Secure Outsourced Computation of Iris Matching

Marina Blanton and Mehrdad Aliasgari
Department of Computer Science and Engineering
University of Notre Dame
{mblanton,maliasga}@cse.nd.edu

Abstract

Today biometric data propagate more heavily into our lives. With more ubiquitous use of such data, computations over biometrics become more prevalent as well. While it is well understood that privacy of biometric data must be protected, often computations over biometric data involve untrusted participants or servers, let it be a cross check between different agencies who are not permitted to share the data or a researcher testing a new biometric matching algorithm on a large scale that forces the computation to be placed on a grid. Unarguably, it would be desirable to secure computation over sensitive biometric data in such environments. Currently, no secure techniques for outsourcing biometric comparisons or searching are readily available, and this work makes the first step at designing solutions for secure outsourcing iris identification to one or more untrusted servers. We develop new solutions for the single-server (i.e., non-interactive) and multiple-server settings that use significantly different techniques. Furthermore, we carry out extensive experimentation on a database of iris codes to both validate the findings and achieve efficiency improvements.

1 Introduction

The need for individual privacy is widely recognized. With biometric authentication becoming more reliable and readily available than before, the need to protect such information is apparent. Furthermore, unlike other types of data used for authentication purposes (passwords, key material, secure tokens, etc.), biometric data cannot be revoked and replaced with a new value, which calls for even stricter protection of such data.

In recent years a significant amount of research effort has been dedicated to protecting biometric data from the server that stores a database with biometric templates for authentication purposes. The idea is, instead of storing the biometrics themselves, to store a function of each biometric such that the value can be used for authentication purposes, but in the case of server compromise it does not lead to compromise of the biometric data. Such solutions include work on fuzzy vault [51, 18, 19], secure sketches and fuzzy extractors [52, 32, 10, 33, 11, 31], shielding functions [58, 73, 70], cancelable or revocable biometrics [65], and many other publications derived from them, especially in the biometrics literature.

In this work we argue that concentrating on authentication alone is not enough, and there is a need to protect sensitive biometric data in other environments and contexts where computation over biometric data, normally for the purposes of identification, is involved. For instance, testing new algorithms for extracting biometric features on large collections of biometric images requires a massive amount of computation and has memory requirements beyond a single machine. This forces such computations to be placed on a grid and be conducted in a distributed manner [13]. Computers comprising the grid, however, are normally less trusted to preserve privacy of the data

than machines controlled by the researcher and thus it is necessary to secure the data used by the machines connected to the grid while computing the task. Other apparent need for privacy protection arises when the computation must be performed across data owned by different entities and/or on outsourced data, e.g., circumstances when an investigator has the need to search biometric databases collected by different agencies; when two organizations that do not completely trust each other with their sensitive data or are prohibited from disclosing the data by law or other provisions would like to carry out computation over their respective databases; when the biometrics data is placed on an external server (for computational or other reasons) and there is a need to compute over obfuscated data; etc.

In all of the above cases, given a particular biometric, the server is to search the database and output information about matches. We wish both the biometric templates stored in the server database and the biometric being searched for to stay hidden from the server performing the search and at the same time not to place a significant computational burden on the client making the query. Existing techniques on biometric-based authentication summarized above achieve secrecy of the stored data from the server when an individual is to authenticate against her own record, but cannot be extended to the case of identification, where the entire database is to be searched for possible matches with the given biometric. This means that no existing tools or techniques are readily available to carry out this task of practical significance. The problem of secure biometric identification or matching with the aid of untrusted servers is thus the focus of this work.

In addressing this problem, we distinguish between the settings where only one untrusted server can be utilized for secure biometric matching and when a larger number of such servers is available. Providing solutions to this problem in either of the above settings presents challenges for the following reasons. When only one untrusted server is available to carry out secure computation, all computation must be performed over secured data in a non-interactive manner[1] which is a difficult task. In particular, it is known that secure non-interactive solutions are not possible for certain types of problems [2]. In this work we show that even using recent powerful techniques, only a limited functionality for biometric matching can be securely realized through non-interactive computation and at a significant cost. This limited computation nevertheless closely approximates the necessary computation and thus achieves the goal we put forward. The multi-server setting, on the other hand, can be modeled as a special case of secure multi-party computation [75, 44], where the participating parties contribute no input and receive no or very limited output. At the time of this writing, no privacy-preserving protocol is known to perform the functionality we seek other than general techniques that can evaluate any computable function. General solutions, however, are known to bear unreasonably high overhead, especially when dealing with large amounts of data (see Section 2.3). Furthermore, while custom privacy-preserving protocols are known for certain types of operations that are more efficient than general techniques (e.g., scalar product [74, 43], set intersection [39, 55], and others), they do not implement the functionality we seek.

Since biometric-based identification techniques heavily depend on the type of biometric used, in this work we concentrate on iris-based matching. Iris is attractive due to a large amount of uncertainty an individual biometric contains (some other types of biometric cannot be used for identification due to high error rates and are only suitable for verification). Additionally, after extracting features from an iris image, it is represented as a binary string, which makes it convenient to work with. Our consecutive work will investigate other popular types of biometric such as faces

[1] Because the client is normally weak, it becomes infeasible for the client to perform work proportional to the size of the biometric database, resulting in limited interaction between the client and the server. When, however, the client is not weak and can carry out the work proportional to the database size, it can, depending on the security requirements, either carry out the computation itself or assume the role of a computational server, resulting in a multi-server setting.

and fingerprints. Throughout this work, we will use biometric reading to denote raw biometric data, and biometric representation or biometric template to denote processed biometric (i.e., after feature extraction).

Organization. We start by describing the model, necessary computation, security requirements, and our contributions in Section 2. Our solution for iris identification in the single-server case is given in Section 3, and Section 4 describes our solution in presence of multiple helper servers. We then discuss the use of approximations for the purposes of reducing computation cost in Section 5. Experimental results performed on an iris database are reported in Section 6. Finally, we describe prior work in Section 7 and conclude in Section 8.

2 Overview of the Model and Solution

2.1 Problem description

Let an iris biometric X be represented as an m-bit binary string. We use X_i to denote ith bit of such a string. We assume that the database owner compiles a database D consisting of biometric templates X. The database is stored at one or more servers, which do not obtain access to the raw data (we specify security assumptions with respect to the servers' behavior in Section 2.2). A client has biometric Y and queries the server for identification purposes. The server (or a number of servers) execute a secure protocol to find all biometric templates stored in the database that match the queried biometric Y and send the result to the client.

For all of the protocols, we assume that there is an initialization phase, Setup, that consists of initializing the system and populating the database and the actual query execution phase, Query, during which a client forms a query, submits it to the server, and obtains information about matches back from the server.

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

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

the computation of the Hamming distance that uses masks becomes [28]:

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

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

Finally, two iris representations can be slightly misaligned, which is caused by head tilt during image acquisition. To account for this, the matching process attempts to compensate for the error and rotates the biometric representation by a fixed amount to determine the lowest distance. This rotation corresponds to circular left and right shifts of the binary representation[2] a small fixed number of times, which we denote by c. The minimum Hamming distance across all runs is then compared to the threshold. In other words, if we let $\mathsf{LS}^j(\cdot)$ (resp., $\mathsf{RS}^j(\cdot)$) denote a circular left (resp., right) shift of the argument by a fixed number of bits (2 bits in experiments conducted by the biometrics group at our institution) j times, the matching process becomes:

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

2.2 Security requirements

As with any cryptographic solution, we require completeness and soundness properties to hold. Therefore, a secure biometric search scheme must meet the following requirements:

Correctness: The secure computation should be performed correctly, i.e., the computation identifies and returns all values that correspond to authentic matches and a minimal number of false matches (which would correspond to the false reject rate (FRR) of 0 and the minimum achievable false accept rate (FAR)).

Security: The servers should not learn information about the data they store in the database D and information about the data contained in the queries. We, however, allow the servers to learn information about matches, i.e., the indices of the records that matched the query, which are consequently returned to the client.

The correctness property requires secure computation to compute the same function as would be computed without applying the privacy-preserving techniques. The error rates of the secure solution then heavily depend on the error rates of the underlying iris codes, but, as we show in this work, can be improved via rather simple means.

The security relaxation above that allows the servers to learn information about the records that matched the client's query (if any) is dictated by efficiency reasons (otherwise, the client's output is proportional to the database size). The indices are then communicated to the client, and this information is considered to be sufficient for the purposes for which it will be used (i.e., for the client it is crucial to know whether there is at least one match, and more information about the matched individuals can be obtained through any other suitable mechanism using the indices it receives). Using the terminology from work on searches over encrypted data, this notion of security is match-revealing (as opposed to the match-concealing type).

In addition, we explicitly specify the efficiency constraints. Communication and computation complexity of a client should be linear in the size of its input (i.e., biometric template) and the output it receives (i.e., the number of matches). In other words, it is unreasonable to require the client to perform computation proportional to the size of the database, which is securely outsourced to the servers. Communication and computation complexity (including round complexity) of the servers should be minimized if possible. In other words, when different realizations of a specific functionality are possible, our goal is to build a solution with best performance.

[2]More precisely, each biometric is represented as a two-dimensional array and during shifting a circular shift is applied to each row. The explanation given so far, however, is sufficient, because if a biometric representation (even in a scrambled form) can be partitioned into individual bits, necessary shifting can always be performed correctly.

As far as trust assumptions with respect to the servers' behavior go, we can distinguish between semi-honest (or passive) and malicious (or active) adversarial models. Semi-honest participants might attempt to learn some information from the data they receive including intermediate results, but will not deviate from the prescribed computation. Malicious participants, on the other hand, can arbitrarily deviate from the computation by altering intermediate results, aborting participation, etc. in the attempt to gain any advantage. We first consider security against semi-honest servers and then also treat the malicious model: we show that our multi-server solution can be made secure against malicious servers using known techniques and we also show that it is impossible to secure a single-server solution against a malicious server without requiring the client to perform work proportional to the database size.

While our framework allows for alternative trust assumptions with respect to the database owner and the client (i.e., they can be trusted parties (or even the same party) who outsource the computation due to computational resources and infrastructure reasons or they can be mutually distrustful parties who cannot reveal the data to each other), both the client's and the database owner's data must be protected from the computational servers. The rapid emergence of cloud computing services introduces a significant shift in the landscape of large-scale or distributed computing. Acquiring necessary computing power for on-demand custom computation is readily available from several industrial providers at cheap cost. While a significant number of prior publications already treat large institutional servers as semi-honest, e.g., [57, 35, 62] (and even non-colluding, e.g., [71, 72]), in the current cloud computing paradigm service providers face liability issues (in case of incompliance with customer contracts, hosting illegal information, etc.). This means that highly reputable and legally bound service providers are expected to guarantee a reasonably high level of security. Therefore, it is realistic to assume that (at least some of) the servers behave semi-honestly. The situation, however, can significantly change in the case of so-called volunteer computing, where individuals or organizations donate their unused CPU cycles toward large-scale computation. This can be common in biometric research where a new biometric matching algorithm needs to be tested on large data sets to collect statistical information about its performance [13]. In that case, techniques for ensuring security in presence of malicious participants would be desirable. In what follows, we start with security in the semi-honest model.

We next formally define security using the standard definition in secure multi-party computation for semi-honest adversaries. Since the helper servers do not contribute any data to the computation, this should be interpreted as no private input to the function they are evaluating. Then for the purposes of the security definition, all data the servers receive before or during the computation (i.e., the database and user queries) will be considered to be a part of the function evaluation and therefore must not leak any information. It, however, must be understood that the servers will be pre-loaded with some data (the biometric database) before any computation takes place. We denote "no data" by a special character $\perp$. The output of each participant can include information about query matches, but no other information.

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 party P_i, respectively. Let $\mathrm{VIEW}_\pi(P_i)$ denote the view of participant P_i during the execution of protocol π. More precisely, P_i's view is formed by its input and internal random coin tosses r_i, as well as messages $m_1, \ldots, m_t$ passed between the parties during protocol execution:*

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

Let $I = \{P_{i_1}, P_{i_2}, \ldots, P_{i_t}\}$ denote a subset of the participants for $t < n$ and $\mathrm{VIEW}_\pi(I)$ denote the combined view of participants in I during the execution of protocol π (i.e., the union of the views

of the participants 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

$$\{S_I(\mathsf{in}_I, f(\mathsf{in}_1, \ldots, \mathsf{in}_n)\} \equiv \{\mathrm{VIEW}_\pi(I), \mathsf{out}_I\},$$

where $\mathsf{in}_I = \bigcup_{P_i \in I}$, $\mathsf{out}_I = \bigcup_{P_i \in I}\{\mathsf{out}_i\}$, *and* $\equiv$ *denotes computational indistinguishability.*

This definition can be interpreted in different ways: when the database owner and the client can trust each other with their data, the parties P_1 through P_n above will correspond to computational servers only who do not contribute any data. That is, for each computational server P_i we have that $\mathsf{in}_i = \perp$. When, however, the client and the database owner are separate distrustful entities, their data must be protected from each other[3] which can be achieved by including them in the set of n participants. In that case, the client's input is $\mathsf{in}_i = \langle Y, M(Y)\rangle$ and the database's owner input is $\mathsf{in}_j = D$.

As a special case, we have that computation can involve only one server. In this case, the security definition must hold as well. Note that in the case of multiple parties, the standard definition of semi-honest participants allows them to combine their views (but the participants will not arbitrarily deviate from the protocol or disrupt it). The security guarantees must hold as long as the coalition size does not exceed a specific threshold t.

In our context, one or more participating parties compute the function f given in Equation 2, where the values X and Y are treated as a part of the function f when the client and the database owner are not among the participants. After evaluation of this function, the parties learn a bit (i.e., whether the distance was below the threshold or not), and no other information including any information about the inputs is available to the computational servers.

2.3 Our contributions

This work is the first to address the problem of secure outsourced biometric identification and develop new techniques for securely comparing iris codes. Our solution depends on whether only one untrusted server or several servers are available to carry out biometric search.

Single server secure biometric search SSSBS. The difficulty in building a non-interactive solution for the required computation securely lies in the fact that the computation involves operations (XOR, addition, and comparison) that often require different techniques (and negative results exist showing that some functionality is impossible to achieve securely in a non-interactive setting). Note that a generic secure circuit evaluation cannot be used because it would require work proportional to the database size from either the database owner or the client on each client query, resulting in no benefit at all from using the helper server. Furthermore, the techniques for secure biometric authentication (against the user's own record) are completely inapplicable as well.

