Secure and Verifiable Outsourcing of Large-Scale Biometric Computations

Marina Blanton and Yihua Zhang
Department of Computer Science and Engineering
University of Notre Dame
{mblanton,yzhang16}@nd.edu

Keith B. Frikken
Computer Science and Software Engineering
Miami University
frikkekb@muohio.edu

***Abstract*—Cloud computing services are becoming prevalent and readily available today, bringing to us economies of scale and making large scale computation feasible. Security and privacy considerations, however, stand on the way of fully utilizing the benefits of such services and architectures. In this work we address the problem of secure outsourcing of large-scale biometric experiments to a cloud, where privacy of the data is preserved and the client can verify that with very high probability the task was computed correctly. We conduct thorough theoretical analysis of the proposed techniques and provide instantiations for concrete biometric types that show that the overhead is modest.**

I. Introduction

Cloud computing enables on-demand access to computing and data storage resources, which can be configured to meet unique constraints of the clients and utilized with minimal management overhead. The recent rapid growth in availability of cloud services makes such services attractive and economically sensible for clients with limited computing or storage resources who are unwilling or unable to procure and maintain their own computing infrastructure. One of the largest possibilities that the cloud enables is computation outsourcing, when the client can utilize any necessary computing resources for its computational task. It has been suggested that the top impediment on the way of harnessing the benefits of cloud computing to the fullest extent is security and privacy considerations that prevent clients from placing their data or computations on the cloud (see, e.g., [1]). While in general sensitive data can be protected by the means of encryption, traditional encryption is not suitable for computation over data. Furthermore, the clients no longer have direct control over the outsourced data and computation and there is a lack of transparency in the current cloud services. The cloud provider can be incentivized to delete rarely accessed data or skip some of the computations to conserve resources (for financial or other reasons), which is especially true for volunteer-based computational clouds. Furthermore, unintentional data or computation corruption might also take place for a variety of reasons including malware, security break-ins, etc. From that perspective, it is important for the clients to be able to verify the correctness of the result of the outsourced task. The verification mechanism should not require the client to perform the computation comparable in size to the outsourced task itself. Secure and verifiable outsourcing of certain types of computation is therefore the focus of this work.

The main motivation for this work comes from the extensive amount of computation involved in biometric research that, due to memory and processing power constraints, inevitably pushes the computation on a computational cloud or grid. The sensitive nature of the data makes its protection throughout the computation a necessary requirement, and to be able to rely on the outcome of the computation, the result needs to be verified. While we present the developed techniques in the context of biometric data processing, our results can be used for any type of computation of similar structure.

The computation, secure and verifiable outsourcing of which we address in this work, can be described as follows. In biometric research, evaluation of a new recognition algorithm amounts to running the algorithm on a very large number of biometric images. Given a data set D of raw biometrics, first each of them needs to be processed to extract the features. Next, the distance between each pair of processed biometrics in D is computed, which is called "all pairs" computation. The result allows users to gather distribution (and any necessary statistical) information about the quality of biometric matching for impostor (different subjects) and authentic (same subject) comparisons. The accuracy of the result depends on the size of D, which can often consist of tens (or even hundreds) of thousands of biometrics. This volume of the computation cannot be performed on a single machine and needs to be partitioned and run by a computational grid or cloud.

Prior work [8] provides a system for seamlessly placing such functionality on a grid. In this work, we extend the framework by integrating security protection in the computation to ensure that it can be placed on a cloud comprised of untrusted machines and the correctness of the computation is verifiable by the client. Throughout this work, we assume that the client is capable of performing work linear in the number biometrics in D, $|D|$, but computation exceeding this linear bound is beyond the client's capabilities. For that reason, we focus on the computation corresponding to $\mathsf{AllPairs}$ and $\mathsf{Analyze}$ functionalities with quadratic complexity, defined as follows: $\mathsf{AllPairs}(S, F)$ compares all members of set S using function F and produces matrix M, where each element $M[i][j] = F(S[i], S[j])$; $\mathsf{Analyze}(M, C)$ extracts statistical or other quality metric data from matrix M and stores the result in C. Furthermore, because secure comparison of biometric templates has been a subject of prior research (see, e.g., [12],

