Oblivious Maximum Bipartite Matching Size Algorithm with Applications to Secure Fingerprint Identification

Marina Blanton(✉) and Siddharth Saraph

Department of Computer Science and Engineering, University of Notre Dame, Notre Dame, IN, USA
{mblanton,ssaraph}@nd.edu

Abstract. The increasing availability and use of biometric data leads to situations when sensitive biometric data is to be handled by entities who may not be fully trusted or otherwise are not 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. Our focus is on privacy-preserving matching of two fingerprints (authentication or identification purposes) using traditional minutia-based representation of fingerprints that leads to the most discriminative fingerprint comparisons. Unlike previous work in the security literature, we would like to focus on algorithms that are guaranteed to find the maximum number of minutiae that can be paired together between two fingerprints leading to more accurate comparisons. To address this problem, we formulate it as a flow network problem and reduce it to finding maximum matching size in bipartite graphs. The resulting problem is in turn reduced to computing the rank of a (non-invertible) matrix, formed as a randomized adjacency matrix of the bipartite graph. We then provide data-oblivious algorithms for matrix rank computation and consecutively finding maximum matching size in a bipartite graph and also extend the algorithms to solve the problem of accurate fingerprint matching. These algorithms lead to their secure counterparts using standard secure two-party or multi-party techniques. Lastly, we implement secure fingerprint matching in the secure two-party computation setting using garbled circuit evaluation. Our experimental results demonstrate that the techniques are efficient, leading to performance similar to that of other fastest secure fingerprint matching techniques, despite higher complexity of our solution that higher accuracy demands.

1 Introduction

The motivation for this work comes from the need to protect sensitive biometric data when it is being used in a growing range of applications. In particular, 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

G. Pernul et al. (Eds.): ESORICS 2015, Part I, LNCS 9326, pp. 384–406, 2015.
DOI: 10.1007/978-3-319-24174-6_20

use today include fingerprint, face, and iris images collected by the US Department of Homeland Security (DHS) from visitors [44]; fingerprint and iris images collected by the government of India from (more than billion) citizens [41]; iris, fingerprint, and face images collected by the United Arab Emirates (UAE) Ministry of Interior from visitors [43]; 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 would like to specifically treat the problem of secure computation with fingerprint data due to popularity and good accuracy of this type of biometry. We would like to cover as many settings where biometric data may used in computation by not fully trusted entities (or data sharing is restricted by law or other provisions) as possible. At the most basic level the problem is formulated as having one party A who possesses private input (fingerprint X) and another party B who possesses another private input (fingerprint Y). The parties would like to compute a function of their private inputs without disclosing any information other than the agreed-upon computation output. In the context of fingerprint matching, this can correspond to biometric authentication (comparing X and Y) and also biometric identification (when one party, e.g., B has a biometric database D and the computation consists of securely comparing X to all $Y \in D$ and identifying all biometrics that matched (if any) or determining the closest match). Another setting in which secure processing of biometric data is relevant is that of computation outsourcing by one or more data owners. In such a case, the computation still consists of comparing two biometrics to each other, but the security requirements are such that the servers that carry out the computation must learn no information about the data they process. Regardless of the setting, the core of the computation consists of comparing two fingerprints to each other, which is what we are to address. When standard secure computation techniques are used for implementing this computation, other described variants can be easily realized.

Prior literature [3,9,10,36] already contains solutions for secure fingerprint matching. All such publications introduce secure two-party computation

protocols for fingerprint comparisons after feature extraction and fingerprint alignment (if any). From this list, [9,10,36] offer solutions for minutia-based representations of fingerprints, which have the most discriminative power and are the only type of fingerprint representation suitable for biometric identification. What, however, we aim in this work is improving the precision of the matching step while maintaining efficiency of the algorithm. In particular, a minutia-based fingerprint representation consists of a number of minutiae in a two-dimensional space.[1] Roughly speaking, matching of minutiae from one fingerprint with minutiae from another fingerprint consists of computing distances between the minutiae and marking two minutiae as a possible match if the corresponding 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 (as suggested in the fingerprint literature), 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 [16,24,45], but incurs higher computational cost.

The latter approach has not been explored in the 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 the maximum flow in a bipartite graph and build techniques for solving it in secure computation context. Thus, the main contribution of this work consists of the design of a data-oblivious algorithm for maximum matching size of a bipartite graph, which proceeds by computing the rank of a randomized adjacency matrix of the graph and has complexity $O(|V|^3 \log(|V|))$. Data-oblivious execution is defined as consisting of instructions and accessing memory locations independent of the data, which makes such algorithms suitable for secure computation and outsourcing. To the best of our knowledge, data-oblivious or privacy-preserving algorithms that protect the structure of the graph for the problem of maximum matching in a flow network have been treated in the literature only for general graphs and the available algorithms have complexities $O(|V|^4)$ and higher. Beyond the application of the solution to fingerprint matching, the algorithm may be applicable to other domains and be of independent interest. When the solution is used for fingerprint matching, despite higher complexity of the algorithm than earlier

[1] In what follows, we refer to a single element of fingerprint representation as a minutia, which typically consists of coordinates in a two-dimensional space, orientation, and optionally minutia type. The fingerprint representation may also be expanded with additional information or extra fields associated with each minutia, which are the result of fingerprint pre-processing.

techniques, we show through experimental evaluation that the solution nevertheless offers good performance.

2 Related Work

Secure Multi-party Computation (SMC). Work on SMC was initiated by Yao's [46] who showed that any 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 only mention that there are a number of tools and compilers (Fairplay [30], VIFF [14], Sharemind [13], PICCO [47], etc.) that can securely evaluate functions on private data in several settings.

