Efficient Privacy-Preserving Biometric Identification

Yan Huang
U. of Virginia
yhuang@virginia.edu

Lior Malka
Intel*
lior34@gmail.com

David Evans
U. of Virginia
evans@virginia.edu

Jonathan Katz
U. of Maryland
jkatz@cs.umd.edu

Abstract

We present an efficient matching protocol that can be used in many privacy-preserving biometric identification systems in the semi-honest setting. Our most general technical contribution is a new backtracking protocol that uses the by-product of evaluating a garbled circuit to enable efficient oblivious information retrieval. We also present a more efficient protocol for computing the Euclidean distances of vectors, and optimized circuits for finding the closest match between a point held by one party and a set of points held by another. We evaluate our protocols by implementing a practical privacy-preserving fingerprint matching system.

1 Introduction

Matching biometric data is critical to many identification systems including fingerprint- and face-recognition systems widely used in law enforcement. Such systems typically consist of a server-side database that holds a set of biometric readings (with associated records), and clients who submit candidate biometric readings to the server for identification. Formally, we assume a server who holds a database $\langle \mathbf{v}_i, p_i \rangle_{i=1}^{M}$, where $\mathbf{v}_i$ denotes the biometric data corresponding to some identity profile p_i, and a client who holds a biometric reading $\mathbf{v}'$ and wants to learn the identity p_{i^*} for which $\mathbf{v}_{i^*}$ is the closest match to $\mathbf{v}'$ with respect to some metric (e.g., Euclidean distance), assuming this match is "close enough" (specifically, within some distance parameter ε).

Our work focuses on *privacy-preserving* biometric identification. The goal is to enable biometric identification of the sort described above without revealing any information about the client's biometric data to the server, and without disclosing anything about the database to the client (other than the closest match, if within distance ε, or the non-existence of any close match). We assume both parties are semi-honest; namely, they are assumed to execute the protocol as specified but may try to learn additional information from the transcript of the protocol execution.

Several researchers have considered similar problems in the context of face recognition [5, 20, 15] and fingerprint matching [1] (see Section 8 for further details). A common structure in those works is to start with a *distance-computation phase* in which the distances $d_1, \ldots, d_M$ between the candidate $\mathbf{v}'$ and each of the potential matches $\mathbf{v}_i$ are implicitly computed. At the end of this phase one party holds M random masks $r_1, \ldots, r_M$ and the other party holds $d_1 + r_1, \ldots, d_M + r_M$. The distance-computation phase is followed by a *matching phase* that computes $i^* = \arg\min_i(d_1, d_2, \ldots, d_M)$. Finally there is a *retrieval phase* (which in some previous work is combined with the matching phase) that computes the output for the client: either a profile p_{i^*} (if $d_{i^*} < \varepsilon$) or $\perp$ (if $d_{i^*} \geq \varepsilon$).

1.1 Our Contributions

We describe new protocols that substantially reduce the computation and bandwidth costs of each of the phases in typical privacy-preserving biometric matching protocols.

Distance-computation phase (Section 4)**.** We present a fast, oblivious, Euclidean-distance protocol appropriate for use in many privacy-preserving applications. Our protocol builds on a previous Euclidean-distance protocol by Erkin et al. [5] and adopts the packing technique from Sadeghi et al. [20]. We provide an order-of-magnitude improvement in both computation time and bandwidth by using packing more aggressively.

Matching phase (Section 5)**.** We use Yao's garbled-circuit technique [22] to perform the matching phase. (However, in our work we do not compute i^* explicitly; rather, this phase merely provides the client with sufficient information to recover the corresponding record in the next phase.) Here, each gate of a circuit is associated with four ciphertexts (a *garbled table*) by one party. The collection of garbled tables (the *garbled circuit*) is sent to the other party, who uses information obtained via oblivious transfer to learn the output of the function on the parties' inputs. The computation and

*Work done while at the University of Maryland.

communication costs for both parties are directly related to the number of gates in the circuit. By carefully integrating the subtraction and comparison computations, and by avoiding the need to propagate indices, we provide a circuit that uses dramatically fewer gates than prior work.

Retrieval phase (Section 6)**.** Our most novel contribution is a new backtracking technique that allows oblivious recovery of the record p_{i^*} corresponding to the closest matching vector $\mathbf{v}_{i^*}$. Separating this retrieval step from the matching phase turns out to be more efficient that computing matching-plus-retrieval as one larger garbled circuit. The reason is simple: the record information can be quite large, and so including the records directly in the garbled circuit would dramatically increase the complexity and cost of the computation. Our main insight is to use the intermediate wire labels, a by-product of evaluating the garbled circuit in the matching phase, to efficiently perform oblivious retrieval.

We have implemented our protocols to build an efficient privacy-preserving fingerprint-matching system. For the underlying matching algorithm we could have used the same PCA (principal component analysis) technique used in several privacy-preserving face-recognition systems [5, 20]; however, this would have required a projection phase in addition to the distance-computation phase which would degrade performance. Instead, we use the FingerCode technique [7] (also used by Barni et al. [1]), which only requires secure computation of Euclidean distances. Aspects of our implementation are described in each of the relevant sections, and we report on the overall performance of the system in Section 7.

2 Background

The primary cryptographic tools we use are homomorphic encryption, oblivious transfer, and garbled circuits. We summarize each of these standard techniques briefly here.

Homomorphic encryption. Given a number a, we write the encryption of a using public key pk as $[\![a]\!]_{\mathsf{pk}}$, or simply $[\![a]\!]$ when the public key is clear from the context. An encryption scheme is *additively homomorphic* if given $[\![a]\!]$ and $[\![b]\!]$ it is possible to compute $[\![a+b \pmod p]\!]$, for some integer p which may depend on pk, without the decryption key. (From now on, we leave p implicit.) It follows that given $[\![a]\!]$ and an integer c, one can also compute $[\![c \cdot a]\!]$.

There are many public-key cryptosystems satisfying this property. In our implementation we use Paillier's cryptosystem [16]. In this scheme, the public key is a modulus n; encryption of $m \in \mathbb{Z}_n$ is done by choosing a random $r \in \mathbb{Z}_n^*$ and computing $(1+n)^m \cdot r^n \mod n^2$.

Oblivious transfer. An oblivious transfer protocol allows a *sender* to send one of a possible set of values to a *receiver*; the receiver selects and learns only one of the values, and the sender does not learn which value the receiver selected. A 1-out-of-2 oblivious transfer protocol, denoted OT_1^2, allows a sender holding strings b_0 and b_1 to interact with a receiver, who holds a selection bit $\sigma \in \{0,1\}$, so the receiver learns b_σ while neither the sender nor receiver learn anything else.

Garbled circuits. Yao introduced the idea of using garbled circuits to perform secure two-party computation [22]; Lindell and Pinkas provide a full description and complete proof of security [10]. Garbled circuits enable two semi-honest parties, P_0 and P_1, holding inputs x_0 and x_1, respectively, to compute $f(x_0, x_1)$ for an arbitrary function f without leaking any information about their respective inputs beyond what is revealed by the outcome itself. The idea is for one party (the *circuit generator*) to represent boolean wire values on each wire with a cryptographic key called that wire's *wire label*, and to replace each gate's truth table with a corresponding garbled gate. Garbled gates are constructed by encrypting outgoing wire labels for each gate using an appropriate combination of the two input wire labels. The second party (the *circuit evaluator*) obtains the input wire labels using oblivious transfer, after which the evaluator can evaluate the rest of the circuit without any further communication. For each garbled gate, the evaluator can decrypt exactly one entry for the outgoing wire based on the wire labels she knows for the two input wires. The circuit generator also sends the mapping from wire labels to boolean values for any output wires, so the evaluator can map its final output-wire labels to boolean values.

3 System Overview

Although the techniques we propose could be applied to many different biometric-matching systems, our implementation targets fingerprint recognition.

Fingerprint recognition (or *fingerprint identification*) is the task of searching for the *best* match in a database of fingerprints with a given candidate fingerprint. In contrast, *fingerprint authentication* seeks to determine if a candidate fingerprint matches a particular registered fingerprint. Techniques for matching fingerprints have been extensively studied, and we only provide a brief summary here; Maltoni et al. [12] provides more comprehensive information.

