal random coin tosses r_i, and messages $m_1, \ldots, m_t$ passed between the parties during protocol execution:*

$$\mathrm{VIEW}_\pi(P_i) = (\mathsf{in}_i, r_i, m_1, \ldots, m_t).$$

We say that protocol π is secure against semi-honest adversaries if for each party P_i there exists a probabilistic polynomial time simulator S_i such that

$$\{S_i(\mathsf{in}_i, f(\mathsf{in}_1, \mathsf{in}_2))\} \equiv \{\mathrm{VIEW}_\pi(P_i), \mathsf{out}_i\},$$

where "$\equiv$" denotes computational indistinguishability.

Homomorphic encryption. Our constructions use a semantically secure additively homomorphic encryption scheme. In an additively homomorphic encryption scheme, $\mathsf{Enc}(m_1) \cdot \mathsf{Enc}(m_2) = \mathsf{Enc}(m_1 + m_2)$ which also implies that $\mathsf{Enc}(m)^a = \mathsf{Enc}(a \cdot m)$. While any encryption scheme with the above properties (such as the well known Paillier encryption scheme [38]) suffices for the purposes of this work, the construction due to Damgård et al. [12, 11] (DGK) is of particular interest here.

We also note that in Paillier encryption scheme, a public key consists of a k-bit RSA modulus $N = pq$, where p and q are prime, and an element g whose order is a multiple of N in $\mathbb{Z}^*_{N^2}$. Given a message $m \in \mathbb{Z}_N$, encryption is performed as $\mathsf{Enc}(m) = g^m r^n \bmod N^2$, where $r \stackrel{R}{\leftarrow} \mathbb{Z}_N$ and notation $a \stackrel{R}{\leftarrow} A$ means that a is chosen uniformly at random from the set A. In DGK encryption scheme [12, 11], which was designed to work with small plaintext spaces and has shorter ciphertext size than other randomized encryption schemes, a public key consists of (i) a (small, possibly prime) integer u that defines the plaintext space, (ii) k-bit RSA modulus $N = pq$ such that p and q are $k/2$-bit primes, v_p and v_q are t-bit primes, and $uv_p|(p-1)$ and $uv_q|(q-1)$, and (iii) elements $g, h \in \mathbb{Z}^*_N$ such that g has order uv_pv_q and h has order v_pv_q. Given a message $m \in \mathbb{Z}_u$, encryption is performed as $\mathsf{Enc}(m) = g^m h^r \bmod N$, where $r \stackrel{R}{\leftarrow} \{0,1\}^{2.5t}$. We refer the reader to the original publications [38] and [12, 11], respectively, for any additional information.

Garbled circuit evaluation. Originated in Yao's work [44], garbled circuit evaluation allows two parties to securely evaluate any function represented as a boolean circuit. The basic idea is that, given a circuit composed of gates, one party P_1 creates a garbled circuit by assigning to each wire two randomly chosen keys. P_1 also encodes gate information in a way that given keys corresponding to the input wires (encoding specific inputs), the key corresponding to the output of the gate on those inputs can be recovered. The second party, P_2, evaluates the circuit using keys corresponding to inputs of both P_1 and P_2 (without learning anything in the process). At the end, the result of the computation can be recovered by linking the output keys to the bits which they encode.

Recent literature provides optimizations that reduce computation and communication overhead associated with circuit construction and evaluation. Kolesnikov and Schneider [32] describe an optimization that permits XOR gates to be evaluated for free, i.e., there is no communication overhead associated with such gates and their evaluation does no involve cryptographic functions.

This optimization is possible when the hash function used for creating garbled gates can be assumed to be correlation robust (see [33, 32] for more detail). Under the same assumptions, Pinkas et al. [39] additionally give a mechanism for reducing communication complexity of binary gates by 25%: now each gate can be specified by encoding only three outcomes of the gate instead of all four. Finally, Kolesnikov et al. [31] improve the complexity of certain commonly used operations such as addition, multiplication, comparison, etc. by reducing the number of non-XOR gates: adding two n-bit integers requires $5n$ gates, n of which are non-XOR gates; comparing two n-bit integers requires $4n$ gates, n of which are non-XOR gates; and computing the minimum of t n-bit integers (without the location of the minimum value) requires $7n(t-1)$ gates, $2n(t-1)$ of which are non-XOR gates.

With the above techniques, evaluating a non-XOR gates involves one invocation of the hash function (which is assumed to be correlation robust). During garbled circuit evaluation, P_2 directly obtains keys corresponding to P_1's inputs from P_1 and engages in the oblivious transfer (OT) protocol to obtain keys corresponding to P_2's inputs.

Oblivious Transfer. In 1-out-of-2 Oblivious Transfer, OT_1^2, one party, the sender, has as its input two strings m_0, m_1 and another party, the receiver, has as its input a bit b. At the end of the protocol, the receiver learns m_b and the sender learns nothing. Similarly, in 1-out-of-N OT the receiver obtains one of the N strings held by the sender. There is a rich body of research literature on OT, and in this work we use its efficient implementation from [36] as well as techniques from [27] that reduce a large number of OT protocol executions to κ of them, where κ is the security parameter. This, in particular, means that obtaining the keys corresponding to P_2's inputs in garbled circuit evaluation by P_2 incurs only small overhead.

4 Secure Iris Identification

4.1 Structural Optimization of the Computation

As indicated in equation 1, computing the distance between two iris codes involves performing the division operation. While techniques for carrying out this operation using secure multi-party computation are known (see, e.g., [1, 8, 6, 9]), their performance in practice even using very recent results is far from satisfactory for this application As an example, [5] reports that two-party evaluation of garbled circuits produced by Fairplay takes several seconds for numbers of length 24–28 bits, but circuits for longer integers could not be constructed due to the rapidly increasing memory requirements of Fairplay. [22] reports that building a multi-party division protocol using homomorphic encryption alone requires on the order of an hour to carry out the operation for 32-bit integers. Fortunately, in our case the computation can be rewritten to completely avoid this operation and replace it with multiplication. That is, using the notation

$$HD(X,Y) = ||(X \oplus Y) \cap M(X) \cap M(Y)|| \,/\, ||M(X) \cap M(Y)|| = D(X,Y) \,/\, M(X,Y),$$

instead of testing whether $HD(X,Y) \stackrel{?}{<} T$, we can test whether $D(X,Y) \stackrel{?}{<} T \cdot M(X,Y)$. While the computation of the minimum distance as used in equation 2 is no longer possible, we can replace it with equivalent computation that does not increase its cost. Now the computation becomes:

$$D(X, \mathsf{LS}^c(Y)) \stackrel{?}{<} T \cdot M(X, \mathsf{LS}^c(Y)) \vee \cdots \vee D(X, \mathsf{RS}^c(Y)) \stackrel{?}{<} T \cdot M(X, \mathsf{RS}^c(Y)) \qquad (5)$$

When this computation is carried over real numbers, T lies in the range [0, 1]. In our case, we need to carry the computation over the integers, which means that we "scale up" all values with the desired level of precision. That is, by using ℓ bits to achieve desired precision, we multiply $D(X, Y)$ by 2^ℓ and let T range between 0 and 2^ℓ. Now $2^\ell D(X, Y)$ and $T \cdot M(X, Y)$ can be represented using $\lceil \log m \rceil + \ell$ bits.

4.2 Base Protocol

In what follows, we first describe the protocol in its simplest form. Section 4.3 presents optimizations and the resulting performance of the protocol.

In our solution, the client C generates a public-private key pair (pk, sk) for a homomorphic encryption scheme and distributes the public key pk. This is a one-time setup cost for the client for all possible invocations of this protocol with any number of servers. During the protocol itself, the secure computation proceeds as specified in equation 5. In the beginning, C sends its inputs encrypted with pk to the server S. At the server side, the computation first proceeds using homomorphic encryption, but later the client and the server convert the intermediate result into a split form and finish the computation using garbled circuit evaluation. This is due to the fact that secure two-party computation of the comparison is the fastest using garbled circuit evaluation [31], but the rest of the computation in our case is best performed on encrypted values.