Secure Computation with Biometric Data. In the context of biometric matching, results available today include work on secure face recognition ([15,35] and others), DNA matching ([7,42], and others), iris code comparisons ([8,9]), fingerprint comparisons ([3,10,36]), and speaker authentication ([1,34]). 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. [3] who utilize the so-called FingerCode approach [23] for comparing two fingerprints and built a solution using a homomorphic encryption scheme. FingerCode-based algorithm is not as discriminative as minutia-based matching and is not suitable for biometric identification, despite offering computational advantages.

Blanton and Gasti [9,10] provide privacy-preserving protocols for both FingerCode and minutia-based fingerprint representations. Their solution utilizes homomorphic encryption and garbled circuit evaluation. To compare fingerprints X and Y consisting of m_X and m_Y minutiae, respectively, the solution in [10] 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 considers each minutia i of X in turn matching it with the closest unmatched minutia j in Y among its possible matches in the adjacency matrix. At the end, the size of the computed matching is compared to the threshold to determine whether the fingerprints are related. As mentioned earlier, this approach may fail to find the best matching for the input fingerprints, and we use it as the starting point for our solution. The protocol's complexity is $O(m_X m_Y)$.

Lastly, Shahandashti et al. [36] design a privacy-preserving protocol for minutia-based fingerprint matching using homomorphic encryption. 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 introduces an error every time a minutia

from X or Y has more than one possible match. The complexity of the solution 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 number of all possible squared Euclidean distances between two points and $|D_a|$ is the number of all possible squared angular distances between point orientations. Because the complexity is quadratic in the domain size of point location values, this approach is substantially slower than others for a typical set of parameters.

Data-Oblivious Protocols. Data-oblivious algorithms and their use in secure computation are receiving an increasing amount of attention. When a data-oblivious algorithm is implemented using secure multi-party computation techniques, where each operation is properly secured, the overall algorithm is guaranteed to leak no information about the data (through its structure or accessed memory locations). In addition to advances in the performance of secure multi-party computation techniques that make performance of complex algorithms practical, the emergence of cloud computing also facilitated work on data-oblivious algorithms suitable for computation outsourcing.

To the best of our knowledge, secure data-oblivious algorithms for maximum flow have appeared in [2,12]. The complexity of the algorithm from [2] that protects the structure of the graph is $O(|V|^5)$ based on Edmonds-Karp algorithm and $O(|V|^4)$ based on Push-Relabel algorithm, where $|V|$ is the number of nodes in the graph. The oblivious algorithm proposed in [12] provides a solution of complexity $O(|V|^3|E|\log(|V|))$ based on Ford-Fulkerson algorithm, where $|E|$ is the number of edges in the graph.

Oblivious RAM (ORAM). ORAM techniques ([19,39] and others) were designed to hide memory access patterns from the external server where the memory resides and are applicable to the secure multi-party computation framework. In our setting, they can be used to protect information about what data item needs to be read (e.g., a specific vertex or edge of the graph) and thus can be applied to make any algorithm data-oblivious. Each ORAM access (reading or writing a data block) has complexity at least $O(\log n^2)$, where n is the total memory size. ORAM constructions assume there is a single client with a small amount of trusted memory who knows what block it needs to read or write. When a non-oblivious algorithm is securely evaluated by a number of computational parties, there is no such client and it now needs to be obliviously simulated by the computational parties. Currently, there are still challenges for efficiently realizing ORAM techniques within the secure computation framework. The publications we are aware on this topic are [29] in the two-party and [25] in the multi-party settings, where the cost of a single ORAM access increases by a factor of $O(\log n)$. This work is complementary to ORAM as it provides an alternative mechanism for achieving data-obliviousness of a number of algorithms (and consecutively their secure versions). Because the benefits of ORAM become significant only for large input sizes [25], in our problem domain alternative techniques will be preferred as providing faster performance (e.g., [25] compares ORAM-based techniques for SMC to a naive oblivious array implementation that touches all elements of the array to retrieve an item at a private

location and suggests that ORAM techniques are faster only when the size of the data is over 1000 items).

3 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 known as honest-but-curious or passive), who follow the prescribed behavior, but might try to compute additional information from the information obtained during protocol execution. The protocols, however, can be extended to achieve stronger security in presence of fully malicious (also known as active) adversaries who can arbitrary deviate from the prescribed computation. Regardless of the model, it is required that the participants do not learn anything about private input data beyond the agreed-upon output. Consequently, security is defined using simulation argument, which we provide in Appendix A due to space considerations. We choose to use a general setup with n parties carrying out the computation. For the problem we study, the most common setting is going to be $n = 2$, but we also would like to offer a solution that works for $n > 2$ and is also suitable for outsourcing to multiple computational nodes.

Because this work treats a graph problem, where the graph is derived from private data, the graph structure graph (e.g., node connectivity) is sensitive information that cannot be revealed to the participants. For that reason, any solution must be data-independent or oblivious, in which the sequence of executed instructions and accessed memory locations must be independent of the data. Achieving data-obliviousness can be realized by using a randomized algorithm (as in ORAM) where these sequences must be indistinguishable for different inputs or a deterministic algorithm where the sequences are the same for all possible inputs. In this work, we pursue the second option.

4 Fingerprint Background

To understand how the solution we develop for maximum matching in bipartite graphs is used to address the problem of fingerprint matching, we present the background related to fingerprint representations and comparisons before moving to the algorithm itself.

