

Secure Fingerprint Identification of High Accuracy

Marina Blanton and Siddharth Saraph
 Department of Computer Science and Engineering
 University of Notre Dame
 mblanton@nd.edu, ssaraph@nd.edu

Abstract—The increasing availability and use of biometric data for authentication and other purposes leads to situations when sensitive biometric data is to be handled or used in computation by entities who may not be fully trusted or otherwise authorized to have full access to such data. This calls for mechanisms of provably protecting biometric data while still allowing the computation to take place. In this work, we treat the problem of privacy-preserving matching of two fingerprints, which can be used for secure fingerprint authentication and identification. We utilize traditional minutia-based representation of fingerprints that leads to the most discriminative (i.e., accurate) fingerprint comparisons. Unlike prior work, we design a data-oblivious algorithm that results in the most accurate outcome of fingerprint matching through a more complex minutia pairing approach based on maximum flow in bipartite graphs. This algorithm then leads to secure fingerprint matching solutions of high security standards. The complexity of our solution is higher than those of some other available protocols, but nevertheless we show that our techniques still efficiently compare two fingerprints with provable security guarantees. That is, they run in a similar amount of time to those with simpler matching mechanisms which are not guaranteed to find the best matching.

I. INTRODUCTION

Biometric authentication and other uses of biometric data are becoming more prevalent today in a variety of applications, which was prompted in part by recent advances in biometric recognition. Large-scale collections of biometric data in use today include, for example, fingerprint, face, and iris images collected by the US Department of Homeland Security (DHS) from visitors [27]; fingerprint and iris images collected by the government of India from (more than billion) citizens [34]; iris, fingerprint, and face images collected by the United Arab Emirates (UAE) Ministry of Interior from visitors [36]; and adoption of biometric passports in several countries. It is evident that biometric authentication and identification have advantages over alternative mechanisms such as good accuracy and unforgeability of biometry. Biometric data, however, is highly sensitive and, once leaked, cannot be revoked or replaced. This

calls for stringent protection of biometric data while at rest and when used in applications.

The above means that biometric data cannot be easily shared between organizations or agencies, but there are often legitimate reasons for computing with biometric data that belong to different entities. As an example, a private investigator can be interested in knowing whether a biometric she captured appears in the government’s criminal database, but without disclosing the biometric if no matches are found. Similarly, two organizations or collaborating governments might want to determine which individuals, if any, appear simultaneously in their respective databases without revealing any additional information. A solution to enabling such computation while protecting privacy of the data is to employ secure multi-party computation techniques, which compute the result without disclosing any additional information.

In this work we focus on fingerprint data due to popularity and good accuracy of this type of biometry. We formulate the problem of private, or secure, fingerprint matching as follows: one party A possesses private fingerprint X and another party B possesses another private fingerprint Y . The parties would like to know whether the fingerprints they possess correspond to the same individual. In the above formulation, this is the problem of biometric authentication. More generally, one of the parties, say, party B can be in possession of a database D . Then the computation consists of securely comparing X to all $Y \in D$ and identifying the biometrics that matched (if any), which is known as the problem of biometric identification. There also could be variants to this computation such as outputting at most one index from the database that corresponds to the biometric with the smallest distance among all biometrics that matched the query. Such variants can be easily derived from a solution that compares two fingerprints and we thus primarily concentrate on techniques for comparing two fingerprints X and Y . We also note that the solution we develop can be used in other settings such as outsourcing the computation to third parties by one or more data owners as no information about the data

is revealed throughout the computation. For concreteness of presentation, we, however, adhere to the two-party formulation of the problem.

Prior literature such as [8], [9], [30] already contains solutions for secure fingerprint matching. What, however, distinguishes our work from prior results is achieving a high level of accuracy while maintaining efficiency of the algorithm. In particular, a minutiae-based fingerprint representation consists of a number of minutiae points in a two-dimensional space together with their orientation. Matching of points from one fingerprint with points from another fingerprint consists of computing Euclidean distances and angle difference between the points and marking two points as a possible match if both distances are within certain thresholds. The next step consists of pairing points from X with “possible match” points from Y and the number of points that could be paired together determines whether the fingerprints were a match or not. A simple way to determine the pairing is to associate a point from X to the closest point from Y that has not already been paired with another point from X . A more involved algorithm, on the other hand, would try to find a pairing of the largest possible size, where a point from X is paired with a “possible match” point from Y , but not necessarily the closest to it. This results in more accurate matching of two fingerprints, but incurs higher computational cost.

The latter approach has not been explored in security literature and requires new techniques for secure processing of the data. We note that the new techniques are necessary even if a general-purpose mechanism for securing computation (such as garbled circuits or secret sharing) are to be used. In this work, we reduce the problem to that of computing the size of maximum flow in a flow network and build techniques for solving it in secure computation context. Beyond application of the solution to fingerprint matching, the algorithm may be applicable to other domains and be of independent interest. Despite higher complexity of the algorithm, we show through experimental evaluation that the solution nevertheless offers fast performance.

II. RELATED WORK

Work on secure multi-party computation was initiated in Yao’s seminal work [38] that showed that any computable function can be securely evaluated by representing it as a boolean circuit. Since then a large number of both general and special-purpose techniques followed and their overview is beyond the scope of this work. We thus mention only the most relevant results. There are currently a number of recent tools and compilers (such as Fairplay [23], VIFF [11], Sharemind [10], PICCO [39], and others), which can securely evaluate functions on private data in a variety of settings.

In the context of biometric matching, results available today include work on secure face recognition ([12], [29] and others), DNA matching ([35], [6], and others), iris code comparisons ([9], [7]), fingerprint comparisons ([2], [8], [30]), and speaker authentication ([28], [1]). Each biometric type has a unique representation and the corresponding algorithm for comparing two biometrics, which prompted the need to design separate solutions for different biometric modalities.

The first privacy-preserving protocol for fingerprint identification is due to Barni et al. [2] who utilize the so-called FingerCode approach [18] for comparing two fingerprints and built a solution using a homomorphic encryption scheme. 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. While the use of FingerCodes can result in fast comparisons of fingerprints, the approach is not suitable for biometric identification.

Blanton and Gasti [8], [9] provide privacy-preserving protocols for both FingerCode and minutiae-based fingerprint representation. Their solution utilizes a combination of homomorphic encryption and garbled circuit evaluation. In the minutiae-based approach, each fingerprint X is represented as a point in a 2-dimensional space together with its orientation. To compare fingerprints X and Y consisting of m_X and m_Y minutiae, respectively, the solution in [8] proceeds by first computing the adjacency matrix of size $m_X m_Y$, which indicates which points from X and Y are a possible match. That is, the cell at row i and column j is set if the spatial (Euclidean) distance between point i in X and point j in Y as well as the difference in their orientation are within specific thresholds. Then the algorithm proceeds by considering each minutia i of X in turn matching it with the closest unmatched minutia j in Y among those which are set as a possible match in the adjacency matrix. At the end, the size of the computed matching is compared to the threshold to determine whether the fingerprints should be treated as related or not. As mentioned before, this approach may fail to find the best matching for the input fingerprint and thus produce an incorrect result, and we use it as the starting point for our solution. The complexity of the approach from [8] for two biometrics X and Y consisting of m_X and m_Y minutiae, respectively, is $O(m_X m_Y)$.

Lastly, Shahandashti et al. [30] design a privacy-preserving protocol for minutiae-based fingerprint matching using homomorphic encryption. As before, each fingerprint consists of a number of minutiae, but with this solution each minutia contains its type (in addition to its

location and orientation) and the types of two minutiae must match for two minutiae to be considered as a possible match. The computation is based on evaluation of polynomials in encrypted form and a pair of minutiae $i \in X$ and $j \in Y$ are added to the matching if they are a possible match. Note that the computation is not accurate and introduces an error every time a minutia from X or Y has more than one possible match. The complexity of the solution from [30] is dominated by $O(m_X m_Y (|T| + |D_E| + |D_a|))$ cryptographic operations, where $|T|$ is the number of minutia types, $|D_E|$ is the set of all possible squared Euclidean distances between two points (i.e., if x_{max} and y_{max} are maximum x and y coordinates, respectively, then $|D_E| = x_{max}^2 + y_{max}^2 + 1$, and $|D_a|$ is the set of all possible squared angular distances between point orientations. Because the complexity is quadratic in the domain size for point representation, this approach is quite bit slower than others for a typical set of parameters.