To compute $D(X, Y) = \sum_{i=1}^{m}(X_i \oplus Y_i) \wedge M(X_i) \wedge M(Y_i)$ using algebraic computation, we use $X_i \oplus Y_i = X_i(1 - Y_i) + (1 - X_i)Y_i$ and obtain:

$$D(X, Y) = \sum_{i=1}^{m}(X_i(1 - Y_i) + (1 - X_i)Y_i)M(X_i)M(Y_i).$$

$M(X, Y)$ is computed as $\sum_{i=1}^{m} M(X_i)M(Y_i)$. Then if S obtains encryptions of $X_iM(X_i)$, $(1 - X_i)M(X_i)$, and $M(X_i)$ for each i from C, the server will be able to compute $D(X, Y)$ and $M(X, Y)$ using its knowledge of the Y_i's and the homomorphic properties of the encryption. Figure 1 describes the protocol, in which after receiving C's encrypted values S produces $\mathsf{Enc}(M(X_i))$'s and proceeds to compute $D(X, Y^j)$ and $M(X, Y^j)$ in parallel for each Y in its database, where Y^j denotes biometric Y shifted by j positions and j ranges from $-c$ to c. At the end of steps 3(a).i and 3(a).ii the server obtains $\mathsf{Enc}(2^\ell D(X, Y^j) + r_S^j)$ for a randomly chosen r_S^j of its choice, and at the end of step 3(a).iii S obtains $\mathsf{Enc}(T \cdot M(X, Y^j) + t_S^j)$ for a random t_S^j of its choice. The server sends these values to the client who decrypts them. Therefore, at the end of step 3(a) C holds $r_C^j = 2^\ell D(X, Y^j) + r_S^j$ and $t_C^j = T \cdot M(X, Y^j) + t_S^j$ and S holds $-r_S^j$ and $-t_C^j$, i.e., they additively share $2^\ell D(X, Y^j)$ and $T \cdot M(X, Y^j)$.

What remains to compute is $2c+1$ comparisons (one per each Y^j) followed by $2c$ OR operations as specified by equation 5. This is accomplished using garbled circuit evaluation, where C enters r_C^j's and t_C^j's and S enters r_S^j's and t_S^j's and they learn a bit, which indicates whether Y was a match.

Note that since r_C^j's, r_S^j's, t_C^j's and t_S^j's are used as inputs to the garbled circuit and will need to be added inside the circuit, we want them to be as small as possible. Therefore, instead of providing unconditional hiding by choosing t_S^j and r_C^j from $\mathbb{Z}_N^*$ (where N is from pk), the protocol achieves statistical hiding by choosing these random values to be κ bits longer than the values that they protect, where κ is a security parameter.

4.3 Optimizations

Pre-computation and offline communication. Similar to prior literature on secure biometric identification [16, 41, 37, 3], we distinguish between offline and online stages, where any computation and communication that does not depend on the inputs of the participating parties can be moved

Input: C has biometric X, $M(X)$ and key pair (pk, sk); S has a database D composed of Y, $M(Y)$ biometrics.
Output: C learns what records in D resulted in match with X if any, i.e., it learns a bit as a result of comparison of X with each $Y \in D$.
Protocol steps:

1. For each $i = 1, \ldots, m$, C computes encryptions $\langle a_{i1}, a_{i2} \rangle = \langle \mathsf{Enc}(X_i M(X_i)), \mathsf{Enc}((1 - X_i)M(X_i)) \rangle$ and sends them to S.
2. For each $i = 1, \ldots, m$, S computes encryption of $M(X_i)$ by setting $a_{i3} = a_{i1} \cdot a_{i2} = \mathsf{Enc}(X_i M(X_i)) \cdot \mathsf{Enc}((1 - X_i)M(X_i)) = \mathsf{Enc}(M(X_i))$.
3. For each record Y in the database, S and C perform the following steps in parallel:
   (a) For each amount of shift $j = -c, \ldots, 0, \ldots, c$, S rotates the bits of Y by the appropriate number of positions to obtain Y^j and proceeds with all Y^j's in parallel.
      i. To compute $(X_i \oplus Y_i^j)M(X_i)M(Y_i^j) = (X_i(1-Y_i^j)+(1-X_i)Y_i^j)M(X_i)M(Y_i^j)$ in encrypted form, S computes $b_i^j = a_{i1}^{(1-Y_i^j)M(Y_i^j)} \cdot a_{i2}^{Y_i^j M(Y_i^j)} = \mathsf{Enc}(X_i M(X_i)(1 - Y_i^j)M(Y_i^j) + (1 - X_i)M(X_i)Y_i^j M(Y_i^j))$.
      ii. S adds the values contained in b_i^j's to obtain $b^j = \prod_{i=1}^m b_i^j = \mathsf{Enc}(\sum_{i=1}^m (X_i \oplus Y_i^j)M(X_i)M(Y_i^j)) = \mathsf{Enc}(||(X \oplus Y^j) \cap M(X) \cap M(Y^j)||)$. S then "lifts up" the result, blinds, and randomizes it as $c^j = (b^j)^{2^\ell} \cdot \mathsf{Enc}(r_S^j)$, where $r_S^j \stackrel{R}{\leftarrow} \{0,1\}^{\lceil \log m \rceil + \ell + \kappa}$, and sends the resulting c^j to C.
      iii. To obtain $T(||M(X) \cap M(Y^j)||)$, S computes $d_i^j = a_{i3}^{M(Y_i^j)} = \mathsf{Enc}(M(X_i) \cdot M(Y_i^j))$ and $d^j = (\prod_{i=1}^m d_i^j)^T = \mathsf{Enc}(T(\sum_{i=1}^m M(X_i)M(Y_i^j)))$. S blinds and randomizes the result as $e^j = d^j \cdot \mathsf{Enc}(t_S^j)$, where $t_S^j \stackrel{R}{\leftarrow} \{0,1\}^{\lceil \log m \rceil + \ell + \kappa}$, and sends e^j to C.
      iv. C decrypts the received values and sets $r_C^j = \mathsf{Dec}(c^j)$ and $t_C^j = \mathsf{Dec}(e^j)$.
   (b) C and S perform $2c + 1$ comparisons and OR of the results of the comparisons using garbled circuit. C enters r_C^j's and t_C^j's, S enters $-r_S^j$'s and $-t_S^j$'s, and C learns bit b computed as $\bigvee_{j=-c}^{c}((r_C^j - r_S^j) \stackrel{?}{<} (t_C^j - t_S^j))$. To achieve this, S creates the garbled circuit and sends it to C. C obtains keys corresponding to its inputs using OT, evaluates the circuit, and S sends to C the key-value mapping for the output gate.

Figure 1: Secure two-party protocol for iris identification.

to the offline stage. In our protocol, first notice that most modular exponentiations (the most expensive operation in the encryption scheme) can be precomputed. That is, the client needs to produce $2m$ encryptions of bits. Because both m and the average number of 0's and 1's in a biometric and a mask are known, the client can produce a sufficient number of bit encryptions in advance. In particular, X normally will have 50% of 0's and 50% of 1's, while 75% (or a similar number) of $M(X)$'s bits are set to 1 and 25% to 0 during biometric processing. Let p_0 and p_1 (q_0 and q_1) denote the fraction of 0's and 1's in an iris code (resp., its mask), where $p_0 + p_1 = q_0 + q_1 = 1$. Therefore, to have a sufficient supply of ciphertexts to form tuples $\langle a_{i1}, a_{i2} \rangle$, the client needs to precompute $(2q_0 + q_1(p_1 + \varepsilon) + q_1(p_0 + \varepsilon))m = (1 + q_0 + 2q_1\varepsilon)m$ encryptions of 0 and $(q_1(p_1 + \varepsilon) + q_1(p_0 + \varepsilon))m = q_1(1 + 2\varepsilon)m$ encryptions of 1, where ε is used as a cushion since the number of 0's and 1's in X might not be exactly p_0 and p_1, respectively. Then at the time of the protocol the client simply uses the appropriate ciphertexts to form its transmission.

Similarly, the server can precompute a sufficient supply of encryptions of r_S^j's and t_S^j's for all records. That is, the server needs for produce $2(2c + 1)|D|$ encryptions of different random values of length $\lceil \log m \rceil + \ell + \kappa$, where $|D|$ denotes the size of the database D. The server also generates one garbled circuit per record Y in its database (for step 3(b) of the protocol) and communicates the circuits to the client. In addition, the most expensive part of the oblivious transfer can also be performed during the offline stage, as detailed below.

Optimized multiplication. Server's computation in steps 3(a).i and 3(a).iii of the protocol can be significantly lowered as follows. To compute ciphertexts b_i^j, S needs to calculate $a_{i1}^{(1-Y_i^j)M(Y_i^j)} \cdot a_{i2}^{Y_i^j M(Y_i^j)}$. Since the bits Y_i^j and $M(Y_i^j)$ are known to S, this computation can be rewritten using one of the following cases:

- $Y_i^j = 0$ and $M(Y_i^j) = 0$: in this case both $(1 - Y_i^j)M(Y_i^j)$ and $Y_i^j M(Y_i^j)$ are zero, which means that b_i^j should correspond to an encryption of 0 regardless of a_{i1} and a_{i2}. Instead of having S create an encryption 0, we set b_i^j to the empty value, i.e., it is not used in the computation of b^j in step 3(a).ii.
- $Y_i^j = 1$ and $M(Y_i^j) = 0$: the same as above.
- $Y_i^j = 0$ and $M(Y_i^j) = 1$: in this case $(1 - Y_i^j)M(Y_i^j) = 1$ and $Y_i^j M(Y_i^j) = 0$, which means that S sets $b_i^j = a_{i1}$.
- $Y_i^j = 1$ and $M(Y_i^j) = 1$: in this case $(1 - Y_i^j)M(Y_i^j) = 0$ and $Y_i^j M(Y_i^j) = 1$, and S therefore sets $b_i^j = a_{i2}$.