Fingerprint identification is a well-studied area with many available approaches [31]. The most popular and widely used techniques extract information about minutiae from a fingerprint and store it as a set of points in the two-dimensional plane. Fingerprint matching normally 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 often represented as a set of m_X points $X = \langle (x_1, y_1, \alpha_1), \ldots, (x_{n_X}, y_{n_X}, \alpha_{n_X}) \rangle$, where x_i and y_i denote the coordinates of minutia i and α_i denotes minutia's orientation. Optionally, a minutia can also have its type included in the fingerprint representation and biometric X might also include secondary features. 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 distance (normally 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 for minutiae X_i and Y_j is computed as:

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

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 fingerprint matching is performed, the two fingerprints need to be aligned, which maximizes the number of matching minutiae. Alignment can be either absolute (each fingerprint is pre-aligned independently using the core point or other information) or relative (fingerprint features are used to guide fingerprint alignment relative to each other). Relative alignment is more accurate that absolute, while absolute alignment is performed much faster in the context of secure computation. In particular, with absolute alignment, each fingerprint is aligned independently and locally without secure computation. To increase the accuracy of matching when absolute alignment is used, a fingerprint can be stored using a small number of slightly different alignments, all of which are compared to another fingerprint, and the result of the comparison is a match if at least one representation matches the second biometric. To the best of our knowledge, relative alignment has not been investigated in secure multi-party computation literature and we leave it as a direction for future research. The matching step, however, always needs to be performed, and this constitutes the focus of this work.

A simple way to determine a pairing between minutiae of fingerprints X and Y consists of considering each minutia X_i from X in turn and pairing it with the closest minutia Y_j in Y that satisfies the matching predicate in Eq. 1 and which has not already 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 Eq. 1 to minutiae X_i and Y_j by $mm(X_i, Y_j)$.

This approach was used in prior secure fingerprint matching solutions, but it does not find the optimum assignment (i.e., the one that maximizes the number of mates). That is, sometimes 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 [24,45], the optimum pairing can be achieved by formulating the problem as an instance of maximum flow, where fingerprints X and Y are used to create a flow network. In particular, we form a bipartite graph in which minutia points from X and Y form the nodes of the first and second partitions, respectively. The edges are created as follows: there is an edge from node $X_i \in X$ 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 [24,45] for additional detail.

The problem of fingerprint matching in the maximum flow formulation can be solved using one of the known algorithms such as Ford-Fulkerson [17] and others. For n-minutia fingerprints, the optimal pairing can be found in $O(n^2)$ time using Ford-Fulkerson algorithm because each node X_i is connected to at most a constant number of nodes 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 algorithm increases at least by a factor n. Finding a pairing of optimal size was considered impractical in [10], but in this work we show that with the techniques we develop, performance of fingerprint matching can be comparable to or even faster than performance of simpler and not as accurate matching in [10] (which is currently the fastest secure minutia-based fingerprint matching).

In this work, we assume that fingerprints X and Y result in a match if the number of paired minutiae exceeds a fixed (known to all parties) threshold T (T can be a function of the number of minutiae in X and Y, but is fixed once the sizes are known).[2]

5 Working Toward the Solution

Our primary objective now is to provide an oblivious algorithm for solving the maximum flow problem in a flow network formed by a bipartite graph (which in the fingerprint matching application corresponds to two fingerprints X and Y). Note that in our application it is not necessary to compute the matching itself and instead the size of the matching is sufficient to determine if two fingerprints X and Y are related. This means that it is sufficient to determine the rank of the matrix formed as described above to solve the fingerprint matching problem.

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 [32] 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 [32] 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 resort to techniques of Ibarra and Moran [21] (which are applicable to bipartite graphs only). The most crucial result listed in [32] that we need is due to Lovasz [28] and can be stated as follows: Let $G = (V, U, E)$ be a bipartite graph with nodes $V \cup U$ and edges

[2] In 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.

Algorithm 1. $A = \mathsf{RandAdjMat}(A' = \{A'_{ij}\}_{1 \le i \le n_X, 1 \le j \le n_Y})$

```
1: for i = 0, ..., n_X do
2:     for j = 0, ..., n_Y do
3:         r_ij <-R [1, R];
4:         A_ij = A'_ij · r_ij;
5:     end for
6: end for
7: return A;
```

Algorithm 2. $B = \mathsf{GE}(A = \{A_{ij}\}_{1 \le i \le n_X, 1 \le j \le n_Y})$

```
1: for i = 1, ..., n_X do
2:     for j = i + 1, ..., n_X do
3:         for k = i, ..., n_Y do
4:             A_jk = A_jk − A_ik · A_ii^{-1} · A_ji
5:         end for
6:     end for
7: end for
```

E, where $|V| = |U| = n/2$, $V = \{v_1, \ldots, v_{n/2}\}$ and $U = \{u_1, \ldots, u_{n/2}\}$. Let an adjacency matrix $A = A(G)$ be formed by setting an element A_{ij} of A at row i and column j to a random value from the set $[1, R]$ for some R if $(v_i, u_j) \in E$ and to 0 otherwise. In other words, a matrix cell is set to a random value of a predefined bitlength if the corresponding nodes are adjacent 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 $1 - \frac{n}{2R}$. This means that if R is set to 2^κ, where κ is a desired correctness parameter, the rank will be equal to the size of the maximum matching with all but at most a negligible probability in κ. For example, if we set $\kappa = 20$, computing the rank of the randomized adjacency matrix will give the solution to the size of the maximum matching with probability $1 - \mathsf{negl}(20)$.

Because the algorithm above assumes a randomized adjacency matrix as the input, the pre-processing step will consist of creating such a matrix. If a regular adjacency matrix is given, it can be randomized using a simple approach shown in Algorithm 1. In other cases, the matrix needs to be computed, and we defer the description of how it can be done in the context of fingerprint matching to Sect. 7. In Algorithm 1, notation $z \xleftarrow{R} S$ denotes that the value of z is chosen uniformly at random from set S.