III. PROBLEM DESCRIPTION

A. Fingerprint background

Fingerprint identification is a well-studied area with a number of available approaches [24]. 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 then 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 $X = \langle (x_1, y_1, \alpha_1), \dots, (x_{m_X}, y_{m_X}, \alpha_{m_X}) \rangle$. 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 (Euclidean) distance between them is smaller than a given threshold d_0 and the orientation difference between them is smaller than a given threshold α_0 . In other words, the matching condition is computed as:

$$\sqrt{(x'_j - x_i)^2 + (y'_j - y_i)^2} < d_0 \quad \wedge \quad (1) \\ \min(|\alpha'_j - \alpha_i|, 360^\circ - |\alpha'_j - \alpha_i|) < \alpha_0.$$

The tolerance values d_0 and α_0 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. Alignment can be either absolute, in which case each fingerprint is pre-aligned independently using the core point or other information, or relative, in which case information contained in the two biometrics is used to guide their alignment relative to each other.

While relative pre-alignment can be more accurate than 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 slightly different alignments, and the result of the comparison is a match if at least one of them matches the queried biometric. A more detailed treatment of alignment procedures is outside 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 that satisfies the matching constraint and which has not already been paired with another minutia in Y . That is, the pairing function considers all minutiae X_i from X in turn and for each X_i finds the closest Y_j that satisfies the matching predicate in equation 1 and which has not been paired with another minutia from X . If no such minutia Y_j from Y exists, X_i is not added to the pairing. We denote the result of applying the minutia matching predicate in equation 1 to minutiae $X_i \in X$ and $Y_j \in Y$ by $mm(X_i, Y_j)$.

This approach was used in prior privacy-preserving fingerprint matching solutions, but it does not find the optimum assignment (i.e., the one that maximizes the number of mates). That is, there are circumstances when a minutia X_i should be paired with another minutia Y_j , which is not the closest to X_i , to result in an assignment of the largest size. According to fingerprint literature [19], [37], 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. In particular, to represent fingerprints X and Y as a flow network, we form a bipartite graph in which minutia points from X and Y form the nodes of the first and second partitions, respectively. The set of edges is created as follows: there is an edge from a node corresponding to minutia $X_i \in X$ to $Y_j \in Y$ iff $mm(X_i, Y_j) = 1$. To use the resulting bipartite graph as a flow network, we create an additional source node s and connect it to all nodes from X using (directional) edges of capacity 1. Similarly, we create a sink node and connect each node from Y to the sink node t using edges of capacity 1. Then each edge from X_i to Y_j also has capacity 1 (in one direction only). We refer the reader to [19], [37] for additional detail.

The problem fingerprint problem in the maximum flow formulation can be solved using one of the known algorithms such as Ford-Fulkerson [13] and others. 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-Fulkerson algorithm increases at least by a factor m . Finding a pairing of optimal size was considered impractical in [8], but in this work we show that with the techniques we develop, performance of fingerprint comparisons can be comparable or faster than performance of simpler and less accurate fingerprint comparisons given in [8] (which is currently the fastest solution for secure minutia-based fingerprint matching).

Throughout this work, we assume that in order for two fingerprints X and Y to result in a match, the number of paired minutiae needs to exceed a fixed (known to all parties) threshold T (which can be a function of the number of minutiae in X and Y , but is fixed once the sizes are known).¹

B. Security model

In this work, we use standard security models for secure multi-party computation. We primarily focus on security in presence of semi-honest participants (also, honest-but-curious or passive), but our solutions can also be extended to achieve stronger security in presence of fully malicious (or active) adversaries. In particular, the semi-honest setting means that the parties follow the prescribed behavior, but might try to compute additional information from the information obtained during protocol execution. Recall that it is required that the participants do not learn anything about private input data beyond the agreed-upon output. Consequently, security in this setting is defined using simulation argument: the protocol is secure if the view of protocol execution for each party is computationally or information-theoretically indistinguishable from the view simulated using that party's input and output only. This implies 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, \dots, P_n engage in a protocol Π that computes function $f(\text{in}_1, \dots, \text{in}_n) = (\text{out}_1, \dots, \text{out}_n)$, where in_i and out_i denote the input and output of P_i , respectively. Let $\text{VIEW}_{\Pi}(P_i)$ denote the view of 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, \dots, m_k passed between the parties during protocol execution: $\text{VIEW}_{\Pi}(P_i) = (\text{in}_i, r_i, m_1, \dots, m_k)$. Let $I = \{P_{i_1}, P_{i_2}, \dots, P_{i_t}\}$ denote a subset of the parties for $t < n$ and $\text{VIEW}_{\Pi}(I)$ denote*

¹In the event that the value of T comes from one of the participants and needs to be protected, the solution can be easily modified to compute with private T .

the combined view of parties in I during the execution of Π (i.e., the union of the views of the parties in I). We say that protocol Π is t -private in presence of semi-honest adversaries if for each coalition of size at most t there exists a probabilistic polynomial time simulator S_I such that $\{S_I(\text{in}_I, f(\text{in}_1, \dots, \text{in}_n))\} \equiv \{\text{VIEW}_{\Pi}(I), \text{out}_I\}$, where $\text{in}_I = \bigcup_{P_i \in I} \{\text{in}_i\}$, $\text{out}_I = \bigcup_{P_i \in I} \{\text{out}_i\}$, and \equiv denotes computational or information-theoretic indistinguishability.

Note that we choose to present a general definition for n parties who carry out the computation. For the problem we study, the most common setting is going to be $n = 2$ and thus $t = 1$. We, however, would like to offer a solution that works for $n > 2$ and is also suitable for outsourcing to multiple computational nodes.

The second standard, and stronger, security model assumes the participants can be fully malicious and thus allows them to behave arbitrarily including deviating from the computation and aborting the execution. Security in this setting is shown using a different security definition, which we omit here due to space considerations and instead refer the reader, e.g., to [14].

IV. WORKING TOWARD THE SOLUTION

Before we start working our way to a solution, we note that our objective now is to solve the maximum flow problem in a flow network formed by two fingerprints X and Y . An important observation here is that this is a graph problem, but the structure of the graph (i.e., information about connectivity of nodes from X to nodes from Y) must be hidden from all parties. This calls for a solution known as data-oblivious in the literature. Data-oblivious execution is defined as having the sequence of instructions and memory accessed to be independent of the data on which the computation is performed. Such a solution can be either deterministic, in which case the sequence of executed instructions and memory accesses are always the same regardless of the data, or probabilistic, in which case their distributions must be indistinguishable for all possible inputs.

In the search for an approach suitable for solving the maximum flow problem in a data-oblivious way, we chose to concentrate on solutions that work with adjacency matrix representation of the graph. Note that because our graph is bipartite, we only need to consider an approach that works for a bipartite graph and not necessarily for a general graph.

Our starting point was the work of Mucha and Sankowski [25] that presents a randomized algorithm for finding maximum matching in an n -node graph in $O(n^\omega)$ time, where ω is the exponent of the best known matrix multiplication algorithm and currently $\omega < 2.38$. The solution of [25] assumes that a perfect matching of size

$n/2$ is present, which it will compute. This is not the case for our application, and to use this solution on a graph without perfect matching, we need to resort to techniques of Ibarra and Moran [16] (which are applicable to bipartite graphs only). The most crucial result listed in [25] that we need is due to Lovasz [22] and can be stated as follows: Let $G = (V, U, E)$ be a bipartite graph with nodes $V \cup U$ and edges E , where $|V| = |U| = n/2$, $V = \{v_1, \dots, v_{n/2}\}$ and $U = \{u_1, \dots, u_{n/2}\}$. Let an adjacency matrix $A = A(G)$ be formed by setting $A_{i,j}$ to a random value from the set $[1, R]$ for some R if $(v_i, u_j) \in E$ and to 0 otherwise. Then the rank of A is at most the size of the maximum matching, where the equality holds with probability at least $1 - (n/2R)$. This means that if the bitlength of R is set according to a correctness parameter, the rank will be equal to the size of the maximum matching with all but at most a negligible probability (in the correctness parameter).

Note that in our application it is not necessary to compute the matching itself and instead the size of the pairing of points from X and Y is sufficient to determine if X and Y result in a match. This means that if we determine the rank of the matrix formed as described above, we will be able to compute the answer to the question whether two fingerprints are related or not.