The above implies that only $q_1 m$ ciphertexts b_i^j need to be added in step 3(a).ii to form b^j (i.e., $q_1 m - 1$ modular multiplications to compute the hamming distance between m-element strings).

Similar optimization applies to the computation of d_i^j and d^j in step 3(a).iii of the protocol. That is, when $M(Y_i^j) = 0$, d_i^j is set to the empty value and is not used in the computation of d^j; when $M(Y_i^j) = 1$, S sets $d_i^j = a_{i3}$. Consequently, $q_1 m$ ciphertexts are used in computing d^j.

To further reduce the number of modular multiplications, we can adopt the idea from [37], which consists of precomputing all possible combinations for ciphertexts at positions i and $i + 1$ and reducing the number of modular multiplications used during processing a database record in half. In our case, the value of $b_i^j b_{i+1}^j$ requires computation only when $M(Y_i^j) = M(Y_{i+1}^j) = 1$. In this case, computing $a_{i1}a_{(i+1)1}$, $a_{i1}a_{(i+1)2}$, $a_{i2}a_{(i+1)1}$, and $a_{i2}a_{(i+1)2}$, for each odd i between 1 and $m - 1$ will cover all possibilities. Note that these values need to be computed once for all possible shift amounts of the biometrics (since only server's Y's are shifted). Depending on the distribution of the set bits in each $M(Y)$, the number of modular multiplication now will be between $q_1 m/2$ (when $M(Y_i) = M(Y_{i+1})$ for each odd i) and $m(q_0 + (1 - 2q_0)/2) = m/2$ (when $M(Y_i) \neq M(Y_{i+1})$ for as many odd i's as possible). This approach can be also applied to the computation of d^j (where only the value of $a_{i3}a_{(i+1)3}$ needs to be precomputed for each odd i) resulting in the same computational savings during computation of the hamming distance. Furthermore, by precomputing the combinations of more than two values additional savings can be achieved during processing of each Y.

Optimized encryption scheme. As it is clear from the protocol description, its performance crucially relies on the performance of the underlying homomorphic encryption scheme for encryption, addition of two encrypted values, and decryption. Instead of utilizing a general purpose encryption scheme such as Paillier, we turn our attention to schemes of restricted functionality which promise to offer improved efficiency. In particular, the DGK additively homomorphic encryption scheme [12, 11] was developed to be used for secure comparison, where each ciphertext encrypts a bit. In that setting, it has faster encryption and decryption time than Paillier and each ciphertext has size k using a k-bit RSA modulus (while Paillier ciphertext has size $2k$). To be suitable for our application, the encryption scheme needs to support larger plaintext sizes. The DGK scheme can be modified to work with longer plaintexts. In that case, at decryption time, one needs to additionally solve the discrete logarithm problem where the base is 2-smooth using Pohlig-Hellman algorithm. This means that decryption uses additional $O(n)$ modular multiplications for n-bit plaintexts. Now recall that in the protocol we encrypt messages of length $\lceil \log m \rceil + \ell + \kappa$ bits. The use of the security parameter κ significantly increases the length of the plaintexts. We, however, notice that the DGK encryption can be setup to permit arithmetic on encrypted values such that all computations on the underlying plaintexts are carried modulo 2^n for any n. For our protocol it implies that (i) the blinding values r_S^j and t_S^j can now be chosen from the range $[0, 2^n - 1]$, where $n = \lceil \log m \rceil + \ell$, and (ii) this provides information-theoretic hiding (thus improving the security properties of the protocol). This observation has a profound impact not only on the client decryption time in step 3(a).iv (which decreases by about an order of magnitude), but also on the consecutive garbled circuit evaluation, where likewise the circuit size is significantly reduced in size.

Circuit construction. We construct garbled circuits using the most efficient techniques from [39] and references therein. By performing addition modulo 2^n and eliminating gates which have a constant value as one of their inputs, we reduce the complexity of the circuit for addition to $n - 1$ non-XOR gates and $5(n - 1) - 1$ total gates. Similarly, after eliminating gates with one constant input, the complexity of the circuit for comparison of n-bit values becomes n non-XOR gates and $4n - 2$ gates overall. Since in the protocol there are two additions and one comparison per each j followed by $2c$ OR gates, the size of the overall circuit is $14(n - 1)(2c + 1) + 2c$ gates, $(3n - 2)(2c + 1) + 2c$ of which are non-XOR gates. Note that this circuit does not use multiplexers, which are required (and add complexity) during direct computation of minimum.

Oblivious transfer. The above circuit requires each party to supply $2n(2c+1)$ input bits, and a new circuit is used for each Y in D. Similar to [21], the combination of techniques from [27] and [36] achieves the best performance in our case. Let the server create each circuit and the client evaluate them. Using the results of [27], performing OT_1^2 the total of $2n(2c + 1)|D|$ times, where the client receives a κ-bit string as a result of each OT for a a security parameter κ, can be reduced to κ invocations of OT_1^2 (that communicates to the receiver κ-bit strings) at the cost of $4\kappa \cdot 2n(2n+1)|D|$ bits of communication and $4n(2c + 1)$ applications of a hash function for the sender and $2n(2c + 1)$ applications for the receiver. Then κ OT_1^2 protocols can be implemented using the construction of [36] with low amortized complexity, where the sender performs $2 + \kappa$ and the receiver performs 2κ modular exponentiations with the communication of $2\kappa^2$ bits and κ public keys. The OT protocols can be performed during the offline stage, while the additional communication takes place once the inputs are known.

Further reducing online communication. If transmitting $2m$ ciphertexts during the online stage of the protocol (which amounts to a few hundred KB for our set of parameters) constitutes a burden, this communication can be performed at the offline stage before the protocol begins. This can be achieved using the technique of [37], where the client transmits $2m$ encryptions of randomly chosen bits $u_1, \ldots, u_{2m}$ during the offline stage, and the online communication consists of $2m$ bits $v_1, \ldots, v_{2m}$. Each bit v_i corresponds to the XOR of the bit w_i that the client wants to use in the protocol with the previously communicated random bit u_i. After receiving the $2m$-bit correction string $w_1 \oplus u_1, \ldots, w_{2m} \oplus u_{2m}$, the server needs to compute encryption of w_i's using $\mathsf{Enc}(u_i)$ and v_i, which is done by XORing u_i and v_i inside the encryption. Using $u_i \oplus v_i = u_i(1 - v_i) + (1 - u_i)v_i = u_i + v_i - 2u_iv_i$, we see that when $v_i = 0$, the server can simply set $\mathsf{Enc}(w_i) = \mathsf{Enc}(u_i)$, but when $v_i = 1$, the server will need to perform subtraction of (encrypted) u_i. While subtraction is usually one of the most expensive operations, note that because of our use of DGK encryption with short plaintexts the subtraction operations can be performed on a ciphertext significantly faster than using generic full-domain encryption schemes such as Paillier. The speed up is on the order of $k/n \approx 50$, where $k \geq 1024$ is the security parameter for a public-key encryption scheme and $n = \lceil \log m \rceil + \ell = 20$ is the length of the values we operate on. Furthermore, this entire computation can be completely removed from the online stage if, upon the receipt of $\mathsf{Enc}(u_i)$, the server computes $\mathsf{Enc}(1 - u_i)$ during the offline stage. Then when the protocol begins, the server sets either $\mathsf{Enc}(w_i) = \mathsf{Enc}(u_i)$ or $\mathsf{Enc}(w_i) = \mathsf{Enc}(1 - u_i)$ depending on the bit v_i it receives.

4.4 Security Analysis

Security of the iris protocol relies on the security of the underlying building blocks. In particular, we need to assume that (i) the DGK encryption scheme is semantically secure (which was shown under a hardness assumption that uses subgroups of an RSA modulus [12, 11]); (ii) garbled circuit evaluation is secure (which was shown assuming that the hash function is correlation robust [32], or if it is modeled as a random oracle); and (iii) the oblivious transfer is secure as well (to achieve this, techniques of [27] require the hash function to be correlation robust and the use of a pseudo-random number generator, while techniques of [36] model the hash functions as a random oracle and use the computational Diffie-Hellman (CDH) assumption). Therefore, assuming the security of the DGK encryption, CDH, and using the random oracle model for hash functions is sufficient for our solution.