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 a bipartite graph with n_X and n_Y nodes in the first and second partition, respectively, its complexity is $O((n_X)^2 n_Y)$ assuming that $n_X \le n_Y$ (and $O((n_Y)^2 n_X)$ otherwise). To fully explore our options, in the full version of this work [11] we also consider an alternative approach for matrix rank computation based on Gram-Schmidt orthogonalization process.

The Gaussian elimination algorithm that takes a randomized adjacency matrix A and converts it to a row echelon form is given in Algorithm 2.

It assumes that $n_X \leq n_Y$; otherwise, the matrix dimensions are swapped by using the transpose of A. Following [21], after forming matrix A, we carry out all operations in a field (of size R) in this and other algorithms. That is, we treat A as consisting of random field elements and all consecutive operations are in a field (which is the reason for using multiplicative inverse in place of division). We present the simplest version of the Gaussian elimination algorithm that works only for invertible matrices (with $n_X = n_Y$) 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 computation, and the matrix does not have to be square. In those cases, during the ith 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. A column may also be "skipped" during the computation if all of its entries at row i and below are zero, i.e., the leftmost non-zero element at row i is at position $> i$. In that case, the computation will be of the form $A_{jk} = A_{jk} - A_{ik} \cdot A_{it}^{-1} \cdot A_{jt}$ for $t > i$. We note that in the 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 that works 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 they cannot contribute to the matrix rank. This reduces complexity of the algorithm for sparse matrices, but is not applicable to secure computation because the size of the reduced matrix is likely to reveal information about the size of the matching.

Returning to our prior discussion of rank computation, recall that its asymptotic complexity can be lower than $O(n^3)$ for $n \times n$ matrices and equal to the complexity of matrix multiplication. Upon examining matrix multiplication algorithms of sub-cubic time, we came to the conclusion that only Strassen's algorithm [40] has practical importance to matrices those size is not huge. Its complexity is $O(n^{\log_2 7}) \approx O(n^{2.807})$ for $n \times n$ matrices or $O(n_X^{\log_2 3.5} n_Y)$ for matrices of size $n_X \times n_Y$ with $n_X \leq n_Y$. 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 [40] is applicable only to invertible matrices. Solodovnikov [38] later showed how the algorithm can be extended to 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. While 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), we choose not to expand on this further due to the limited

applicability of the algorithm to fingerprint matching. In particular, Strassen's matrix multiplication outperforms the standard $O(n^3)$ matrix multiplication on matrices with sizes ≥ 100 for each dimension, but the number of minutiae in a fingerprint (which define the matrix size) is normally much lower.

6 Oblivious Rank Computation Algorithms

Developing data-oblivious algorithms for computing the rank of a non-invertible matrix of size $n_X \times n_Y$ constitutes the core of this work. In this section, we describe the intuition behind our solution for rank determination followed by its detailed description.

To ensure data-oblivious execution, we must require that the sequence of executed instruction does not depend on the data. This, in particular, means that execution associated with conditional statements is to be modified. In all algorithms we develop, we always execute both branches of conditional statements and the values which may be modified inside conditional statements will be set based on the result of evaluating the condition. In more detail, 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

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

Here, $\vee$ and $\wedge$ denote bitwise OR and AND, respectively. When only one branch is present, the branch gets executed, but the affected values are either updated or kept unchanged depending on the result of the condition evaluation. For example, statements of the type "if ($cond$) then $a = v_1$" can be rewritten as "$a = a + cond \cdot (v_1 - a)$".

When working on Gaussian elimination suitable for secure computation, a major issue we are 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 ith 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 its correctness.

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 $i \leq k \leq n_Y$ can be non-zero and are checked. Similarly, for column $j \geq i$ only cells at rows $i \leq k \leq n_X$ are relevant and checked. After this step, all zero rows and columns are pushed to the end using oblivious compaction.

At this point we know 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, we add all rows with indices $i+1$ 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 element $A_{ii} \neq 0$. 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 $[1, R]$) results in 0 is $1/R$. Thus, with overwhelming probability (in R's bitlength) $A_{ii} \neq 0$ 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 been processed. That is, for some iteration i of the algorithm all remaining (partial) rows and columns will be zero. To ensure that the algorithm can execute exactly the same steps without revealing this fact and without affecting correctness, the only place we have to modify is computation of the inverse of A_{ii}. When $A_{ii} = 0$, A_{ii} does not have a multiplicative inverse, and we set $A_{ii} = 1$ in that case. Then because $1^{-1} = 1$, multiplying any value by 1^{-1} (as on line 4 of Algorithm 2) 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 bit corresponding to the result of comparing A_{ii} to 0.

The overall oblivious algorithm for computing matrix rank based on Gaussian elimination is given in Algorithm 3. Lines 2–5 and 7–10 compute row and column flags, respectively, that indicate whether the (partial) rows/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 ith 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, after which the rank is returned.

To realize oblivious compaction, we build on tight order-preserving compaction from [20], which was subsequently used for SMC in [6]. The algorithm proceeds in $\log_2 n$ rounds on input of a sequence of n values. At round i (for $0 \leq i \leq \log n - 1$), an element at position j is either obliviously moved 2^i elements left or is not moved. The former happens when the ith least significant bit in the number $count_j$ of zero elements that precede the jth element is 1. We refer the reader for additional details to [6,20] and provide our realization of it

Algorithm 3. $\mathsf{rank} = \mathsf{OblGERank}(\{A_{ij}\}_{1 \le i \le n_X, 1 \le j \le n_Y})$