Our solution in the single-server setting is to employ predicate encryption. The most powerful existing predicate encryption schemes permit only computation of scalar product computation. Counter-intuitively, we were able to implement most of necessary operations including computation of minimum and comparison using a scalar vector computation alone. The only operation that cannot be carried out is the division. To address this, we use approximations for the division operation and empirically show that the introduced error is very small. We prove security when the server follows the prescribed behavior, and we also show that if the server deviates from the computation correctness cannot be achieved with the desired complexity on the client's (or database owner's) part.

[3]We can also allow for one-sided security guarantees, when, for instance, the database must be protected from the client, but the server can learn information about the client's queries.

This single-server solution is far from practical for large databases or biometric representations of large size, but can be used as a proof-of-concept solution when only one server is available. Thus, we next build a more practical multiple-server solution when more than one server is available.

Multiple server secure biometric search MSSBS**.** As mentioned in the introduction, this setting can be modeled as a special case of secure multi-party computation [75, 44]. General techniques for secure function evaluation are applicable, but are known to result in unbearable overheads especially when dealing with massive amounts of data. Such a solution in our case would mean that a new oblivious circuit must be generated for *each client's query* (i.e., circuits cannot be reused if the security guarantees are to be maintained). This means that, besides the circuit evaluation itself, either the database owner must transfer all of the data by performing a public-key operation per bit of the database or each participating server must perform one public-key operation per bit of the database, which clearly results in unreasonable cost. Our solution avoids public-key cryptography, and all computations involve much shorter numbers (which is important since a lot of computation is carried out over bits).

In our solution, the database is shared among the servers in a split form, and biometric search takes the form of a secure multi-party computation over shared data. We implement all operations necessary for carrying out a search over iris codes. While the division operation requires the most involved protocol, we restructure the computation to avoid the division operations, which substantially improves performance of the solution. As designing the building blocks to be as efficient as possible is an important aspect of this work, we perform careful analysis and evaluation of the techniques. An interesting element of our solution is that the number of interactive operations for comparing two iris codes of size m is only logarithmic in m, where the bulk of the computation is carried out locally. In our solution the client incurs minimal overhead (i.e., only splits its biometric query among the servers using a secret sharing scheme); the same holds for the database owner as well. The solution is secure in both semi-honest and malicious models. We also suggest approximations that reduce the cost of the protocol.

Empirical validation. Unique empirical experiments and analysis using a database of iris codes constitute a significant part of this work. We first develop two majority coding algorithms for iris codes that take into account iris representations used in practice and apply them to sample data to improve performance of the original biometric matching. The iris data is then used to evaluate the accuracy of several approximation and optimization techniques. All of these techniques, as our results suggest, introduce only a small error and thus have practical relevance.

3 Single-Server Solution

3.1 Preliminaries

For this setting, we utilize predicate encryption – a new type of encryption introduced in [53] – which allows for fine-grained control over access to encrypted data. The first public-key predicate encryption was given in [53], and a symmetric key predicate encryption with additional privacy guarantees was introduced in [67]. As the privacy properties we seek cannot be achieved using public-key encryption (see below), a brief description of predicate encryption we provide here will be in the context of the symmetric key setting.

In a symmetric key predicate encryption scheme, the owner of the master key can create and issue secret key tokens to others. Tokens correspond to some predicates and ciphertexts are associated with attributes. Decryption of a ciphertext associated with attribute I using a token for predicate f is successful if f evaluates to 1 on I. Predicate encryption must provide *plaintext*

privacy, where no information about the plaintext with attribute I can be learned using tokens for predicates that evaluate to 0 on I. Additionally, we are interested in *predicate privacy* achieved in [67], which ensures that, given a token for predicate f, no information about the predicate can be discovered beyond the result of application of the token to encrypted data.[4] The most powerful constructions of predicate encryption known to date (i.e., [53, 67]) support evaluation of inner products (over $\mathbb{Z}_N$ for a large integer N), which in turn enables construction of predicates corresponding to disjunctions, polynomials, etc. These types of constructions can be used as predicate-only schemes, with no messages included in a ciphertext, and this is the variant we use in our construction.

When we use a predicate encryption scheme, we refer to it by its four algorithms, namely:

- $\mathsf{PESetup}(1^\kappa)$ which, on input security parameter κ, generates public parameters[5] $param$ and the master key mk;
- $\mathsf{PEEncrypt}$ which, on input the master key mk and a plaintext attribute I, produces a ciphertext C_I;
- $\mathsf{PEGenToken}$ which, on input the master key mk and a predicate f, produces a token T_f;
- $\mathsf{PEQuery}$ which, on input parameters $param$, a token T_f for predicate f and a ciphertext C_I for plaintext I, outputs 0 or 1 which indicates the value of the predicate evaluated on the underlying plaintext.

The security requirements are such that ciphertext C_I does not reveal information about the corresponding plaintext I (plaintext privacy) and token T_f does not reveal information about the corresponding predicate f (predicate privacy).

For the type of predicate encryption that supports evaluation of inner products, let attribute I be represented as a vector of integers $\vec{a} = \langle a_1, \ldots, a_n \rangle \in \mathbb{Z}_N^n$ and predicate f be represented as a vector $\vec{b} = \langle b_1, \ldots, b_n \rangle \in \mathbb{Z}_N^n$. The predicate f then evaluates to true on I if the inner product $\vec{a} \odot \vec{b} = a_1 b_1 + \ldots + a_n b_n \bmod N$ is equal to zero. In other words, $f_{\vec{b}}(\vec{a}) = 1$ iff $\vec{a} \odot \vec{b} = 0$. The inner product functionality allows one to evaluate any polynomial $p(x) = c_t x^t + \ldots + c_1 x + c_0$ on a specific point x_0 by creating vectors $\vec{a} = \langle c_t, \ldots, c_1, c_0 \rangle$ and $\vec{b} = \langle x_0^t, \ldots, x_0, 1 \rangle$, so that $\vec{a} \odot \vec{b} = p(x_0)$. Then we can test whether a polynomial evaluates to a specific value d by setting I to be $\vec{a'} = \langle c_t, \ldots, c_1, c_0, -d \rangle$ and f to be $\vec{b'} = \langle x_0^t, \ldots, x_0, 1, 1 \rangle$ (or vice versa). In this case, $f(I)$ is true (and decryption is successful) if and only if $\vec{a'} \odot \vec{b'} = p(x_0) - d = 0$ or equivalently $p(x_0) = d$. In general, the polynomial p can be over any number of variables.

Boolean OR $f_1(I_1) \vee f_2(I_2)$ can be computed by first representing the predicate computation as (possibly multi-variate) polynomials p_1 and p_2 and then computing a new polynomial $p_3 = p_1 \cdot p_2$. Polynomial p_3 evaluates to 0 (which means that the predicate evaluates to true) if at least one of p_1 and p_2 evaluates to 0.

The constructions of predicate encryption in [53, 67] use bilinear groups whose order is a product of three distinct secret primes. Performing $\mathsf{PEQuery}$ in [67] involves $2n + 2$ bilinear pairing operations.

[4] Notice that in the public-key setting, an adversary can create any number of ciphertexts corresponding to attributes I of its choice. The adversary can then evaluate a token for an unknown predicate f on these ciphertexts in the attempt to learn as much information about the predicate as possible based on the success of the decryption. In the symmetric-key setting, on the other hand, the owner of the key has control over what ciphertexts are available to the adversary and therefore can mitigate information leakage.

[5] We note that the scheme in [67] was not specified to produce public parameters, but the group description is necessary for evaluating predicates on ciphertexts, and we therefore explicitly include public parameters in the output of the algorithm.

3.2 Protocol

Recall that the functionality we would like to compute is given in equation 2. We use the templates stored in the database to form ciphertexts (i.e., each template will be used to construct an attribute) and user query data to form secret key token (to form predicates). Evaluation of the inner product of the vector corresponding to an attribute and the vector corresponding to a predicate amounts to testing whether the Hamming distance is below some threshold T. As the computation that can be performed with a single invocation of a scalar product protocol is limited, we provide a solution that computes an approximation of the computation of equation 2 and later estimate the error introduced by the approximation.

Let M_i denote the ith bit of mask $M(X)$ and M'_i the ith bit of mask $M(Y)$. Then we can compute $||(X \oplus Y) \cap M(X) \cap M(Y)||$ using a scalar product of vectors $\vec{x}$ and $\vec{y}$, where $\vec{x} = \langle x_1, \ldots, x_{2m} \rangle = \langle M_1(1-2X_1), M_1X_1, M_2(1-2X_2), M_2X_2, \ldots, M_m(1-2X_m),\ M_mX_m \rangle$ and $\vec{y} = \langle y_1, \ldots, y_{2m} \rangle = \langle M_1Y_1, M_1, M_2Y_2, M_2, \ldots,\ M_mY_m, M_m \rangle$. That is, for each $i = 1, \ldots, m$, $\langle M_i(1-2X_i), M_iX_i \rangle \odot \langle M'_iY_i, M'_i \rangle = M_i(1-2X_i)M'_iY_i + M_iX_iM'_i = (X_i + Y_i - 2X_iY_i) \cdot M_i \cdot M'_i = (X_i \oplus Y_i) \wedge M_i \wedge M'_i$, and the scalar product of entire $\vec{x}$ and $\vec{y}$ produces the overall Hamming distance $HD(X,Y) = \sum_{i=1}^{m} ((X_i \oplus Y_i) \wedge M_i \wedge M'_i)$.

This computation can be represented as a polynomial p, where the x_i's serve the role of variables, and the y_i's the role of coefficients (or vice versa). Then to find out whether the Hamming distance falls below the threshold T, we form a new polynomial that allows the difference to be anywhere in the range $[0, T-1]$ as a new polynomial $q = p(p-1)\cdots(p-(T-1))$. That is, q corresponds to the Boolean OR of predicates testing whether the Hamming distance takes a specific value i between 0 and $T-1$.

For the computation of the minimum among all shifts, since several comparisons cannot be implemented directly within the scalar product, we have to restructure the computation to perform the following: we form polynomials $q_{-c}, \ldots, q_c$, where q_i denotes polynomial q with i amount of right shift in biometric Y, and compute their OR by taking their product. Thus, instead of directly computing the minimum distance and testing whether it lies in the interval $[0, T-1]$, we test whether some Hamming distance among all shifts lies in the interval $[0, T-1]$. Then if at least one Hamming distance is in the range $[0, T-1]$, the overall predicate will evaluate to true.

The given tools do not allow us to divide the computed distance by the size of the mask overlap $||M(X) \cap M(Y)||$. Therefore, we experimentally compute this size and multiply the threshold T according to that estimate. Additionally, we consider skipping the template rotation for efficiency reasons. Since both of these changes can degrade the quality of the answer, we empirically evaluate their impact and report it in Section 6.2. Our experiments indicate that such approximations can result in a small error.

Protocol SSSBS

Setup:

1. The database owner runs PESetup using security parameter κ and obtains public parameters $param$ and master key mk.

2. For each $\langle X, M(X) = M \rangle \in D$, represent the biometric as a vector of length $2m$ formed as $\vec{x} = \langle M_1(1-2X_1), M_1X_1, M_2(1-2X_2), M_2X_2, \ldots, M_m(1-2X_m), M_mX_m \rangle$. Use $\vec{x}$, T, and c to form ciphertext C_X (with no message) corresponding to the coefficients of $Q_X = \prod_{i=-c}^{c} q_i$, where each q_i is computed from $\vec{x}$ and T as described above, using PEEncrypt and master key mk.

3. Store the ciphertexts C_X for each X at the server.

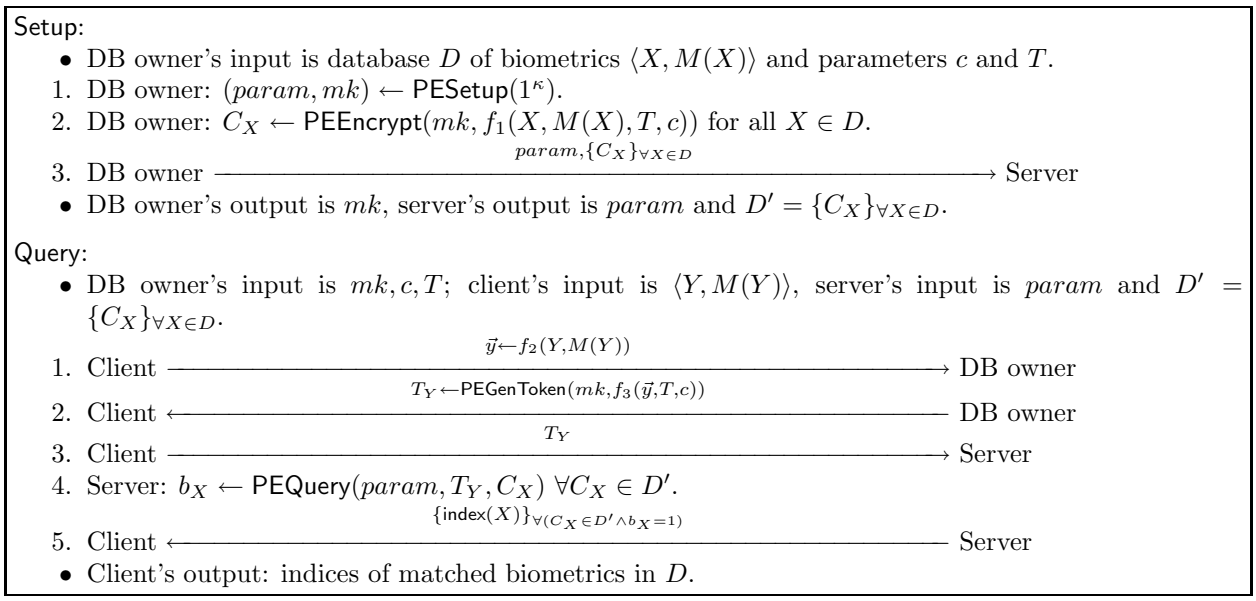

Figure 1: Summary of SSSBS protocol.

Query:

1. Given biometric representation Y with mask $M(Y) = M$, the client forms a vector of length $2m$ as $\vec{y} = \langle M_1 Y_1, M_1, M_2 Y_2, M_2, \ldots, M_m Y_m, M_m\rangle$ and sends it to the database owner.

2. The database owner $\vec{y}$, T, and c to construct the predicate P_Y for the desired computation, which is a vector of variables in $\vec{y}$ and their higher degrees such that the polynomial encoded in Q_X could be evaluated on Y. The database owner then generates a token T_Y for P_Y by executing $\mathsf{PEGenToken}(mk, P_Y)$ and sends T_Y to the client.

3. The client sends T_Y to the server.

4. The server applies this token to each ciphertext in the database using PEQuery and returns indices (if any) of templates which could be decrypted (i.e., on which the predicate evaluated to 1).