To show the security of the protocol, we sketch how to simulate the view of each party using its inputs and outputs alone. If such simulation is indistinguishable from the real execution of the protocol, for semi-honest parties this implies that the protocol does not reveal any unintended information to the participants (i.e., they learn only the output and what can be deduced from their respective inputs and outputs).

First, consider the client C. The client's input consists of its biometric X, $M(X)$ and the private key, and its outputs consists of a bit b for each record in S's database D. A simulator that is given these values simulates C's view by sending encrypted bits of C's input to the server as prescribed in step 1 of the protocol. It then simulates the messages received by the client in step 3(a).iii using encryptions of two randomly chosen strings r_C^j and t_C^j of length n. The simulator next creates a garbled circuit for the computation given in step 3(b) that, on input client's r_C^j's and t_C^j's computes bit b, sends the circuit to the client, and simulates the OT. It is clear that given secure implementation of garbled circuit evaluation in the real protocol, the client cannot distinguish simulation from real protocol execution. Furthermore, the values that C recovers in step 3(a).iv of the protocol are distributed identically to the values used in the real protocol execution that uses DGK encryption (and they are statistically indistinguishable when other encryption schemes are used).

Now consider the server's view. The server has its database D consisting of Y, $M(Y)$ and the threshold T as the input and no output. In this case, a simulator with access to D first sends to S ciphertexts (as in step 1 of the protocol) that encrypt bits of its choice. For each $Y \in D$, S performs its computation in step 3(a) of the protocol and forms garbled circuits as specified in step 3(b). The server and the simulator engage in the OT protocol, where the simulator uses arbitrary bits as its input to the OT protocol and the server sends the key-value mapping for the output gate. It is clear that the server cannot distinguish the above interaction from the real protocol execution. In particular, due to semantic security of the encryption scheme S learns no information about the encrypted values and due to security of OT S learns no information about the values chosen by the simulator for the garbled circuit.

	Setup	Offline		
		enc	circuit	total
Server	$c = 5$	1398 msec + 71 msec/rec	1780 msec + 8.5 msec/rec	3178 msec + 79.5 msec/rec
	$c = 0$	1398 msec + 6.5 msec/rec	1457 msec + 0.75 msec/rec	2855 msec + 7.25 msec/rec
	$c = 5$ with [38]	131.37 sec + 780 msec/rec	1780 msec + 8.5 msec/rec	131.37 sec + 993.5 msec/rec
Client	$c = 5$	11.93 sec	1693 msec + 3.39 msec/rec	13.62 sec + 3.39 msec/rec
	$c = 0$	11.93 sec	1055 msec + 0.34 msec/rec	12.99 sec + 0.34 msec/rec
	$c = 5$ with [38]	161.37 sec	1693 msec + 3.39 msec/rec	163.06 sec + 3.39 msec/rec
Comm	$c = 5$	512KB	11.6KB + 22.1KB/rec	524KB + 22.1KB/rec
	$c = 0$	512KB	11.6KB + 2KB/rec	524KB + 2KB/rec
	$c = 5$ with [38]	1024KB	11.6KB + 22.1KB/rec	1036KB + 22.1KB/rec

	Setup	Online		
		enc	circuit	total
Server	$c = 5$	108 msec + 148 msec/rec	1.25 msec/rec	89 msec + 149.25 msec/rec
	$c = 0$	108 msec + 13.6 msec/rec	0.11 msec/rec	89 msec + 13.71 msec/rec
	$c = 5$ with [38]	427 msec + 586 msec/rec	1.25 msec/rec	427 msec + 587.25 msec/rec
Client	$c = 5$	20 msec/rec	2.61 msec/rec	22.61 msec/rec
	$c = 0$	1.8 msec/rec	0.22 msec/rec	2.02 msec/rec
	$c = 5$ with [38]	197 msec/rec	2.61 msec/rec	199.61 msec/rec
Comm	$c = 5$	0.5 KB + 2.7 KB/rec	17.2 KB/rec	0.5 KB + 19.9 KB/rec
	$c = 0$	0.5 KB + 0.2 KB/rec	1.6 KB/rec	0.5 KB + 1.8 KB/rec
	$c = 5$ with [38]	0.5 KB + 5.5 KB/rec	17.2 KB/rec	0.5 KB + 22.7 KB/rec

Table 1: Breakdown of the performance of the iris identification protocol.

4.5 Implementation and Performance

We implemented the secure iris identification protocol in C using MIRACL library [25] for cryptographic operations. The implementation used DGK encryption scheme [12, 11] with a 1024-bit modulus and another security parameter t set to 160, as suggested in [12, 11]. To illustrate the advantage of the tools we utilize, we also run selected experiments using Paillier encryption [38]. The Paillier encryption scheme was implemented using a 1024-bit modulus and a number of optimizations suggested in [38] for best performance. In particular, small generator $g = 2$ was used to achieve lower encryption time, and decryption is sped up using precomputation and Chinese remainder computation (see [38], section 7 for more detail). To simplify comparisons with prior work, throughout this work we use $k = 1024$ security parameter for public-key cryptography and $\kappa = 80$ for symmetric and statistical security. The experiments were run on an Intel Core 2 Duo 2.13 GHz machine running Linux (kernel 2.6.35) with 3GB of RAM and *gcc* version 4.4.5.

Table 1 shows performance of the secure iris identification protocol and its components. The performance was obtained using the following set of parameters: the size of iris code and mask $m = 2048$ (this value of m is used in commercial iris recognition software), 75% of bits are reliable in each iris code, and the length n of values is 20 bits. All optimizations described earlier in this section were implemented. In our implementation, upon receipt of client's data, the server precomputes all combinations for pairs of ciphertexts $b_i b_{i+1}$ in step 3(a).ii (one-time cost of the total of $4(m/2)$ modular multiplications) and all combinations of 4 elements $d_i d_{i+1} d_{i+2} d_{i+3}$ in step 3(a).iii (one-time cost of $11(m/4)$ modular multiplications). This cuts the server's time for processing each Y by more than a half. Furthermore, the constant overhead associated with the OT (circuit) can be reduced in terms of both communication and computation for both parties if public-key operations are implemented over elliptic curves.

The table shows performance using three different configurations: (i) the amount of rotation c was set to 5, (ii) no rotation was used by setting $c = 0$ (this is used when the images are well aligned, which is the case for iris biometrics collected at our institution), and (iii) with $c = 5$ using Paillier encryption [38] instead of DGK scheme. In the table, we divide the computation and communication into offline precomputation and online protocol execution. No inputs are assumed to be known by any party at precomputation time. All times are shown in seconds (or fraction, where specified) and communication is shown in KB. Some of the overhead depends on the server's database size, in which case the computation and communication are indicated per record (using notation "/rec"). The overhead associated with the part of the protocol that uses homomorphic encryption is shown separately from the overhead associated with garbled circuits. The offline and online computation for the part based on homomorphic encryption is computed as described in Section 4.3. For circuits, garbled circuit creation, communication, and some of OT is performed at the offline stage, while the rest of OT (as described in Section 4.3) and garbled circuit evaluation takes place during the online protocol execution.

It is evident that our protocol design and optimizations allow us to achieve notable performance. In particular, comparison of two iris codes, which includes computation of $2(2c+1) = 22$ Hamming distances over 2048-bit biometrics in encrypted form, is done in 0.15 sec. This is noticeably lower than 0.3 sec online time per record reported by the best currently known face recognition protocol SCiFI [37], which computes a single Hamming distance over 900-bit values. That is, despite an order of magnitude larger number of operations and more complex operations such as division, computation of minimum, etc., we are able to outperform prior work by roughly 50%. This in particular implies that using the techniques suggested in this work (and DGK encryption scheme in particular) performance of SCiFI and other existing protocols can be improved to a fraction of the previously reported time. When iris images are well aligned and no rotation is necessary our protocol requires only 14 msec online computation time and under 2KB of data to compare two biometrics.

5 Secure Fingerprint Identification

Before proceeding with the novel protocol for fingerprint identification based on minutiae pairing, we first illustrate how a number of the techniques developed in this work for iris identification can be applied to other types of biometric computations such as FingerCodes. In particular, we show that the efficiency of the secure protocol for FingerCode identification [3] can be improved by an order of magnitude.