The next step is to compute the rank of A . A standard way to achieve this is to apply Gaussian elimination (LU decomposition) to A . The simplest algorithm for doing so runs in $O(n^3)$ time for an $n \times n$ matrix and asymptotically lower solutions (of the same complexity as that of matrix multiplication) are possible. Before we proceed with further discussion, we include a (non-secure) solution of complexity $O(n^3)$ based on Gaussian elimination. When it is applied to biometrics X and Y with m_X and m_Y minutiae, respectively, its complexity is $O(m_X^2 m_Y)$ assuming that $m_X \leq m_Y$.

Note that the first step of fingerprint matching based on rank computation using any approach consists of computing the randomized adjacency matrix. Because this pre-processing step is common to all solutions, we separately present it in Algorithm 1. In the algorithm, notation $z \stackrel{R}{\leftarrow} S$ denotes that the value of z is chosen uniformly at random from set S .

The Gaussian elimination algorithm that takes the resulting adjacency matrix A and converts it to a row echelon form is given in Algorithm 2. It assumes that $m_X \leq m_Y$; otherwise, the roles of X and Y are swapped. Following [16], after forming adjacency matrix A , we carry out all operations in a field (of size R) in this and other algorithms. That is, we treat the matrix as consisting of random field elements and all consecutive operations are in a field (which in particular is the reason for using multiplicative inverse in place of division). We present the simplest version of the Gaussian elimination

Algorithm 1 $A = \text{AdjMat}(X = \langle x_i, y_i, \alpha_i \rangle_{1 \leq i \leq m_X}, Y = \langle x'_i, y'_i, \alpha'_i \rangle_{1 \leq i \leq m_Y})$

```

1: for  $i = 0, \dots, m_X$  do
2:   for  $j = 0, \dots, m_Y$  do
3:     if  $(\sqrt{(x'_j - x_i)^2 + (y'_j - y_i)^2} < d_0) \wedge$ 
        $(\min(|\alpha'_j - \alpha_i|, 360^\circ - |\alpha'_j - \alpha_i|) < \alpha_0)$  then
4:        $A_{ij} = 1;$ 
5:     else
6:        $A_{ij} = 0;$ 
7:     end if
8:      $r_{ij} \stackrel{R}{\leftarrow} [1, R];$ 
9:      $A_{ij} = A_{ij} \cdot r_{ij};$ 
10:  end for
11: end for
12: return  $A;$ 

```

Algorithm 2 $B = \text{GE}(A = \{A_{ij}\}_{1 \leq i \leq m_X, 1 \leq j \leq m_Y})$

```

1: for  $i = 1, \dots, m_X$  do
2:   for  $j = i + 1, \dots, m_X$  do
3:     for  $k = i, \dots, m_Y$  do
4:        $A_{jk} = A_{jk} - A_{ik} \cdot A_{ii}^{-1} \cdot A_{ij}$ 
5:     end for
6:   end for
7: end for

```

algorithm that works only for invertible matrices (with $m_X = m_Y$) and which results in a matrix with only non-zero entries on the diagonal formed by elements A_{ii} and zero elements below the diagonal. In a more general case, some of the matrix rows or columns may either be initially zero or become zero during the execution of the algorithm, and the matrix does not have to be square. In those cases, during the i th iteration, the algorithm may swap row $i + 1$ with another row at a higher index so that row $i + 1$ contains a non-zero element at the leftmost position (or lowest column index) among all rows with indices $i + 1$ and higher. This implies that a column will be “skipped” if all of its entries below row i are zero (i.e., the computation will be of the form $A_{jk} = A_{jk} - A_{ik} \cdot A_{ii}^{-1} \cdot A_{jt}$ for $t > i$). We note that in our application of fingerprint matching the adjacency matrix is likely to contain a large number of zero elements and we need to use the general algorithm for arbitrary matrices. Then the rank of the matrix is computed as the number of non-zero rows (or columns) once the matrix has been converted to a row echelon form.

Lastly, we note that in the traditional setting, when some rows and/or columns are initially zero, they can be eliminated from the matrix before the algorithm is executed because such rows/columns cannot contribute to the matrix rank. This reduces complexity of the algorithm for sparse matrices in the regular setting, but

cannot be used in the context of secure computation. This is because the size of the resulting matrix is likely to reveal information about the size of the matching.

Returning to our prior discussion of rank computation, recall that the asymptotic complexity of rank computation can be lower (the same as that of matrix multiplication). Upon examining alternative matrix multiplication algorithms of sub-cubic time, we came to the conclusion that only Strassen’s algorithm [33] has practical importance to matrices whose size is not huge. The complexity of such an algorithm is $O(n^{\log_2 7}) \approx O(n^{2.807})$ for an $n \times n$ matrix and it would give us a solution that runs in $O(m_X^{\log_2 3.5} m_Y)$ time. While this algorithm has reduced numerical stability, it is not an issue when the computation is carried out in a finite field (i.e., on integers without rounding errors).

The original Strassen’s algorithm proposed in [33] is applicable only to invertible matrices. Solodovnikov [32] later showed how the algorithm can be extended for solving an arbitrary system of equations or finding the rank of an arbitrary matrix, which can be used as a starting point for a secure solution. The algorithm is rather complex involving several matrix transformations and produces matrices the size of which determines the rank. We note that it is possible to make the algorithm oblivious (the most important change will be to force matrices to always be of the same size by padding them with dummy rows or columns when necessary), but we choose not to expand on this further due to the limited applicability of the algorithm to fingerprint data. In particular, Strassen’s matrix multiplication outperforms the standard matrix multiplication of cubic complexity on matrices with sizes 100 and higher for each dimension, but the number of minutiae in a fingerprint (which will define the matrix size) is normally much lower.

To fully explore our options, we consider another approach that uses Gram-Schmidt orthogonalization process for computation of the rank of a matrix. In particular, Gram-Schmidt process takes a set of vectors $\{\mathbf{x}_1, \dots, \mathbf{x}_n\}$ and produces an orthogonal set of basis vectors $\{\mathbf{y}_1, \dots, \mathbf{y}_n\}$ that span the same subspace as the original vectors (i.e., the basis for $\text{span}\{\mathbf{x}_1, \dots, \mathbf{x}_n\}$). The algorithm proceeds with re-expressing all \mathbf{x}_i ’s in terms of new orthogonal basis starting from \mathbf{x}_1 . It starts by setting $\mathbf{y}_1 = \mathbf{x}_1$ and for $i = 2, \dots, n$ computes $\mathbf{y}_i = \mathbf{x}_i - \sum_{j=1}^{i-1} \text{proj}_{\mathbf{y}_j}(\mathbf{x}_i)$. Here $\text{proj}_{\mathbf{y}_j}(\mathbf{x}_i)$ denotes orthogonal projection of vector \mathbf{x}_i onto the line spanned by vector \mathbf{y}_j and thus $\sum_{j=1}^{i-1} \text{proj}_{\mathbf{y}_j}(\mathbf{x}_i)$ is the orthogonal projection of \mathbf{x}_i onto $\text{span}\{\mathbf{y}_1, \dots, \mathbf{y}_{i-1}\}$. The projection operator is defined as $\text{proj}_{\mathbf{y}_j}(\mathbf{x}_i) = \frac{\langle \mathbf{x}_i, \mathbf{y}_j \rangle}{\langle \mathbf{y}_j, \mathbf{y}_j \rangle} \mathbf{y}_j$, where $\langle \mathbf{x}_i, \mathbf{y}_j \rangle$ denotes the inner product of vectors \mathbf{x}_i and \mathbf{y}_j . Also, when $\mathbf{y}_j = 0$, $\text{proj}_{\mathbf{y}_j}(\mathbf{x}_i) = 0$ for any vector \mathbf{x}_i . If we treat vectors \mathbf{x}_i ’s collectively as a matrix and apply