Figure 1 summarizes the computation and interaction. For compactness, it slightly abuses the notation by using $\forall X \in D$ to mean $\forall \langle X, M(X)\rangle \in D$. Functions f_1, f_2, and f_3 are detailed in the full description. Each bit b_X is set to the result of evaluating the predicate on the ciphertext, and notation $\mathsf{index}(X)$ denotes position of X in D.

If template rotation is not used, as an alternative solution, each database entry can be stored as T randomly permuted ciphertexts (each testing for a specific value between 0 and $T-1$). The benefit of this approach is that the query size is now at most $2m+1$ group elements (and each record stored at the database also has the same size), which significantly reduces the overhead associated with the scheme and makes it significantly more practical (see analysis below). This modification, however, slightly weakens the secrecy guarantees of the scheme, where the server would obtain the value of the Hamming distance whenever it is below T.

3.3 Analysis

Theorem 1 *Assuming that the predicate encryption scheme* (PESetup, PEEncrypt, PEGenToken, PEQuery) *achieves ciphertext and predicate privacy, protocol* SSSBS *is secure in the presence of a semi-honest*

server.

Proof To show the security, we build a simulator S that, given the server's input and output, produces the server's view which indistinguishable from the protocol execution. Recall that the server learns bit b_X for each X in D as its output, and the server's input consists only of computation parameters m, T, and c.[6]

S first forms a vector $\vec{I_X}$ for each X in D of the same length as used in real protocol execution. This vector will be used as an attribute in forming a simulated ciphertext for X. S also forms a predicate vector $\vec{P}$ of the same length, which would correspond to the simulated predicate for Y. All $\vec{I_X}$ and $\vec{P}$ can be chosen arbitrarily with the constraint that $\vec{I_X} \odot \vec{P} = 0$ iff $b_X = 1$ (this particular means that the elements of $\vec{I_X}$ and $\vec{P}$ can be from a different range than during the real protocol execution, where many elements would correspond to a bit). The simulator executes $(param, mk) \leftarrow \mathsf{PESetup}$ and then $\mathsf{PEEncrypt}(mk, \vec{I_X})$ for each $\vec{I_X}$ and obtains ciphertext C_X encoding the attribute. Similarly, S executes $\mathsf{PEGenToken}(mk, \vec{P})$ to obtain token T_Y for $\vec{P}$. S simulates the Setup protocol by sending $param$ and ciphertexts C_X to the server. To simulate the Query protocol, S sends T_Y to the server. Then the server participates in multiple Query protocols, S can be modified to produce C_X's and T_Y's for each query execution such that the output of $\mathsf{PEQuery}$ matches the values of b_X that the server obtains as a result of predicate evaluation on each ciphertext.

We now analyze the server's ability to distinguish this simulation from the real protocol execution. The value of $param$ that the server receives during simulation is distributed identically to the real protocol execution and therefore, the server is unable to tell the difference. The server is also unable to distinguish the ciphertexts C_X it receives during the simulation from the ciphertexts in real execution due to the ciphertext privacy property of the predicate encryption scheme. Finally, the server is unable to distinguish between the token(s) T_Y it receives during simulation and real protocol execution because of the predicate privacy property of the encryption scheme. □

Correctness of the computation approximation is empirically evaluated in Section 6.2.

Before proceeding with complexity analysis of our solution, we discuss security in presence of a fully malicious server. We next show that any protocol secure in the malicious model would impose computation and communication requirements on the database owner and/or client which exceed the efficiency requirements put forward in this work. In particular, recall that the purpose of using an external computational server is to eliminate the need to the database owner or the client to perform work proportional to the size of the biometric database D. Suppose that the server is malicious. This means that even if the server performed comparisons with each record in D honestly, it can always modify the result and return an incorrect set of indices to the client. In order for the client to ensure that each returned index indeed resulted in a match and no indices are missing from the set that the client receives, the client will need to verify the result of comparison its Y with each record in D. This means that $O(|D|)$ computation and communication is unavoidable. This result is not specific to our solution. Any possible solution that guarantees verifiable computation would have to impose the same complexity on the client (or the database owner).

The overhead of the solution in Section 3.2 is determined by the size of the representation of the templates and queries as polynomials for scalar product computation. As was shown earlier, comparing a Hamming distance (for m-bit biometrics) with a fixed value produces vectors of size $2m+1$ group elements. It might seem that constructing the polynomial $q = p(p-1)\cdots(p-T+1)$

[6]Note that the values of m, T, and c might not be explicitly given to the server, but the server will likely be able to approximate them from the ciphertexts it stores. Because such values are a part of the algorithm, they normally would not be considered secret and are included explicitly.

to compare the distance to the threshold would result in vectors of size $2mT + 1$ (and of size $2mT(2c + 1) + 1$ with shifting and computation of minimum), but unfortunately for multivariate polynomials the number of monomials exhibits exponential growth after such polynomial multiplication. Because the value of the threshold T is normally linear in m, this solution would be impractical in most settings (i.e., when m is significant and/or the database size is large), but can be used as a proof of concept for a fully secure outsourcing of biometric computation to a single server.

If we employ the alternative mechanism also outlined in Section 3.2, each query would indeed consist of $2m + 1$ group elements and each template would be stored as T vectors of the same size, resulting in a much more realistic solution. Furthermore, by allowing more servers to participate in this computation, we can remove multiplicative dependence on T in the computation and make other performance improvements making the solution efficient, as described in the next section.

4 Multiple-Server Solution

4.1 Preliminaries

Notation. Given biometric representation X, we use X_i to denote its leftmost ith bit, $1 \leq i \leq m$. For a data item x, we use $[x]$ to denote x secret shared among n parties using a linear secret sharing scheme in a field $\mathbb{F}$ of size p. Throughout the discussion, we will assume that p is a prime (so that the field is $\mathbb{Z}_p$) larger than the maximum value that will be secret-shared among the participants and larger than the number of participants n. To achieve the best resilience against collusion, we will assume that a (t, n) linear secret sharing scheme is used for the highest possible value of t (i.e., each value is secret shared among n parties and $t + 1$ shares are required to reconstruct it, while t or fewer participants learn nothing). We use $[x]^j$ to denote the share held by party P_j. We also use ℓ to denote the bitlength of the values on which operations are performed (note that ℓ may or may not coincide with the bitlength of p).

Known techniques. We will assume that an information theoretically secure linear secret sharing scheme is used, where each secret is represented by a polynomial of degree t and $t < \frac{n}{2}$. Throughout this work, we will use notation $[x] \leftarrow \mathsf{RSS}(x)$ to denote creation of n random shares of x using the secret sharing scheme. The properties of such schemes allow the parties to compute any linear combination (including addition, subtraction, multiplication by a constant, etc.) of secret-shared values without any interaction. For instance, computing $[s] \leftarrow [a]+[b]$ involves locally adding shares of a and b (in $\mathbb{F}$); computing $[b] \leftarrow [a]+c$ amount to locally adding the (public) value c to the share of a, and $[b] \leftarrow c[a]$ involves locally multiplying the share of a by c. Additionally, efficient protocols for computing a product of two shared values exist for threshold linear secret sharing schemes; we denote multiplication as $[ab] \leftarrow \mathsf{Mult}([a], [b])$. Because the details of a multiplication protocol will be relevant in the subsequent description, we will assume that the multiplication protocol from [42] is used: To multiply $[a]$ and $[b]$, each participant first locally multiplies her shares of $[a]$ and $[b]$ and then distributes random shares of the result to the remaining participants. Upon receipt of messages from all parties, each participant computes her share of $[ab]$ as a linear combination of received values. The first multiplication step raises the degree of the polynomial representing ab to $2t$ and the subsequent steps re-randomize the resulting polynomial and reduce its degree back to t through polynomial interpolation. What will be important to us is that in the first step instead of a single multiplication, the parties can evaluate any multivariate polynomial of degree 2 locally and then apply the (interactive) re-sharing and interpolation operations only once.

Following prior literature, we evaluate performance of our protocols in terms of interactive operations – termed invocations – which dominate the overall cost. Such operations include mul-

Source	Rounds	Complexity	Security
[61]	15	$279\ell + 5$	perfect
[14]	$2\ell + 10$	$24\ell + 5$	perfect
[17]	4	$4\ell + 1$	statistical

Table 1: Known techniques for comparison of ℓ-bit values without the need for bit decomposition.

Source	Rounds	Complexity	Security
Tree-based	$\log k$	$k - 1$	perfect
[23]	5	$6k$	perfect
[17]	3	$5\log(k) + 2$	statistical

Table 2: Known techniques for computing Boolean OR of k shared bits.

tiplication, share distribution, and reconstruction of a value from its shares. In addition to the overall complexity measured in the total number of invocation, round complexity is an important indicator of efficiency of a protocol. We therefore also measure round complexity, which is the number of sequential interactive invocations.

In this work, we use some existing protocols as building blocks, which are:

LT : given two shared values $[x]$ and $[y]$, performs less-than and outputs shared bit $b = (x \stackrel{?}{<} y)$, where $b = 1$ if $x < y$ and 0 otherwise. The length l can take any value $\leq \ell$. The complexity of this protocol is proportional to the bitlength ℓ of x and y. Comparison protocols received a fair amount of attention in prior literature, but several techniques (e.g., [23, 40]) assume that the operands will be available in the bitwise form (i.e., each bit of x and y is shared by the participants). This means that expensive bit decomposition first needs to be applied to both x and y. The techniques that perform comparison without the need to bit-decompose the operands first provide a more efficient option in our case, and we list performance of known solutions in Table 1. Because the recent solution of [17] is the most efficient, this is what we adopt in our protocol. It, however, should be noted that the comparison protocols in [17] provide statistical privacy and rely on the size of the field $\mathbb{F}$ to be substantially larger than the size of the values. In particular, it needs to be κ bits longer than ℓ, where κ is a statistical security parameter. This primarily affect computation rather than communication complexity. We provide more detail about the comparison protocols from [17] in Appendix B.

OR : given a number of shared bits $[x_1], \ldots, [x_k]$, performs Boolean OR of all input bits and produces a shared bit b (i.e., b is 1 iff at least one of $x_1, \ldots, x_k$ is 1). A straightforward implementation of this function allows us to compute the OR of k bits in a binary tree fashion using $k - 1$ multiplications in $\lceil \log k \rceil$ rounds. Results from the literature also provide constant-round protocols, which are reported in Table 2. In [23], the OR is computed as a symmetric Boolean function using unbounded fan-in multiplication. And in [17] the computation cleverly uses the fact that $\bigvee_{i=1}^{k} a_i = 0$ iff $\sum_{i=1}^{k} a_i = 0$, which can be tested using equality of a value of length $\log k$. As before, the computation in [17] is over a larger field.

4.2 Overall protocol

Before presenting the overall protocol, we note that the computation to be performed can be restructured to result in a more efficient solution. The most computation-intensive operation in equation 2 is the division operation. While in general the use of this operation can be unavoidable, in our case the computation can be rewritten to avoid it. In particular, we change the computation in equation 2

to the following, using the notation $HD(X,Y) = (||(X \oplus Y) \cap M(X) \cap M(Y)||) \,/\, (||M(X) \cap M(Y)||) = D(X,Y)/M(X,Y)$:

$$\left(D(X, \mathsf{LS}^c(Y)) \stackrel{?}{<} T \cdot M(X, \mathsf{LS}^c(Y))\right) \wedge \cdots \wedge \left(D(X, \mathsf{RS}^c(Y)) \stackrel{?}{<} T \cdot M(X, \mathsf{RS}^c(Y))\right) \qquad (3)$$

The above restructuring allows us to replace $2c + 1$ division operations with the same number of comparisons which can be carried out more efficiently. In the rest of this section we thus describe our protocol corresponding to the computation in equation 3. Note that this type of computation restructuring cannot be used to enable the exact computation in the single-server solution, as predicate encryption allows us to only compare a computed value with a constant.

At high level, our MSSBS protocol follows the computation of equation 3. The database is stored among the servers in an secret-shared form and the client also distributes shares of her query to the servers. Because all computation must proceed on integer values, there is a need to discretize the data. In particular, the threshold T needs to be represented as an integer using a desired level of precision. That is, the floating point value of the threshold (where $0 < T < 1$) is multiplied by constant h which represents the level of precision. This also implies that once the Hamming distance is computed on binary data, the result is multiplied by h. That is, the computation will be performed as $(D(X,Y) \cdot h) \stackrel{?}{<} ((T \cdot h) \cdot M(X,Y))$ (and the conventional implementation that uses division would compute $\lfloor (D(X,Y) \cdot h)/M(X,Y) \rfloor \stackrel{?}{<} \lfloor T \cdot h \rfloor$). In what follows, we will assume that T already corresponds to its integer representation for the desired level of precision. The above also means that the length of the numbers on which we operate is $\ell = \lceil \log(m \cdot h) \rceil$.

To minimize the interactive computation associated with computing $D(X,Y)$ and $M(X,Y)$, both the database owner and the client communicate their data in a special form. In particular, instead of sending bits X_i and $M(X_i)$ to the servers, the database owner distributes $A_i = X_i \wedge M(X_i) = X_i M(X_i)$ and $B_i = \overline{X_i} \wedge M(X_i) = (1 - X_i)M(X_i)$, after which the servers also compute $C_i = A_i + B_i = M(X_i)$. The client then also distributes its data Y and $M(Y)$ as $U_i = Y_i M(Y_i)$ and $V_i = (1 - Y_i)M(Y_i)$, after which the servers compute $W_i = U_i + V_i = M(Y_i)$. This representation allows the servers to compute both $D(X,Y)$ and $M(X,Y)$ as two-degree polynomials over variables A_i, B_i, C_i, U_i, V_i, and W_i, which would require only one invocation per $D(X,Y)$ and $M(X,Y)$ instead of $O(m)$ invocations in other implementations. In more detail, $M(X,Y)$ is computed as $M(X,Y) = \sum_{i=1}^{m} C_i W_i = \sum_{i=1}^{m} M(X_i)M(Y_i)$ and $D(X,Y)$ is computed as $D(X,Y) = \sum_{i=1}^{m}(A_i V_i + B_i U_i) = \sum_{i=1}^{m}(X_i(1 - Y_i) + (1 - X_i)Y_i)M(X_i)M(Y_i) = \sum_{i=1}^{m}(X_i \oplus Y_i) \wedge M(X_i) \wedge M(Y_i)$. Both of these functions can be represented as inner product computation (on C and W, and $\langle A, B \rangle$ and $\langle V, U \rangle$, respectively), and we define sub-protocol Inner that proceeds as follows:

Protocol $[a] \leftarrow \mathsf{Inner}([x_1] \ldots [x_k], [y_1] \ldots [y_k])$:

1. Each party P_j locally computes $[a']^j = \sum_{i=1}^{k} [x_i]^j [y_i]^j$.
2. $P_1, \ldots, P_n$ engage in re-sharing and polynomial interpolation steps of the multiplication protocol using their $[a']^j$'s as input to produce a proper secret-shared inner product $[a]$.