5.1 FingerCode Identification

The computation involved in FingerCode comparisons is very simple, which results in an extremely efficient privacy-preserving realization. Similar to [3], we rewrite the computation in equation 4 as $\sum_{i=1}^{m}(x_i - y_i)^2 = \sum_{i=1}^{m}(x_i)^2 + \sum_{i=1}^{m}(y_i)^2 - \sum_{i=1}^{m} 2x_iy_i < T^2$. In our protocol, the Euclidean distance is computed using homomorphic encryption, while the comparisons are performed using garbled circuits. The secure FingerCode protocol is given in Figure 2: the client contributes encryptions of $-2x_i$ and $\sum(x_i)^2$ to the computation, while the server contributes $\sum(y_i)^2$ and computes encryption of $-2x_iy_i$ from $-2x_i$. Note that by using $\mathsf{Enc}(-2x_i)$ instead of $\mathsf{Enc}(x_i)$, the server's work for each Y is reduced since negative values use significantly longer representations. The protocol in Figure 2 uses DGK encryption with the plaintext space of $[0, 2^n - 1]$. To be able to represent the Euclidean distance, we need to set $n = \lceil \log m \rceil + 2\ell + 1$, where ℓ is the bitlength of elements x_i and y_i. This implies that all computation on plaintexts is performed modulo 2^n; for instance, $2^n - 2x_i$ is used in

Input: C has biometric $X = (x_1, \ldots, x_m)$ and DGK encryption key pair (pk, sk); S has a database D composed of biometrics $Y = (y_1, \ldots, y_m)$.
Output: C learns what records in D resulted in match with X if any, i.e., it learns a bit as a result of comparison of X with each $Y \in D$.
Protocol steps:

1. C computes and sends to S encryptions $\mathsf{Enc}(-2x_1), \ldots, \mathsf{Enc}(-2x_m), \mathsf{Enc}(\sum_{i=1}^{m} x_i^2)$.
2. For each $Y = (y_1, \ldots, y_m) \in D$, S and C perform in parallel:
   (a) S computes the encrypted distance d between X and Y as $d = \mathsf{Enc}(\sum_{i=1}^{m} x_i^2) \cdot \mathsf{Enc}(\sum_{i=1}^{m} y_i^2) \cdot \prod_{i=1}^{m} \mathsf{Enc}(-2x_i)^{y_i} = \mathsf{Enc}(\sum_{i=1}^{m}(x_i - y_i)^2)$, blinds it as $d' = d \cdot \mathsf{Enc}(r_S)$, where $r_S \stackrel{R}{\leftarrow} \{0,1\}^n$, and sends d' to C.
   (b) C decrypts the value it receives and sets $r_C = \mathsf{Dec}(d')$.
   (c) C and S engage in a secure protocol that computes $((r_C - r_S) \bmod 2^n) \stackrel{?}{<} T^2$ using garbled circuit evaluation. S creates the circuit and sends it to C along with the key-value mapping for the output gate. C obtains keys corresponding to its inputs from S using OT, evaluates the circuit, and learns the result.

Figure 2: Secure two-party protocol for FingerCode identification.

step 1 to form $\mathsf{Enc}(-2x_i)$. The circuit used in step 2(c) takes two n-bit values, adds them modulo 2^n, and compares the result to a constant as described in Section 4.3.

Finally, some of the computation can be performed offline: for the client it includes precomputing the random values used in the $m+1$ ciphertexts it sends in step 1 (computation of $h^r \bmod N$), and for the server includes precomputing $\mathsf{Enc}(r_S)$ and preparing a garbled circuit for each Y, as well as one-time computation of random values for $\mathsf{Enc}(\sum_{i=1}^{m}(y_i)^2)$ since the reuse of such randomness does not affect security. The client and the server also perform some of OT functionality prior to protocol initiation.

In the FingerCode protocol of [3], each fingerprint in the server's database is represented by c FingerCodes that correspond to different orientations of the same fingerprint, which improves the accuracy of matching. The protocol of [3], however, reports all matches within the c FingerCodes corresponding to the same fingerprint, and this is what our protocol in Figure 2 computes. If it is desirable to output only a single bit for all c instances of a fingerprint, it is easy to modify the circuit evaluated in step 2(c) of the protocol to compute the OR of the bits produced by the original c circuits.

Security. The security of this protocol is straightforward to show and we omit the details of the simulator from the current description. As before, by using only tools known to be secure and protecting the information at intermediate stages, neither the client nor the server learns information beyond what the protocol prescribes.

Implementation and performance. The FingerCode parameters can range as $m = 16$–640, $\ell = 4$–8, and $c = 5$. We implement the protocol using parameters $m = 16$ and $\ell = 7$ (the same as in [3]) and therefore $n = 19$. The performance of our secure FingerCode identification protocol is given in Table 2. No inputs (X or Y) are assumed to be known at the offline stage when the parties compute randomization values of the ciphertexts. For that reason, a small fixed cost is inquired in the beginning of the protocol to finish forming the ciphertext using the data itself. We also note that, based on our additional experiments, by using Paillier encryption instead of DGK encryption, the server's online work increases by an order of magnitude, even if packing is used.

It is evident that the overhead reported in the table is minimal and the protocol is well suited

	Offline		
	enc	circuit	total
Server	3.6 msec + 3.9 msec/rec	1448 msec + 0.37 msec/rec	1451.6 msec + 4.3 msec/rec
Client	61 msec	1025 msec + 0.15 msec/rec	1086 msec + 0.15 msec/rec
Comm	0	11.6 KB + 1.26 KB/rec	11.6 KB + 1.26 KB/rec

	Online		
	enc	circuit	total
Server	0.22 msec + 1.37 msec/rec	0.05 msec/rec	0.22 msec + 1.42 msec/rec
Client	4.7 msec + 0.92 msec/rec	0.16 msec/rec	4.7 msec + 1.08 msec/rec
Comm	2.12 KB + 0.12 KB/rec	0.74 KB/rec	2.12 KB + 0.86 KB/rec

Table 2: Breakdown of the performance of the FingerCode identification protocol.

for processing fingerprint data in real time. In particular, for a database of 320 records used in prior work (64 fingerprints with 5 FingerCodes each used in [3]), client's online work is 0.35 sec and the server's online work is 0.45 sec, with online communication of 277KB. As can be seen from these results, computation is no longer the bottleneck and this secure two-party protocol can be carried out extremely efficiently. Compared to the solution in [3] that took 16 sec for the online stage with the same setup, the computation speed up is by a factor of 35. Communication efficiency, however, is what was specifically emphasized in the protocol of [3] resulting in 10101KB online overhead for a database of size 320. Our solution therefore improves such result by a factor of 35. We also would like to note that all offline work in [3] is for ciphertext precomputation (since no garbled circuits are used) and is non-interactive, while in our protocol circuit transmission and input-independent portions of OT can be done prior to the protocol itself and involve interaction. We, however, note that the overall (offline and online) computation for $|D| = 320$ is 1.48 sec for the client and 3.27 sec for the server with the total of 692KB communication, which is still at least several times lower than the online portion of the time and communication in [3].

Comparison with [41]. Privacy-preserving face recognition techniques by Sadeghi et al. [41] can also be adapted to perform secure FingerCode comparisons. Even though they were developed for different applications, they present some analogies with our approach. In particular, they involve computing Euclidean distances using homomorphic encryption, followed by garbled circuits-based comparisons of the results. Although at the high-level the techniques are similar, the optimizations employed in this work allow us to achieve superior performance. For the distance computation, [41] reports runtime of 6.08 sec for the client and 0.47 sec for the server for a database of 320 records; distance computation in our protocol (including precomputation) is 0.36 sec for the client and 1.69 sec for the server for the same database size. In [41] the number of dimensions is $m = 12$, while we have $m = 16$, but the length of values is $n = 50$ in [41], and $n = 19$ in our tests. The computation itself in [41] is more expensive (including interaction between the parties, which we do not have) due to the need to transform client's data, but [41] uses faster machines.

We performed additional experiments, in which we adapted the solution of Sadeghi et al. to carry out FingerCode computation and implemented it. These experiments show that our technique is several times faster than that of Sadeghi et al. while requiring a comparable amount of communication bandwidth.

5.2 Minutiae-based Fingerprint Identification

Our secure protocol for minutiae-based fingerprint identification preserves the high-level idea of using homomorphic encryption for computing the distance between minutia points and garbled circuit evaluation for comparisons, but introduces a number of new techniques. At high-level, computing the pairing between minutiae of fingerprints $X = \langle(x_1, y_1, \alpha_1), \ldots, (x_{m_X}, y_{m_X}, \alpha_{m_X})\rangle$ and $Y = \langle(x'_1, y'_1, \alpha'_1), \ldots, (x'_{m_Y}, y'_{m_Y}, \alpha'_{m_Y})\rangle$ based on minimum distances proceeds in iterations as follows. C and S maintain an m_Y-bit array M, the i-th bit of which indicates whether minutia Y_i has been marked or not. Initially, all bits of M are set to 0. For $i = 1, \ldots, m_X$, perform:

1. Compute the set S of minutiae from Y matching X_i that have not been marked, i.e., $S = \{Y_j \mid mm(X_i, Y_j) \text{ and } M[j] = 0\}$.
2. Compute the minutia Y_k (if any) from S with the minimum (spatial) distance from X_i and set $M[k] = 1$.