Depending on the sensing technology, fingerprint images exhibit traits at different levels of image quality. At the global level, ridge-lines shapes fall into one of several patterns such as *loop*, *whorl*, and *arch*. At the local level, there are about 150 different types of local ridge characteristics (*minute detail*). At an even finer level, intra-ridge details are identified and used in high-end fingerprint applications.

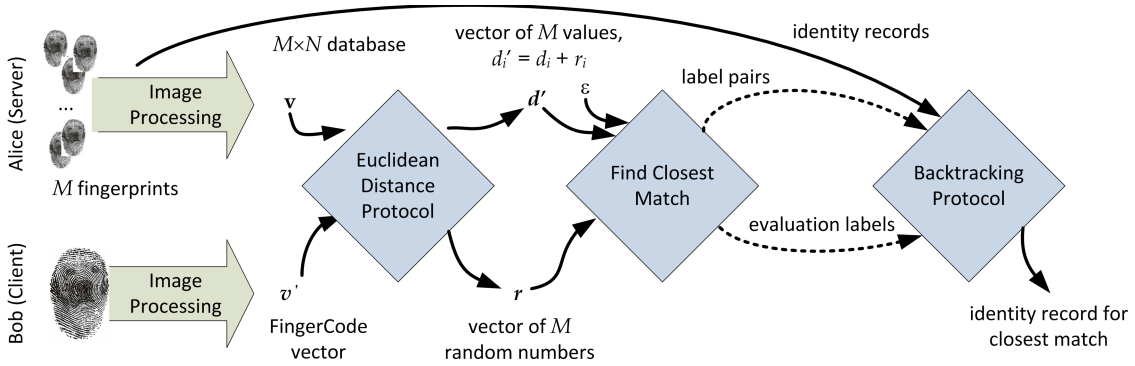

Figure 1. System Overview

In the last decade, many fingerprint-recognition techniques have been developed that combine various features of the fingerprint [2, 18, 19, 21]. Most of them involve sophisticated training and classification algorithms, which are not suitable for developing an efficient privacy-preserving fingerprint recognition system.

In our work we use the filterbank-based approach [3] (also used by Barni et al. [1]) because it provides good accuracy and leads to an efficient privacy-preserving protocol. In this approach, fingerprints are represented by a *FingerCode* derived from the raw fingerprint image. For our purposes, it is important only to know that the FingerCode of each fingerprint is an N-dimensional (typically $N = 640$) feature vector, each entry of which is an 8-bit integer. The distance between two fingerprints is defined as the Euclidean distance between the two corresponding feature vectors.

We assume a server ("Alice") holding a database that contains M fingerprint feature vectors, each of which is associated with corresponding profile information (e.g., name, age, criminal record). Given a candidate fingerprint image, a client ("Bob") first locally derives the associated FingerCode feature vector. An advantage of using the filterbank-based algorithm is that fingerprint images can be transformed to their feature vector representation locally; hence, each party can do the relevant image-processing on its own using a standard image-processing program.

Our system design can be decomposed into the three stages shown in Figure 1: a secure Euclidean-distance protocol (Section 4), a secure closest-match protocol (Section 5), and an oblivious retrieval protocol (Section 6). In our implementation, the first two phases are each divided into a preparation stage which can be performed off-line (i.e., independently of the client's candidate fingerprint) and an on-line execution stage.

Looking only at the feature vectors (and ignoring the associated profile records for now), we may view the server's database as an $M \times N$ matrix $[v_{i,j}]_{M\times N}$, where each $v_{i,j}$ is an 8-bit integer. Each of the M row vectors, written as $\mathbf{v}_i$ (where $1 \le i \le M$), represents the vector corresponding to some fingerprint. The database can also be viewed as N column vectors $[\mathbf{c}_1, \ldots, \mathbf{c}_N]$. The client's input is denoted by a vector $\mathbf{v}' = [v'_1, \ldots, v'_N]$, where each v'_j is an 8-bit integer.

Our Euclidean-distance protocol is based on an additively homomorphic encryption scheme. The server's input to this protocol is the matrix $[v_{i,j}]_{M\times N}$, and the client's input is a single feature vector $[v'_1, \ldots, v'_N]$. The squared Euclidean distance between $\mathbf{v}_i$ and $\mathbf{v}'$ is denoted d_i:

$$d_i = \sum_{j=1}^{N} (v_{i,j} - v'_j)^2.$$

At the end of this protocol, the server obtains a list of M random numbers $\mathbf{r} = [r_1, \ldots, r_M]$ and the client obtains $\mathbf{d}' = [d'_1, \ldots, d'_M]$, where $d'_i = d_i + r_i$. (Addition here is done over the integers, but statistical masking of the d_i values can be achieved by setting the bit-length of r_i large enough relative to the maximum possible value of d_i.)

In the second phase, the client and server (implicitly) compute the minimum difference between the candidate fingerprint vector and the vectors in the database, if this distance is less than some threshold ε. If no fingerprint in the database is within distance ε of the candidate fingerprint, the client (implicitly) receives a "no match" response; otherwise, the client (implicitly) learns the index i^* of the closest match. We implement this phase using a garbled circuit which takes as inputs $\mathbf{r}$ and $\mathbf{d}'$ as output by the previous phase. Conceptually, the garbled circuit computes $\mathbf{d}' - \mathbf{r}$ to produce $\mathbf{d}$=$[d_1, \ldots, d_M]$, which is then fed into a minimum circuit to find the minimal component d_{i^*}. Finally, d_{i^*} is compared to ε to see if it is a close-enough match. For efficiency, our design combines the difference, comparison, and threshold check into one circuit.

In the final phase, the client learns the record associated with the closest match (if a close enough match exists). This is done using the wire labels from the garbled circuit used in the previous phase. Section 6 describes the backtracking

protocol we use to efficiently and obliviously retrieve the matching profile record.

As in other scenarios where garbled circuits are used, our system is flexible in terms of who learns the result. If only the server should learn the outcome, then the client can just send back the wire labels of the final outputs. Then the server, who knows the label-to-signal mappings, learns the index of the closest match (if that match is close enough).

On the other hand, if the client is supposed to be the only party learning the final outcome, the server only needs to send to the client a pair $\langle r'_0 \| H_{\lambda^0}(r'_0), r'_1 \| H_{\lambda^1}(r'_1) \rangle$, where H is a random oracle, $r'_0, r'_1 \xleftarrow{U} \{0,1\}^n$, and λ^0, λ^1 are the wire labels denoting 0 and 1 respectively, for each of the final output wires, then followed by the backtracking tree protocol. In this case, the client knows from the additional pairs how his final output wire labels binds to wire signals. After seeing if a match really happens, he then decides whether to go on evaluating the backtracking tree.

Note that even a secure two party computation can leak information about both participants' private inputs via revealing the correct final output. For both fingerprint recognition system setups above, the server's threshold value ε can be used as a security parameter that controls the information leakage through final outputs. By choosing a small enough ε, the server can ensure that no information (other than the absence of a close match) is revealed unless the client has a candidate fingerprint that is a close match to one in the database. This satisfies the requirements for applications such as identifying a criminal while keeping the database of known criminals secret while preserving the privacy of non-criminals.

4 Euclidean-Distance Protocol

As has been observed before (see, e.g., Erkin et al. [5]), computation of the squared Euclidean distance d_i between $\mathbf{r}_i$ (one of the vectors in the server's database) and $\mathbf{v}'$ (the candidate vector) can be broken into three parts:

$$\begin{aligned} d_i = \|\mathbf{v}_i - \mathbf{v}'\|^2 &= \sum_{j=1}^{N} (v_{i,j} - v'_j)^2 \\ &= \underbrace{\sum_{j=1}^{N} v_{i,j}^2}_{S_{i,1}} + \underbrace{\sum_{j=1}^{N} (-2v_{i,j} \cdot v'_j)}_{S_{i,2}} + \underbrace{\sum_{j=1}^{N} {v'_j}^2}_{S_3} \end{aligned}$$

(Note that the last component does not depend on i.)