1: **for** $i = 1, \ldots, n_X - 1$ **do**
2: **for** $j = i, \ldots, n_X$ **do**
3: Set $\mathsf{rowflag}_j = \bigvee_{k=i}^{n_Y} A_{jk}$;
4: Set $r_j = (\mathsf{rowflag}_j \stackrel{?}{\neq} 0)$;
5: **end for**
6: Use compaction to "sort" partial rows $i, \ldots, n_X$ using keys r_j, where row j is $(A_{ji}, \ldots, A_{jn_Y})$, so that all rows with $r_j = 0$ are moved to the bottom of A.
7: **for** $j = i, \ldots, n_Y$ **do**
8: Set $\mathsf{colflag}_j = \bigvee_{k=i}^{n_X} A_{kj}$;
9: Set $c_j = (\mathsf{colflag}_j \stackrel{?}{\neq} 0)$;
10: **end for**
11: Use compaction to "sort" partial columns $i, \ldots, n_Y$ using keys c_j, where column j is $(A_{ij}, \ldots, A_{n_X j})$, so that all columns with $c_j = 0$ are moved in the right in A.
12: **for** $j = i + 1, \ldots, n_X$ **do**
13: **for** $k = i, \ldots, n_Y$ **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** $j = i + 1, \ldots, n_X$ **do**
20: **for** $k = i, \ldots, n_Y$ **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 $\mathsf{ranksum} = 0$;
26: **for** $i = 1, \ldots, n_X$ **do**
27: Set $\mathsf{ranksum} = \mathsf{ranksum} + (A_{ii} \stackrel{?}{\neq} 0)$;
28: **end for**
29: Return $\mathsf{ranksum}$;

Algorithm 4. $\langle y_1, \ldots, y_n \rangle = \mathsf{Comp}(\langle x_1, \ldots, x_n \rangle)$

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

with new optimizations in Algorithm 4. It is written for the special case when the input consists of bits and moves all non-zero elements to the beginning of the input sequence. When this algorithm is used in Algorithm 3, it will take 1-bit r_j's or c_j's 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 x_{j+2^i} during the ith 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 $b_{i,j}$ is the ith least significant bit of $count_j$. At most one of x_j and x_{j+2^i} is guaranteed to be 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

Algorithm 5. $A = \mathsf{AdjMat}(X = \langle x_i, y_i, \alpha_i \rangle_{1 \le i \le m_X}, Y = \langle x'_i, y'_i, \alpha'_i \rangle_{1 \le i \le m_Y})$

1: **for** $i = 0, \ldots, m_X$ **do**
2: **for** $j = 0, \ldots, m_Y$ **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: $A_{ij} \xleftarrow{R} [1, R]$;
5: **else**
6: $A_{ij} = 0$;
7: **end if**
8: **end for**
9: **end for**
10: **return** A;

do not need to multiply x_j by $(1 - b_{i,j})$ and instead set $x_j = x_j + b_{i,j+2^i} \cdot x_{j+2^i}$. This logic (for one general and two special cases) is presented on lines 8–13 of Algorithm 4 in an optimized form.

The complexity of the oblivious compaction algorithm 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., $m_Y - i + 1$ 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 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 Sect. 8, the cost of compaction is small compared to the remaining computation.

In [11] we also present an alternative algorithm based on Gram-Schmidt process.

7 Oblivious Fingeprint Macthing Algorithms

Now we proceed with showing how the above rank computation algorithm can be used to realize oblivious fingerprint matching. To accomplish this, we first need to obliviously build a randomized adjacency matrix from the information contained in two fingerprints. We also need to modify the rank computation algorithms to implement its over-the-threshold version, in which instead of reporting the rank, the output consists of a single bit indicating whether the rank is above the desired threshold.

The regular (non-oblivious) way of computing the adjacency matrix according to Eq. 1 is presented in Algorithm 5. It simply consists of comparing each minutia from X to each minutia in Y and setting the corresponding matrix cell to a random element if the minutiae are a possible match and to 0 otherwise. To make the algorithm oblivious, we have to restructure the computation associated with the conditional statement. Our oblivious algorithm for computation of the adjacency matrix is given as Algorithm 6. For performance reasons, we eliminate

Algorithm 6. $A = \mathsf{OblAdjMat}(X = \langle x_i, y_i, \alpha_i \rangle_{1 \le i \le m_X}, Y = \langle x'_i, y'_i, \alpha'_i \rangle_{1 \le i \le m_Y})$

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

square root computation when computing the Euclidean distance (squared distances are used). Also, in the algorithm a_3 corresponds to $|\alpha_i - \alpha'_j|$ and $(c_3 \vee c_4)$ to $\min(|\alpha'_j - \alpha_i|, 360° - |\alpha'_j - \alpha_i|) \overset{?}{<} \alpha_0$.

To obtain an over-the-treshold version of the rank computation algorithm, we note that the necessary changes are rather simple. In particular, to produce of an over-the-threhold version of Algorithm 3, all we need is to return the result of comparison ($\mathsf{ranksum} \overset{?}{\geq} T$) on line 29 instead of $\mathsf{ranksum}$ itself.

Given our oblivious algorithm, it is now not difficult to realize it in the secure computation framework. Because of space considerations, we refer the reader to Appendix B and [11] for our secure protocols in both two-party and multi-party settings.

8 Implementation and Performance

To evaluate performance of our techniques, we implement our oblivious fingerprint matching algorithm in a secure computation framework. Our implementation is based on two-party garbled circuit evaluation and utilizes a tool called JustGarble [5] for efficient circuit garbling and garbled circuit evaluation.

We build Boolean circuits for Algorithm 6 followed by the over-the-threshold version of Algorithm 3 (as described in Sect. 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 them to become virtually free [26]. This means that a circuit that minimizes the use of non-XOR gates will have performance advantages over other circuits of comparable size with a 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 Eq. 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 $\overline{c} \wedge v_2$) can be non-zero at any time and thus

Table 1. Performance of secure two-party fingerprint matching using JustGarble.