This protocol requires only one invocation in one round, which allows us to significantly reduce the overhead of computing Hamming distances from linear in biometric size to a small constant. The overall protocol for multi-server biometric search is given next.

Protocol MSSBS

Setup:

1. For each biometric $\langle X, M(X)\rangle \in D$, the database owner distributes information about it to the parties $P_1, \ldots, P_n$ as follows:
   (a) for $i = 1, \ldots, m$, the database owner executes $[A_i] \leftarrow \mathsf{RSS}(X_i M(X_i))$ and $[B_i] \leftarrow \mathsf{RSS}((1 - X_i)M(X_i))$.
   (b) for $i = 1, \ldots, m$, the database owner communicates shares $[A_i]^j$ and $[B_i]^j$ to party P_j for $j = 1, \ldots, n$.
2. The database owner also executes $[T] \leftarrow \mathsf{RSS}(T)$ and sends c, h, and $[T]^j$ to party P_j for $j = 1, \ldots, n$.
3. Upon receipt of $[A_i]^j$ and $[B_i]^j$, each party P_j sets $[C_i]^j = [A_i]^j + [B_i]^j$ for each $i = 1, \ldots, m$ and each $\langle X, M(X)\rangle \in D$. This results in sharing the value of $C_i = M(X_i)$ among the parties.

As a result of this setup, let each shared biometric be denoted by $\langle A, B, C\rangle$ and the collection of shared biometrics be denoted by D'.

Query:

1. The client computes $[U_i] \leftarrow \mathsf{RSS}(Y_i M(Y_i))$ and $[V_i] \leftarrow \mathsf{RSS}[(1-Y_i)M(Y)_i]$ for $1 \le i \le m$, and communicates $[U_i]^j$, $[V_i]^j$ to P_j for $1 \le j \le n$ and $1 \le i \le m$.
2. Each P_j locally sets $[W_i] \leftarrow [U_i] + [V_i]$ using its shares for $i = 1, \ldots, m$.
3. For each $\langle A, B, C\rangle \in D'$, run in parallel:
   (a) For each $k = -c, \ldots c$, run in parallel:
      i. Each P_j locally performs a circular shift of its shares of U, V, and W according to the value of k and the step size of the shift. Denote the resulting values as U', V', and W'.
      ii. $P_1, \ldots, P_n$ execute $[a] \leftarrow \mathsf{Inner}(\langle A, B\rangle, \langle V', U'\rangle)$.
      iii. $P_1, \ldots, P_n$ execute $[b] \leftarrow \mathsf{Inner}(C, W')$.
      iv. $P_1, \ldots, P_n$ execute $[c] \leftarrow \mathsf{Mult}([b], [T])$.
      v. $P_1, \ldots, P_n$ execute $[d_k] \leftarrow \mathsf{LT}(h[a], [c])$.
   (b) $P_1, \ldots, P_n$ execute $[f] \leftarrow \mathsf{OR}([d_{-c}], \ldots, [d_c])$ and reconstruct the value of f.
4. For each $\langle A, B, C\rangle \in D'$ that returned 1, party P_j sends the index of the record in D' to the client.

In this protocol, the parties compute steps 3.a.ii and 3.a.iii in parallel and obtain shares of $D(X, Y')$ and $M(X, Y')$, respectively, where Y' is the rotated biometric Y. This takes one round. In the second round, the parties compute $T \cdot M(X, Y')$ (step 3(a).iv). Step 3.a.v involves comparing $D(X, Y')$ and $T \cdot M(X, Y')$. Finally in step 3.b the parties compute the OR of the outcome of the comparisons across all shifts of Y and recover the result. Figure 2 summarizes the interaction between the database owner, the client, and the servers in the protocol, where g denotes the function the servers compute in order to compare two biometrics.

This protocol assumes that the precision value h is a part of the algorithm and is public. If this is not the case, the protocol can be easily modified to work with a split value. Then during the setup, h will be distributed to the participants as $[h]p$, and during the query execution, local computation $h[a]$ will be replaced with interactive $\mathsf{Mult}([h], [a])$ in the second round of the protocol.

Setup:

- DB owner's input is database D of biometrics $\langle X, M(X)\rangle$ and parameters c, h and T.

1. DB owner: $A_X \leftarrow \langle \mathsf{RSS}(X_i M(X_i))\rangle_{i=1}^m$, $B_X \leftarrow \langle \mathsf{RSS}((1-X_i)M(X_i))\rangle_{i=1}^m$ for all $X \in D$.
2. DB owner: $[T] \leftarrow \mathsf{RSS}(T)$.
3. DB owner $\xrightarrow{\langle [A_X],[B_X]\rangle_{\forall X\in D},[T],c,h}$ Servers
4. Servers: $[C_X] \leftarrow [A_X] + [B_X]$ for all $X \in D$.

- Servers' output is $D' = \langle [A_X], [B_X], [C_X]\rangle_{\forall X\in D}$, $[T]$, c, and h.

Query:

- Client's input is $\langle Y, M(Y)\rangle$, servers' input is $[T]$, c, h, and $D' = \langle [A_X], [B_X], [C_X]\rangle_{\forall X\in D}$.

1. Client: $[U_Y] \leftarrow \langle \mathsf{RSS}(Y_i M(Y_i))\rangle_{i=1}^m$, $[V_Y] \leftarrow \langle \mathsf{RSS}((1-Y_i)M(Y_i))\rangle_{i=1}^m$.
2. Client $\xrightarrow{[U_Y],[V_Y]}$ Servers
3. Servers: $[W_Y] \leftarrow [U_Y] + [V_Y]$.
4. Servers: $f_X \leftarrow g([A_X], [B_X], [C_X], [U_Y], [V_Y], [W_Y], [T], c, h)\ \forall X \in D$.
5. Client $\xleftarrow{\{\mathsf{index}(X)\}_{\forall (X\in D \wedge f_X = 1)}}$ Servers

- Client's output: indices of matched biometrics in D.

Figure 2: Summary of MSSBS protocol interaction.

4.3 Analysis

Security. In our multi-party biometric search we utilize a number of sub-protocols which are known to be secure in the semi-honest model and can also be shown to be secure in the fully malicious model. A standard approach in showing that a protocol, which consists of secure building blocks, is secure is to employ the composition theorem of [16]. It states that the composition of secure protocols results in a secure overall protocol. Then security of the overall solution reduces to ensuring that sub-protocols are secure. In what follows, we first show that our MSSBS protocol is secure against semi-honest participants, and then extend its security to fully malicious participants through the use of additional tools.

Theorem 2 *Assuming security of the secret sharing scheme with $t < \frac{n}{2}$ and secure channels between the participants, the* MSSBS *protocol achieves security in presence of semi-honest servers.*

Proof First, we note that the linear secret sharing scheme achieves perfect secrecy in presence of collusions of size at most t (i.e., zero information can be learned about secret-shared values by t or fewer parties). Similarly, the multiplication and inner product protocols do not reveal any information, as the only information transmitted to the participants are the shares. This means that we can build perfect simulators for Mult and Inner, where the simulators can use arbitrary values at intermediate stages of the protocol (as long as they messages comply with expected output) and the view of a coalition of size at most t will be identically distributed to the real protocol execution. Similarly, due to security of LT and OR protocols from [17], we will invoke their corresponding simulators to build the simulator for the overall protocol.[7]

A simulator S for MSSBS (which will simulate the view of a coalition of up to t servers) is then given m, c, and h as the input and bit f for each record in D' as the output. S first constructs T, arbitrary tuples $\langle X_i M(X_i)\rangle_{i=1}^m$, $\langle (1-X_i)M(X_i)\rangle_{i=1}^m$ for each record in D and arbitrary tuples for client queries $\langle Y_i M(Y_i)\rangle_{i=1}^m$, $\langle (1-Y_i)M(Y_i)\rangle_{i=1}^m$ such that the result of comparing X and Y using c

[7] If the tree-based OR protocol is used, its computation consists of a combination of multiplications and linear combinations of shares without revealing any information beyond secret shared values, and therefore this protocol also achieves perfect secrecy and has a corresponding simulator.

and h would produce the desired output bit f. S distributes the shares of T and database records to the servers, followed by the shares of client biometric Y. As there is no interaction in the Setup phase, S does not need to generate any additional view information for the servers. To simulate the servers' view in the Query phase, S uses simulators for sub-protocols Inner, Mult, LT, and OR. As these simulators are secure (up to t colluding servers cannot learn additional information and the servers cannot distinguish simulation from real execution) and there are no additional messages that the servers receive, we arrive at security of the overall protocol. □

The above result shows that the computational servers cannot learn additional information about the database owner's and client's data. The result, however, can be easily extended to also show security against both the database owner and the client. That is, the database owner does not learn any information about clients' queries and the client does not learn any information about the database beyond what it can conclude from information about matches that it learns as its output. The security follows from the fact that neither the database owner nor the client receive any intermediate results, and therefore the database owner/client colluding with at most t servers gains no information.

To show security in presence of malicious adversaries, we need to ensure that (i) all participants prove that each step of their computation was performed correctly and that (ii) if some dishonest participants quit, others will be able to reconstruct their shares and proceed with the rest of the computation. The above is normally achieved using a verifiable secret sharing scheme (VSS), and a large number of results have been developed over the years (e.g., [42, 20, 45, 46, 6, 26, 24, 25] and many others). In particular, because any linear combination of shares is computed locally, each participant is required to prove that it performed each multiplication correctly on its shares. Such results normally work for $t < \frac{n}{3}$ in the information theoretic or computational setting with different communication overhead and under a variety of assumptions about the communication channels. Additional proofs associated with robust multi-party computation might include proofs that shares of a private value were distributed correctly among the participants (when the dealer is dishonest) and proofs of proper reconstruction of a value from its shares (when not already implied by other techniques). In addition, if at any point of the computation the participants are required to input values of a specific form, they would have to prove that the values they supplied are well formed.

Security of our MSSBS protocol against active adversaries can be shown as follows. First of all, we note that the efficiency of the protocols from [17] is in part due to its use of Pseudorandom Replicated Secret Sharing [21] (PRSS) which allows the parties to generate shares of random values without any interaction (and thus reduce communication), after which the shares are converted to shares in a regular linear secret sharing scheme. If the PRSS-based random value generation is replaced with a traditional interactive version,[8] standard VSS techniques can be applied to achieve security against active adversaries. In this case, the (interactive) complexity of the protocol would increase, as well as the number of rounds (we detail that below). In addition, because VSS techniques perform verification for each multiplication operation, the need to verify multiplications in Inner would result in that protocol executed as regular multiplication operations followed by a local sum. Finally, one building block in both LT and OR (namely, PRandInt, see Appendix B) requires the parties to input random values of a particular length. To ensure that the malicious participants comply with the length requirements, a range check proof should be used. Several techniques from the literature can be used for that purpose (in combination with VSS) with techniques from [64, 63] being among the most recent. To summarize, security in the malicious model can be achieved by

[8] In this case, each participant P_i chooses a random secret r_i and distributes its shares to other participants. The resulting random value r is computed as the sum of individual random values r_i, and its shares are obtained by adding all of the received shares.

using standard VSS techniques, e.g., [42, 22], in combination with a range proof, e.g., [64], where PRSS-based random generation is replaced with regular share generation. These VSS techniques would also work with a malicious database owner and/or client, who would need to prove that they generate legitimate shares of their data.

In addition to achieving security as described above, Cramer et al. [21] shows how security against active adversaries can be built using PRSS in two rounds of communication. This, however, requires representing the computation as a polynomial of degree 3 using a generic conversion procedure, which would compromise the efficiency of the protocol. This would also require $t < n/5$ [21]. In this case, the techniques of [27] should be used for the range proof (where [27] also utilizes PRSS).

Complexity. In our analysis we distinguish between overhead of the client and computational servers per execution of Query protocol. For the client, communication consists of sending the shares of the biometric Y to the servers and therefore is $2nm$ field elements. The communication received as a result of the computation is proportional to the number of matches (and therefore is small, if non-zero).

For each participating server, processing each record in the database requires 2 invocations and one round for steps 3.a.ii and 3.a.iii, one invocation and one round for step 3.a.iv, and additional 3 rounds and $4\ell+1$ invocations for step 3.a.v (one round of LT can be carried out in parallel with the previous steps as it is input-independent, see [17] for more detail). In addition, the constant-round OR protocol from [17] adds two more rounds and $5\log(c)+2$ invocations in step 3.b (like in the case with LT, one round can be carried out in parallel with the previous steps). We obtain the overall complexity of $(2c+1)(4\ell+4)+5\log(c)+2$ invocations in 7 rounds, which is very low and independent of the length m of biometric representations. The typical values for c and ℓ are $c=10$ and $\ell \approx 20$, resulting in about 1,800 invocations overall.

If the shifting can be skipped when computing the Hamming distance (which is a realistic assumption, see Section 6), this reduces to $4\ell+4$ invocations in 5 rounds, which would give fewer than 100 invocations overall for typical ℓ.

If the PRSS-based random value generation is not used (e.g., for the purposes of using standard VSS techniques in the malicious model), the complexities of LT and OR protocols, and thus the overall MSSBS protocol, change. In Appendix B we provide a detailed analysis of the new complexities, and obtain $10\ell+6$ invocations for LT in 7 rounds and $10\log(c)+5$ invocations for OR in 7 rounds. This would change the overall complexity of MSSBS to $(2c+1)(10\ell+9)+10\log(c)+5$ invocations in 11 rounds, which is still low.

Performance. To further investigate the performance of our solution in practice, we implement the multiplication protocol, the most basic building block of the solution, and evaluate the performance of the MSSBS protocol. The execution time of the protocol will heavily depend (i) on the computational power of the participating servers and, perhaps more importantly, (ii) on the speed of the networks connecting them. Recall that a multiplication protocol requires a participant to send one message to each other participating party and wait for a reply from all of them. Because communication between each pair of participants requires a secure channel, multiplication can be viewed as the most expensive type of a single invocation with $n-1$ encryptions and $n-1$ decryptions per participant (reconstructing a value from its shares, for instance, requires no encryption, but is also counted as one invocation).