We will compute each of the above values in encrypted form. In our application (in contrast to the settings considered in [5, 20]) the client holds $\mathbf{v}'$ and so can compute $[\![S_3]\!]$ locally. Similarly, the server can compute $[\![S_{i,1}]\!]$ locally. Thus, all that remains is to provide a secure method for evaluating $[\![S_{i,2}]\!]$.

The distance protocol in the basic settings of both Erkin's secure face recognition systems begins with the server (the server) having all the $v_{i,j}$ values while the client (the client) has the v'_j value encrypted under the server's public key (pk_S). For the secure face recognition algorithm, the client cannot learn v'_j's because v'_j's can carry information about the eigenfaces that must be kept private to the server. The $\left[\!\left[v'_j\right]\!\right]$'s are derived from eigenfaces using secure dot-product computations. In our application, the client can compute $\mathbf{v}'$ from the fingerprint image directly, so can compute $[\![S_3]\!]$ locally. This is similar to the *Public Eigenfaces* scenario mentioned by Erkin et al. [5].

The next subsection describes the previous privacy-preserving Euclidean-distance protocols [5, 20]. Section 4.2 presents our improved protocol that reduces the bandwidth and computation cost by an order of magnitude.

4.1 Prior Euclidean-Distance Protocols

A basic version of the protocol begins by having the client publish a public key pk_C for a homomorphic encryption scheme. The client computes $[\![-2v'_1]\!], \ldots, [\![-2v'_N]\!]$ and $[\![S_3]\!]$ and sends these ciphertexts to the server. The server computes $[\![S_{i,1}]\!]$ (for $1 \le i \le M$) by herself. The server then computes $[\![S_{i,2}]\!]$ (for $1 \le i \le M$) using the following formula:

$$[\![S_{i,2}]\!] = \prod_{j=1}^{N} [\![-2v'_j]\!]^{v_{i,j}} = \left[\!\left[\sum_{j=1}^{N} (-2v_{i,j} \cdot v'_j) \right]\!\right].$$

Finally, for $1 \le i \le M$ the server chooses random r_i from some appropriate range and computes:

$$\begin{aligned} [\![d'_i]\!] &= [\![d_i + r_i]\!] = [\![S_{i,1} + S_{i,2} + S_3 + r_i]\!] \\ &= [\![S_{i,1}]\!] \cdot [\![S_{i,2}]\!] \cdot [\![S_3]\!] \cdot [\![r_i]\!]. \end{aligned}$$

The $\{[\![d'_i]\!]\}$ are sent to the client, who decrypts and recovers $d'_1, \ldots, d'_M$; the server outputs $r_1, \ldots, r_M$.

As noted by Sadeghi et al. [20], *packing* can be applied to save bandwidth in the second round of the protocol. The basic idea is to send $\frac{M}{\ell}$ ciphertexts of the form $[\![d'_i \| d'_{i+1} \| \cdots \| d'_{i+\ell}]\!]$ instead of M ciphertexts of the form $[\![d'_i]\!]$, where the maximum value of ℓ depends on the maximum range of the d'_i and the bit-length of plaintexts in the encryption scheme being used. If each d'_i satisfies $0 \le d'_i < 2^\sigma$, then ciphertexts of the required form can be computed as,

$$[\![d'_1 \| d'_2 \| \cdots \| d'_{\ell+1}]\!] = \prod_{j=1}^{\ell+1} [\![d'_j]\!]^{2^{(\ell+1-j)\cdot\sigma}}.$$

Note that this method of packing cannot be applied to the initial message from the client to the server in the basic protocol described above, hence it only reduces the bandwidth required for the final response from the server to the client.

Improved Privacy-Preserving Euclidean-Distance Protocol

Input to the server: a matrix $\{v_{i,j}\}_{M\times N}$.
Input to the client: a vector $\mathbf{v}' = [v'_1, \ldots, v'_N]$.

Output of the server: M random integers $[d'_1, \ldots, d'_M]$, where $d'_i = d_i + r_i$.
Output of the client: M integers $[r_1, \ldots, r_M]$.

Preparation:

1. The server generates a key pair $\langle \mathsf{pk}_S, \mathsf{sk}_S \rangle$.
2. For $1 \le j \le N$, the server computes $[\![2\mathbf{c}_j]\!]_{\mathsf{pk}_S}$.
3. The server computes $[\![\mathbf{S}_1]\!] = [\![S_{1,1} \| S_{2,1} \| \cdots \| S_{M,1}]\!]$.
4. The server sends $[\![2\mathbf{c}_1]\!], \ldots, [\![2\mathbf{c}_N]\!]$, and $[\![\mathbf{S}_1]\!]$ to the client.

Execution:

Server		Client
③ The server decrypts to get $[d'_1, \ldots, d'_M]$.	② Send $[\![d'_1 \| d'_2 \| \cdots \| d'_M]\!]$. ⟵	① The client chooses $[r_1, \ldots, r_M]$, and computes $[\![\mathbf{d}']\!] = [\![d'_1 \| d'_2 \| \cdots \| d'_M]\!]$.

Figure 2. Improved Euclidean-distance protocol.

4.2 Improved Euclidean-Distance Protocol

Our improved protocol, summarized in Figure 2, uses packing more aggressively throughout the protocol to reduce both computation and bandwidth. One key idea is to move as much computation as possible to a pre-processing step that can be done by the server alone (independent of the client's input). Since we expect in most applications the database changes infrequently, the costs of this step are amortized over a series of queries on the same, fixed database. Second, we slice the fingerprint database in columns (that cross-cut individual fingerprint vectors) instead of rows. This enables more efficient use of packing throughout the protocol.

In contrast to the protocol from the previous section, in our protocol most of the computation is done on the client and data is encrypted with the server's public key pk_S. The protocol begins by having the server pack an entire column vector into a single ciphertext. (Here, for simplicity, we assume that M is small enough to pack M scalars of sufficient bit-length into a single ciphertext. If not, the protocol is repeated on sub-matrices of the server's original matrix.) Namely, if $\mathbf{c}_j = \left[v_{1,j}, v_{2,j}, \cdots, v_{M,j}\right]$ is the jth column of the server's input matrix, the server computes an encryption of $\mathbf{c}_j$ as

$$[\![2\mathbf{c}_j]\!] \stackrel{\text{def}}{=} [\![(2v_{1,j}) \,\|\, (2v_{2,j}) \,\|\, \cdots \,\|\, (2v_{M,j})]\!] .$$

Note that this involves only a single encryption, as the server can concatenate the values (padded as necessary) before encrypting. The server also computes

$$[\![\mathbf{S}_1]\!] = [\![S_{1,1} \| S_{2,1} \| \cdots \| S_{M,1}]\!] .$$

The server sends $[\![2\mathbf{c}_1]\!], \ldots, [\![2\mathbf{c}_N]\!]$, and $[\![\mathbf{S}_1]\!]$ to the client.

The client computes $[\![\mathbf{S}_3]\!] \stackrel{\text{def}}{=} [\![S_3 \| S_3 \| \cdots \| S_3]\!]$, followed by $\left[\!\left[2v'_j \cdot \mathbf{c}_j\right]\!\right] = [\![2\mathbf{c}_j]\!]^{v'_j}$ for $1 \le j \le N$. Multiplying these latter ciphertexts together, the client obtains a packed encryption of the $\{S_{i,2}\}$:

$$\begin{aligned}[\![\mathbf{S}_2]\!] &\stackrel{\text{def}}{=} [\![S_{1,2} \| S_{2,2} \| \cdots \| S_{M,2}]\!] \\ &= [\![2v'_1 \cdot \mathbf{c}_1]\!] \cdot [\![2v'_2 \cdot \mathbf{c}_2]\!] \cdots [\![2v'_N \cdot \mathbf{c}_N]\!] .\end{aligned}$$

The client then chooses values $r_1, \ldots, r_M$ from some appropriate range, and computes $[\![\mathbf{r}]\!] \stackrel{\text{def}}{=} [\![r_1 \| \cdots \| r_M]\!]$. Finally, the client sets

$$\begin{aligned}[\![\mathbf{d}']\!] &= [\![\mathbf{S}_1]\!] \cdot [\![\mathbf{S}_2]\!]^{-1} \cdot [\![\mathbf{S}_3]\!] \cdot [\![\mathbf{r}]\!] \\ &= [\![d_1 + r_1 \| \cdots \| d_M + r + M]\!]\end{aligned}$$

and sends this to the server.