Algorithm 3 rank = GSRank($\{A_{ij}\}_{1 \leq i \leq m_X, 1 \leq j \leq m_Y}$)

```

1: for  $i = 1, \dots, m_X$  do
2:   for  $j = 1, \dots, m_Y$  do
3:      $B_{ij} = A_{ij}$ ;
4:   end for
5: end for
6: for  $i = 2, \dots, m_X$  do
7:   for  $j = 1, \dots, i - 1$  do
8:     set  $zt_j = \bigvee_{k=1}^{m_Y} B_{ik}$ 
9:     if ( $zt_j \neq 0$ ) then
10:      set  $\langle \mathbf{y}_j, \mathbf{y}_j \rangle = \sum_{k=1}^{m_Y} (B_{jk})^2$  and  $\langle \mathbf{x}_i, \mathbf{y}_j \rangle = \sum_{k=1}^{m_Y} A_{ik} B_{jk}$ 
11:      for  $k = 1, \dots, m_Y$  do
12:        set  $B_{ik} = B_{ik} - \langle \mathbf{x}_i, \mathbf{y}_j \rangle \langle \mathbf{y}_j, \mathbf{y}_j \rangle^{-1} B_{jk}$ 
13:      end for
14:    end if
15:  end for
16: end for
17: set rank = 0;
18: for  $i = 1, \dots, m_X$  do
19:   if ( $zt_i \neq 0$ ) then
20:     rank = rank + 1;
21:   end if
22: end for
23: return rank

```

Gram-Schmidt process to it to obtain another matrix formed by vectors \mathbf{y}_i ’s, then the number of non-zero \mathbf{y}_i ’s will correspond to the rank of the matrix. We provide our rank computation algorithm in a finite field based on Gram-Schmidt process in Algorithm 3. In the algorithm, lines 1–16 correspond to the Gram-Schmidt process and lines 17–22 count non-zero rows in the resulting matrix. The variable zt_j tests whether vector \mathbf{y}_j contains at least one non-zero element. If \mathbf{y}_j is zero, then any projection $\text{proj}_{\mathbf{y}_j}$ is zero and \mathbf{y}_i does not need to be modified (which corresponds to skipping lines 10–13 when zt_j is zero). As before, we assume that $m_X \leq m_Y$, and the algorithm’s complexity is $O(m_X^2 m_Y)$.

Lastly, we note that with finite precision arithmetic the algorithm can be unstable. This, however, is not an issue for us because we use precise computation in a field. The use of field operations, however, may introduce an error which is specific to that type of arithmetic. In particular, for a non-zero vector \mathbf{y}_j , it is possible that with a small probability $\langle \mathbf{y}_j, \mathbf{y}_j \rangle$ will evaluate to 0 on line 10 of the algorithm. The probability of this happening in any given round of computation can be shown to be less than $1/R$. If this however happens, it means that the inverse $\langle \mathbf{y}_j, \mathbf{y}_j \rangle^{-1}$ is not going to exist and an error may be introduced into the result (i.e., the computed rank may be off by 1). To eliminate the error, we suggest that the value of $\langle \mathbf{y}_j, \mathbf{y}_j \rangle$ is compared to 0 after its computation

Algorithm 4 $A = \text{AdjMat}(X = \langle x_i, y_i, \alpha_i \rangle_{1 \leq i \leq m_X}, Y = \langle x'_i, y'_i, \alpha'_i \rangle_{1 \leq i \leq m_Y})$

```

1: for  $i = 0, \dots, m_X$  do
2:   for  $j = 0, \dots, m_Y$  do
3:      $c_1 = ((x'_j - x_i)^2 + (y'_j - y_i)^2 \stackrel{?}{<} (d_0)^2);$ 
4:      $c_2 = (\alpha_i \stackrel{?}{\geq} \alpha'_j);$ 
5:      $a_1 = \alpha_i - \alpha'_j;$ 
6:      $a_2 = \alpha'_j - \alpha_i;$ 
7:      $a_3 = c_2 \cdot a_1 + (1 - c_2)a_2;$ 
8:      $c_3 = (a_3 \stackrel{?}{<} \alpha_0);$ 
9:      $c_4 = ((360 - a_3) \stackrel{?}{<} \alpha_0);$ 
10:     $c = c_1 \wedge (c_3 \vee c_4);$ 
11:     $r_{ij} \stackrel{R}{\leftarrow} [1, R];$ 
12:     $A_{ij} = c \cdot r_{ij};$ 
13:  end for
14: end for

```

on line 10. If it is 0 at any iteration of the algorithm, then the algorithm can be restarted using new randomness in forming A . Note that because of the small probability of this happening, the expected number of times that this algorithm will have to be restarted is near 0.

V. OBLIVIOUS FINGERPRINT MATCHING ALGORITHMS

In order to design a secure solution for the fingerprint matching problem, we need to ensure that all parts of the computation can be performed obliviously. This, in particular, implies that both branches of conditional statements will always be executed and the values will be set based on the result of evaluating the condition. In particular, statements of the type

if (*cond*) then $a = v_1$ else $a = v_2$

will be transformed into evaluating the condition *cond* first and then setting

$$a = \text{cond} \cdot v_1 + (1 - \text{cond}) \cdot v_2 = (\text{cond} \wedge v_1) \vee (\overline{\text{cond}} \wedge v_2) \quad (2)$$

Here, \vee and \wedge denote bitwise OR and AND, respectively. Then, for instance, oblivious computation of the adjacency matrix A can be performed as given in Algorithm 4. For performance reasons, we eliminate square root computation when computing the Euclidean distance (instead, squared distance is used). Also, a_3 corresponds to $|\alpha_i - \alpha'_j|$ and $(c_3 \vee c_4)$ corresponds to $\min(|\alpha'_j - \alpha_i|, 360^\circ - |\alpha'_j - \alpha_i|) \stackrel{?}{<} \alpha_0$.

Modifying rank computation algorithms, however, requires more significant changes to the way the computation is carried out. In what follows, we describe the intuition behind our solutions for rank determination and also provide their detailed description.

A. Rank determination using Gaussian elimination

When working on Gaussian elimination algorithm suitable for secure processing, a major issue we have to overcome is to make the execution oblivious in presence of zero rows and columns. That is, regardless of having zero columns (that need to be skipped) or zero rows (that need to be swapped), we want the algorithm to always execute the same instructions and always access the same matrix cells. For that reason, at each iteration of the solution, we choose to push all zero columns to the right and all zero rows to the bottom. This will allow us to work with row i and column i during the i th iteration of the algorithm (assuming that some non-zero row and column still remain at iteration i). Furthermore, once all non-zero rows and columns have been processed, we cannot reveal this fact and have to continue the computation without affecting correctness of the result.

To realize swapping of zero rows and columns in an oblivious way, we utilize data-oblivious compaction. Compaction of a sequence of values allows one to move all non-zero elements to the beginning and thus any zero element will appear only after all non-zero elements. Our goal of pushing zero columns and rows to the end can also be achieved by using oblivious sorting, but we choose compaction for performance reasons. Thus, as the first step of each iteration i , we compute whether any given (partial) row and column at position i and higher contains at least a single non-zero element. Note that we only need to consider matrix elements with both row and column indices i and higher and this is why only a part of each row and column is checked. For example, for row $j \geq i$ only cells at position $i \leq k \leq m_Y$ can be non-zero and are checked. Similarly, for column $j \geq i$ only cells at rows $i \leq k \leq m_X$ are relevant and checked. After this step, all zero rows and columns are pushed to the end using oblivious compaction.

At this point we have that the current row and column (with index i) have at least one non-zero element, but the algorithm requires that the leading coefficient of the (partial) current row i is non-zero. To satisfy this constraint, we add all rows with index $i + 1$ and higher to the current row i . This has no effect on correctness of the computation (and is a common operation in Gaussian elimination), but ensures that the matrix element A_{ii} is non-zero. This is because when (partial) column i has at least one non-zero element, the probability that the sum of its elements (which are random values from the range $[1, R]$) results in 0 is $1/R$. Thus, with overwhelming probability (in the bitlength of R) $A_{ii} \neq 0$ when (partial) column i has at least one non-zero element and correctness of the computation is preserved.

The only part of the algorithm that remains to be modified for oblivious execution is ensuring that the computation can proceed in exactly the same way once

all non-zero rows and columns have already been processed. That is, for some iteration of the algorithm i we will have that all remaining (partial) rows and columns are zero. To ensure that the algorithm can execute exactly the same steps without revealing this fact and affecting correctness of the computation, the only place we have to modify is computation of the inverse of A_{ii} . Because when $A_{ii} = 0$, it does not have a multiplicative inverse, we set A_{ii} to 1 in that case. Then because $1^{-1} = 1$, multiplying any value by 1^{-1} will have no effect. To ensure that A_{ii} is unchanged when $A_{ii} \neq 0$, we set A_{ii} to $A_{ii} + c$, where c is the result of comparing A_{ii} to 0.