In our implementation we used a LAN with $n=5$. While this configuration has low latency and permits a large number of rounds within a given time frame than slower networks, participants on a WAN will be able to utilize a higher degree of parallelism per round of communication. Because our protocol involves a significantly larger number of invocations than the number of rounds,

and additional parallelism can be gained by simultaneous processing of multiple X from D, both configurations are suitable for the protocol. In general, the computational servers are assumed to have sufficiently high-speed connections available, but reside on different networks.

In our setup, commodity 1GHz Sun workstations with dual core AMD Opteron processors 180 and 1214 running Red Hat Linux were used for running the experiments on a 100Mbps switched LAN. The timings were gathered and averaged over five hundred executions. We obtain that the communication overhead per multiplication was near 3ms, which means that on a LAN all rounds of the protocol can be finished in about 20ms. Due to technical reasons with out setup, we were unable to obtain reliable measurements for the delay with a varying number of participants n. We, however, note that all necessary information is already available in prior literature, and our result for $n = 5$ is in agreement with it. In particular, Geisler [41, Section 4.5] reports on the results of performing multiplication with n between 4 and 31 using slightly faster machines and network. The times range between 0.7ms with $n = 4$ and $t = 1$ and 9.5ms for $n = 31$ and $t = 10$. While communication in the multiplication protocol is linear in n for each party, multiplication time exhibits faster than linear growth with quadratic function $f(x) = 0.799 - 0.006x + 0.0009x^2$ resulting in the best fit. This is due to the fact that during multiplication re-sharing the computed value requires $n - 1$ evaluations of a polynomial of degree t, which results in quadratic number of operations overall (assuming that $t = O(n)$). We note that [41] also provides performance results for multiplication in presence of active adversaries, and the online portion of the multiplication techniques implemented in [41] exhibits only linear in n growth. This is due to the fact that each multiplication (after pre-processing step) involves two linear-time reconstructions plus constant work.

When the servers reside on different networks, the communication delay increases and the round trip time between well-connected machines would be in the range of 50–80 ms across a large country (e.g., between East and West Coasts in the US). In that case, the round complexity of the solution will result in communication delay under 0.5 second (with the ability to process a very large number of records in the database in parallel).

As far as the ability to parallelize multiple executions of the multiplication protocol goes, another (optimized) secure multi-party implementation, Sharemind [1], achieves more than a million multiplications per second with $n = 3$ on a fast network. While the underlying protocol used for multiplication in Sharemind is different from what we utilize, for small values of n the computational overhead will be dominated by encryption/decryption operations and therefore would be comparable. This in particular means that with our MSSBS protocol on the order of 1000 biometric records can be securely compared in a second with that setup, but the degree of parallelism would decrease for a larger n or slower connections. Furthermore, AES hardware implementations such as that of Tillich and Großschädl [69] allow for about 5-fold speedup as compared to conventional software implementations. Since in our application the helper servers would mostly execute multiplication protocols, it would be logical to assume that they will be able to use optimized hardware implementations of AES with possibly extended instruction set for improved performance. Thus, the estimated value above can be further reduced and can used as the upper bound on the computational costs.

5 Using Approximations to Reduce Cost

As efficiency is one of our goals, it seems natural to apply approximation algorithms to improve the computation or communication used per comparison with a database record. In particular, the protocols we described in the previous sections examine all m bits of each biometric and compute

all operations whenever possible, while an approximation might involve only a fraction of that work at the expense of some error in the computation. Here our goal is not to use an approximation algorithm to replace all of the computation, but rather have a faster way to filter out most of records that definitely constitute a mismatch. In other words, the computation can be performed in two stages:

1. Run an approximation algorithm to filter out the majority of records;
2. Run the exact algorithm to filter out mismatches within the error of the approximation algorithm.

This two-stage matching process might allow us to tolerate a larger error with more coarse and faster approximations than using a solely good approximation algorithm. If the approximation is fast enough on the majority of the records, the average time spent per record has the potential to significantly go down. Recall that we are interested in the FRR of 0 and in as low a FAR as possible (i.e., return all authentic records that matched and as few mismatches as possible).

In recent years, several results appeared that permit private evaluation of approximations with provable bounds of the error [37, 38, 49]. There are two main reasons, however, why prior solutions cannot be directly applied here. First, they assume that, in two-party computation, one party holds one database and the other party holds another database, and the computation produces an approximation of the distance between the databases. In our case, no party has access to either string being compared, which rules out some of the approaches used (in particular, the sublinear hashing scheme of [37, 38]). Second, such approximation (more precisely, sampling) algorithms result in savings only when the size of the string is very large. In our case, the size of the data string m is rather small (12800 bits in our database, commercial software produces iris codes of 2048 bits), and even setting the error bounds to generous values results in the number of sampled bits to be significantly larger than m. Thus, we chose to depart from the theoretical bounds and, as the initial step, empirically evaluate the sampling technique on smaller values. We report the results of experiments in Section 6.2.3 and leave more involved approximation algorithms as a direction for future work.

6 Experimental Validation

In this section we report on the experiments we conducted to estimate the accuracy of the techniques. As is evident from the prior description, some of our techniques introduce errors in the matching process, and we are interested in estimating and minimizing the errors they introduce. All data reported in this section should be treated as an initial study that gains insight into the impact of approximations on the performance of biometric matching and corresponding error rates. It would be desirable to know if the magnitude of robustness to approximations using averaged images as described in this section would be maintained on data sets gathered under different conditions, of different quality, etc.

For the purposes of this work, the experiments were run on biometric data from 170 subjects, each having about 50–100 biometric samples. Authentic data were gathered using multiple biometric templates for each individual, and impostor data were gathered by pairwise comparisons between different individuals. Thus, the volume of impostor comparisons significantly exceeds the data collected for authentic comparisons.

	Authentic				Impostor			
	Min	Max	Avg	St dev	Min	Max	Avg	St dev
Original	0.002	0.553	0.289	0.123	0.083	0.744	0.374	0.080
MC-5	0.121	0.492	0.311	0.092	0.233	0.5670	0.422	0.044
MC-9	0.119	0.479	0.281	0.075	0.241	0.5720	0.421	0.043
MC-13	0.126	0.386	0.243	0.062	0.263	0.5580	0.422	0.044
MC-17	0.108	0.348	0.224	0.052	0.257	0.5700	0.422	0.043
MC-21	0.108	0.326	0.215	0.051	0.250	0.5610	0.422	0.044
MC-25	0.116	0.319	0.204	0.049	0.242	0.5630	0.421	0.044
MC-29	0.104	0.297	0.188	0.047	0.266	0.5590	0.423	0.044
MC-33	0.103	0.299	0.172	0.046	0.228	0.5500	0.422	0.044

Table 3: Performance of Any-Support-Alg majority algorithm on authentic and impostor data.

6.1 Widening the gap between authentic and impostor distributions

As a first step in the experimentation, we attempt to increase the gap between authentic and impostor distributions by using repeated sampling and majority encoding for each bit in the biometric representation, as suggested in [29] (i.e., acquire multiple biometric readings and set bit X_i in biometric representation using the majority from the readings). The goal is to have a gap between the authentic and impostor distributions so that the errors introduced by approximations will fall within the gap rather than cause larger overlap of such distributions. This technique was theoretically evaluated (in [29]) on a different number of samples used for majority encoding (assuming an error rate of 10% for authentic data). Such analysis implies that the noise is uniformly distributed over the code, which is not true for biometric data, i.e., some bits are more likely to stay consistent over multiple acquisitions than others. To the best of our knowledge, this is the first work that empirically evaluates effectiveness of this technique (as far as we know, this rather simple technique is not used (or even known) in the biometric community).

While the idea of this technique is very simple, applying it to iris codes is less straightforward than it might seem due to usage of masks and template rotation in the matching process. In particular, given a number of templates, only a subset of them can have any particular bit marked as reliable (this also implies that, given an odd number of templates, it is no longer guaranteed that the majority for each bit will be computed using an odd number of samples). Furthermore, it might be desirable to pre-align the biometric representations used in computing the majority template prior to performing majority encoding.

For the purposes of this work, we designed two algorithms for computing majority codes, which we describe next. Suppose we are given k iris codes (for an odd k) and would like to compute a majority code using them. For each jth bit of the biometric representations ($1 \leq j \leq m$), let b_{ij} denote the jth biometric bit from the ith code and b_j the computed majority bit. Similarly, we use m_{ij} to denote the jth mask bit of the ith code and m_j the corresponding mask bit in the majority code. The first algorithm, to which we refer as Any-Support-Alg, sets the mask bit m_j to 1 if a least one bit among the m_{ij}'s ($1 \leq i \leq k$) is equal to 1. The bit b_j is then set to the majority of the b_{ij}'s that had the corresponding mask bit set to 1 (when their number is even and there is no majority, it is set to 0). The second algorithm, to which we refer as Threshold-Support-Alg, sets the mask bit m_j to 1 only if the number of templates that have this mask bit set is above a certain threshold (which we set to $k/4$). Furthermore, if the number of templates with the set mask bit exceeds the threshold, but there is no majority among such b_{ij}'s, m_j is also set to 0. We provide pseudo-code for these algorithms in Appendix A.

Tables 3 and 4 show the results of the distance computation (i.e., as in equation 1) on the

	Authentic				Impostor			
	Min	Max	Avg	St dev	Min	Max	Avg	St dev
Original	0.002	0.605	0.295	0.135	0.015	0.654	0.375	0.082
MC-5	0.019	0.521	0.243	0.116	0.148	0.609	0.398	0.063
MC-9	0.068	0.478	0.242	0.090	0.186	0.585	0.407	0.055
MC-13	0.077	0.416	0.217	0.078	0.202	0.579	0.408	0.053
MC-17	0.085	0.470	0.194	0.070	0.191	0.571	0.411	0.051
MC-21	0.076	0.354	0.188	0.061	0.220	0.576	0.414	0.050
MC-25	0.062	0.301	0.182	0.054	0.213	0.572	0.414	0.050
MC-29	0.046	0.303	0.171	0.054	0.234	0.566	0.416	0.049
MC-33	0.067	0.321	0.166	0.054	0.211	0.558	0.416	0.048

Table 4: Performance of Threshold-Support-Alg majority algorithm on authentic and impostor data.

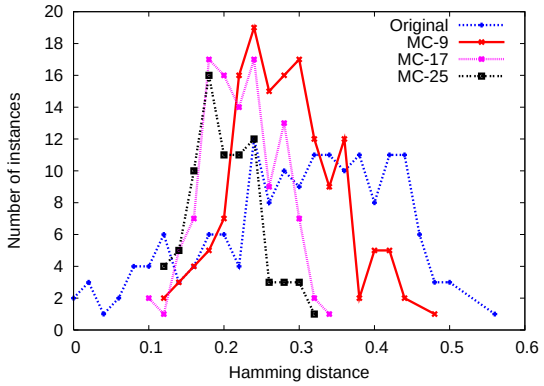

Figure 3: Authentic comparisons with Any-Support-Alg.

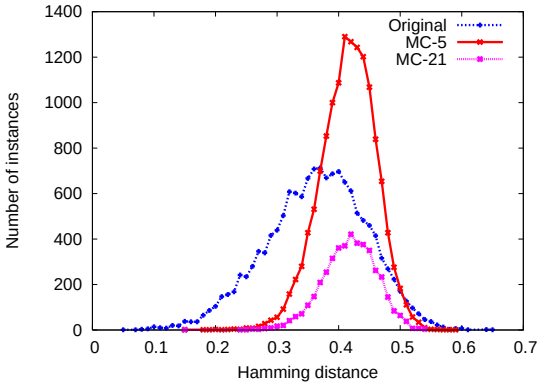

Figure 4: Impostor comparisons with Any-Support-Alg.

data produced using these algorithms, as well as the original data.[9] In the tables, MC-k stands for majority code computed from k iris codes. We also illustrate performance of Any-Support-Alg in figures 3 and 4. As is evident from the tables and figures, applying either majority algorithm enlarges the distance between the authentic and impostor distributions and also decreases the variance of both distributions. Both of these outcomes are expected. What is also very important for our application is that the maximum error for authentic comparisons drops significantly and takes values significantly lower than most values for impostor error.

To gain better understanding how these findings align with theoretical analysis (which assumes uniformly distributed noise), we provide theoretical expectations here for the value of 0.29 (the original average error rate for authentic data in our data sample). The theoretically expected error rate for the majority code would then be:

MC-5	MC-9	MC-13	MC-17	MC-21	MC-25	MC-29	MC-33
0.150	0.087	0.053	0.033	0.021	0.013	0.008	0.005

which is clearly very different from the actual data we observe.

Threshold-Support-Alg provides slightly better separation between the means of authentic and impostor data than Any-Support-Alg, but has a higher variance. Also, Any-Support-Alg stabilized

[9]The exact numbers for the original data can slightly differ between the tables as they were produced using two randomly chosen biometrics for each subject to be comparable with other experiments that used two majority codes per subject.

	Authentic				Impostor			
	Min	Max	Avg	St dev	Min	Max	Avg	St dev
Original	0.148	0.740	0.484	0.121	0.125	0.838	0.469	0.167
MC-5	0.911	1.272	1.153	0.078	0.816	1.272	1.090	0.108
MC-9	1.131	1.280	1.246	0.029	1.139	1.280	1.218	0.037
MC-13	1.238	1.280	1.267	0.011	1.212	1.280	1.251	0.018
MC-17	1.258	1.280	1.271	0.006	1.240	1.280	1.263	0.011
MC-21	1.267	1.280	1.275	0.004	1.262	1.280	1.271	0.005
MC-25	1.268	1.280	1.276	0.004	1.261	1.280	1.271	0.006
MC-29	1.273	1.280	1.277	0.002	1.268	1.280	1.275	0.004
MC-33	1.273	1.280	1.277	0.002	1.271	1.280	1.276	0.003

Table 5: Mask overlap size of comparisons of iris codes (in 10^4 bits) with Any-Support-Alg.

the impostor data very quickly (with virtually no changes as the number of templates used during the computation of the majority codes increases), but did not appear to have significant effect on the authentic data when a small number of templates were used to form majority codes. One advantage of Any-Support-Alg is that it produces biometric codes with a much higher number of mask bits set than in the original biometric representations. This allows us to approximate division with high precision as we show in the next section.

6.2 Validating accuracy of approximation algorithms

We now proceed with the empirical evaluation of different types of approximations described in this work on the iris codes computed using the majority algorithms. Recall that the single-server solution does not implement the division operation, and this is what we evaluate next. We also investigate the effect of skipping the template rotation, as well as report the results of applying sampling techniques on the majority iris codes in order to reduce computational cost of biometric search solutions.