Table 1 compares the online computation and communication required for the two protocols. Compared to the protocol from Section 4.1 (even when using packing there), our protocol has several advantages:

1. The initial round of our protocol can be pre-computed by the server based only on its database.
2. Our protocol saves substantial computation because it performs arithmetic over several *packed* scalars using

Protocol	Encryptions	Decryptions	Exponentiations	Bandwidth
Previous (4.1)	$N+M+1$	M	$M(N+1)$	$N+M/\kappa+1$
Improved (4.2)	M/κ	M/κ	MN/κ	M/κ

Table 1. Comparison of two Euclidean-distance protocols.
For our improved protocol, we measure complexity of the second round only (i.e., we assume pre-processing is being done, and tabulate the complexity per query). We let κ denote the number of scalars that can be packed into a single ciphertext. Bandwidth is tabulated in terms of the number of ciphertexts communicated.

a single homomorphic operation. The number of homomorphic operations is reduced by the number of scalars, κ, that can be packed into a single ciphertext.

3. Our protocol uses less bandwidth. If we assume a client making multiple queries, then the communication cost of the first round can be amortized over a large number of queries (with the client only sending a new second-round message for each query).

Security. Security of this protocol (as well as the prior protocols [5, 20]) depends on the **r** values adequately masking the entries of **d**. We stress that addition here is computed *over the integers* rather than modulo some value; thus, we obtain statistical hiding rather than perfect hiding. Concretely, if each d_i is a δ-bit integer and r_i is a uniform ρ-bit integer, then releasing $d'_i = d_i + r_i$ gives statistical security roughly $2^{\delta-\rho}$ for the value of d_i. (Formally, for any fixed d_i^0, d_i^1 the statistical difference between the random variables $d_i^0 + r_i$ and $d_i^1 + r_i$ is approximately $2^{\delta-\rho}$.) By choosing ρ suitably, we can make this probability arbitrarily low. Increasing the maximum mask value, however, reduces κ as discussed next.

Packing Implementation. Suppose each of $v_{i,j}$ and v'_j is a σ-bit integer and each additive random mask r_i is a ρ-bit integer. From the formulas for computing d_i and d'_i, it is clear that $\delta = 2\sigma + \lceil \log N \rceil$ bits are sufficient to represent d_i; as for d'_i, since we need $\rho > \delta$ for statistical security we see that $\delta' = \rho + 1$ bit suffice to represent d'_i.

Let $\theta \geq \delta'$ denote the number of bits designated for each of the κ units being "packed" in one ciphertext. We cannot set $\theta = \delta'$ because we need to handle possible overflow of intermediate values in the computation. We allocate $\sigma + \lceil \log N \rceil$ bits for overflow; thus,

$$\begin{aligned} \theta &= \delta' + \sigma + \lceil \log N \rceil \\ &= \rho + 1 + \sigma + \lceil \log N \rceil. \end{aligned}$$

As a concrete example, for $\sigma = 8, N = 640, \rho = 32$ we get $\theta = 51$. Therefore, if a 1024-bit modulus is used in Paillier's cryptosystem, $\kappa = 20$ units can be packed into a single ciphertext.

Results. Table 2 provides a comparison between our improved distance protocol and the standard protocol (from Section 4.1) as the size of the modulus for Paillier's encryption varies. Details on the implementation and experimental setup are provided in Section 7.

The results are consistent with our quantitative analysis in Table 1 and demonstrate nearly twenty-fold improvements in both time and bandwidth for a 1024-bit Paillier modulus. Our distance protocol also has better scalability with respect to the security parameter of Paillier's cryptosystem. This is because as the length of the modulus increases we can pack more values into each ciphertext: thus, e.g., each encryption with a 2048-bit modulus can be used to perform twice as many underlying computations as with a 1024-bit modulus.

5 Finding the Closest Match

This stage begins with the server knowing $[d'_1, \ldots, d'_M]$ and the client knowing $[r_1, \ldots, r_M]$. This stage can be viewed as allowing the client to learn the index i^* minimizing $d_i = d'_i - r_i$, assuming $d_{i^*} < \varepsilon$. In fact, though, the client does not learn i^* explicitly; rather, the client learns wire labels for "active" wires in a garbled circuit prepared by the server, and this will be sufficient to enable to client to learn the desired record (that corresponds to index i^*) in the backtracking stage we describe in Section 6.

Section 5.1 describes the circuit we evaluate securely to find the closest match, and Section 5.2 explains how we implement the matching phase using this circuit.

5.1 Circuit Design

The overall functionality of this stage is implemented by the circuit SubReduceMin shown in Figure 3. The server's inputs are $[d'_1, \ldots, d'_M]$ and the client's inputs are $[r_1, \ldots, r_M]$, where each of these values is an l-bit integer; the parties also know ε, a k-bit threshold value ($k < l$).

We use several sub-circuits to implement the desired functionality. The SubReduce circuit is used, roughly, to compute the difference $d'_i - r_i$ and then output either the low-order k bits of the result or ε. Formally, this sub-circuit

Paillier Modulus		1024		2048		3072	
Time/Bandwidth		s	KB	s	KB	s	KB
Exec.	Standard	123.6	190.8	574.3	350.6	1510.6	511.4
	Improved	6.8	9.8	14.4	8.9	23.9	8.3
	Savings	94.5%	94.9%	97.5%	97.5%	98.4%	98.4%

Table 2. Execution Phase Costs for Euclidean Distance Protocols.

computes the function

$$\mathsf{SubReduce}(d_i', r_i, \varepsilon) = \begin{cases} \varepsilon, \text{if } d_i' - r_i \geq 2^k \\ k \text{ low-order bits of} \\ \qquad d_i' - r_i, \text{otherwise} \end{cases}$$

Figure 4(a) shows how the SubReduce circuit. Our starting point in building this sub-circuit was the work of Kolesnikov et al. [8], and the SUB and MUX sub-circuits are taken directly from their work. However, we reduce the size of the overall circuit by avoiding unnecessary comparisons and removing the need to propagate indexes.

First, instead of computing a full comparison between $d_i' - r_i$ and ε, we compute the logical OR of the high-order $l-k$ bits of the difference. If the result is 1, then $d_i' - r_i$ is greater that ε, and there is no need to compare the other bits. The output of the $(l-k)$-bit OR gate (implemented as a tree of binary OR gates) is used as the selector bit for a MUX that selects between the low-order k bits of the difference and ε. Since the bit length of ε is significantly smaller than that of $d_i + r_i$, this modification substantially reduces the number of gates.

Next, we compute the minimum of the values output by the M SubReduce circuits. This is done using a M-to-1 Min circuit which is simply a tree of 2-to-1 Min circuits (with the latter being constructed as in [8]). The output is then compared with ε using another 2-to-1 Min circuit.

In contrast to the work of [8], we only need to compute the minimum value rather than the *index* of the minimum value. This is a consequence of the backtracking protocol that we present in the next section. This allows us to reduce the number of gates by roughly $M - \log M$ overall.

Table 3 summarizes the number of non-XOR gates in each of our circuits (using the free-XOR technique [9], XOR gates do not contribute significantly to the cost of the garbled circuit since they do not require any encryption operations).

SubReduce	2-to-1 Min	M-to-1 Min	SubReduceMin
$2l-1$	$2k$	$2k(M-1)$	$(2l+2k-1)M$

Table 3. The number of non-free binary gates in each circuit.

5.2 Implementation

To implement the matching phase, we follow the standard garbled-circuit methodology with the exception that there is no need to send the client the semantic wire mappings at the end of the protocol. Our implementation has the following stages:

1. The server prepares a garbled version of the circuit described in the previous section, in the standard way. The resulting garbled circuit is sent to the client. The server also sends the client the wire labels corresponding to its own input bits.