The overall oblivious algorithm for computing over-the-threshold matrix rank based on Gaussian elimination is given in Algorithm 5. Lines 2–5 and 7–10 compute row and column flags, respectively, that indicate whether the (partial) rows or columns consist of only zero elements. These flags are used in row-wise and column-wise compaction on lines 6 and 11, respectively. Lines 12–16 update row i to ensure that its leading element is non-zero if non-zero rows still remain. Line 17 adjusts the element A_{ii} for the purpose of computing its inverse as described above. Next, lines 18–23 compute the i th iteration of Gaussian elimination. After executing lines 1–24, matrix A is in a row echelon form and all that remains is to compute its rank by adding the number of non-zero elements on the diagonal A_{ii} . This is performed on lines 25–28. Lastly, the algorithm compares the computed rank to some predefined threshold (which is assumed to be a constant for given input sizes) and outputs a bit, which indicates whether the compared biometrics should be treated as a match.

To realize oblivious compaction, we utilize the general approach of tight order-preserving compaction algorithm from [15], which was subsequently used in the secure multi-party computation framework in [5]. The algorithm proceeds in $\log_2 n$ rounds for a sequence of n values. At round i (for $0 \leq i \leq \log n - 1$), an element at position j is either obviously moved 2^i elements left or is not moved at all. The former happens when the i th least significant bit in the number $count_j$ of zero elements that precede the element at position j is 1. We refer the reader for additional details regarding the algorithm to [15], [5] and provide our realization of it with new optimizations in Algorithm 6. It is written for the special case when the input consists of 1-bit elements and moves all non-zero elements to the beginning of the input sequence. When this algorithm is used for rank computation in Algorithm 5, it will still take 1-bit keys r_j or c_j according to which the values need to be moved, but instead of moving individual elements, the entire (partial) rows or columns are moved.

In the most general case, the element x_j at position j is either kept unchanged or replaced with element at

Algorithm 5 $b = \text{OTGERank}(\{A_{ij}\}_{1 \leq i \leq m_X, 1 \leq j \leq m_Y})$

```

1: for  $i = 1, \dots, m_X - 1$  do
2:   for  $j = i, \dots, m_X$  do
3:     Set  $\text{rowflag}_j = \bigvee_{k=i}^{m_Y} A_{jk}$ ;
4:     Set  $r_j = (\text{rowflag}_j \neq 0)$ ;
5:   end for
6:   Use compaction to “sort” partial rows  $i, \dots, m_X$ 
   using keys  $r_j$ , where row  $j$  is  $(A_{ji}, \dots, A_{jm_Y})$ , so
   that all rows with  $r_j = 0$  are moved to the bottom
   of  $A$ .
7:   for  $j = i, \dots, m_Y$  do
8:     Set  $\text{colflag}_j = \bigvee_{k=i}^{m_X} A_{kj}$ ;
9:     Set  $c_j = (\text{colflag}_j \neq 0)$ ;
10:  end for
11:  Use compaction to “sort” partial columns
    $i, \dots, m_Y$  using keys  $c_j$ , where column  $j$  is
    $(A_{ij}, \dots, A_{m_X j})$ , so that all columns with  $c_j = 0$ 
   are moved to the right of  $A$ .
12:  for  $j = i + 1, \dots, m_X$  do
13:    for  $k = i, \dots, m_Y$  do
14:      Set  $A_{ik} = A_{ik} + A_{jk}$ ;
15:    end for
16:  end for
17:  Set  $A_{ii} = A_{ii} + (A_{ii} \stackrel{?}{=} 0)$ ;
18:  Compute  $A_{ii}^{-1}$ ;
19:  for  $j = i + 1, \dots, m_X$  do
20:    for  $k = i, \dots, m_Y$  do
21:      Set  $A_{jk} = A_{jk} - A_{ik} \cdot A_{ii}^{-1} \cdot A_{ji}$ ;
22:    end for
23:  end for
24: end for
25: Set  $\text{ranksum} = 0$ ;
26: for  $i = 1, \dots, m_X$  do
27:   Set  $\text{ranksum} = \text{ranksum} + (A_{ii} \stackrel{?}{\neq} 0)$ ;
28: end for
29: Compute and return  $b = (\text{ranksum} \stackrel{?}{\leq} T)$ ;

```

position $j + 2^i$ at the i th iteration of the algorithm. This corresponds to the computation $x_j = (1 - b_{i,j})x_j + b_{i,j+2^i} \cdot x_{j+2^i}$, where $b_{i,j}$ represents the i th least significant bit of $count_j$ and at most one of x_j and x_{j+2^i} is non-zero at any given time. When, however, $j + 2^i$ exceeds the total number of elements, x_j is either kept or erased, i.e., $x_j = (1 - b_{i,j})x_j$. In addition, for the first $2^i - 1$ elements of the sequence, $b_{i,j}$ is always 0, which means that we do not need to multiply x_j by $(1 - b_{i,j})$. This logic is presented on lines 8–13 of the algorithm.

The complexity of the oblivious compaction algorithm on which we build is $O(n \log n)$ for an n -element input. In our case, each invocation of compaction is executed on $m_X - i + 1$ rows (resp., $m_Y - i + 1$

Algorithm 6 $\langle y_1, \dots, y_n \rangle = \text{Comp}(\langle x_1, \dots, x_n \rangle)$

```

1:  $count_1 = 1 - x_1$ ;
2: for  $i = 2, \dots, n$  do
3:    $count_i = count_{i-1} + 1 - x_i$ ;
4: end for
5: Let  $b_{i,j}$  denote the  $i$ th least significant bit of  $count_j$ 
   for  $j = 1, \dots, n$  and  $i = 0, \dots, \lceil \log n \rceil - 1$ 
6: for  $i = 0, \dots, \lceil \log n \rceil - 1$  do
7:   for  $j = 1, \dots, n$  do
8:     if  $j \geq 2^i$  then
9:        $x_j = (1 - b_{i,j})x_j$ ;
10:    end if
11:    if  $j + 2^i \leq n$  then
12:       $x_{j+2^i} = x_{j+2^i} + b_{i,j+2^i} \cdot x_j$ ;
13:    end if
14:  end for
15: end for
16: Return  $\langle x_1, \dots, x_n \rangle$ ;

```

columns) each of size $m_Y - i + 1$ (resp., $m_X - i + 1$). This gives us that the total cost of compaction at all iterations i of the algorithm is $O(m_X^2 m_Y \log m_X)$ for rows and $O(m_X^2 m_Y \log m_Y)$ columns. This dominates the algorithm's complexity, as the remaining work is $O(m_X^2 m_Y)$. However, according to our experimental results in section VII, the cost of compaction is small compared to the remaining computation (e.g., the cost of multiplying two elements of the field is noticeably higher than the cost of moving an element multiple times).

B. Rank determination using Gram-Schmidt process

To obtain a data-oblivious algorithm for matrix rank computation based on the Gram-Schmidt process, we notice that the only computation that is not data-oblivious in the conventional approach in Algorithm 3 is the conditional statement on lines 9–14. To eliminate this conditional logic, we can use a similar mechanism to that utilized in oblivious rank computation based on Gaussian elimination. That is, when vector \mathbf{y}_j is zero, so is $\langle \mathbf{y}_j, \mathbf{y}_j \rangle$, in which case we set $\langle \mathbf{y}_j, \mathbf{y}_j \rangle$ to 1. Then the inverse of 1 is always computable (and equal to 1), but the projection is still 0 because all elements of \mathbf{y}_j are 0 (and thus $\langle \mathbf{x}_i, \mathbf{y}_j \rangle$ is also 0).