6.2.1 Division

As was mentioned above, the Any-Support-Alg majority algorithm produces codes with very high percentage of mask bits set (i.e., much higher than 75% in the original codes). The small variance in the mask size thus makes it ideal for approximating the division operation. That is, instead of dividing the distance between two codes by the mask overlap size, we multiply the threshold T, with which the hamming distance is compared, by a constant that corresponds to the average mask overlap size. Since the distribution of mask overlap sizes for authentic comparisons differ from the distribution of impostor comparisons, we consider averaging the means and using that value in approximating the division.

Table 5 shows the distribution of mask overlap sizes produced on authentic and impostor comparisons that used majority codes (generated by Any-Support-Alg majority algorithm), from the total of $m = 12800$ bits. As can be seen from the table, the variance reduces dramatically as the number of templates k used to create majority codes increases. As the benefits of such majority algorithms are most pronounced on larger values of k (and can bear very small advantage otherwise), if such algorithms are deployed, we expect that rather high values of k to be used (e.g., 20 and above). In such cases, the spread of mask overlap values is minimal: e.g., when $k = 21$ the mask overlap sizes range from 12620 to 12800 across all of impostor and authentic comparisons. This means that the maximum error that division approximation can produce is bounded

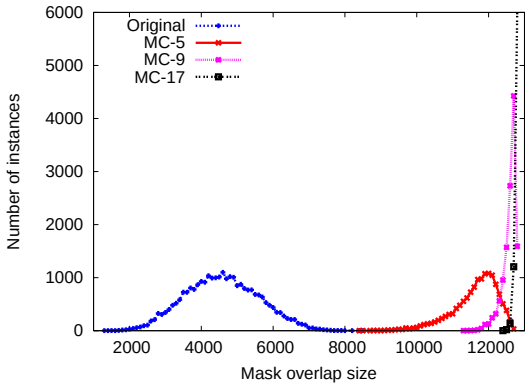

Figure 5: Distribution of mask overlap size for impostor comparisons with Any-Support-Alg.

	Authentic				Impostor			
	Min	Max	Avg	St dev	Min	Max	Avg	St dev
Original	0.027	0.691	0.360	0.161	0.103	0.867	0.477	0.111
MC-5	0.121	0.584	0.332	0.099	0.253	0.720	0.473	0.060
MC-9	0.119	0.484	0.302	0.077	0.256	0.680	0.466	0.058
MC-13	0.136	0.436	0.269	0.069	0.254	0.689	0.466	0.059
MC-17	0.108	0.453	0.238	0.066	0.267	0.696	0.467	0.060
MC-21	0.108	0.373	0.222	0.058	0.251	0.686	0.467	0.060
MC-25	0.119	0.311	0.210	0.050	0.256	0.713	0.467	0.060
MC-29	0.104	0.318	0.198	0.050	0.265	0.657	0.468	0.060
MC-33	0.103	0.330	0.192	0.054	0.253	0.676	0.468	0.060

Table 6: Performance of Any-Support-Alg on authentic and impostor data when no shifting is used during computation of Hamming distance.

by 180/12800 or 1.4%. Furthermore, for most comparisons the error will be even lower due to the uneven distribution of the mask overlap sizes, which are higher concentrated near the upper bound. Figure 5 illustrates mask overlap size distribution for select cases. We chose to plot mask overlap sizes corresponding to impostor data because they have larger spread than values of authentic data.

6.2.2 No shifting

The next type of experiments we performed is to estimate the error introduced by skipping the shifting and minimum operations. As the iris codes in our database are rather well-aligned (i.e., finding the minimum value among all shifts for authentic data often results in choosing the original code with no shifting), the expected error is small. Table 6 shows the results of such an approximation.

If we compare these results with the data reported in Table 3, it is clear that the difference is small. The average distance for impostor data increased by over 4% in all cases (and significantly higher when no majority coding is used), while it increased by about 2% for authentic data (except when no majority coding is used). This tells us that (i) the effect of shifting is higher on the original data than when majority coding is used and (ii) with majority coding, the effect of shifting is higher on impostor data than authentic data (i.e., shifting is more likely to accidently lower the Hamming distance of impostor data than bring authentic codes closer by aligning them). The variance, however, is larger in all cases in Table 3 than in Table 6, but not significantly larger.

Finally, to obtain the full picture of the performance of the optimized SSSBS protocol, we combined approximations for division with skipping the shifting. Table 7 shows statistics for

	Authentic				Impostor			
	Min	Max	Avg	St dev	Min	Max	Avg	St dev
Original	0.021	0.992	0.373	0.193	0.028	1.068	0.459	0.152
MC-5	0.108	0.620	0.343	0.106	0.192	0.805	0.490	0.068
MC-9	0.124	0.495	0.306	0.081	0.278	0.699	0.477	0.060
MC-13	0.138	0.444	0.272	0.069	0.244	0.695	0.472	0.059
MC-17	0.109	0.453	0.240	0.067	0.275	0.681	0.470	0.059
MC-21	0.108	0.375	0.223	0.058	0.256	0.687	0.469	0.060
MC-25	0.120	0.312	0.215	0.051	0.273	0.687	0.469	0.060
MC-29	0.105	0.319	0.200	0.051	0.267	0.690	0.468	0.059
MC-33	0.104	0.331	0.192	0.054	0.262	0.678	0.468	0.060

Table 7: Performance of Any-Support-Alg on authentic and impostor data when no division and no shifting is used during computation of Hamming distance.

authentic and impostor comparisons when mask overlap size was approximated by a constant during division and no shifting was used during the computation of Hamming distance. As can be seen from the table, we obtain that even by applying both of these approximation to the majority codes, the results still significantly outperform the performance of original data (with no majority coding) using the precise computation. When these approximations are used on the original data, however, the performance is very poor (due to the large effect of approximating division and removal of minimum computation on such data). Thus, we conclude that such approximations are feasible not only for the single-server case, but can also be used to significantly reduce the computation cost of the multi-server solution.

6.2.3 Sampling

The purpose of this last section is to gain insights on how reducing the number of bits used in the computation of the Hamming distance reduces the accuracy of the matching. These experiments were run on different values for the sample size $s < m$. Given s, during each comparison, a random subset of size s was chosen from the m bits, while the rest of the computation proceeded unchanged. As an interesting implementation issue, the care must be taken when such a subset is chosen not to interfere with the circular shifts. Our biometric representations are stored as 20×640 bit arrays, and we draw $s/20$ columns at random during each comparison. Doing so does not disrupt the row-wise shifting. Also, since an excessive amount of shifting reduces impostor distance more than authentic, in our experiments the amount of shifting c was scaled down with the size of the sample set (e.g., $c = 2$ for $s = m/10$ and $c = 1$ for $s = m/20$).

Table 8 presents the results of applying the sampling technique to iris data with the sample size of 2% of the original bitlength m, $s = 260$. As the data suggest, while the original biometric templates were significantly affected by this approximations, the templates computed using majority coding still exhibited a very stable behavior. To evaluate the overhead of the two-stage search process described in Section 5, we compute the overlap of the authentic and impostor distributions as the percent of impostor comparisons that fall below the maximum value of authentic Hamming distances. The table below shows such numbers for different values of the sample set s using majority codes with good performance.

Sample size	MC-21	MC-25	MC-29	MC-33
$s = 12800$	2.01%	1.36%	0.28%	0.36%
$s = 640$	7.00%	1.50%	1.20%	0.18%
$s = 260$	6.50%	1.26%	0.50%	0.48%

	Authentic				Impostor			
	Min	Max	Avg	St dev	Min	Max	Avg	St dev
Original	0.000	0.720	0.379	0.183	0.068	0.955	0.474	0.123
MC-5	0.101	0.591	0.339	0.108	0.150	0.777	0.471	0.069
MC-9	0.105	0.541	0.291	0.080	0.146	0.747	0.465	0.065
MC-13	0.119	0.446	0.262	0.074	0.150	0.696	0.464	0.065
MC-17	0.085	0.350	0.228	0.061	0.159	0.681	0.463	0.066
MC-21	0.092	0.365	0.215	0.066	0.123	0.700	0.465	0.066
MC-25	0.115	0.320	0.208	0.056	0.242	0.696	0.467	0.067
MC-29	0.104	0.296	0.192	0.054	0.262	0.695	0.469	0.066
MC-33	0.073	0.288	0.178	0.054	0.250	0.677	0.468	0.066

Table 8: Performance of Any-Support-Alg with the sample size $s = 260$.

Protocol	Optimization type					
	No division		No shifting		Sampling	
	accuracy degradation	performance improvement	accuracy degradation	performance improvement	accuracy degradation	performance improvement
SSSBS	required		small	significant	small	significant
MSSBS	small	minimal	small	significant	small	minimal

Table 9: The effect of approximations techniques on the accuracy of comparing averaged biometrics and performance of secure matching protocols.

Given the above, it is clear that a small sample size can be used to filter out a great majority of false matches, after which a rigorous comparison can be conducted on the remaining records. This would result in significant computational savings of the SSSBS protocol and modest computational savings of the MSSBS protocol.

A summary of approximation techniques and their impact on both accuracy of matching and performance of single-server and multi-server protocols is given in Table 9.

7 Comparison with Prior Work

A lot of relevant literature is cited throughout this work, and in this section we compare our result with other related publications. We would like to note that prior work does not allow one to perform all of the computation realized in this work. In particular, outsourcing the computation to a single server is a very challenging task, and even the use of a very recent and powerful notion of predicate encryption utilized in this work provides only a limited solution to the problem. Other techniques from secure multi-party computation such as general Boolean circuit evaluation and customized privacy-preserving protocols are interactive and require knowledge of inputs. Literature on secure computation outsourcing (see, e.g., [60, 54, 15, 7, 56, 48, 50, 36, 47, 30]) is mostly concerned with outsourcing modular exponentiations, which are normally the most computation-expensive part of a cryptographic scheme. The goal of this work, on the other hand, is to use (untrusted) servers for the entire computation, which is not based on modular exponentiations. Other results related to secure computational outsourcing [34, 4, 3] do not treat biometric computations or operations used in such computations.

Our multiple-server setting is closer to the traditional secure multi-party computation setting, with the exception that no inputs are known to any single participant (which might render some tools inapplicable). While general Boolean circuits can theoretically be built to evaluate any func-

tion in a secure manner, their performance is particularly poor when dealing with large volumes of data and would incur several orders of magnitude larger overheads on the client and database owner than in our protocol.[10] Additionally, other custom privacy-preserving tools cannot be readily adapted to perform the required computation. In particular, solutions for computing the hamming distance [34, 12] do not extend to the computation used in iris matching, set intersection protocols [39, 55] do not achieve the functionality we seek and are not composable (i.e., cannot be used as a sub-protocol). We build our solution on general tools with attractive characteristics, i.e., secret sharing is more suitable in this context than secure computation techniques based on homomorphic encryption due to the small size of the numbers we manipulate and the computational overhead associated with public-key cryptography. Finally, literature on searches on encrypted data (see, e.g., [9, 68] among many others) treats exact, wildcard, or range matching and cannot support the computation used in iris comparisons (predicate encryption used here is conceptually close to such schemes).

In parallel with this work, secure protocols for comparing other biometric types such as faces and certain representations of fingerprints have been developed [35, 66, 62, 5]. Besides realizing different types of computation, these publications consider a two-party setting that differs from the setting of this work: in a two-party setup, each party has access to its biometric(s) in the clear, and the parties run a secure two-party protocol to compare their respective data. Notice that in certain cases, access to the data in the clear form can aid efficiency and/or functionality.

8 Conclusions

This work motivates the problem of secure computation (or outsourcing) of biometric matching when neither the database nor the biometric value being searched for is available in the clear to the server. We introduce first solutions to the problem for iris codes, where we distinguish between a single-server solution, SSSBS, and a multiple-server solution, MSSBS. Our results show that (i) a close approximation of the necessary computation is possible in the non-interactive single-server case; and (ii) the exact secure computation can be carried out more efficiently in the setting of multiple servers. We also introduce majority coding algorithms for iris codes and implement them using a sample iris database, which noticeably widens the gap between distributions of authentic and impostor data. Finally, we study various types of approximation techniques and achieve favorable results.

As this is the first work in this direction, many interesting problems remain. For example, other types of approximation algorithms beyond simple sampling are worth exploring. This work concentrated on iris codes, but many other types of biometric data remain. Finally, larger-scale experimentation with biometric data is needed to confirm the findings of this work and further explore effect of applying different majority algorithms to biometric data.

9 Acknowledgments

This work benefited from useful discussion with Keith Frikken at early stages of this work. We also would like to acknowledge the help of Patrick Flynn, Tanya Peters, and Karen Hollingsworth with conducting our experiments on biometric data, and anonymous reviewers for their valuable

[10] In addition, any computation that uses division might be infeasible to carry out by such means. For example, [8] reports that building a Boolean circuit for division using existing tools such as Fairplay [59] already requires resources beyond an ordinary workstation in the two-party setting, while this work is concerned with more involved multi-party setup.

feedback. This work was partially supported by the grant FA9550-09-1-0223 from the Air Force Office of Scientific Research.

References

[1] Sharemind. http://sharemind.cyber.ee/.

[2] M. Abadi, J. Feigenbaum, and J. Killian. On hiding information from an oracle. *Journal of Computer and System Sciences*, 39:21–50, 1989.

[3] M. Atallah and J. Li. Secure outsourcing of sequence comparisons. *International Journal of Information Security*, 4(4):277–287, 2005.

[4] M. Atallah, K. Pantazopoulos, J. Rice, and E. Spafford. Secure outsourcing of scientific computations. *Advances in Computers*, 54(6):215–272, 2001.

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

[6] Z. Beerliová-Trubíniová and M. Hirt. Perfectly-secure MPC with linear communication complexity. In *Theory of Cryptography Conference (TCC)*, pages 213–230, 2008.

[7] P. Beguin and J. J. Quisquater. Fast server-aided RSA signatures secure against active attacks. In *Advances in Cryptology – CRYPT0 95*, pages 57–69, 1995.

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

[9] D. Boneh and B. Waters. Conjunctive, subset, and range queries on encrypted data. In *Theory of Cryptography Conference*, volume 4392 of *LNCS*, pages 535–554, 2007.

[10] X. Boyen. Reusable cryptographic fuzzy extractors. In *ACM Conference on Computer and Communications Security (CCS'04)*, pages 82–91, 2004.

[11] X. Boyen, Y. Dodis, J. Katz, R. Ostrovsky, and A. Smith. Secure remote authentication using biometric data. In *Advances in Cryptology – EUROCRYPT'05*, volume 3493 of *LNCS*, pages 147–163, 2005.