[3], [6] and others), this work is primarily dedicated to techniques for computation verification. These techniques are independent of the mechanisms for securing the computation (e.g., those based on secret sharing or homomorphic encryption) and can be used with any suitable solution for protecting privacy of the data. Nevertheless, to illustrate that all of the computation considered in this work can indeed be carried out privately in an outsourced scenario, we describe secure protocols that can be derived from prior literature. For concreteness, throughout this work we assume that unconditionally secure techniques based on linear secret sharing are used.

In the current version of this work due to space constraints we focus on the verification techniques for the Analyze functionality, which is the more interesting and difficult among AllPairs and Analyze. The full version [7] shows how to enforce verifiability of AllPairs computation and integrate the techniques with those of Analyze functionality. It also provides implementation results.

Our contributions. We design a mechanism for verification of outsourced Analyze computation for three distinct distance metrics used in biometric computation: the Hamming distance (Section III), the Euclidean distance (Section IV), and the set intersection cardinality (Section V). The rigorous analysis that allows the client to set the security parameters as to achieve the desired probability of misbehavior detection is omitted due to space constraints and can be found in the full version [7]. We, however, demonstrate to what values the security parameters should be set to achieve desired security guarantees (according to the analysis) and show that the overhead of our schemes is acceptable. The computation is assumed to be carried out on protected data, and we illustrate a way of achieving privacy-preserving outsourcing for all functionality used in this work.

II. Problem Description

Computation description. A client has a pre-selected large collection S of biometric templates, which are to be compared and analyzed. To accomplish the AllPairs functionality (which comes before Analyze), the matrix M is partitioned across multiple computational servers such that each server receives a computational task which can be performed within its memory and computational capacity constraints. Let a server receive a job of the form of two sets S_1 and S_2 of n items each and its task is to perform comparisons of each pair of items $x \in S_1$ and $y \in S_2$, producing an $n \times n$ matrix as the output. When we refer to this matrix, we will assume that the items from the first set correspond to rows of the matrix and the items from the second set correspond to columns. The server is not assumed to follow the computation as prescribed, but it might be interested in attempting to avoid being detected that (some of) the computation was not performed. The computation is appropriately secured, so that the server does not learn any information about the data it handles, but has the description of the computation. Depending on how the secure computation is realized, a single task might take the form of multi-party computation, in which case it should be understood that when we refer to server's computation it involves (interactive) computation by multiple entities. Also, because biometric comparisons amount to computing the distance between two items which normally consist of multiple elements, we assume that each item consists of m elements.

To accomplish the Analyze functionality, the computational servers will need to post-process the distance values in the matrix to compute statistical information. In this work we propose that each server computes the number of times each particular distance value appeared among the n^2 computed distances. This information fully defines the distribution and will allow the client to compute any necessary statistics from it. (When the computational task is partitioned among multiple servers, the client can easily merge the returned data by simply adding the counts from multiple servers for each given distance value.) To compute the count for each distance, the server obliviously compares a distance it computed for a given cell to all possible distance values and increments one of them that matched (without knowing which count was incremented). We refer to the data structure that stores distribution information (i.e., an array of protected counts) as C. Note that the number of counts that C contains will depend on the range of distances information about which is being collected. For instance, for biometrics represented as m-bit binary strings the distance between which is computed as the Hamming distance, the range of distances will be $[0, m]$. Let $C = \langle c_0, \ldots, c_{v-1} \rangle$, where v is defined by the range of the distances and is a function of m. The pseudo-code below shows how C is updated for each cell. Below, notation $[x]$ means that the value of x is not known to the server, the result of the equality testing operation ($a \overset{?}{=} b$) is a bit, and d_i is the distance value associated with count c_i. Initially, all c_i's are set to 0.

```
$[d] := \mathsf{dist}([x],[y])$;
for $i = 0, \ldots, v-1$
    $[b_i] := ([d] \overset{?}{=} [d_i])$;
    $[c_i] := [c_i] + [b_i]$;
```

Note that because the server does not have access to the distances d_i associated with each c_i, the values of d_i do not have to equal to i or even be in the range $[0, v-1]$.