Correctness parameter	Metric	Biometric size (in minutiae)				
		10	15	20	25	30
10	T_G	81.80	84.05	85.29	85.99	87.14
	T_E	50.49	53.18	53.91	54.85	54.74
	Gates	1,843,602	5,238,622	11,543,713	21,741,388	36,796,263
15	T_G	81.30	83.18	84.33	85.25	85.05
	T_E	52.12	53.37	54.16	54.57	54.12
	Gates	4,307,707	11,496,802	24,619,823	45,690,373	76,695,248
20	T_G	80.66	82.35	83.15	82.77	82.85
	T_E	53.02	53.92	54.25	53.80	53.56
	Gates	8,392,862	21,156,282	43,964,983	80,226,158	133,311,283

XOR would accomplish the same functionality as OR or addition. This applies to computation in all of Algorithms 3, 4, and 6. 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. Note that using synthetic data affects neither performance nor accuracy of the secure algorithm. 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 [28], 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-minitia 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 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 range $[0, 359]$ 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 4 GB of memory and are given in Table 1. Each experiment was run 100 times, and the double median (i.e., the median of 10 medians) is reported.

In Table 1, T_G denotes the time it takes to garble the circuit measured in the average number of CPU cycles per gate (as in [5]). 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 is often because circuits contain different percentages of XOR gates (which require substantially less work to

create and evaluate than other gates). From Table 1, we can 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 can be attributed to the varying composition of the circuits from XOR and non-XOR gates. For example, when the correctness parameter increases, a larger portion of the circuit corresponds to field operations that have a higher percentage of XOR gates than other operations. We also observed that paritioning a circuit into small circuits and evaluating the sub-circuits circuits results in slightly faster overall per-gate time compared to the original time, which is due to improved cache performance.

We note that the overall execution consists of circuit garbling, oblivious transfer (OT) for one of the parties' inputs, and garbled circuit evaluation. Circuit garbling and transfer of the garlbed 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 an OT extension [22], the number of public-key operations associated with any number of input bits is reduced to a constant corresponding to a security parameter (on the order of 96–128). Furthermore, all 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 of the circuit evaluator 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.

To provide additional information about runtime of our secure fingerprint identification protocols, we translate the numbers from Table 1 into execution times in Fig. 1 in Appendix C. We obtain runtimes on the order of a second or less, which is an acceptable delay for fingerprint authentication. Additional results can be found in [11].

Before we conclude this section, we comment on the performance of our solution compared to that of other secure fingerprint matching protocols. As mentioned earlier, the only secure fingerprint matching protocols that use minutia representations we are aware of are from [10,36]. They are based on pairing a minutia with the closest possible match minutia and all possible match minutiae, respectively, which does not achieve the same accuracy as in our solution and requires substantially less work. Implementation results are only given in [10] and the runtimes are similar to what we obtain in our solution. (And while no implementation results were reported in [36], we anticipate that performance of that solution will be substantially slower than the solution from [10].) The computation in [10] 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

fingerpring Y, the one-time cost will still be present. For fingerprints consisting of 20 minutiae, [10] reports about 5 sec 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 [10] because of higher complexity of the algorithm we use.

9 Conclusions

This work is motivated by privacy-preserving fingerprint matching in the secure computation framework, using standard minutia-based representation of fingerprints. We show that the maximum (optimal) number of minutiae that match between two fingerprints can be determined by modeling the problem as a flow network in bipartite graphs. Towards this end, we investigate the problem of maximum matching size in a bipartite graph and reduce it to the problem of finding the rank of an adjacency matrix, which has the same complexity as that of matrix multiplication. We build a data-oblivious algorithm for rank computation based on Gaussian elimination, the complexity of which is cubic in the number vertices in the graph (or 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, we choose to concentrate on simpler algorithms because of smaller constants behind the big-O notation. More advanced techniques of lower asymptotic complexity are also of limited applicability to the problem of fingerprint matching because simpler solutions with higher complexity outperform them on rather small input sizes (the number of minutiae) used in fingerprint matching. Our data-oblivious algorithms for matrix rank, maximum flow size in bipartite graphs and fingerprint matching consequently lead to secure protocols for respective problems using available secure two-party and multi-party techniques. Our implementation builds and evaluates secure two-party protocol for fingerprint matching put forward in this work. Despite having more complex computation to achieve improved 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.

Acknowledgments. This work was supported in part by grants CNS-1223699 and CNS-1319090 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.

A Security Definitions

Security in the semi-honest 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 this:

Definition 1. *Let parties $P_1, \ldots, P_n$ engage in a protocol Π that computes function $f(\mathsf{in}_1, \ldots, \mathsf{in}_n) = (\mathsf{out}_1, \ldots, \mathsf{out}_n)$, 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 Π. That is, P_i's view is formed by its input, internal random coin tosses r_i, and messages $m_1, \ldots, m_k$ passed between the parties during protocol execution: $\text{VIEW}_\Pi(P_i) = (\mathsf{in}_i, r_i, m_1, \ldots, m_k)$. Let $I = \{P_{i_1}, P_{i_2}, \ldots, P_{i_t}\}$ denote a subset of the parties for $t < n$ and $\text{VIEW}_\Pi(I)$ denote the combined view of the 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 I of size at most t there exists a probabilistic polynomial time simulator S_I such that $\{S_I(\mathsf{in}_I, f(\mathsf{in}_1, \ldots, \mathsf{in}_n)\} \equiv \{\text{VIEW}_\Pi(I), \mathsf{out}_I\}$, where $\mathsf{in}_I = \bigcup_{P_i \in I}\{\mathsf{in}_i\}$, $\mathsf{out}_I = \bigcup_{P_i \in I}\{\mathsf{out}_i\}$, and $\equiv$ denotes computational or information-theoretic indistinguishability.*

The second standard, and stronger, malicious security model assumes the participants can 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 [18].