As discussed in section IV, there is a small possibility that $\langle \mathbf{y}_j, \mathbf{y}_j \rangle = 0$ when \mathbf{y}_j is non-zero. Then to ensure that the algorithm always computes the result correctly, we would like to implement error reporting, which will notify the parties that the result may be unreliable and let them restart the execution using new randomness for the adjacency matrix. Note that unlike non-oblivious execution, we cannot report the error at the point of its occurrence because the fact that an error happened can

Algorithm 7 $b = \text{OTGSRank}(\{A_{ij}\}_{1 \leq i \leq m_X, 1 \leq j \leq m_Y})$

```

1: for  $i = 1, \dots, m_X$  do
2:   for  $j = 1, \dots, m_Y$  do
3:      $B_{ij} = A_{ij}$ ;
4:   end for
5: end for
6: set error = 0
7: for  $i = 2, \dots, m_X$  do
8:   compute  $zt_i = (\bigvee_{j=1}^{m_Y} B_{ij} \stackrel{?}{=} 0)$ 
9:   compute  $\langle \mathbf{y}_i, \mathbf{y}_i \rangle = \sum_{j=1}^{m_Y} (B_{ij})^2$ 
10:  compute  $b = (\langle \mathbf{y}_i, \mathbf{y}_i \rangle \stackrel{?}{=} 0)$  and set  $\langle \mathbf{y}_i, \mathbf{y}_i \rangle + b$ 
11:  set error = error +  $(1 - zt_i)b(1 - \text{error})$ 
12:  compute  $\langle \mathbf{y}_i, \mathbf{y}_i \rangle^{-1}$ 
13:  for  $j = 1, \dots, i - 1$  do
14:    compute  $\langle \mathbf{x}_i, \mathbf{y}_j \rangle = \sum_{k=1}^{m_Y} A_{ik} B_{jk}$ 
15:    for  $k = 1, \dots, m_Y$  do
16:      set  $B_{ik} = B_{ik} - \langle \mathbf{x}_i, \mathbf{y}_j \rangle \langle \mathbf{y}_j, \mathbf{y}_j \rangle^{-1} B_{jk}$ 
17:    end for
18:  end for
19: end for
20: set ranksum = 0;
21: for  $i = 1, \dots, m_X$  do
22:   ranksum = ranksum +  $zt_i$ ;
23: end for
24: Set  $b = (\text{ranksum} \stackrel{?}{\leq} T)$ ;
25: return  $(1 - \text{error})b + 2 \cdot \text{error}$ ;

```

reveal unintended information. For instance, if an error is reported during iteration i of the algorithm, at the very least, the participants will learn that the i th row of the adjacency matrix was non-zero, which cannot be deduced from the output that the parties learn. Thus, the computation needs to proceed until the end even if an error is detected, and the error is returned at the end of the computation by outputting a special value. Once again, the probability of the error happening is negligible in the correctness parameter (which determines the bitlength of the field elements), which means that the computation almost never will need to be restarted.

We provide our data-oblivious algorithm for over-the-threshold rank computation based on Gram-Schmidt process in Algorithm 7. The algorithm is structured in such a way as to minimize the overall amount of computation; namely, any quantity used in the computation multiple times without being modified in between is computed only once. In particular, because row i is updated only during iteration i of the loop, each \mathbf{y}_i is updated only once and we can take the computation of $\langle \mathbf{y}_j, \mathbf{y}_j \rangle$ and its inverse outside of the loop over j and instead compute $\langle \mathbf{y}_i, \mathbf{y}_i \rangle$ once for each i .

The algorithm checks whether $\langle \mathbf{y}_i, \mathbf{y}_i \rangle = 0$ and if so, sets it to 1. Also, if \mathbf{y}_i is non-zero, but $\langle \mathbf{y}_i, \mathbf{y}_i \rangle = 0$,

an error is set. We maintain that variable error is a bit, which after being set to 1 remains 1 until the end of the algorithm. That is, its value is incremented (from 0 to 1) only when the current row \mathbf{y}_i is non-zero, $\langle \mathbf{y}_i, \mathbf{y}_i \rangle = 0$, and error is currently 0 (line 11). The algorithm outputs either 0, 1, or 2. In the exceptional case of when the error was set, the algorithm outputs 2. Otherwise, either 0 or 1 is returned based on the result of the computation.

VI. SECURE FINGERPRINT MATCHING PROTOCOLS

The data-oblivious fingerprint matching algorithms presented above lead us to secure realization of this functionality in the secure multi-party computation framework. In particular, because the execution is now data-oblivious, we can combine it with one of the available secure arithmetic techniques to result in protocols that provably protect privacy of biometric data. We outline two such possibilities.

The first solution that we list is to employ two-party garbled circuit evaluation techniques (originally proposed in [38]). This technique represents the function to be evaluated as a Boolean circuit and allows one participant, circuit generator, to encode the circuit using two random labels for each (binary) wire. Then the second participant, circuit evaluator, evaluates the garbled circuit on private inputs in a way that it sees the labels used during function evaluation, but their meaning (i.e., 0 or 1) is not known to that party. After the evaluator computes the labels for the output wires, it sends them to the circuit generator, who determines their meaning and (optionally) announces the output to the second party. To choose labels corresponding to the private inputs, the parties engage in Oblivious Transfer (OT), as a result of which the evaluator obtains labels corresponding to its inputs and the other party learns nothing. This allows the evaluator to obtain labels for its private inputs and labels for the circuit generator’s inputs are sent directly to the evaluator (who does not know their meaning). There are many available OT realizations and their extensions such as, e.g., [26] and [17].

Theorem 1. *Assuming the existence of secure garbled circuit evaluation techniques and oblivious transfer, our algorithms result in 1-private protocols for fingerprint matching with two participants P_1 and P_2 .*

Proof. Given secure implementation of garbled circuit evaluation and OT, two participants can 1-privately evaluate any function f on their private inputs in_1 and in_2 and learn outputs out_1 and out_2 , respectively. The only part that has to be shown is that the function can be represented as a Boolean circuit. We specify our algorithms using a data-independent sequence of instructions consisting of Boolean and integer operations, which have known Boolean circuit representations. This means that

the functions can be securely evaluated in the garbled circuit framework, resulting in 1-private protocols for fingerprint matching. \square

The second technique that we suggest to use is threshold linear secret sharing in the multi-party setting (such as Shamir’s secret sharing [31]). It allows $n > 2$ parties to securely evaluate a function on shares of private data. Then before the computation commences, all private data are split into shares and the shares are distributed among the computational parties who carry out the computation on protected data. After the computation, the shares of the result are communicated to the participants who are entitled to learning the result and reconstruct the output from the shares. Note that in this formulation, the participants who provide the data do not have to coincide with computational parties, but rather the set of input providers, output recipients, and computational parties can be arbitrary with respect to their relationship to each other. This makes the framework suitable for a variety of settings including secure computation outsourcing by one or more clients to a number of servers.

With linear secret sharing techniques, any linear combination of secret shared data is computed locally by each participant, but multiplication requires their interaction and constitutes a basic (interactive) building block of larger computations. With (n, t) -threshold linear secret sharing techniques, each private value is split into n shares (and distributed to n participants) such that t or fewer shares information-theoretically reveal no information about the shared value, while $t + 1$ shares allow the value to be reconstructed. For semi-honest participants, it is typically the case that $t < n/2$. Any function can be expressed in this framework, and optimized designs of commonly used operations are available.

Theorem 2. *Assuming the existence of secure (n, t) -threshold linear secret sharing scheme, our algorithms result in t -private protocols for fingerprint matching with n participants and $t < n/2$.*

The argument for this proof proceeds in the same way as for Theorem 1: because our are oblivious, the straightforward application of threshold linear secret sharing techniques results in secure protocols for fingerprint matching in the multi-party setting.

For both two-party techniques based on garbled circuit evaluation and multi-party techniques based on secret sharing, there are general mechanisms for converting solutions secure in the semi-honest model to solutions resilient to fully malicious behavior in the stronger malicious adversarial model (see, e.g., [21] for garbled circuits and [3] for secret sharing among many others). This means that if we apply such techniques to our computation, we automatically obtain protocols secure in the malicious model. We omit the details here.

VII. IMPLEMENTATION AND PERFORMANCE

To evaluate performance of our techniques, we implement our oblivious fingerprint matching algorithms using both Gaussian elimination and Gram-Schmidt approaches in a secure computation framework. Our implementation is based on two-party garbled circuit evaluation and utilizes a tool called JustGarble [4] for efficient circuit garbling and garbled circuit evaluation.