Security model. As mentioned above, our goal is to achieve secure and verifiable computation outsourcing. Because computation verification techniques is the focus of this work, throughout most of it we assume that the computation can be carried out in a secure manner. For completeness of this work, we, however, show below how the types of computation used in this work can be carried out in a private manner by the servers to whom tasks are being outsourced. We thus obtain that the server performing the computation is unable to learn information about the data it handles, but might deviate from the computation by skipping a portion of it or returning incorrect results. In particular, we assume that the server computes fraction p of its task, where $0 \le p \le 1$, and attempts to manipulate the result so as to make the client believe that it computed its task as prescribed. The client's goal is then to devise a mechanism which detects the server's misbehavior even when the portion of skipped computation, $1-p$, is small. Let D denote the event that the client detects server's cheating (and $\overline{\mathsf{D}}$ that the client does not). Then the client's goal is to achieve $\Pr[\mathsf{D}] \geq 1 - \frac{1}{2^\kappa}$ for the desired security parameter κ whenever p is below the specified threshold. It is assumed that the server knows the verification procedure and knows (or can sufficiently well approximate) all of the parameters used by the client for devising its task verification mechanism. These include $\Pr[\mathsf{D}]$, p, m, n, and all other security parameters derived from them as detailed later in this work.

Achieving privacy of distance and statistics computation. Three different distance metrics are treated in this work: the Hamming distance (used for iris), the Euclidean distance (used for faces and other types of biometrics), and the set intersection cardinality (used for fingerprints). Secure outsourcing of iris code comparisons to one or multiple servers has been addressed in a recent work [6]. The computation considered in that work is more complex than the Hamming distance alone, but for the purposes of this work we simplify the computation and show how the Hamming distance and statistics computation can be achieved in the multi-party setting. Based on that, we also provide a protocol for the Euclidean distance computation. Lastly, [5] develops currently the most efficient multi-party implementation of secure set intersection and set intersection cardinality which can be used in outsourced environment. We show how the solution can be adopted to our work. All protocols for private distance and statistics computation are given in Appendix A.

III. Verification of Statistics Computation for Hamming Distance

We now proceed with describing our techniques for verifiable Analyze functionality. In the current description we use the Hamming distance as the distance metric for computing distances between two items. Modifications to this method that apply to other distance metrics are given afterwards.

A. *First attempt*

At a high level, the idea consists of introducing fake items, the distances associated with which fall into a range different from original distances and can be verified by the client. The client has two data sets and inserts n_1 (n_2) fake items into S_1 (S_2, resp.), where the resulting sets are of size n each. Each original item consists of m elements, where every element is a bit (i.e., the Hamming distance between any two items is in the range $[0, m]$). The client inserts a new $(m+1)$st element in each item as follows: if $x = \langle x_1, \ldots, x_m \rangle$ is a real item, the client modifies it to $x = \langle x_1, \ldots, x_m, 0 \rangle$. To form a fake item y, the client sets its elements to $\langle y_1, \ldots, y_m, m+1 \rangle$, where bits $y_1, \ldots, y_m$ are chosen according to any desired distribution. The server will be unable to distinguish two types of values because they are not accessible to the server in the clear.

We obtain that the distances between two real items remain in $[0, m]$, while distances between a real and fake items now lie in the range $[m+1, 2m+1]$. The distances between two fake items can be made not to overlap with the range $[0, 2m+1]$ since the Hamming distance is computed using arithmetic operations.

The client can verify the correctness of the server's computation using the number of computed distances that fall in each range. In particular, the server computes the distance distribution by comparing each computed distance to $2m+2$ values from 0 to $2m+1$ and incrementing the count for the distance that matched. The client's verification consists of adding the counts for the distances in the range $[0, m]$ and the counts for the distances in the range $[m+1, 2m+1]$. The client compares the aggregate counts to their expected values $(n-n_1)(n-n_2)$ and $(n-n_1)n_2+n_1(n-n_2)$, respectively, and the computation is considered correct if both of them match.