B Secure Protocols

The data-oblivious algorithms that we developed lead to secure protocols for computing maximum bipartite matching size, matrix rank, and fingerprint matching in secure multi-party computation framework. That is, because the execution is now data-oblivious, we can combine each algorithm with available secure arithmetic techniques to provably protect private data throughout the computation. We list two possibilities.

Our first solution is to employ two-party garbled circuit evaluation (originally proposed in [46]). This technique represents the function to be evaluated as a Boolean circuit and one participant, circuit generator, encodes the circuit using two random labels for each (binary) wire. 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. After the evaluator computes the output labels, it sends them to the circuit generator, who determines their meaning (it is also possible for the

evaluator to learn the output or for both parties to learn the same or individual outputs). 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. The 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., [33] and [22]. Using garbled circuit evaluation, we can state the following result:

Theorem 1. *Assuming the existence of secure garbled circuit evaluation techniques and OT, our algorithms result in 1-private protocols for maximum bipartite matching size, matrix rank, and fingerprint matching with two participants* P_1 *and* P_2.

We refer the reader to [11] for the proofs of Theorems 1 and 2.

The second technique we suggest is threshold linear secret sharing in the multi-party setting (such as Shamir's secret sharing [37]). It allows $n > 2$ parties to securely evaluate a function on shares of private data. Before the computation starts, 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 the participants who provide the data do not have to coincide with computational parties, but instead the sets of input providers, output recipients, and computational parties can be arbitrary with respect 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. 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 maximum bipartite matching size, matrix rank, and fingerprint matching with* n *participants and* $t < n/2$.

For both two-party techniques based on garbled circuit evaluation and multi-party techniques based on linear secret sharing, there are general mechanisms for converting solutions secure in the semi-honest model to solutions secure in the stronger malicious model (see, e.g., [27] for garbled circuits and [4] 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.

C Additional Performance Results

In Fig. 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 as described in Sect. 8.

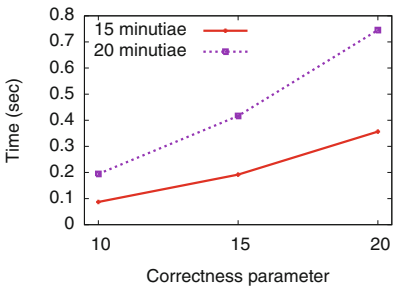

Fig. 1. Performance of garbled circuit evaluation for fingerprint matching.

References

1. Aliasgari, M., Blanton, M.: Secure computation of Hidden Markov Models. In: International Conference on Security and Cryptography (SECRYPT) (2013)
2. Aly, A., Cuvelier, E., Mawet, S., Pereira, O., Van Vyve, M.: Securely solving simple combinatorial graph problems. In: Financial Cryptography, pp. 239–257 (2013)
3. Barni, M., Bianchi, T., Catalano, D., Di Raimondo, M., Labati, R., Failla, P., Fiore, D., Lazzeretti, R., Piuri, V., Scotti, F., Piva, A.: Privacy-preserving fingercode authentication. In: ACM Workshop on Multimedia and Security (MM&Sec), pp. 231–240 (2010)
4. Beerliová-Trubíniová, Z., Hirt, M.: Perfectly-secure MPC with linear communication complexity. In: Canetti, R. (ed.) TCC 2008. LNCS, vol. 4948, pp. 213–230. Springer, Heidelberg (2008)
5. Bellare, M., Hoang, V., Keelveedhi, S., Rogaway, P.: Efficient garbling from a fixed-key blockcipher. In: IEEE Symposium on Security and Privacy, pp. 478–492 (2013)
6. Blanton, M., Aguiar, E.: Private and oblivious set and multiset operations. Cryptology ePrint Archive Report 2011/464 (2011)
7. Blanton, M., Aliasgari, M.: Secure outsourcing of DNA searching via finite automata. In: DBSec, pp. 49–64 (2010)
8. Blanton, M., Aliasgari, M.: Secure outsourced computation of iris matching. J. Comput. Secur. **20**(2–3), 259–305 (2012)