2. The client and server use oblivious transfer so that the client can obtain the wire labels corresponding to its

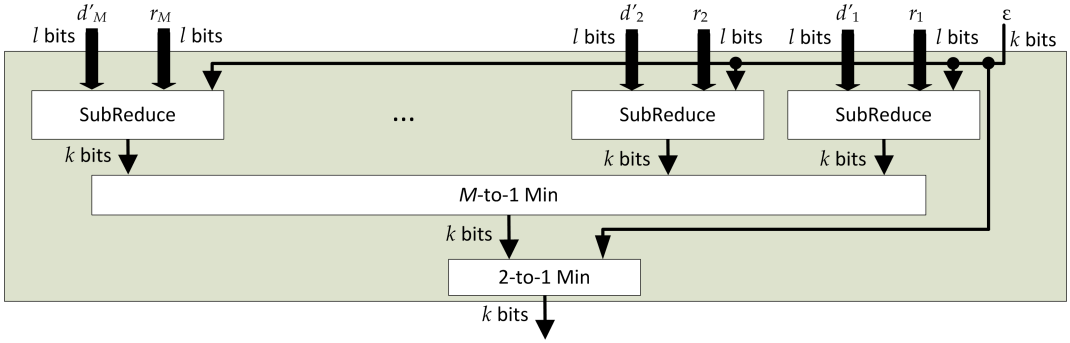

Figure 3. SubReduceMin **circuit for finding the closest match.**

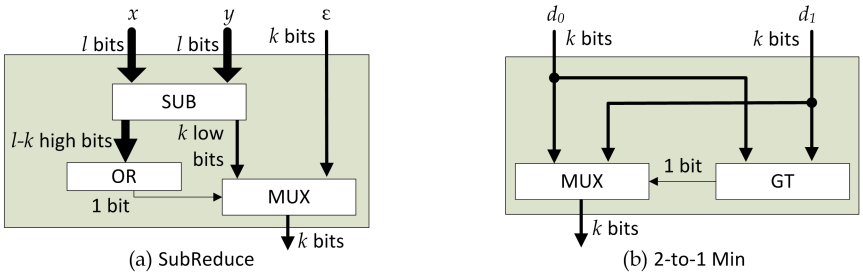

Figure 4. Circuits for SubReduce **and** 2-to-1 Min**.**

own input bits. We use OT extension [6] to reduce an arbitrary number of OTs to secure evaluation of just k OTs, where k is a statistical security parameter. As our base OT we use the Naor-Pinkas protocol [14] which achieves semi-honest security based on the decisional Diffie-Hellman assumption. We also use the standard technique of pre-processing [4] to push most of the cost of the oblivious transfer step into a pre-computation phase. The details of our oblivious transfer protocol are described in the Appendix.

Differing from prior work, however, in our implementation the server does *not* send the client the mappings from any of the wire labels to actual bits. Thus, the client receives no semantic output from this phase. Nevertheless, we show in the following section how the client can use the wire labels that it learned in this step to recover the record corresponding to the closest match.

6 Backtracking Protocol

At the end of the circuit evaluation, the client has one of each pair of wire labels. In traditional garbled circuit protocols, the last step is to convert the final output wire labels to meaningful values. Our solution shows that the overhead of propagating the indexes and retrieving the information can be avoided while enabling arbitrary information about the match to be transmitted obliviously.

In a conventional garbled circuit, wire signals (0 or 1) are denoted by randomly-chosen nonces known as *wire labels*. The bindings between wire labels and the wire signals are known by the circuit generator, but hidden from the circuit evaluator (except for the bindings for the final output wires which are disclosed to reveal the result). Wire labels are merely used for intermediate computation. However, these apparently meaningless nonces can be exploited in later stages of the protocol. We take advantage of the key property of garbled circuit evaluation: the evaluator only learns one of the two possible output wire labels for each gate, as determined by the obliviously-selected input wire labels and evaluation of the rest of the circuit. These wire labels can serve as keys for encrypting useful information.

The wire labels of the output wires of GT comparison circuits in the n-to-1 Minimum tree can be used to reveal a path from the inputs to the minimum value. Figure 5 shows an example comparison tree for a four record database. In each of the 2-to-1 Min circuits the GT circuit takes two inputs and outputs a bit indicating which value is greater. We denote that bit as $g_{h,i}$ and the corresponding wire labels $\lambda^0_{h,i}$ (when the greater than comparison is false) and $\lambda^1_{h,i}$ (when the comparison is true). When the more closely matching entry match is on the left side of this gate, the client learns $\lambda^0_{h,i}$; when it is on the right side he learns $\lambda^1_{h,i}$. The final gate in the diagram compares the best match with ε. We use the g_ε output to prevent the client from learning any information from the backtracking tree when there is no match within the ε threshold.

Our backtracking tree protocol involves a tree generator (the server), who produces and sends a tree encoding encrypted paths to the profile records, and a tree evaluator (the client), who follows a single path through the tree to open the best matching profile record. To generate a backtracking tree, the server starts by filling the leaf nodes (level 0) with the desired information corresponding to each database entry. Then, she fills in the internal nodes of a binary tree with those leaves, as illustrated by the left tree in Figure 6. Note that the structure of this tree is identical to that of the comparison tree in Figure 5.

Next, she generates new nonces for each non-leaf node in the tree, and encrypts those nonces with keys that combine the appropriate wire labels and the nonce of its parent node. The wire label used for node h,i (the i^{th} node at level h) is either $\lambda^0_{h,i}$ or $\lambda^1_{h,i}$, depending on whether it is the left or right child of its parent). Thus, the label pair the server uses for each node comes from the labels of $g_{h,i}$ in the corresponding 2-to-1 Min circuit she generated for the match-finding protocol. The root node is encrypted using λ^0_ε, the label the client will learn when the closest match is closer than ε. The right tree in Figure 6 shows the backtracking tree corresponding to the example circuit in Figure 5.

Starting from the root of the tree, which the client can only open when $g_\varepsilon = 0$, the client can follow a single path

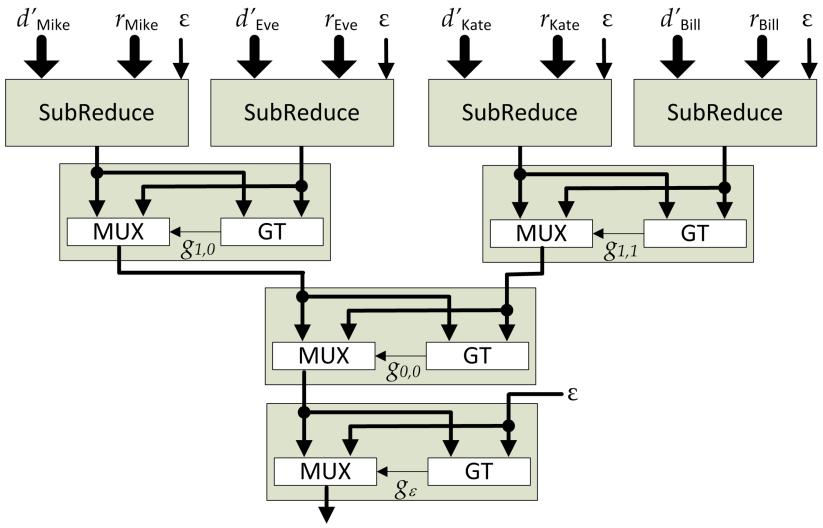

Figure 5. Example Find Closest Match Circuit

through the tree, learning the keys along that path, and eventually the key needed to decrypt the encrypted record information at the leaf. When he evaluates the garbled SubReduceMin circuit, the client obtains either $\lambda^0_{h,i}$ or $\lambda^1_{h,i}$ from each label pair. Since each key in the backtracking tree depends on a complete path from the root to that tree, this means the client can only open a single path through the tree; specifically, the single path from the root to the leaf corresponding to the closest match.