Now suppose that the server computes pn^2 distances and would like to avoid detection. Instead of trying to guess the locations of fake items, it simply returns $2m+2$ counts, such that counts c_0 through c_m add to $(n-n_1)(n-n_2)$ and counts c_{m+1} through c_{2m+1} add to $(n-n_1)n_2+n_1(n-n_2)$. Because it is reasonable to assume that the server knows (or can guess) n_1 and n_2, which are set by the client to achieve a certain level of security, the server can always be successful in avoiding the detection. This means that a different solution is needed.

B. *Improved solution*

To improve security of the above solution, we employ two ideas: (i) (protected) distances used for computing statistics are given to the server in randomized order and (ii) the client verifies a larger number of aggregate counts. By itself, the first modification still results in insufficiently high detection probability, but in combination with the second it leads to the client's ability to achieve a desired level of protection.

In detail, both client's procedures for preparing the data and verifying the computation change. Before the computation is sent to the server, the client adds extra k elements to each real item (with m original elements). The $m+k$ elements are randomly permuted, but consistently across all items. Let $i_1, \ldots, i_k$ denote positions of extra elements. All artificial elements are set to 0 in real items.

To form fake items, the client first chooses a small integer ℓ, which will be used as a security parameter. The client next chooses ℓ values larger than m; each value will be used to increase the distance between certain fake items and real items. For concreteness, and without loss of generality, let these values be $m+1, \ldots, m+\ell$. Then to create a fake item, the client first chooses a distance d at random from the range $[m+1, m+\ell]$. Next, the client chooses k values $d_1, \ldots, d_k$ such that $\sum_{i=1}^{k} d_i = d$, and sets the element at position i_j in the fake item to d_j for $j = 1, \ldots, k$ and all original elements to 0. This setup implies that the distance between this fake item and any real item is in the range $[d, d+m]$, and the distances between any fake item and any real time will always be in the range $[m+1, 2m+\ell]$. The client also records the number of times each d was used in a fake item in the set S_1 and S_2, respectively (both formed in the above described way). Let the counts be denoted c_i^j, where $i \in [m+1, m+\ell]$ and $j \in [1, 2]$.

Before the client will be able to verify the computation performed by the server using sets S_1 and S_2, the client needs to compute additional information as follows: For each real item in S_1, the client computes the number of bits sets to 1 in that biometric (i.e., its Hamming weight) and counts the number of instances of each distance across all real items. Let $s_0^1, \ldots, s_m^1$ denote the distribution of the distances, where s_i^1 indicates the number of real items in S_1 with the Hamming weight of i. Similarly, the client produces the counts for the real items in S_2, which are denoted by $s_0^2, \ldots, s_m^2$.

After the server receives sets S_1 and S_2 from the client, it computes the distances between all pairs. Afterwards, when the server produces statistics, it will need to compare each computed distance to the values in the range $[0, 2m+\ell]$. Since the comparisons are performed obliviously without the server knowing to what value a distance is being compared, the client randomizes the order in which all possible distances are communicated to the server. This means that the server will not be able to know what positions within the set of $2m+\ell+1$ values correspond to original distances from 0 to m and which correspond to artificial distances above m.

After the server returns statistics data to the client, the client performs the verification procedure in Figure 1 (which is also used for other distance metrics) using parameters $[l_r, l_u] = [0, m]$, $[l_f, u_f] = [m+1, 2m+\ell]$, and $d_o = m$. Notation $C[i]$ denotes the count returned by the server for distance i and $d_o = m$ indicates that the original elements of real items contribute distances in $[0, m]$ to the distances between real and fake items. Step 2 in the figure represents the number of distances contributed by the intersection of fake items in S_1 with real items in S_2 and fake items in S_2 with real items in S_1. For example, $c_{m+1} = c_{m+1}^1 s_0^2 + c_{m+1}^2 s_0^1$, $c_{m+2} = c_{m+1}^1 s_1^2 + c_{m+2}^1 s_0^2 + c_{m+1}^2 s_1^1 + c_{m+2}^2 s_0^1$, etc. If all checks succeed, the client treats the obtained distribution as correct.