To preserve secrecy of the data, each bit of the array M is maintained by C and S in XOR-split form, i.e., C stores $M_C[i]$ and S stores $M_S[i]$ such that $M[i] = M_C[i] \oplus M_S[i]$. During each iteration of the computation, at the end of step 2 above, C and S obtain XOR-shares of an array A that has bit k set to 1 and all other bits set to 0 (or all bits set to 1 if no pairing for X_i exists). Both C and S update their share of M by XORing the share of A that they received with the current share of M. This ensures that the array M is properly maintained.

In the beginning of the protocol the client sends information about its fingerprint X . For best performance, we utilize DGK encryption with two pairs of keys. The first pair (pk_1, sk_1) is used for encrypting spatial coordinates x_i, y_i and computing Euclidean distance between points, and the second pair (pk_2, sk_2) is used for encrypting directional information α_i and directional difference. Therefore, we set $u = 2^{2\ell+2}$ in pk_1, where ℓ is the bitlength of coordinates x_i, y_i, and $u = 360$ in pk_2. This implies that computing $\alpha'_j - \alpha_i$ on encrypted values will automatically result in the value being reduced modulo 360, which simplifies computation with the directional difference in this form. Also note that, while decryption in the DGK encryption scheme involves solving the discrete logarithm, when $u = 360$ this can be achieved at low cost using Pohlig-Hellman algorithm because 360 has only small factors.

Our secure fingerprint identification protocol is given in Figure 3. At iteration i, after computing the distances in encrypted form (step 2(b).i) and decrypting them in a split form (step 2(b).ii), the parties engage in garbled circuit evaluation using a circuit that performs the main computation and produces an m_Y-bit vector A with at most one bit set to one indicating the position of the mate of minutia X_i. This (optimized) circuit is the most involved part of the protocol and is discussed in detail below. At the end of each iteration the vector M is updated with the output of the circuit, and after all iterations have been performed the rest of the protocol consists of counting the number of marked elements in M comparing that number to the threshold T. This is done using an additional garbled circuit, where the client learns the output bit.

Note that the protocol requires that both parties know the number of minutiae in client's X and server's Ys, which is assumed not to leak information about the fingerprints themselves. While biometric images of similar quality are expected to have similar numbers of minutiae, if for the purposes of this computation m_X and m_Y are considered to be sensitive information, the fingerprints can be slightly padded to always use the same number m of minutia points. This can achieved by agreeing on a fixed m and inserting fake elements into each fingerprint until its size becomes m. The fake elements should not affect the result of the computation, which means that the fake elements of client's X should be matching either original or fake elements of any Y. The easiest way to ensure this is by setting fake x_i in X to its maximum value plus d_0 and by setting fake x'_j in each Y to its maximum value plus $2d_0$.

We design the circuit evaluation in step 2(b).iii of the protocol to minimize the number of comparisons. In particular, each directional difference $\alpha'_j - \alpha_i$ is compared to the threshold α_0 in

Input: C has biometric $X = \langle(x_1, y_1, \alpha_1), \ldots, (x_{m_X}, y_{m_X}, \alpha_{m_X})\rangle$ and DGK encryption key pairs (pk_1, sk_1) and (pk_2, sk_2); S has a database D composed of biometrics $Y = \langle(x'_1, y'_1, \alpha'_1), \ldots, (x'_{m_Y}, y'_{m_Y}, \alpha'_{m_Y})\rangle$.
Output: C learns what records in D resulted in match with X if any, i.e., it learns a bit as a result of comparison of X with each $Y \in D$.
Protocol steps:

1. C computes encryptions $\langle a_{i1}, a_{i2}, a_{i3}, a_{i4}\rangle = \langle \mathsf{Enc}_{pk1}(-2x_i), \mathsf{Enc}_{pk_1}(-2y_i), \mathsf{Enc}_{pk_1}(x_i^2 + y_i^2), \mathsf{Enc}_{pk_2}(-\alpha_i)\rangle$ for each $i = 1, \ldots, m_X$ and sends them to S.
2. For each $Y = \langle(x'_1, y'_1, \alpha'_1), \ldots, (x'_{m_Y}, y'_{m_Y}, \alpha'_{m_Y})\rangle \in D$, S and C perform in parallel:
   (a) S and C setup m_Y-bit vector M, where initially S's and C's shares M_S and M_C, respectively, are set to all 0's.
   (b) For $i = 1, \ldots, m_X$ S and C perform the following computation:
      i. S computes the encrypted spatial distance s_j between X_i and each Y_j in Y as $s_j = (a_{i1})^{x'_j} \cdot (a_{i2})^{y'_j} \cdot a_{i3} \cdot \mathsf{Enc}_{pk_1}((x'_j)^2 + (y'_j)^2)$ and encrypted directional distance as $d_j = (a_{i4})^{\alpha'_j}$. S blinds all pairs as $s'_j = s_j \cdot \mathsf{Enc}(r_S^j)$, where $r_S^j \stackrel{R}{\leftarrow} \{0,1\}^{2\ell+2}$ and $d'_j = d_j \cdot \mathsf{Enc}(t_S^j)$ where $t_S^j \stackrel{R}{\leftarrow} \mathbb{Z}_{360}$ and sends s'_j, d'_j to C.
      ii. C decrypts received pairs for all $j = 1, \ldots, m_Y$ and sets $r_C^j = \mathsf{Dec}_{sk_1}(s'_j)$ and $t_C^j = \mathsf{Dec}_{sk_2}(d'_j)$.
      iii. C and S engage in garbled circuit evaluation, where S inputs the bits of M_S and $-r_S^j \pmod{2^{2\ell+2}}, -t_S^j \pmod{360}$ for $j = 1, \ldots, m_Y$, C inputs the bits of M_C and r_C^j, t_C^j for $j = 1, \ldots, m_Y$, S learns m_Y-bit A_S, and C learns m_Y-bit A_C. The vector $A = A_S \oplus A_C$ has at most one bit set which indicates the index of the mate of minutia X_i in Y.
      iv. S updates its M_S as $M_S = M_S \oplus A_S$, and C updates its M_C as $M_C = M_C \oplus A_C$.
   (c) C and S engage in the garbled circuit evaluation where, on input M_S from S and M_C from C, C learns the bit corresponding to the computation $||M_S \oplus M_S|| \stackrel{?}{<} T$.

Figure 3: Secure two-party protocol for minutiae-based fingerprint identification.

the beginning, and if it exceeds the threshold, the corresponding distance between X_i and Y_j is modified so that it will not be chosen as the minimum. This is done by prepending the resulting bit of computation $((\alpha'_j - \alpha_i) \geq \alpha_0) \wedge ((\alpha'_j - \alpha_i) \leq (360 - \alpha_0))$ to the spatial distance between X_i and Y_j (as the most significant bit). The same technique is used to ensure that marked minutiae from Y is not selected as well. What remains to compute is to verify what spatial distances fall below the threshold and computing the minimum of such values. In the (oblivious) garbled circuit, instead of first comparing each distance to the threshold and then computing the minimum of (possibly modified) distances, we directly compute the minimum and then compare the minimum to the threshold. This reduces the number of distance comparisons from $2m_Y - 1$ to m_Y. The two previously prepended bits are preserved throughout comparisons, and the final result will have no mate for X_i selected if the computed minimum (squared) distance is not below the threshold $(d_0)^2$.

Both the computation of the minimum and creation of vector A require the use of multiplexers in the circuit. In particular, after comparing two values a_1 and a_2 one type of multiplexer used in our circuit chooses either the bits of a_1 or a_2 based on the resulting bit of the comparison. This permits the computation of the minimum in a hierarchical manner using a small number of non-XOR gates as described in [31]. We also use multiplexers to collect information about A throughout the circuit. In particular, after a single comparison of distances a_1 and a_2, the portion of A corresponding to these two bits will be chosen to be either 01 or 10. Suppose that after comparing a_1 and a_2 this value is 01 and after comparing a_3 and a_4 the value is 10. Then after performing the comparison add add mod 2^n

$\geq 360?$

subtract

$\geq \alpha_0?$ $\geq 360 - \alpha_0?$

$<?$

mux_1 mux_2

Figure 4: Component of circuit in fingerprint identification protocol performed for each value of $j \in [1, m_Y]$.