Figure 7 summarizes the backtracking protocol. The algorithms to generate and evaluate the tree are shown in Algorithm 1 and Algorithm 2. For the tree generation algorithm (Algorithm 1), the inputs a vector of the profile data to send and an array of the wire label pairs for each comparison gate in the generated circuit. The output is the backtracking tree, but only the node labels are transmitted to the evaluator. For the tree evaluation algorithm (Algorithm 2), the inputs are the tree (where the label for each node is the encrypted label in the generated tree) and the wire labels learned by the evaluator in evaluating the garbled circuit. The output is the decrypted profile information for the closest matching entry, if there is one within ε.

Security. The backtracking tree protocol is secure if both of the following properties hold:

1. The generator (the server) gains nothing.
2. The evaluator (the client) gains nothing other than the data associated with the closest match.

The first property trivially holds since the client sends nothing back to the server. The second property follows from two facts:

1. With a *semantically secure* encryption scheme, no information is leaked by the encryption (i. e., without also revealing the keys).
2. *Wire labels* in a garbled circuit convey no information unless their mappings to *wire signals* are known somehow. This follows from the garbled circuit security proof [10].

In every iteration of the loop in Algorithm 2, the client only gets to know the nonce of one of current_node's two children, and proceeds using that value. The whole subtree of the failed branch remains unknown to the client since the nonce is needed to open any configurations on that subtree. Thus, the client can only follow a single path in the tree corresponding to the path leading to the closest match.

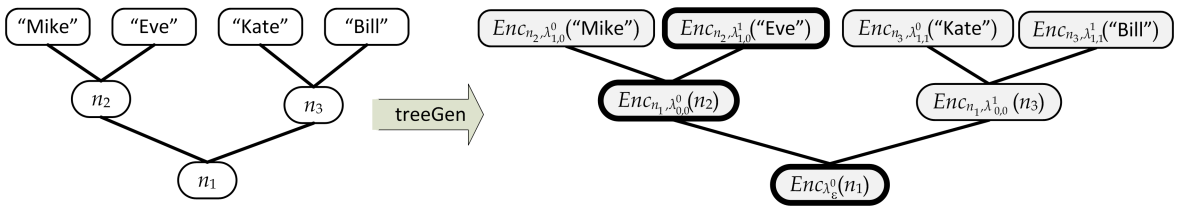

Figure 6. Backtracking Tree Example

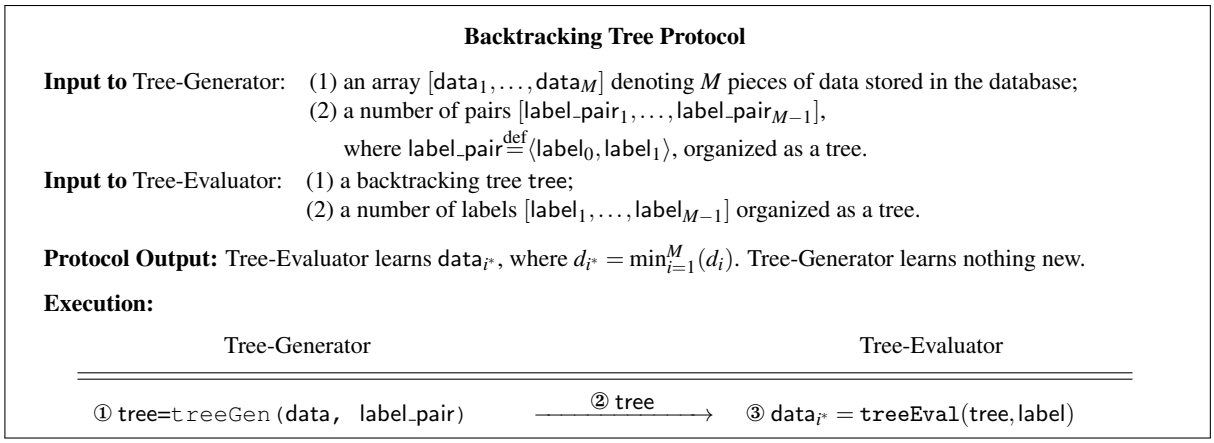

Figure 7. The Backtracking Tree Protocol.

Note that if the server's database is released in encrypted format beforehand as in the Improved Euclidean Distance protocol, the server may not want the client to learn extra information from the position of the matched records. Hence, the server should randomly permute the order of database records before beginning the Euclidean distance protocol. This random permutation needs only to be done once, since once the database is permuted relative positions of opened records reveal no information.

Multiple matches. In the unlikely case where multiple fingerprints match the candidate fingerprint equally well, a straightforward implementation of garbled circuits always returns either the left-most or the right-most matched leaf node. This poses potential threats to the server's privacy. For example, if the minimum tree is known to always return the left-most matched leaf node, then the client learns there can't be another equally-well matched fingerprint in the server's database. A straightforward but costly way to fix this vulnerability would be to add an equality test circuit and a MUX for each 2-to-1 Min to randomly choose a number to output. Instead, we use a simple fix that does not require adding any new circuitry. We modify the circuit generator to randomly choose the internal carry-in bit for GT, instead of always using signal 0. Setting this internal carry-in bit to 1 is equivalent to making the GT test $x+1 > y$. This change does not affect the functionality when $x > y$. When $x = y$, the modified GT outputs 0 (if the internal carry-in bit is 0) and 1 (if the internal carry-in bit is 1) with equal probability $\frac{1}{2}$.

7 Evaluation

To measure the impact of our improvements and evaluate the practicality of privacy-preserving biometrics, we implemented a privacy-preserving fingerprint matching system. Our implementation comprises about 5400 lines of Java 1.6 code, available from http://mightbeevil.org.

We set up the server and the client on separate machines

Algorithm 1 treeGen(data, label_pairs)

Require: data.length $= M$; label_pairs.length $= M-1$; $M = 2^{\mathsf{h}}$, where h is an integer.
1: Generate a perfect tree tree of size $2M-1$.
2: Fill the M leaf nodes with the M values in data.
3: **for all** node in tree **do**
4: node.nonce $\xleftarrow{U} \{0,1\}^k$;
5: **end for**
6: **for** $\ell \leftarrow \mathsf{h}-1$ to 1 **do**
7: **for all** node at level ℓ **do**
8: lp $\leftarrow$ label_pairs[pos(node)], the labels for the gate corresponding to node in the tree;
9: node.leftChild.label $\leftarrow \mathrm{Enc}_{\text{node.nonce}||\text{lp.label}_0}$(node.leftChild.nonce);
10: node.rightChild.label $\leftarrow \mathrm{Enc}_{\text{node.nonce}||\text{lp.label}_1}$(node.rightChild.nonce);
11: **end for**
12: **end for**
13: **return** tree;

Algorithm 2 `treeEval(tree, wire_labels)`

```
1: msg ← 0;
2: current_node ← tree.root;
3: while current_node has children do
4:    m ← Dec_{msg||wire_labels[pos(current_node)]}(current_node.leftChild.label);
5:    if m is valid then
6:       msg ← m;
7:       current_node ← current_node.leftChild;
8:    else
9:       msg ← Dec_{msg||wire_labels[pos(current_node)]}(current_node.rightChild.label);
10:      current_node ← current_node.rightChild;
11:   end if
12: end while
13: return msg;
```

connected by a LAN. Both machines are homogeneously configured, each with an Intel Xeon CPU (E5504) running at 2.0GHz. The JVMs are configured with a memory cap of 4GB, both on the server and the client.

We use randomly generated 640-entry vectors as our benchmark. Note that we are not evaluating the fingerprint matching algorithm here, since our privacy-preserving protocol uses exactly the original filterbank-based fingerprint matching algorithm which has been extensively evaluated [7]. The evaluation time is independent of the actual fingerprint vectors. In our experiments, the client's feature vector is randomly picked from the feature vectors in the database.

Our implementation used the following parameters: in the Euclidean-distance protocol, the bit length allocated for each packed value (i.e., θ) was 64 and the bit length of the random mask was 45. We use Paillier encryption with a 1024-bit modulus. We set ε to be a 16-bit integer. In our garbled circuit implementation, 80-bit wire labels are used.