9. Blanton, M., Gasti, P.: Secure and efficient protocols for iris and fingerprint identification. In: Atluri, V., Diaz, C. (eds.) ESORICS 2011. LNCS, vol. 6879, pp. 190–209. Springer, Heidelberg (2011)
10. Blanton, M., Gasti, P.: Secure and efficient iris and fingerprint identification. In: Ngo, D., Teoh, A., Hu, J. (eds.) Biometric Security (2015)
11. Blanton, M., Saraph, S.: Secure and oblivious maximum bipartite matching size algorithm with applications to secure fingerprint identification. Cryptology ePrint Archive Report 2014/596 (2014)
12. Blanton, M., Steele, A., Aliasgari, M.: Data-oblivious graph algorithms for secure computation and outsourcing. In: ASIACCS, pp. 207–218 (2013)
13. Bogdanov, D., Laur, S., Willemson, J.: Sharemind: a framework for fast privacy-preserving computations. In: Jajodia, S., Lopez, J. (eds.) ESORICS 2008. LNCS, vol. 5283, pp. 192–206. Springer, Heidelberg (2008)
14. Damgård, I., Geisler, M., Krøigård, M.: Asynchronous multiparty computation: Theory and implementation. In: Public Key Cryptography (PKC), pp. 160–179 (2009)
15. Erkin, Z., Franz, M., Guajardo, J., Katzenbeisser, S., Lagendijk, I., Toft, T.: Privacy-preserving face recognition. In: Goldberg, I., Atallah, M.J. (eds.) PETS 2009. LNCS, vol. 5672, pp. 235–253. Springer, Heidelberg (2009)
16. Fan, K.-C., Liu, C.-W., Wang, Y.-K.: A fuzzy bipartite weighted graph matching approach to fingerprint verification. IEEE Trans. Syst. Man Cybern. **5**, 4363–4368 (1998)
17. Ford, L., Fulkerson, D.: Flows in Networks. Princeton University Press (1962)
18. Goldreich, O.: Foundations of Cryptography: Volume 2, Basic Applications. Cambridge University Press, Cambridge (2004)
19. Goldreich, O., Ostrovsky, R.: Software protection and simulation on oblivious RAMs. J. ACM (JACM) **43**(3), 431–473 (1996)
20. Goodrich, M.: Data-oblivious external-memory algorithms for the compaction, selection, and sorting of outsourced data. In: SPAA, pp. 379–388 (2011)
21. Ibarra, O., Moran, S.: Deterministic and probabilistic algorithms for maximum bipartite matching via fast matrix multiplication. Inf. Process. Lett. **13**(1), 12–15 (1981)
22. Ishai, Y., Kilian, J., Nissim, K., Petrank, E.: Extending oblivious transfers efficiently. In: Boneh, D. (ed.) CRYPTO 2003. LNCS, vol. 2729, pp. 145–161. Springer, Heidelberg (2003)
23. Jain, A., Prabhakar, S., Hong, L., Pankanti, S.: Filterbank-based fingerprint matching. IEEE Trans. Image Process. **9**(5), 846–859 (2000)
24. Jea, T.-Y., Govindaraju, V.: A minutia-based partial fingerprint recognition system. Pattern Recogn. **38**(10), 1672–1684 (2005)
25. Keller, M., Scholl, P.: Efficient, oblivious data structures for MPC. Cryptology ePrint Archive Report 2014/137 (2014)
26. Kolesnikov, V., Schneider, T.: Improved garbled circuit: free XOR gates and applications. In: Aceto, L., Damgård, I., Goldberg, L.A., Halldórsson, M.M., Ingólfsdóttir, A., Walukiewicz, I. (eds.) ICALP 2008, Part II. LNCS, vol. 5126, pp. 486–498. Springer, Heidelberg (2008)
27. Kreuter, B., shelat, A., Shen, C.: Billion-gate secure computation with malicious adversaries. In: USENIX Security Symposium (2012)
28. Lovasz, L.: On determinants, matchings and random algorithms. Fundam. Comput. Theor. **79**, 565–574 (1979)

29. Lu, S., Ostrovsky, R.: Distributed oblivious RAM for secure two-party computation. In: Sahai, A. (ed.) TCC 2013. LNCS, vol. 7785, pp. 377–396. Springer, Heidelberg (2013)
30. Malkhi, D., Nisan, N., Pinkas, B., Sella, Y.: Fairplay - a secure two-party computation system. In: USENIX Security Symposium, pp. 287–302 (2004)
31. Maltoni, D., Maio, D., Jain, A., Prabhakar, S.: Handbook of Fingerprint Recognition, 2nd edn. Springer, London (2009)
32. Mucha, M., Sankowski, P.: Maximum matchings via Gaussian elimination. In: IEEE Symposium on Foundations of Computer Science, pp. 248–255 (2004)
33. Naor, M., Pinkas, B.: Efficient oblivious transfer protocols. In: SODA (2001)
34. Pathak, M., Portelo, J., Raj, B., Trancoso, I.: Privacy-preserving speaker authentication. In: Gollmann, D., Freiling, F.C. (eds.) ISC 2012. LNCS, vol. 7483, pp. 1–22. Springer, Heidelberg (2012)
35. Sadeghi, A.-R., Schneider, T., Wehrenberg, I.: Efficient privacy-preserving face recognition. In: Lee, D., Hong, S. (eds.) ICISC 2009. LNCS, vol. 5984, pp. 229–244. Springer, Heidelberg (2010)
36. Shahandashti, S.F., Safavi-Naini, R., Ogunbona, P.: Private fingerprint matching. In: Susilo, W., Mu, Y., Seberry, J. (eds.) ACISP 2012. LNCS, vol. 7372, pp. 426–433. Springer, Heidelberg (2012)
37. Shamir, A.: How to share a secret. Commun. ACM **22**(11), 612–613 (1979)
38. Solodovnikov, V.: Extension of Strassen's estimate to the soultion of arbitrary systems of linear equations. USSR Comput. Maths. Math. Phys. **19**, 21–33 (1978)
39. Stefanov, E., van Dijk, M., Shi, E., Fletcher, C., Ren, L., Yu, X., Devadas, S.: Path ORAM: An extremely simple oblivious RAM protocol. In: CCS, pp. 299–310 (2013)
40. Strassen, V.: Gaussian elimination is not optimal. Numer. Math. **13**, 354–356 (1969)
41. The Corbett Report. India fingerprints, iris scanning over one billion people. http://www.corbettreport.com/india-fingerprinting-iris-scanning-over-one-billion-people/
42. Troncoso-Pastoriza, J., Katzenbeisser, S., Celik, M.: Privacy preserving error resilient DNA searching through oblivious automata. In: CCS, pp. 519–528 (2007)
43. UAE Iris Collection. http://www.cl.cam.ac.uk/~jgd1000/UAEdeployment.pdf
44. U.S. Dhs Office of Biometric Identity Management. http://www.dhs.gov/obim
45. Wang, C., Gavrilova, M., Luo, Y., Rokne, J.: An efficient algorithm for fingerprint matching. In: International Conference on Pattern Recognition (ICPR), pp. 1034–1037 (2006)
46. Yao, A.: How to generate and exchange secrets. In: FOCS, pp. 162–167 (1986)
47. Zhang, Y., Steele, A., Blanton, M.: PICCO: a general-purpose compiler for private distributed computation. In: CCS, pp. 813–826 (2013)