$<?$

mux_1 mux_3

$< d_0^2?$

mux_4

A_S

A_C

Figure 5: Computation of minimum and its index in circuit of fingerprint identification.

of $\min(a_1, a_2)$ and $\min(a_3, a_4)$ either 0100 or 0010 will be chosen as the current portion of A. This process continues until the overall minimum and the entire A is computed. This value of A will have a single bit set to 1, and after the final comparison of the minimum with the threshold A will either remain unchanged or will be reset to contain all 0s.

Figure 4 shows the initial computation in the circuit performed for each value of j, where $n = 2\ell + 2$, and Figure 5 shows the computation of the minimum and the output for a toy example of $m_Y = 4$. In Figure 4, after adding t_C^j and $-t_S^j$ (mod 360) together, the sum is compared to 360. If it exceeds the value, 360 is subtracted from the sum (in our implementation the subtracted value is bitwise AND of the outcome of comparison and each bit of the binary representation of 360). Finally, the resulting value is compared to two thresholds and the result is prepended to the spatial distance $r_C^j - r_S^j$. In Figure 4, multiplexer mux_1 chooses the smaller value based on the result of the comparison, mux_2 chooses either 01 or 10 based on the result of the comparison, mux_3 chooses a 4-bit string based on its inputs from two multiplexers mux_2 and the outcome of another comparison, and mux_4 chooses either its input from mux_3 or a zero string based on the result of the final comparison. The server (circuit creator) supplies a stream of random bits A_S to the circuit, and the client learns the outcome of the XOR of that stream and the output of the last multiplexer.

	Setup	Offline		
		enc	circuit	total
Server	$m = 20$	72 msec + 2990 msec/rec	1868 msec + 1159 msec/rec	1940 msec + 4149 msec/rec
	$m = 32$	114 msec + 7682 msec/rec	2114 msec + 2117 msec/rec	2228 msec + 9799 msec/rec
Client	$m = 20$	288 msec	1866 msec + 212 msec/rec	2154 msec + 212 msec/rec
	$m = 32$	460 msec	2380 msec + 552 msec/rec	2840 msec + 552 msec/rec
Comm	$m = 20$	0	11.6KB + 83KB/rec	11.6KB + 83KB/rec
	$m = 32$	0	11.6KB + 133KB/rec	11.6KB + 133KB/rec

	Setup	Online		
		enc	circuit	total
Server	$m = 20$	3.6 msec + 100 msec/rec	30 msec/rec	3.6 msec + 130 msec/rec
	$m = 32$	6 msec + 262 msec/rec	77 msec/rec	6 msec + 339 msec/rec
Client	$m = 20$	15 msec + 580 msec/rec	145 msec/rec	15 msec + 725 msec/rec
	$m = 32$	25 msec + 1502 msec/rec	374 msec/rec	25 msec + 1876 msec/rec
Comm	$m = 20$	10KB + 100KB/rec	22.3KB/rec	10KB + 122.3KB/rec
	$m = 32$	16KB + 256KB/rec	38.2KB/rec	16KB + 294.2KB/rec

Table 3: Breakdown of the performance of the fingerprint identification protocol.

Precomputation. Precomputation in this protocol takes similar form as in the FingerCode protocol. Namely, the random values ($h^r \bmod N$) in the ciphertexts are precomputed and the server chooses all r_S^j and t_S^j in advance and encrypts them. Furthermore, encrypted values $\mathsf{Enc}((x'_j)^2 + (y'_j)^2)$ are formed by the server once for each j (independent of m_Y or the size of D) and can also reuse (or use no) randomness. In addition, all garbled circuits are created and transferred in advance, as well as the expensive portion of the OT is performed in advance. Note that it is sufficient to have two input wires to implement all constants in the circuit such as 360, α_0, d_0^2, inputs to multiplexers, etc.

Security. As before, it is easy to show that the protocol is secure, where the simulator relies on the security of the encryption scheme, garbled circuits, and OT.

Implementation and performance. We implement the protocol using a grid of size 250×250 for fingerprint images, which means that each $x_i, y_i \in [0, 249]$ and ℓ is set to 8. In our experiments we use $m = m_X = m_X$ with two values of 20 and 32 minutiae per fingerprint. It is clear that the protocol incurs cost quadratic in m and is expected to have higher overhead than two previous protocols. Table 3 shows performance of the protocol. The online work is dominated by $2m^2$ decryptions at the client side and adds up to 0.73 sec per fingerprint comparison for $m = 20$ and 1.88 sec for $m = 32$. The circuit evaluated by the client in step 2(b).iii of the protocol has size of 2372 non-XOR gates (and 8836 gates total) for $m = 20$ and 3820 non-XOR gates (and 14212 gates total) for $m = 32$. It is evaluated m times by the client for each Y. The circuit evaluated by the client in step 2(c) of the protocol has size of 39 non-XOR gates (and 153 gates total) for $m = 20$ and 63 non-XOR gates (and 246 gates total) for $m = 32$. It is evaluated once for each Y.

We also would like to mention that a protocol solely based on garbled circuit evaluation for this type of computation is likely to result in comparable performance. That is, the circuit would need to perform additional $2m^2$ multiplications (as well as additional additions and subtractions) per Y, with the additional number of gates exceeding the number of gates in the current circuit. This means that the offline work associated with circuit construction (per Y) will increase, but the online communication should decrease.

6 Summary of Design Principles and Conclusions

The protocol design presented in this work suggests certain principles that lead to an efficient implementation of a privacy-preserving protocol for biometric identification, which we summarize next. First, notice that in the computation used in this work, as well as in prior literature, first a distance between biometric X and each biometric Y in the database is computed followed by a comparison operation. The comparison can be performed to either (i) determine whether the distance $\mathsf{dist}(X, Y)$ is below a certain threshold (where the threshold can be specific to each Y or fixed for all Y) or (ii) determine whether the minimum of all distances $\mathsf{dist}(X, Y)$ is below a certain threshold. In both cases an equivalent number of comparisons is performed. The most efficient protocols known to date compute the distance function using homomorphic encryption, but then resort to a different technique for the comparisons. Therefore, the client first communicates its encrypted biometric X to the server, the server next computes the distances, and both the client and the server are involved in the comparison protocol. We thus obtain the following:

1. *Representation of client's biometric matters.* The server's work for processing each record in its database can be significantly reduced if the client's data is provided in the form that optimizes server's computation (for instance, computing $\mathsf{Enc}(-a)$ from $\mathsf{Enc}(a)$ could be one of the most expensive operations). This one-time cost at the client's side has far-reaching consequences for the performance of the overall protocol.

2. *Operations that manipulate bits are the fastest outside encryption.* Any protocol for biometric identification is expected to use comparisons. Despite recent advances in the techniques for carrying out secure comparisons over encrypted data which make them practical (as, e.g., in [12]), garbled circuit evaluation is better suited for a large volume of such operations. Furthermore, when the range of values being compared is small and many comparisons are necessary, additional techniques such as OT can be utilized at low cost [37].

3. *The largest speedup can be seen from proper tuning of encryption tools.* Privacy-preserving protocols that rely on homomorphic encryption can benefit immensely from a wise choice of encryption scheme and its usage. Traditionally, packing was used to reduce overhead of privacy-preserving protocols including asymptotic complexity (see, e.g., [18] for an example). When computation is carried out on integers of small size, alternative encryption schemes such as DGK or additively homomorphic ElGamal implemented over elliptic curves can significantly improve performance. Our results would not be possible without our choice of encryption schemes.

Using these principles and a number of new techniques in this work we develop and implement secure protocols for iris and fingerprint identification that use standard biometric recognition algorithms. The optimization techniques employed in this work allow us to achieve notable performance results for three secure biometric identification protocols:

- We develop the first privacy-preserving two-party protocol for iris codes using current biometric recognition algorithms. Despite the length of iris codes' representation and complexity of their processing, our protocol allows a secure comparison between two biometrics to be performed in 0.15 second with communication of under 18KB. Furthermore, when the iris codes are known to be well-aligned and their rotation is not necessary, the overhead decreases by an order of magnitude to 14 msec computation and 2KB communication per comparison.

- Two FingerCodes used for fingerprint recognition can be compared at low cost, which allowed us to develop an extremely efficient privacy-preserving protocol. Comparing two fingerprints requires approximately 1 msec of computation, allowing thousands of biometrics to be processed in a matter of seconds. Communication overhead is also very modest with less than 1KB per biometric comparison. Compared to prior privacy-preserving implementation of FingerCode [3], we simultaneously improve computation and communication by a factor of 30 or more.