We build Boolean circuits for Algorithms 5 and 7 with optimizations tailored to specifics of modern garbling techniques. In particular, recent garbled circuit-based techniques allow for XOR gates to be implemented without any use of cryptographic operations, which allows XOR gates to become virtually free [20]. This means that a circuit that minimizes the use of non-XOR gates will have performance advantages over other circuits of comparable size with smaller percentage of XOR gates. One specific optimization that we were able to apply is minimizing the number of non-XOR gates in evaluation of conditional statements. In detail, recall that conditional statements are re-written as given in equation 2. The second formula expressed in terms of Boolean operations is more suitable for use in Boolean circuits, but we also notice that the bitwise OR operation can be replaced with bitwise XOR operation. This is due to the fact that at most one clause (i.e., $c \wedge v_1$ or $\bar{c} \wedge v_2$) can be non-zero at any time and thus XOR would accomplish the same functionality as OR or addition. This applies to computation in all of Algorithms 5, 6, and 7. Also note that in compaction algorithm extracting individual bits of counts requires no computation because of bitwise representation of all values.

We measure performance of the algorithms for different numbers of minutiae in fingerprints being compared and different values of the correctness parameter. We varied the number of minutiae in both fingerprints from 10 to 30 and also varied the size of the field \mathbb{F}_R with R 's bitlength ranging from 10 to 20. Recall that according to [22], the probability that the rank of a randomized adjacency matrix is not equal to the size of the maximum matching is at most n/R for n -minutia fingerprints. This means that in our experiments the probability that the result is incorrect is approximately between $\leq n/10^3$ and $\leq n/10^6$. In the implementation, we assume that coordinates x_i, y_i of each minutia point are represented in a 2-dimensional space of size 250×250 (i.e., $x_i, y_i \in [0, 249]$) and thus the bitlength of each coordinate is 8. Then angle α_i is provided in degrees from space $[0, 359]$ and thus each α_i is represented using 9 bits. In our experiments, circuits with 30 million gates and larger were divided into sub-circuits as the current implementation of JustGarble requires that the entire circuit resides in memory for garbling/evaluation. All experiments were run on a 3.2 GHz machine with Red

Hat Linux and 4GB of memory and are given in Table I. Each experiment was run 100 times, and the double median (i.e., the median of 10 medians) is reported.

In Table I, T_G denotes the time it takes to garble the circuit measured in the average number of CPU cycles per gate (as in [4]). Similarly, T_E indicates evaluation time, also measured in the number of CPU cycles per gate. We also provide the total number of gates for each circuit. Note that the number of cycles per gate can vary in different circuits, which at least in part is due to the fact that circuits can contain different percentage of XOR gates (which require substantially less work to create and evaluate than other gates). From Table I, we can often see a slight increase in the per-gate runtimes as the number of minutiae in fingerprints increases and a slight decrease in the runtimes as the correctness parameter decreases. This perhaps can be explained by the varying composition of the circuits between XOR and non-XOR gates. For example, when the security parameter increases, a larger portion of the circuit corresponds to field operations that have a higher percentage of XOR gates than other operations. For two circuits of the same sizes, we also observed that partitioning a circuit into small pieces results in slightly faster per-gate times, which is likely due to improved cache performance. We can see from the table that the circuits corresponding to the GE solution are by a factor of 2–3 are smaller than circuits corresponding to the GS solution. This is likely due to a larger number of field operations (while the structure of the algorithms is similar).

We note that the overall execution consists of circuit garbling, oblivious transfer for one of the parties' inputs, and garbled circuit evaluation. Circuit garbling and transfer of garbled circuit can typically be performed in advance, assuming that the sizes of inputs are known. Similarly, the most expensive portion of OT (which uses public-key operations) can be performed in advance. This means that the online phase will consist of garbled circuit evaluation and communication of inputs associated with the remaining portion of OT. Using the OT extension from [17], the number of public-key operations associated with any number of OT invocations is reduced to approximately κ , where κ is the security parameter (on the order of 96–128). Furthermore, all these public-key operations can be performed in the offline phase and the online phase involves only communicating a number of bits linear in the number of inputs the circuit evaluator has and performing a similar number of hash function operations. Recall that in our application the number of inputs for each party is the number of bits in fingerprint representation (i.e., $25m_X$ or $25m_Y$), which is very small compared to the size of the computation. This means that the cost of OT will not have a noticeable impact on the overall runtime of our solution.

| Biometric size | Correctness parameter | GE Scheme | | | GS Scheme | | |
|----------------|-----------------------|-----------|-------|-------------|-----------|-------|-------------|
| | | T_G | T_E | Gates | T_G | T_E | Gates |
| 10 minutiae | 10 | 81.80 | 50.49 | 1,843,602 | 81.24 | 51.84 | 3,830,374 |
| | 15 | 81.30 | 52.12 | 4,307,707 | 80.69 | 52.70 | 8,818,644 |
| | 20 | 80.66 | 53.02 | 8,392,862 | 80.74 | 53.32 | 16,450,214 |
| 15 minutiae | 10 | 84.05 | 53.18 | 5,238,622 | 81.27 | 53.03 | 11,798,434 |
| | 15 | 83.18 | 53.37 | 11,496,802 | 81.72 | 53.68 | 26,427,929 |
| | 20 | 82.35 | 53.92 | 21,156,282 | 81.03 | 53.71 | 47,858,974 |
| 20 minutiae | 10 | 85.29 | 53.91 | 11,543,713 | 82.96 | 54.00 | 26,865,777 |
| | 15 | 84.33 | 54.16 | 24,619,823 | 81.44 | 53.32 | 59,565,747 |
| | 20 | 83.15 | 54.25 | 43,964,983 | 80.46 | 53.29 | 106,524,267 |
| 25 minutiae | 10 | 85.99 | 54.85 | 21,741,388 | 83.27 | 54.03 | 51,343,112 |
| | 15 | 85.25 | 54.57 | 45,690,373 | 81.67 | 53.50 | 113,323,432 |
| | 20 | 82.77 | 53.80 | 80,226,158 | 81.44 | 53.23 | 201,405,552 |
| 30 minutiae | 10 | 87.14 | 54.74 | 36,796,263 | 82.93 | 53.96 | 87,541,197 |
| | 15 | 85.05 | 54.12 | 76,695,248 | 81.56 | 53.75 | 192,792,367 |
| | 20 | 82.85 | 53.56 | 133,311,283 | 81.32 | 53.63 | 341,462,337 |

TABLE I
PERFORMANCE OF FINGERPRINT MATCHING USING JUSTGARBLE BASED ON GAUSSIAN ELIMINATION (GE) OR GRAM-SCHMIDT (GS) APPROACHES.

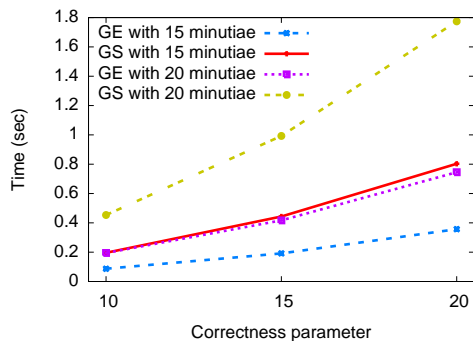


Fig. 1. Performance of garbled circuit evaluation for fingerprint matching.

To provide additional information about runtime of our privacy-preserving fingerprint identification protocols, we translate the numbers from Table I into execution times. In Figure 1, we report circuit evaluation times for experiments with 15 and 20 minutiae. The runtimes were computed from the circuit sizes, per-gate evaluation times, and the machine’s clock rate. We can see that the runtimes are on the order of a second or less, which is an acceptable delay for fingerprint authentication.

Before we conclude this section, we would like to comment on the performance of our solution compared to performance of other secure fingerprint matching protocols. As mentioned earlier, the only secure fingerprint matching protocols that use minutia representations we are aware of are those from [8] and [30]. They are based on pairing a minutia with the closest possible match minutia and all possible match minutiae, respectively, which does not guarantee an accurate result and requires

substantially less work. Implementation results are only given in [8] and the runtimes are similar to what we obtain in our solution. (And while no implementation results were reported in [30], we anticipate that performance of that solution will be substantially slower than the solution from [8].) The computation in [8] was structured as comparing fingerprint X to a number of fingerprints Y in a database D . This incurs a one-time cost (per X) and a recurring cost per record Y in D . This means that if we compare X to a single fingerprint Y , the one-time cost will still be present. For fingerprints consisting of 20 minutiae, [8] reports about 5 seconds of offline work per Y (total for both parties) and about 4 more seconds for one-time offline work. The online work is approximately 0.85 second per Y . We note that our solution requires even lower overall work for the same fingerprint sizes, but the online work may be higher for large values of the correctness parameter. If we increase the number of minutia points in a fingerprint, the runtime of our solution is expected to increase more rapidly than the runtime of the solution from [8] because of higher complexity of the algorithm we use.