[12] J. Bringer, H. Chabanne, D. Pointcheval, and Q. Tang. Extended private information retrieval and its application in biometrics authentications. In *Cryptology and Network Security*, volume 4856 of *LNCS*, pages 175–193, 2007.

[13] H. Bui, M. Kelly, C. Lyon, M. Pasquier, D. Thomas, P. Flynn, and D. Thain. Experience with BXGrid: A data repository and computing grid for biometrics research. *Journal of Cluster Computing*, 2009.

[14] M. Burkhart, M. Strasser, D. Many, and X. Dimitropoulos. SEPIA: Privacy-preserving aggregation of multi-domain network events and statistics. In *USENIX Security Symposium*, pages 223–240, 2010.

[15] J. Burns and C. Mitchell. Parameter selection for server-aded RSA computation schemes. *IEEE Transactions on Computing*, 43(2):163–174, 1994.

[16] R. Canetti. Security and composition of multiparty cryptographic protocols. *Journal of Cryptology*, 13(1):143–202, 2000.

[17] O. Catrina and S. de Hoogh. Improved primitives for secure multi-party integer computation. In *Security in Communication Networks (SCN)*, pages 182–199, 2010.

[18] E.-C. Chang and Q. Li. Small secure sketch for point-set difference. ePrint Archive Report 2005/145, 2005.

[19] E.-C. Chang and Q. Li. Hiding secret points amidst chaff. In *Advances in Cryptology – EUROCRYPT'06*, volume 4004 of *LNCS*, pages 59–72, 2006.

[20] R. Cramer, I. Damgård, S. Dziembowski, M. Hirt, and T. Rabin. Efficient multiparty computations secure against an adaptive adversary. In *Advances in Cryptology – EUROCRYPT*, pages 311–326, 1999.

[21] R. Cramer, I. Damgård, and Y. Ishai. Share conversion, pseudorandom secret-sharing and applications to secure computation. In *Theory of Cryptography Conference (TCC'06)*, pages 342–362, 2005.

[22] R. Cramer, I. Damgård, and U. Maurer. General secure multi-party computation from any linear secret-sharing scheme. In *Advances in Cryptology – EUROCRYPT*, pages 316–334, 2000.

[23] I. Damgård, M. Fitzi, E. Kiltz, J. Nielsen, and T. Toft. Unconditionally secure constant-rounds multi-party computation for equality, comparison, bits and exponentiation. In *Theory of Cryptography Conference (TCC'06)*, volume 3876 of *LNCS*, pages 285–304, 2006.

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

[25] I. Damgård, Y. Ishai, and M. Krøigaard. Perfectly secure multiparty computation and the computational overhead of cryptography. In *Advances in Cryptology – EUROCRYPT*, pages 445–465, 2010.

[26] I. Damgård, Y. Ishai, M. Krøigaard, J. Nielsen, and A. Smith. Scalable multiparty computation with nearly optimal work and resilience. In *Advances in Cryptology – CRYPTO*, pages 241–261, 2008.

[27] I. Damgård and R. Thorbek. Non-interactive proofs for integer multiplication. In *Advances in Cryptology – EUROCRYPT*, pages 412–429, 2007.

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

[29] G. Davida, Y. Frankel, and B. Matt. On enabling secure applications through off-line biometric identification. In *IEEE Symposium on Security and Privacy*, pages 148–157, 1998.

[30] M. Dijk, D. Clarke, B. Gassend, G. Suh, and S. Devadas. Speeding up exponentiation using an untrusted computational resource. *Des Codes Cryptography*, 39(2):253–273, 2006.

[31] Y. Dodis, R. Ostrovsky, L. Reyzin, and A. Smith. Fuzzy extractors: How to generate strong keys from biometrics and other noisy data. *SIAM Journal of Computing*, 38(1):97–139, 2008.

[32] Y. Dodis, L. Reyzin, and A. Smith. Fuzzy extractors: How to generate strong keys from biometrics and other noisy data. In *Advances in Cryptology – EUROCRYPT'04*, volume 3027 of *LNCS*, pages 523–540, 2004.

[33] Y. Dodis and A. Smith. Correcting errors without leaking partial information. In *ACM Symposium on Theory of Computing (STOC'05)*, pages 654–663, 2005.

[34] W. Du and M. Atallah. *Protocols for secure remote database access with approximate matching*, chapter 6. Kluwer Academic Publishers, 2001.

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

[36] A.-M. Ernvall and K. Nyberg. On server-aided computation for RSA protocols with private key splitting. In *Nordsec*, 2003.

[37] J. Feigenbaum, Y. Ishai, T. Malkin, K. Nissim, M. Strauss, and R. Wright. Secure multiparty computation of approximations. In *International Colloquium on Automata, Languages and Programming (ICALP'01)*, pages 927–938, 2001.

[38] J. Feigenbaum, Y. Ishai, T. Malkin, K. Nissim, M. Strauss, and R. Wright. Secure multiparty computation of approximations. *ACM Transactions on Algorithms*, 2(3):435–472, 2006.

[39] M. Freedman, K. Nissim, and B. Pinkas. Efficient private matching and set intersection. In *Advances in Cryptology – EUROCRYPT'04*, volume 3027 of *LNCS*, pages 1–19, 2004.

[40] J. Garay, B. Shoenmakers, and J. Villegas. Practical and secure solutions for integer comparison. In *Conference on Theory and Practice of Public Key Cryptography (PKC)*, pages 330–342, 2007.

[41] M. Geisler. *Cryptographic Protocols: Theory and Implementation*. PhD thesis, Aarhus University, February 2010. Available from https://bitbucket.org/mg/dissertation/downloads/.

[42] R. Gennaro, M. Rabin, and T. Rabin. Simplified VSS and fast-track multiparty computations with applications to threshold cryptography. In *ACM Symposium on Principles of Distributed Computing (PODC)*, pages 101–111, 1998.

[43] B. Goethals, S. Laur, H. Lipmaa, and T. Mielikainen. On private scalar product computation for privacy-preserving data mining. In *ICISC*, volume 3506 of *LNCS*, pages 104–120, 2004.

[44] O. Goldreich, S. Micali, and A. Wigderson. How to play any mental game. In *ACM Symposium on Theory of Computing (STOC)*, pages 218–229, 1987.

[45] M. Hirt and U. Maurer. Robustness for free in unconditional multi-party computation. In *Advances in Cryptology – CRYPTO*, pages 101–118, 2001.

[46] M. Hirt and J. Nielsen. Robust multiparty computation with linear communication complexity. In *Advances in Cryptology – CRYPTO*, pages 463–482, 2006.

[47] S. Hohenberger and A. Lysyanskaya. How to securely outsource cryptographic computations. In *Thory of Cryptography Conference (TCC)*, volume 3378 of *LNCS*, pages 264–282, 2005.

[48] S.-M. Hong, J.-B. Shin, H. Lee-Kwang, and H. Yoon. A new approach to server-aided secret computation. In *Informaiton Security and Cryptology (ISC)*, pages 33–45, 1998.

[49] P. Indyk and D. Woodruff. Polylogarithmic private approximations and efficient matching. In *Theory of Cryptography Conference (TCC'06)*, volume 3876 of *LNCS*, pages 245–264, 2006.

[50] M. Jakobsson and S. Wetzel. Secure server-aided signature generation. In *International Workshop on Practice and Theory of Public Key Cryptography (PKC)*, pages 383–401, 2001.

[51] A. Juels and M. Sudan. A fuzzy vault scheme. In *International Symposium on Information Theory*, 2002.

[52] A. Juels and M. Wattenberg. A fuzzy commitment scheme. In *ACM Conference on Computer and Communications Security*, pages 28–36, 1999.

[53] J. Katz, A. Sahai, and B. Waters. Predicate encryption supporting disjunctions, polynomial equations, and inner products. In *Advances in Cryptology – EUROCRYPT 2008*, volume 4965 of *LNCS*, pages 146–162, 2008.

[54] S. Kawamura and A. Shimbo. Fast server-aided secret computation protocols for modular exponentiation. *IEEE Journal on Selected Areas in Communications*, 11(5):778–784, 1993.

[55] L. Kissner and D. Song. Privacy-preserving set operations. In *Advances in Cryptology – CRYPTO'05*, volume 3621 of *LNCS*, pages 241–257, 2005.

[56] C. Lim and P. Lee. Security and performance of server-aided RSA computation protocols. In *Advances in Cryptology – CRYPT0 95*, pages 70–83, 1995.

[57] Y. Lindell and B. Pinkas. Privacy-preserving data mining. *Journal of Cryptology*, 15(3):177–206, 2002.

[58] J. Linnartz and P. Tuyls. New shielding functions to enhance privacy and prevent misuse of biometric templates. In *International Conference on Audio and Video Based Biometric Person Authentication*, June 2003.

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

[60] T. Matsumoto, K. Kato, and H. Imai. Speeding up secret computations with insecure auxiliary devices. In *Advances in Cryptology – CRYPT0 88*, pages 497–506, 1988.

[61] T. Nishide and K. Ohta. Multiparty computation for interval, equality, and comparison without bit decomposition protocol. In *Conference on Theory and Practice of Public Key Cryptography (PKC)*, pages 343–360, 2007.

[62] M. Osadchy, B. Pinkas, A. Jarrous, and B. Moskovich. SCiFI – A system for secure face identification. In *IEEE Symposium on Security and Privacy*, pages 239–254, 2010.

[63] K. Peng and F. Bao. Achieving high efficiency in membership proof without compromising or weakening any security property. In *IEEE International Conference on Computer and Information Technology*, pages 1044–1049, 2010.

[64] K. Peng and F. Bao. An efficient range proof scheme. In *IEEE International Conference on Information Privacy, Security, Risk and Trust (PASSAT)*, pages 826–833, 2010.

[65] N. Ratha, J. Connell, and R. Bolle. Enhancing security and privacy in biometrics-based authentication systems. *IBM Systems Journal*, 40(3), 2001.

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

[67] E. Shen, E. Shi, and B. Waters. Predicate privacy in encryption systems. In *Theory of Cryptography Conference (TCC'09)*, volume 5444 of *LNCS*, pages 457–473, 2009.

[68] E. Shi, J. Bethencourt, H. Chan, D. Song, and A. Perrig. Multi-dimensional range query over encrypted data. In *IEEE Security and Privacy Symposium*, pages 350–364, 2007.

[69] S. Tillich and J. Großschädl. VLSI implementation of a functional unit to accelerate ECC and AES on 32-bit processors. In *Arithmetic of Finite Fields*, volume 4547 of *LNCS*, pages 40–54, 2007.

[70] P. Tuyls, A. Akkermans, T. Kevenaar, G.-J. Schrijen, A. Bazen, and G. Veldhuis. Practical biometric authentication with template protection. In *Audio- and Video-Based Biometric Person Authentication (AVBPA'05)*, volume 3546 of *LNCS*, pages 436–446, 2005.

[71] J. Vaidya and C. Clifton. Privacy-preserving k-means clustering over vertically partitioned data. In *ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 206–215, 2003.

[72] J. Vaidya, M. Kantarcioglu, and C. Clifton. Privacy-preserving naive bayes classification. *VLDB Journal*, 17(4):879–898, 2008.

[73] E. Verbitskiy, P. Tuyls, D. Denteneer, and J. Linnartz. Reliable biometric authentication with privacy protection. In *Benelux W.I.C. Symposium on Information Theory*, pages 125–131, May 2003.

[74] R. Wright and Z. Yang. Privacy-preserving Bayesian network structure computation on distributed heterogeneous data. In *ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 713–718, 2004.

[75] A. Yao. Protocols for secure computations. In *IEEE Symposium on Foundations of Computer Science*, pages 160–164, 1982.

A Majority Algorithms

Let $b_1, \ldots, b_k$ denote the ith bit of biometric representations used in forming the template, and $m_1, \ldots, m_k$ denote the ith mask bit in the corresponding representations. The algorithms for forming ith biometric bit b of the template and its corresponding mask bit m are given in Figure 6.

Any-Support-Alg is a simple algorithm that sets the mask bit to 0 if at least one mask bit m_i is set. The biometric bit is set according to the majority of bits marked as reliable and to 0 is no majority exists. In Threshold-Support-Alg, when sufficient evidence of bit reliability cannot be gathered, the mask bit is set to 0. In both algorithms, for simplicity we set b to 0 if $m = 0$. If, however, all biometric bits are used in some computation or analysis regardless of the mask bit value (e.g., to evaluate the distribution of bits in templates), it is more accurate to set the bit in the same way as in the case when the bit is considered to be sufficiently reliable.

Any-Support-Alg:

```
1.  n_0 := 0
2.  n_1 := 0
3.  for (i from 1 to k) {
4.    if (m_i = 1) then
5.      if (b_i = 0) then n_0 := n_0 + 1
6.      else n_1 := n_1 + 1
7.  }
8.  if (n_1 = 0 and n_0 = 0) {
9.    m := 0
10.   b := 0
11. }
12. else {
13.   m := 1
14.   if (n_1 > n_0) then b := 1
15.   else b := 0
16. }
```

Threshold-Support-Alg:

```
1.  c := sum_{i=1}^{k} m_i
2.  if (c ≤ ⌊k/4⌋) then {
3.    m := 0
4.    b := 0
5.  }
6.  else {
7.    n_0 := 0
8.    n_1 := 0
9.    for (i from 1 to k) {
10.     if (m_i = 1) then
11.       if (b_i = 0) then n_0 := n_0 + 1
12.       else n_1 := n_1 + 1
13.   }
14.   if (n_0 = n_1) then {
15.     m := 0
16.     b := 0
17.   }
18.   else {
19.     m := 1
20.     if (n_0 > n_1) then b := 0
21.     else b := 1
22.   }
23. }
```

Figure 6: Algorithms for computing majority codes.

B Review of Secure Protocols from [17]

B.1 Secure comparison protocol

In this section we review the secure comparison protocol of [17] which can be performed without decomposing the operands into their bitwise representation. This protocol is used in our multiple-server MSSBS solution, and the description given in this section will be used to analyze the security of our protocol.

To compare two values $[a]$ and $[b]$ of bitlength k, a protocol for determining whether a shared integer is less than zero, LTZ, is used: $\mathsf{LTZ}([a], k)$ computes 1 if $a < 0$ and 0 otherwise. Then comparison of $[a]$ and $[b]$ is reduced to LTZ as follows:

$$(a \stackrel{?}{<} b) = \mathsf{LT}(a, b) = \mathsf{LTZ}(a - b) \qquad (a \stackrel{?}{\leq} b) = \mathsf{LE}(a, b) = 1 - \mathsf{LTZ}(b - a)$$
$$(a \stackrel{?}{>} b) = \mathsf{GT}(a, b) = \mathsf{LTZ}(b - a) \qquad (a \stackrel{?}{\geq} b) = \mathsf{GE}(a, b) = 1 - \mathsf{LTZ}(a - b)$$

To be able to represent negative values, the size of the field needs to be at least twice as large as the range of the integers to be represented (of bitlength k). The protocols in [17], however, achieve only statistical security, which means that the size of the field q must be at least as large as $2^{k+\kappa}$, where κ is a security parameter for statistical privacy. This implies that the field size will always be sufficiently large for the correctness of the protocol.