- Fingerprint recognition based on minutiae pairings utilizes most complex algorithms over unordered sets with spatial and directional differences, and in our secure implementation fingerprint identification can be performed using approximately 1 second per record.

7 Acknowledgments

We would like to thank Keith Frikken for suggestion to reduce communication of the iris identification protocol from $3m$ to $2m$ ciphertexts. Portions of this work were sponsored by the Air Force Office of Scientific Research grant AFOSR-FA9550-09-1-0223.

References

[1] M. Atallah, M. Bykova, J. Li, K. Frikken, and M. Topkara. Private collaborative forecasting and benchmarking. In *ACM Workshop on Privacy in the Electronic Society (WPES)*, pages 103–114, 2004.

[2] M. Barbosa, T. Brouard, S. Cauchie, and S. de Sousa. Secure biometric authentication with improved accuracy. In *Australasian conference on Information Security and Privacy (ACISP)*, pages 21–36, 2008.

[3] M. Barni, T. Bianchi, D. Catalano, M. Di Raimondo, R. Labati, P. Failla, D. Fiore, R. Lazzeretti, V. Piuri, F. Scotti, and A. Piva. Privacy-preserving fingercode authentication. In *ACM Workshop on Multimedia and Security (MM&Sec)*, pages 231–240, 2010.

[4] N. Barzegar and M. Moin. A new user dependent iris recognition system based on an area preserving pointwise level set segmentation approach. *EURASIP Journal on Advances in Signal Processing*, pages 1–13, 2009.

[5] M. Blanton. Empirical evaluation of secure two-party computation models. Technical Report TR 2005-58, CERIAS, Purdue University, 2005.

[6] M. Blanton and M. Aliasgari. Secure computation of biometric matching. Technical Report 2009-03, Department of Computer Science and Engineering, University of Notre Dame, 2009.

[7] J. Bringer, H. Chabanne, M. Izabachene, D. Pointcheval, Q. Tang, and S. Zimmer. An application of the Goldwasser-Micali cryptosystem to biometric authentication. In *Australasian conference on Information Security and Privacy (ACISP)*, volume 4586 of *LNCS*, pages 96–106, 2007.

[8] P. Bunn and R. Ostrovsky. Secure two-party k-means clustering. In *ACM Conference on Computer and Communications Security (CCS)*, pages 486–497, 2007.

[9] O. Catrina and A. Saxena. Secure computation with fixed-point numbers. In *Financial Cryptography and Data Security*, pages 35–50, 2010.

[10] R. Cramer, I. Damgård, and J. Nielsen. Multiparty computation from threshold homomorphic encryption. In *Advances in Cryptology – EUROCRYPT*, pages 280–300, 2001.

[11] I. Damgård, M. Geisler, and M. Krøigård. A correction to efficient and secure comparison for on-line auctions. Cryptology ePrint Archive, Report 2008/321, 2008.

[12] I. Damgård, M. Geisler, and M. Krøigård. Homomorphic encryption and secure comparison. *Journal of Applied Cryptology*, 1(1):22–31, 2008.

[13] I. Damgård, M. Geisler, and M. Krøigård. Asynchronous multiparty computation: Theory and implementation. In *Public Key Cryptography (PKC)*, pages 160–179, 2009.

[14] J. Daugman. How iris recognition works. *IEEE Transactions on Circuits and Systems for Video Technology*, 14(1):21–30, 2004.

[15] E. De Cristofaro and G. Tsudik. Practical private set intersection protocol with linear computational and badwidth complexity. In *Financial Cryptography and Data Security*, volume 6052 of *LNCS*, pages 143–159, 2010.

[16] Z. Erkin, M. Franz, J. Guajardo, S. Katzenbeisser, I. Lagendijk, and T. Toft. Privacy-preserving face recognition. In *Privacy Enchancing Technologies Symposium (PETS)*, pages 235–253, 2009.

[17] L. Ford and D. Fulkerson. *Flows in Networks*. Princeton University Press, 1962.

[18] M. Freedman, K. Nissim, and B. Pinkas. Efficient private matching and set intersection. In *Advances in Cryptology – EUROCRYPT*, pages 1–19, 2004.

[19] K. Frikken. Privacy-preserving set union. In *Applied Cryptography and Network Security (ACNS)*, pages 237–252, 2007.

[20] O. Goldreich, S. Micali, and A. Wigderson. How to play any mental game or a completeness theorem for protocols with honest majority. In *ACM Symposium on Theory of Computing (STOC)*, pages 218–229, 1987.

[21] W. Henecka, S. Kogl, A.-R. Sadeghi, T. Schneider, and I. Wehrenberg. TASTY: Tool for Automating Secure Two-partY computations. In *ACM Conference on Computer and Communications Security (CCS)*, pages 451–462, 2010.

[22] T. Hoens, M. Blanton, and N. Chawla. A private and reliable recommendation system using a social network. In *IEEE International Conference on Information Privacy, Security, Risk and Trust (PASSAT)*, pages 816–825, 2010.

[23] K. Hollingsworth, K. Bowyer, and P. Flynn. The best bits in an iris code. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 31(6):964–973, June 2009.

[24] U.S. DHS US-VISIT `http://www.dhs.gov/files/programs/usv.shtm`.

[25] Multiprecision Integer and Rational Arithmetic C/C++ Library. http://www.shamus.ie/.

[26] IrisGuard Press Release. http://cl.ly/3KIB.

[27] Y. Ishai, J. Kilian, K. Nissim, and E. Petrank. Extending oblivious tranfers efficiently. In *Advances in Cryptology – CRYPTO*, pages 145–161, 2003.

[28] A. Jain, S. Prabhakar, L. Hong, and S. Pankanti. Filterbank-based fingerprint matching. *IEEE Transactions on Image Processing*, 9(5):846–859, 2000.

[29] T.-Y. Jea and V. Govindaraju. A minutia-based partial fingerprint recognition system. *Pattern Recognition*, 38(10):1672–1684, 2005.

[30] L. Kissner and D. Song. Privacy-preserving set operations. In *Advances in Cryptology – CRYPTO*, pages 241–257, 2005.

[31] V. Kolesnikov, A.-R. Sadeghi, and T. Schneider. Improved garbled circuit building blocks and applications to auctions and computing minima. In *Cryptology and Network Security (CANS)*, pages 1–20, 2009.

[32] V. Kolesnikov and T. Schneider. Improved garbled circuit: Free XOR gates and applications. In *International Colloquium on Automata, Languages and Programming (ICALP)*, pages 486–498, 2008.

[33] Y. Lindell, B. Pinkas, and N. Smart. Implementing two-party computation efficiently with security against malicious adversaries. In *Security and Cryptography for Networks (SCN)*, pages 2–20, 2008.

[34] D. Malkhi, N. Nisan, B. Pinkas, and Y. Sella. Fairplay – a secure two-party computation system. In *USENIX Security Symposium*, pages 287–302, 2004.

[35] D. Maltoni, D. Maio, A. Jain, and S. Prabhakar. *Hanbook of Fingerprint Recognition*. Springer, second edition, 2009.

[36] M. Naor and B. Pinkas. Efficient oblivious transfer protocols. In *ACM-SIAM Symposium On Discrete Algorithms (SODA)*, pages 448–457, 2001.

[37] 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.

[38] P. Paillier. Public-key cryptosystems based on composite degree residuosity classes. In *Advances in Cryptology – EUROCRYPT'99*, volume 1592 of *LNCS*, pages 223–238, 1999.

[39] B. Pinkas, T. Schneider, N. Smart, and S. Williams. Secure two-party computation is practical. In *Advances in Cryptology – ASIACRYPT*, volume 5912 of *LNCS*, pages 250–267, 2009.

[40] T. Rabin and M. Ben-Or. Verifiable secret sharing and multiparty protocols with honest majority. In *ACM Symposium on Theory of Computing (STOC)*, pages 73–85, 1989.

[41] A.-R. Sadeghi, T. Schneider, and I. Wehrenberg. Efficient privacy-preserving face recognition. In *International Conference on Information Security and Cryptology (ICISC)*, pages 229–244, 2009.

[42] J. Troncoso-Pastoriza, S. Katzenbeisser, and M. Celik. Privacy preserving error resilient DNA searching through oblivious automata. In *ACM Conference on Computer and Communications Security (CCS)*, pages 519–528, 2007.

[43] C. Wang, M. Gavrilova, Y. Luo, and J. Rokne. An efficient algorithm for fingerprint matching. In *International Conference on Pattern Recognition (ICPR)*, pages 1034–1037, 2006.

[44] A. Yao. How to generate and exchange secrets. In *IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 162–167, 1986.