In order for distances between two fake items not to interfere with the counts being verified, we can ensure that $\mathsf{dist}(x, y)$ between fake x and y are outside of the range $[0, 2m+\ell]$ using the following: Because in secure computation modular arithmetic is normally used, the client chooses d_i's uniformly at random from $\mathbb{Z}_q$, when the arithmetic is carried out modulo q, while maintaining $\sum_{i=1}^{k} d_i \bmod q = d$. When $q \gg 2m+\ell$, with high probability all $n_1 n_2$ distances between two fake items will fall outside of the range $[0, 2m+\ell]$. If, however, $\mathsf{dist}(x, y)$ happens to be in $[0, 2m+\ell]$ for some x and y, the client can choose a different set of d_i's for x or y. We, however, allow $\mathsf{dist}(x, y)$ to be in $[0, 2m+\ell]$, in which case the client needs to adjust the counts it expects from the server and also needs to compensate for the error when using statistical data computed by the server. This incurs minimal overhead on the client, but allows to keep q low.

C. Performance

To evaluate the overhead introduced by our techniques, we need to compute parameters n_1, k, and ℓ for a fixed m, variable n, and desired values of Pr[D] and p. Our analysis also uses an additional parameter γ that serves a role similar to Pr[D] that ensures that the client's security guarantees hold with probability at least $1-\gamma$. Therefore, the parameters are computed for the desired values of Pr[D], γ, and p. We use $m = 1000$ in our computation, which is similar to the length of iris codes used in commercial software.

We show the values of k, ℓ, and n_1 according to our (omitted) analysis for three different settings of Pr[D], γ, and p in Table I. As can be seen from the table, for $m = 1000$ it is sufficient to set ℓ to 1. Also, the value of m has a low impact on the value of k. Similarly, the value of n_1 increases slowly as a function of n and approaches a constant.

The cost associated with our verification techniques consists of the overhead due to (i) the addition of fake elements which amounts to the fraction k/m for tasks of all sizes n, (ii) the addition of fake items which amounts to the fraction n_1/n that decreases as n grows, and (iii) expanding the size of C from m to $2m+\ell$ which doubles the work associated with processing each pair of items. The client's task preparation and verification effort is minimal and amounts to a couple of seconds for tasks with n set to several thousand. We refer the reader to [7] for additional results.

IV. Verification of Statistics Computation for Euclidean Distance

We now proceed with the verification techniques for the Euclidean distance. This time, each item x is represented as an m-dimensional vector $\langle x_1, x_2, \ldots x_m \rangle$, where every element $x_i \in [0, t]$ for $1 \le i \le m$. Instead of following the distance computation exactly, in this metric, we have the server compute the distribution of squared distances and send the result back to the client. That is, we define $\mathsf{dist}(x, y) = \sum_{i=1}^{m} (x_i - y_i)^2$. The client will then either produce a mapping between regular and squared distances as it forms a task assignment for the server, will take the square root of each returned result, or will operate directly on squared distances.

The solution is very similar to that for the Hamming distance. The client, as before, inserts $n_1 = n_2$ fake items into S_1 and S_2 and also inserts k fake elements into real and fake items. The resulting $m+k$ elements are randomly permuted, and all artificial elements are set to 0 in the real items.