Then the functionality of $\mathsf{LTZ}([a], k)$ is implemented using the following observation: if $a < 0$, then $\lfloor a/2^{k-1} \rfloor = -1$; and if $a \geq 0$, then $\lfloor a/2^{k-1} \rfloor = 0$. Therefore, the sign of $[a]$ is determined by

computing $[s] = -\mathsf{Trunc}([a], k, k-1)$, where the truncation function $\mathsf{Trunc}([a], k, m)$ computes (the shares of) $\lfloor a/2^m \rfloor$. The protocols for LTZ and Trunc are given next.

Protocol B.1. $[s] \leftarrow \mathsf{LTZ}([a], k)$

1. $[s] \leftarrow -\mathsf{Trunc}([a], k, k-1)$;
2. return $[s]$;

Protocol B.2. $[d] \leftarrow \mathsf{Trunc}([a], k, m)$

1. $[a'] \leftarrow \mathsf{Mod2m}([a], k, m)$;
2. $[d] \leftarrow ([a] - [a'])(2^{-m} \bmod q)$;
3. return $[d]$;

As one can see from the above, the protocol Trunc makes a use of additional protocol $\mathsf{Mod2m}([a], k, m)$ that computes $[a'] = [a \bmod 2^m]$. Protocol $\mathsf{Mod2m}$ in turn utilizes three additional building blocks defined as follows:

- $\mathsf{Output}([a])$ corresponds to the participants broadcasting their shares and reconstructing the value of a.
- $\mathsf{PRandM}(k, m)$ allows the parties to generate a partially decomposed random value. The parties produce $[r'']$ and $[r']$ along with bit decomposition of $[r']$, where r'' is of length $k+\kappa-m$ and r' has length m. This ensures that only the minimum necessary number of random bits has been generated.
- $\mathsf{BitLT}(a, ([b_k], \ldots, [b_1]))$ corresponds to the comparison (less-than) operation where the first operand is public and the second one is available in the bit-decomposed form. The protocol outputs a bit $[s]$ which is equal to 1 iff $a < b$.

A precise description of $\mathsf{Mod2m}$ and its building block $\mathsf{PRandM}(k, m)$ is given next:

Protocol B.3. $[a'] \leftarrow \mathsf{Mod2m}([a], k, m)$

1. $([r''], [r'], [r'_{m-1}], \ldots, [r'_0]) \leftarrow \mathsf{PRandM}(k, m)$;
2. $c \leftarrow \mathsf{Output}(2^{k-1} + [a] + 2^m[r''] + [r'])$;
3. $c' \leftarrow c \bmod 2^m$;
4. $[u] \leftarrow \mathsf{BitLT}(c', ([r'_{m-1}], \ldots, [r'_0]))$;
5. $[a'] \leftarrow c' - [r'] + 2^m[u]$;
6. return $[a']$;

Protocol B.4. $([r''], [r'], [b_{m-1}], \ldots, [b_0]) \leftarrow \mathsf{PRandM}(k, m)$

1. $[r''] \leftarrow \mathsf{PRandInt}(k + \kappa - m)$;
2. for $i = 0$ to $m - 1$ do $[b_i] \leftarrow \mathsf{PRandBit}(q)$;
3. $[r'] \leftarrow \sum_{i=0}^{m-1} 2^i[b_i]$;
4. return $[r''], [r'], [b_{m-1}], \ldots, [b_0]$;

As can be seen from the above, in addition to local operations, PRandM makes calls to two primitives that allows parties to produce shares of random values: $\mathsf{PRandInt}(k)$ that allows the parties to compute shares of a random k-bit value and $\mathsf{PRandBit}$ that allows the parties to compute shares of a random bit. $\mathsf{PRandInt}$ is a basic primitive that allows the parties to agree on shares of a random value without any communication (followed some initial setup). We refer the reader to [17] for

Protocol	Rounds	Invocations
$c \leftarrow [a]b$; $c \leftarrow [a] + b$	0	0
$[c] \leftarrow [a] + [b]$	0	0
$[c] \leftarrow [a][b]$	1	1
$a \leftarrow \mathsf{Output}([a])$	1	1
$[r] \leftarrow \mathsf{PRandFld}(\mathbb{F})$	0	0
$[r] \leftarrow \mathsf{PRandInt}(k)$	0	0
$c \leftarrow \mathsf{MulPub}([a], [b])$	1	1

Table 10: Complexity of core protocols.

more information. RRandBit, on the other hand, is more complex and is presented below. In the PRandBit protocol, PRandFld allows the parties to generate shares of a random element of the field ($\mathbb{Z}_q$) with no communication, and MulPub($[a], [b]$) corresponds to a protocol for multiplies a and b and opening the result. MulPub is represented as a single primitive rather than two operations of multiplication and opening because it can be achieved with one invocation and, more importantly, in one round (instead of two invocations that require two rounds). Table 10 lists core protocols and their cost. We note that the last three protocols – PRandFld, PRandInt, and MulPub – can achieve lower than usual performance due to the use of Pseudorandom Replicated Secret Sharing (PRSS) in [17], where the parties generate PRSS shares of random values without any interaction and then convert them to shares in the linear secret sharing scheme.

Protocol B.5. $[b] \leftarrow \mathsf{PRandBit}()$

1. $[r] \leftarrow \mathsf{PRandFld}(\mathbb{Z}_q)$;
2. $u \leftarrow \mathsf{MulPub}([r], [r])$;
3. $v \leftarrow u^{-(q+1)/4} \bmod q$;
4. $[b] \leftarrow (v[r] + 1)(2^{-1} \bmod q)$;
5. return $[b]$;

As can be seen from the above, PRandBit requires one invocation in one round, which means that PRandM requires m invocations in one round (since the interactive steps 1 and 2 can be carried out in parallel). To complete the description of the comparison operation, we present protocol BitLT and its building blocks next. Note that [17] provides alternative implementations of this functionality which run in constant and logarithmic number of rounds. The log-round protocol, BitLTL, achieves perfect privacy in which most operations can be performed over a small field. For simplicity of presentation and consecutive analysis, we, however, present only the constant-round version BitLTC.

Protocol B.6. $[u] \leftarrow \mathsf{BitLTC}(a, [b_k], \ldots, [b_1])$

1. for $i = 1$ to k do $[d_i] \leftarrow a_i + [b_i] - 2a_i[b_i]$;
2. $([p_k], \ldots, [p_1]) \leftarrow \mathsf{PreMulC}([d_k] + 1, \ldots, [d_1] + 1)$;
3. for $i = 1$ to $k - 1$ do $[s_i] \leftarrow [p_i] - [p_{i+1}]$;
4. $[s_k] \leftarrow [p_k] - 1$;
5. $[s] \leftarrow \sum_{i=1}^{k} [s_i](1 - a_i)$;
6. $[u] = \mathsf{Mod2}([s], k)$;
7. return $[u]$;

All operations in BitLTC are local, except for protocols Mod2 and PreMulC which are presented next. $\mathsf{Mod2}([a], k)$ computes $[a \bmod 2]$, and $\mathsf{PreMulC}([a_1], \ldots, [a_k])$ computes prefix-AND or prefix multiplications of its input bits in constant round and outputs $[p_1], \ldots, [p_k]$, where $[p_i] = \prod_{j=1}^{i} [a_j]$.

Protocol B.7. $[a_0] \leftarrow \mathsf{Mod2}([a], k)$

1. $([r''], [r'], [r'_0]) \leftarrow \mathsf{PRandM}(k, 1)$;
2. $c_0 \leftarrow \mathsf{Output}(2^{k-1} + [a] + 2[r''] + [r'_0])$;
3. $[a_0] \leftarrow c_0 - [r'_0] + 2c_0[r'_0]$;
4. return $[a_0]$;

Protocol B.8. $[p_1], \ldots, [p_k] \leftarrow \mathsf{PreMulC}([a_1], \ldots, [a_k])$

1. for $i = 1$ to k do in parallel
2. $\quad [r_i] \leftarrow \mathsf{PRandFld}(\mathbb{F})$;
3. $\quad [s_i] \leftarrow \mathsf{PRandFld}(\mathbb{F})$;
4. $\quad u_i \leftarrow \mathsf{MulPub}([r_i], [s_i])$;
5. for $i = 1$ to $k - 1$ do in parallel $[v_i] \leftarrow [r_{i+1}][s_i]$;
6. $[w_1] \leftarrow [r_1]$;
7. for $i = 2$ to k do $[w_i] \leftarrow [v_{i-1}](u_{i-1}^{-1} \bmod q)$;
8. for $i = 1$ to k do $[z_i] \leftarrow [s_i](u_i^{-1} \bmod q)$;
9. for $i = 1$ to k do in parallel $m_i \leftarrow \mathsf{MulPub}([w_i], [a_i])$;
10. $[p_1] \leftarrow [a_1]$;
11. for $i = 2$ to k do $[p_i] \leftarrow [z_i](\prod_{j=1}^{i} m_j)$;
12. return $([p_1], \ldots, [p_k])$;

This completes the presentation of all protocols and we obtain that the complexity of BitLTC is $3k + 1$ invocations in 3 rounds (i.e., two more invocations and one more rounds than that of PreMulC due to a call to Mod2). The overall complexity of (constant-round) Mod2m is thus $4m + 1$ invocations in 4 rounds, and this is the overall complexity of the constant-round version of the comparison protocol LTZ.

B.2 Secure OR protocol

Protocol B.9. $[b] \leftarrow \mathsf{OR}([a_1], \ldots, [a_k])$

1. $m \leftarrow \lceil \log(k) \rceil$;
2. $([r''], [r'], [r'_{m-1} \ldots, [r'_0]) \leftarrow \mathsf{PRandM}(k, m)$;
3. $c \leftarrow \mathsf{Output}(2^m[r''] + [r'] + \sum_{i=1}^{k} [a_i])$;
4. $(c_m, \ldots, c_1) \leftarrow Bits(c, m)$;
5. for $i = 1$ to m do $[d_i] \leftarrow c_i + [r'_i] - 2c_i[r'_i]$;
6. $[e] \leftarrow \mathsf{OR\text{-}SF}([d_1], \ldots, [d_m])$;
7. return $[e]$;

OR-SF denotes a protocol in which OR is evaluated as a symmetric function using the approach of Damgård et al. [23]. The cost of any symmetric function evaluation $[f(a_1, \ldots, a_k)] \leftarrow f([a_1], \ldots, [a_k])$ is the cost of one unbounded fan-in multiplication $([a_1], \ldots, [a_1 a_2 \cdots a_k]) \leftarrow \mathsf{Mul}^*([a_1], \ldots, [a_k])$. The latter can be implemented as follows:

Protocol	Rounds	Invocations
$[r] \leftarrow \mathsf{PRandFld}(\mathbb{F})$	1	1
$[r] \leftarrow \mathsf{PRandInt}(k)$	1	1
$c \leftarrow \mathsf{MulPub}([a], [b])$	2	2

Table 11: Modified complexity of core protocols without PRSS.

Protocol B.10. $[a_1], \ldots, [a_1 a_2 \cdots a_k] \leftarrow \mathsf{Mul}^*([a_1], \ldots, [a_k])$

1. for $i = 0$ to k do $[b_i], [b_i^{-1}] \leftarrow \mathsf{PRandFldInv}()$;
2. for $i = 1$ to k do $[c_i] \leftarrow [b_i][a_i]$;
3. for $i = 1$ to k do $d_i \leftarrow \mathsf{MulPub}([c_i], [b_i^{-1}])$;
4. for $i = 1$ to k do $[a_{1,i}] \leftarrow (\prod_{j=1}^{i} d_j)[b_0^{-1}][b_i]$;
5. return $[a_{1,1}], \ldots, [a_{1,k}]$;

The above protocol uses notation $a_{i_1,i_2} = \prod_{i=i_1}^{i_2} a_i$. It in turn makes calls to PRandFldInv procedure that generates a random (invertible) field element together with its multiplicative inverse at the cost of one round and one invocation, as described next:

Protocol B.11. $[a], [a^{-1}] \leftarrow \mathsf{PRandFldInv}()$

1. $[a] \leftarrow \mathsf{PRandFld}()$;
2. $[b] \leftarrow \mathsf{PRandFld}()$;
3. $c \leftarrow \mathsf{MulPub}([a], [b])$;
4. if $(c = 0)$ abort;
5. $[a^{-1}] \leftarrow c^{-1}[a]$;
6. return $[a], [a^{-1}]$;

We thus obtain that the overall complexity of Mul^* is $4k + 1$ invocations in 2 rounds (where the interactive parts of steps 1 and 2 and steps 3 and 4 can be carried out in parallel). Recall that in our case Mul^* is called on $m = \log k$ arguments, which gives us 3 rounds and $5m + 2 = 5\log(k) + 2$ invocations for the OR protocol.

B.3 Performance of protocols without PRSS

Here we analyze the complexity of LT and OR protocols if the PRSS is not used. This is relevant for carrying out the MSSBS protocol in the malicious model. In this case, the complexity of the last three protocols in Table 10 change to the values listed in Table 11. This gives us that the complexity of PRandBit changes to 3 invocations in 3 rounds (all can be precomputed), complexity of PRandM changes to $3m+1$ invocations in 3 rounds (all can be precomputed), complexity of Mod2 changes to 5 invocations in 4 rounds (3 can be precomputed), complexity of PreMulC changes to $7k - 1$ invocations in 4 rounds (2 can be precomputed), complexity of BitLTC changes to $7k + 4$ invocations in 5 rounds (2 can be precomputed), and complexity of Mod2m changes to $10k + 6$ invocations in 7 rounds (3 can be precomputed). We include values for input-independent rounds that can be precomputed, which means that for any given protocol those rounds can be run in parallel with computation preceding the protocol. We obtain that the complexity of LT is that of Mod2m and is $10k + 6$ invocations in 7 rounds.

Also, the complexity of PRandFldInv becomes 3 invocations in 3 rounds (all can be precomputed),

that of Mul^* becomes $7k + 3$ invocations in 5 rounds (2 can be precomputed), and therefore the complexity of OR becomes $10\log(k) + 5$ invocations in 7 rounds (3 can be precomputed).