VIII. CONCLUSIONS

This work investigates privacy-preserving fingerprint matching in secure computation framework. We explore fingerprint matching approaches based on standard minutia-based representation of fingerprints and settle on the most accurate algorithm which models the problem as a flow network in bipartite graphs and is guaranteed to find a matching of maximal size. We show how the problem can be solved using rank computation of an adjacency matrix, which has the same complexity as

that of matrix multiplication. We build data-oblivious algorithms for rank computation based on Gaussian elimination and Gram-Schmidt process, the complexity of which is cubic in the number of minutiae in fingerprints. While it is possible to make algorithms of lower asymptotic complexity (such as Strassen’s matrix multiplication and its extension to rank computation for non-singular matrices) data oblivious, such algorithms are of limited applicability to this problem because of larger constants behind the big-O notation and a rather low number of minutia points in a fingerprint. Our data-oblivious algorithms consequently lead to secure fingerprint authentication or identification using available secure two-party and multi-party techniques. Despite having more complex computation to achieve the desired accuracy, we show through experiments that performance of our techniques is suitable for this application and is comparable to the performance of other secure fingerprint matching techniques that perform simpler minutia matching (and which is not guaranteed to be optimal).

ACKNOWLEDGMENTS

This work was supported in part by grants CNS-1223699 from the National Science Foundation and FA9550-13-1-0066 from the Air Force Office of Scientific Research. Any opinions, findings, and conclusions or recommendations expressed in this publication are those of the authors and do not necessarily reflect the views of the funding agencies.

REFERENCES

- [1] M. Aliasgari and M. Blanton. Secure Computation of Hidden Markov Models. In *SECRYPT*, 2013.
- [2] M. Barni, T. Bianchi, D. Catalano, M. Di Raimondo, R. Labati, P. Failla, D. Fiore, R. Lazzaretto, V. Piuri, F. Scotti, and A. Piva. Privacy-preserving fingerprint authentication. In *ACM Workshop on Multimedia and Security (MM&Sec)*, pages 231–240, 2010.
- [3] Z. Beerliova-Trubiniova and M. Hirt. Perfectly-secure MPC with linear communication complexity. In *Theory of Cryptography Conference (TCC)*, pages 213–230, 2008.
- [4] M. Bellare, V. Hoang, S. Keelveedhi, and P. Rogaway. Efficient garbling from a fixed-key blockcipher. In *IEEE Symposium of Security and Privacy*, pages 478–492, 2013.
- [5] M. Blanton and E. Aguiar. Private and oblivious set and multiset operations. Cryptology ePrint Archive Report 2011/464, 2011. <http://eprint.iacr.org/2011/464>.
- [6] M. Blanton and M. Aliasgari. Secure Outsourcing of DNA Searching via Finite Automata. In *Conference on Data and Applications Security (DBSec)*, pages 49–64, 2010.
- [7] M. Blanton and M. Aliasgari. Secure outsourced computation of iris matching. *Journal of Computer Security*, 20(2–3):259–305, 2012.
- [8] M. Blanton and P. Gasti. Secure and Efficient Protocols for Iris and Fingerprint Identification. Cryptology ePrint Archive, Report 2010/627, 2010. <http://eprint.iacr.org/>.
- [9] M. Blanton and P. Gasti. Secure and Efficient Protocols for Iris and Fingerprint Identification. In *ESORICS*, pages 190–209, 2011.
- [10] D. Bogdanov, S. Laur, and J. Willemson. Sharemind: A framework for fast privacy-preserving computations. In *ESORICS*, pages 192–206, 2008.
- [11] I. Damgård, M. Geisler, and M. Krøigård. Asynchronous multiparty computation: Theory and implementation. In *Public Key Cryptography (PKC)*, pages 160–179, 2009.
- [12] Z. Erkin, M. Franz, J. Guajardo, S. Katzenbeisser, I. Lagendijk, and T. Toft. Privacy-preserving face recognition. In *PETS*, pages 235–253, 2009.
- [13] L. Ford and D. Fulkerson. *Flows in Networks*. Princeton University Press, 1962.
- [14] O. Goldreich. *Foundations of Cryptography: Volume 2, Basic Applications*. Cambridge University Press, 2004.
- [15] M. Goodrich. Data-oblivious external-memory algorithms for the compaction, selection, and sorting of outsourced data. In *SPAA*, pages 379–388, 2011.
- [16] O. Ibarra and S. Moran. Deterministic and probabilistic algorithms for maximum bipartite matching via fast matrix multiplication. *Information Processing Letters*, 13(1):12–15, 1981.
- [17] Y. Ishai, J. Kilian, K. Nissim, and E. Petrank. Extending oblivious transfers efficiently. In *CRYPTO*, pages 145–161, 2003.
- [18] A. Jain, S. Prabhakar, L. Hong, and S. Pankanti. Filterbank-based fingerprint matching. *IEEE Transactions on Image Processing*, 9(5):846–859, 2000.
- [19] T.-Y. Jea and V. Govindaraju. A minutia-based partial fingerprint recognition system. *Pattern Recognition*, 38(10):1672–1684, 2005.
- [20] V. Kolesnikov and T. Schneider. Improved garbled circuit: Free XOR gates and applications. In *JCALP*, pages 486–498, 2008.
- [21] B. Kreuter, a. shelat, and C. Shen. Billion-gate secure computation with malicious adversaries. In *USENIX Security Symposium*, 2012.
- [22] L. Lovasz. On determinants, matchings and random algorithms. *Fundamentals of Computation Theory*, pages 565–574, 1979.
- [23] D. Malkhi, N. Nisan, B. Pinkas, and Y. Sella. Fairplay – a secure two-party computation system. In *USENIX Security Symposium*, pages 287–302, 2004.
- [24] D. Maltoni, D. Maio, A. Jain, and S. Prabhakar. *Handbook of Fingerprint Recognition*. Springer, second edition, 2009.
- [25] Marcin Mucha and Piotr Sankowski. Maximum matchings via Gaussian elimination. In *FOCS*, pages 248–255, 2004.
- [26] M. Naor and B. Pinkas. Efficient oblivious transfer protocols. In *SODA*, pages 448–457, 2001.
- [27] U.S. DHS Office of Biometric Identity Management. <http://www.dhs.gov/obim>.
- [28] M. Pathak, J. Portelo, B. Raj, and I. Trancoso. Privacy-preserving speaker authentication. In *ISC*, pages 1–22, 2012.
- [29] A.-R. Sadeghi, T. Schneider, and I. Wehrenberg. Efficient privacy-preserving face recognition. In *ICISC*, pages 229–244, 2009.
- [30] S. Shahandashti, R. Safavi-Naini, and P. Ogunbona. Private fingerprint matching. In *ACISP*, pages 426–433, 2012.
- [31] A. Shamir. How to share a secret. *Communications of the ACM*, 22(11):612–613, 1979.
- [32] V. Solodovnikov. Extension of Strassen’s estimate to the solution of arbitrary systems of linear equations. 19:21–33, 1978.
- [33] V. Strassen. Gaussian elimination is not optimal. *Numer. Math.*, 13:354–356, 1969.
- [34] The Corbett Report. India fingerprints, iris scanning over one billion people. <http://www.corbettreport.com/india-fingerprinting-iris-scanning-over-one-billion-people/>.
- [35] J. Troncoso-Pastoriza, S. Katzenbeisser, and M. Celik. Privacy preserving error resilient DNA searching through oblivious automata. In *CCS*, pages 519–528, 2007.
- [36] UAE Iris Collection. <http://www.cl.cam.ac.uk/~jgd1000/UAEdeployment.pdf>.
- [37] 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.
- [38] A. Yao. How to generate and exchange secrets. In *FOCS*, pages 162–167, 1986.
- [39] Y. Zhang, A. Steele, and M. Blanton. PICCO: A general-purpose compiler for private distributed computation. In *CCS*, pages 813–826, 2013.