Table 4 shows the running time and bandwidth usage for our protocol as a function of M, the number of entries in the database. We report the computation time and bandwidth for each of eight protocol sub-stages. The first three sub-stages are preparation sub-stages, which do not depend on the candidate fingerprint and only need to be done once: (1) Euclidean distance preparation (*Distance*): the server computing and transmitting the encrypted packed columns and $[\![\mathbf{S}_1]\!]$); (2) Garbled circuit preparation (*Circuit*): the server generating the SubReduceMin garbled circuit (except for the wire labels and garbled tables, which must be regenerated for each execution); (3) OT preparation (*OT*): the preparation steps for the oblivious transfer. The preparation time is dominated by the time required to compute the encrypted distance vectors, which scales approximately linearly with the size of the database. Since this is done only once by the server, though, it is not prohibitively expensive even for large databases. The preparation phase of the OT protocol depends only on the security parameters we choose, so its cost does not scale with M.

The final five sub-stages are the execution sub-stages which must be done once for each candidate fingerprint execution: (4) Euclidean distance protocol execution (*Distance*), (5) the server resetting the initial input wire labels and transmitting the wire labels representing her input to the client (*Reset Labels*), (6) the client learning the wire labels corresponding to his inputs obliviously (*OT*); (7) garbled circuit evaluation, including the server's generating and transmitting the intermediate wire labels and garbled truth tables and the client's evaluating the circuit (*Circuit*), and (8) the backtracking protocol (*Backtracking*), which comprises generating, transmitting, and evaluating the backtracking step.

The distance protocol dominates the execution time. The other two substantial sub-stages are circuit and OT, and the time for the backtracking protocol is negligible. As expected, the results in Table 4 confirm that every sub-stage of the execution phase scales approximately linearly with the size of the database. The bandwidth in the execution phase is dominated by the circuit, due to transmitting a large number of garbled truth tables. This is dominated by traffic from the server to the client, which accounts for 88% of the overall traffic.

8 Related Work

The protocols presented here build upon, and could be applied to improve the efficiency of, several previous systems for privacy-preserving biometric identification.

Erkin et al. [5] developed an efficient privacy-preserving face recognition system based on the standard Eigenfaces recognition algorithm. Similar to our work, it also computes Euclidean distances between vectors using additive homomorphic encryption. Their work does not use garbled circuits but relies heavily on homomorphic encryption for

Database Size (M)		128		256		512		1024	
Time/Bandwidth		s	KB	s	KB	s	KB	s	KB
Prep.	Distance	145.08	1288.99	277.35	2577.25	555.87	5153.77	1089.42	10306.81
	Circuit	0.35	None	1.09	None	2.95	None	6.28	None
	OT	0.52	21.91	0.48	21.91	0.48	21.91	0.45	21.91
Exec.	Distance	1.68	2.93	3.36	5.21	6.82	9.79	13.33	18.95
	Reset Labels	0.01	57.94	0.03	115.72	0.08	231.28	0.24	462.40
	OT	0.13	237.13	0.25	467.69	0.61	928.82	1.38	1851.06
	Circuit	0.39	656.14	0.67	1313.64	1.58	2628.64	3.12	5258.63
	Backtracking	0.01	12.70	0.02	25.45	0.03	50.95	0.04	101.94
Exec Sub-Total		2.22	966.84	4.33	1927.71	9.12	3849.48	18.11	7692.98

Table 4. Running Time (seconds) and Bandwidth (KB) for Protocol Phases

all the core computation. This limits the scalability of their system. The length of vectors is 12 in their system (compared to 640 in our experiments). For a database of 320 faces, the whole system takes 18 seconds online computation to serve a query, and generates about 7.25MB network traffic.

Sadeghi et al. [20] improved the efficiency of Erkin's work by constructing a hybrid protocol that uses homomorphic encryption to compute Euclidean distances and garbled circuits for minimum. They also devised the idea of *packing*, which, however, they use merely used to save communication cost. Our Euclidean distance protocol builds on this protocol, but improves its efficiency by also incorporating packing in the computation steps. They do not use any minimum circuit to identify the best match. Instead, matches were found by securely comparing distance values with individual threshold values. In contrast to our work, Sadeghi et al. [20] generated their garbled circuit using a generic compiler FairplaySPF [17], which is in turn based on Fairplay [11]. Such compilers are convenient, but cannot take advantage of application-specific properties to develop more efficient custom circuits.

SCiFI [15] is a practical privacy-preserving face identification system. It uses a component-based face identification technique with a binary index into a vocabulary representation. The distance between faces is the Hamming distance between their bit-vectors. One notable design choice the authors made for SCiFI is that both the secure Hamming distance and secure minimum algorithms are purely based on additive homomorphic encryption and oblivious transfer. The authors present several optimization techniques specific to their application. In contrast, we argue that a hybrid scheme combining both homomorphic encryption and garbled circuits tends to be superior. They report that identification takes 31 seconds of online computation for a database of size 100 (with 900-bit vectors, in comparison to our 640-byte = 5120-bit vectors), while no bandwidth consumption is reported. Rather than computing the global minimum, their implementation produces the indexes of all entries within a threshold value of the candidate.

The most similar work to ours is the privacy-preserving fingerprint authentication by Barni et al. [1] which uses the same FingerCode biometric as we do. Like Erkin et al.'s approach, it is also a system based purely on homomorphic encryption. They do not support the computation of global minimum, but instead output the indexes of all matches within some threshold. They report results from an experiment with a database of 320x16 7-bit where their protocol completes in 16 seconds and uses 9.11MB bandwidth. In contrast, our system's performance results for the most comparable but larger experiment are 3.47s and 3.76MB for a database of 512x16 8-bit integers.

9 Conclusion

Privacy-preserving computation offers the promise of obtaining results dependent on private data without exposing that private data. The main drawback is that current protocols for privacy-preserving computations are very expensive and impractical for real-scale problems. In this work, we have shown that those costs can be substantially reduced for a large class of biometric matching applications by developing efficient protocols for Euclidean distance, finding the closest match, and retrieving the associated record. Our approach involves using the normal by-products of a garbled circuit evaluation to enable very efficient oblivious information retrieval, and we believe this technique can be extended to many other applications. Our experimental results support the hope that privacy-preserving biometrics are now within reach for practical applications.

Acknowledgements

This work was partially supported by a MURI award from the Air Force Office of Scientific Research, and grants from the National Science Foundation and DARPA. The contents of this paper do not necessarily reflect the position or the policy of the US Government, and no official endorsement should be inferred. The authors thank Yikan Chen and Aaron Mackey for insightful discussions about this work.

References

[1] M. Barni, T. Bianchi, D. Catalano, M. D. Raimondo, R. D. Labati, P. Faillia, D. Fiore, R. Lazzeretti, V. Piuri, F. Scotti, and A. Piva. Privacy-Preserving Fingercode Authentication. In *12th ACM Multimedia and Security Workshop*, 2010.

[2] A. Bazen and S. Gerez. Systematic Methods for the Computation of the Directional Fields and Singular Points of Fingerprints. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2002.

[3] A. Bazen, G. Verwaaijen, S. Gerez, L. Veelenturf, and B. van Der Zwaag. A Correlation-Based Fingerprint Verification System. In *ProRISC2000 Workshop on Circuits, Systems and Signal Processing*, 2000.

[4] D. Beaver. Precomputing Oblivious Transfer. In *15th International Conference on Cryptology (CRYPTO)*, 1995.

[5] Z. Erkin, M. Franz, J. Guajardo, S. Katzenbeisser, I. Lagendijk, and T. Toft. Privacy-Preserving Face Recognition. In *9th International Symposium on Privacy Enhancing Technologies*, 2009.

[6] Y. Ishai, J. Kilian, K. Nissim, and E. Petrank. Extending Oblivious Transfers Efficiently. In *23rd International Conference on Cryptology (CRYPTO)*, 2003.

[7] A. Jain, S. Prabhakar, L. Hong, and S. Pankanti. Filterbank-Based Fingerprint Matching. *IEEE Transactions on Image Processing*, 2000.

[8] V. Kolesnikov, A. Sadeghi, and T. Schneider. Improved Garbled Circuit Building Blocks and Applications to Auctions and Computing Minima. In *Cryptology and Network Security*, 2009.

[9] V. Kolesnikov and T. Schneider. Improved Garbled Circuit: Free XOR Gates and Applications. In *35th International Colloquium on Automata, Languages and Programming (ICAPL)*, 2008.

[10] Y. Lindell and B. Pinkas. A Proof of Security of Yao's Protocol for Two-Party Computation. *Journal of Cryptology*, 22(2), 2009.

[11] D. Malkhi, N. Nisan, B. Pinkas, and Y. Sella. Fairplay — A Secure Two-Party Computation System. In *USENIX Security Symposium*, 2004.

[12] D. Maltoni, D. Maio, A. Jain, and S. Prabhakar. *Handbook of Fingerprint Recognition*. Springer, 2009.

[13] M. Naor and B. Pinkas. Efficient Oblivious Transfer Protocols. In *ACM-SIAM Symposium on Discrete Algorithms*, 2001.

[14] M. Naor and B. Pinkas. Computationally Secure Oblivious Transfer. *Journal of Cryptology*, 18(1), 2005.

[15] M. Osadchy, B. Pinkas, A. Jarrous, and B. Moskovich. SCiFI: A System for Secure Face Identification. In *IEEE Symposium on Security and Privacy (Oakland)*, 2010.

[16] P. Paillier. Public-key Cryptosystems Based on Composite Degree Residuosity Classes. In *17th International Conference on Theory and Application of Cryptographic Techniques (EUROCRYPT)*, 1999.

[17] A. Paus, A. R. Sadeghi, and T. Schneider. Practical Secure Evaluation of Semi-Private Functions. In *International Conference on Applied Cryptography and Network Security (ACNS)*, 2009.

[18] S. Prabhakar and A. Jain. Decision-Level Fusion in Fingerprint Verification. *Pattern Recognition*, 2002.

[19] A. Ross, A. Jain, and J. Reisman. A Hybrid Fingerprint Matcher. *Pattern Recognition*, 2003.

[20] A. Sadeghi, T. Schneider, and I. Wehrenberg. Efficient Privacy-Preserving Face Recognition. In *International Conference on Information Security and Cryptology*, 2009.

[21] H. Xu, R. Veldhuis, A. Bazen, T. Kevenaar, T. Akkermans, and B. Gokberk. Fingerprint Verification using Spectral Minutiae Representations. *IEEE Transactions on Information Forensics and Security*, 2009.

[22] A. C. Yao. How to Generate and Exchange Secrets. In *27th Symposium on Foundations of Computer Science*, 1986.

Appendix: Oblivious Transfer Protocol

Our protocol is summarized in Figure 8. It combines the protocols from Naor and Pinkas [13] (which we refer to as *NPOT*) and Ishai et al. [6] using an aggressive pre-computation strategy to produce an efficient OT protocol.

We denote the i^{th} column vector of a matrix T by $\mathbf{t}^i$, and the i^{th} row vector of T by $\mathbf{t}_i$. The preparation phase can be done before any of the selection bits are known. At the end of the preparation phase, SNDER has k_1 keys key_{i,s_i}, and RCVER has k_1 key pairs $\langle \mathsf{key}_{i,0}, \mathsf{key}_{i,1} \rangle$, where $(1 \le i \le k_1)$. These keys are later used to transmit the matrix Q efficiently. By using pre-computation, the on-line phase of our OT implementation requires only $2(k_1 + m)$ symmetric encryptions and $k_1 + m$ symmetric decryptions.

The correctness and security of this protocol follow directly from the proofs for NPOT [13] and Ishai et al.'s extended OT protocol [6].

Correctness. The RCVER can learn x_{i,r_i} for all $1 \le i \le m$ following this case analysis:

1. If $r_i = 0$, then $\mathbf{q}_i = \mathbf{t}_i$ no matter what value s_i takes. Thus, RCVER knows the key $\mathbf{t}_i$, which is used to encrypt $x_{i,0}$.
2. When $r_i = 1$, the value of s_i selects whether $\mathbf{q}^i = \mathbf{t}^i$, or $\mathbf{r} \oplus \mathbf{t}^i$. However, this "selection" effect is canceled by xor-ing $\mathbf{s}$ and $\mathbf{q}_i$, so that it is always true that $\mathbf{s} \oplus \mathbf{q}_i = \mathbf{t}_i$, which is the key used to encrypt $x_{i,1}$.

Security. The security of our protocol follows from these two points:

1. The RCVER can never learn anything about $x_{i,\overline{r_i}}$ because it is encrypted using a different secret key which differs from that used for x_{i,r_i} by $\mathbf{s}$, the SNDER's random bit vector that is never revealed to the RCVER. The security property of NPOT used in the preparation phase guarantees that the selection bits of $\mathbf{s}$ are not revealed to RCVER.
2. In the first round of communication, either $\mathbf{t}^i$ or $\mathbf{r} \oplus \mathbf{t}^i$ is sent to the SNDER, but not both. Thus, the fact that the SNDER can never learn anything about the RCVER's selection bits $\mathbf{r}$ is derived directly from the security property of NPOT used in the preparation phase [13].

The Oblivious Transfer Protocol

Input to SNDER: m pairs $\langle x_{i,0}, x_{i,1} \rangle$ of l-bit strings, where $1 \le i \le m$.
Input to RCVER: m selection bits $\mathbf{r} = [r_1, \ldots, r_m]$.

Protocol Output: RCVER outputs $\{x_{1,r_1}, x_{2,r_2}, \cdots x_{m,r_m}\}$ while knowing nothing of $\{x_{1,\overline{r_1}}, x_{2,\overline{r_2}}, \cdots x_{m,\overline{r_m}}\}$. SNDER learns nothing.

Preparation:

1. RCVER generates a $m \times k_1$ matrix T of random bits.
2. RCVER generates k_1 pairs $\langle \mathsf{key}_{i,0}, \mathsf{key}_{i,1} \rangle$ of k_2-bit strings, where $1 \le i \le k_1$.
3. SNDER generates a vector $\mathbf{s} = [s_1, \ldots, s_{k_1}]$ of random bits.
4. RCVER and SNDER execute NPOT for k_1 times, where RCVER acts as the sender, SNDER as the receiver. At the i^{th} execution of OT_1^2, the message pair to send is $\langle \mathsf{key}_{i,0}, \mathsf{key}_{i,1} \rangle$, and the selection bit is s_i.

Execution:

SNDER		RCVER
	② $[\mathsf{msg}_1, \ldots, \mathsf{msg}_{k_1}]$. ←	① RCVER prepares k_1 pairs $\mathsf{msg}_i = \langle \mathsf{msg}_{i,0}, \mathsf{msg}_{i,1} \rangle = \langle \mathbf{Enc}_{\mathsf{key}_{i,0}}(\mathbf{t}^i), \mathbf{Enc}_{\mathsf{key}_{i,1}}(\mathbf{r} \oplus \mathbf{t}^i) \rangle$, where $1 \le i \le k_1$.
③ SNDER derives a bit matrix Q, where $\mathbf{q}^i = \mathbf{Dec}_{\mathsf{key}_{i,s_i}}(\mathsf{msg}_{i,s_i})$. Then the SNDER prepares m pairs $\mathsf{msg'}_i = \langle \mathsf{msg'}_{i,0}, \mathsf{msg'}_{i,1} \rangle = \langle \mathbf{Enc}_{\mathbf{q}_i}(x_{i,0}), \mathbf{Enc}_{\mathbf{s} \oplus \mathbf{q}_i}(x_{i,1}) \rangle$, where $1 \le i \le m$.		
	④ $[\mathsf{msg'}_1, \ldots, \mathsf{msg'}_m]$. →	⑤ $\forall 1 \le i \le m$, RCVER outputs $x_{i,r_i} = \mathbf{Dec}_{\mathbf{t}_i}(\mathsf{msg'}_{i,r_i})$.

Figure 8. The Oblivious Transfer Protocol.