To form fake items, the client chooses a small integer ℓ as the security parameter and then chooses ℓ values larger than mt^2 with each of them being used to increase the distance between real and fake items. For concreteness, we set these values to $mt^2+1, mt^2+2, \ldots, mt^2+\ell$. To form a fake item, the client first randomly chooses a distance d from these ℓ candidates. Next, the client chooses k fake elements at random from $\mathbb{Z}_q$, denoted by d_i for $1 \le i \le k$, so that the constraint $\sum_{i=1}^{k} {d_i}^2 = d$ is satisfied. In detail, the client chooses the first $k-1$ d_i's uniformly at random from $\mathbb{Z}_q$, sets $(d_k)^2$ to $d - \sum_{i=1}^{k-1} {d_i}^2$, and computes d_k. For a prime q, there is about 50% probability that square root computation fails (i.e., $(q-1)/2$ values from $\mathbb{Z}_q$ are quadratic nonresidues). In this case, the client chooses a new d_{k-1} at random and tries again until d_k is successfully found. Finally, the client sets the remaining m (original) elements in that item to 0. To aid producing the

Input:

- the range of distances $[l_r, u_r]$ between two real items in data structure C;
- the range of distances $[l_f, u_f]$ between real and fake items in data structure C;
- the maximum distance d_o that the original elements of real items contribute to the distance between real and fake items.

Algorithm steps:

1) The client adds the counts for the distances between l_r and u_r using $C[l_r]$ through $C[u_r]$ and compares the result to $(n-n_1)(n-n_2)$.
2) The client computes the expected distribution of the distances between real and fake items as follows: for each $i = l_f, \ldots, u_f$, the client sets $c_i = \sum_{j=\max(l_f, i-d_o)}^{\min(u_f-l_f+1, i)} (c_j^1 s_{i-j}^2 + c_j^2 s_{i-j}^1)$. The client then compares each c_i with $C[i]$.

Fig. 1. Description of client's computation of the expected statistics.

Security setting	Computed parameters							
	k	ℓ	n_1					
	any n	any n	$n = 200$	$n = 400$	$n = 600$	$n = 800$	$n = 1000$	$n = 2000$
$p \leq 0.9$, Pr[D] ≥ 0.95, $\gamma \leq 0.05$	28	1	27	28	28	28	29	29
$p \leq 0.95$, Pr[D] ≥ 0.95, $\gamma \leq 0.05$	57	1	51	55	56	57	57	58
$p \leq 0.95$, Pr[D] ≥ 0.99, $\gamma \leq 0.01$	87	1	73	81	84	85	86	88

TABLE I

VALUES OF PARAMETERS k, ℓ, AND n_1 FOR VERIFICATION OF HAMMING DISTANCE-BASED STATISTICS WITH $m = 1000$.

expected statistics for computation verification purposes, the client records the number of times each d was used in a fake item in the set S_1 and S_2, respectively. Let the counts be denoted c_i^j, where $i \in [mt^2+1, mt^2+\ell]$ and $j \in [1, 2]$.

By forming a task as described above, the distances between two real items fall into the range $[0, mt^2]$, the distances between a real and fake items fall into the range $[mt^2 + 1, 2mt^2 + \ell]$, and the distances between two fake items can be anywhere in $\mathbb{Z}_q$. Because the range of distances between two fake items might now overlap with the range of distances between two real (or real and fake) items, for verification purposes, the client needs to precompute the distribution of distances between two fake items that fall into the range $[0, 2mt^2 + \ell]$, and subtract it from the statistics returned by the server. Note that the fake items could be reused for each task without lowering the detection probability, thus this computation should be treated as a one-time overhead.

Before the client is able to verify the computation performed by the server, it needs to compute additional information as follows: for each real item x in S_1 and S_2, the client computes the sum of squares of its elements $\sum_{i=1}^{m} x_i^2$, and counts the number of instances of each occurred value across all items. Let s_i^1 (s_i^2) denote the number of items with computed value i in S_1 (S_2, resp.). After acquiring this distribution, the client computes the expected statistics and verifies the results returned by the servers using the algorithm in Figure 1 with the inputs: $[l_r, u_r] = [0, mt^2]$, $[l_f, u_f] = [mt^2 + 1, 2mt^2 + \ell]$, and $d_o = mt^2$. If all the checks in the algorithm succeed, the client treats the obtained statistics data as correct.

The number of dimensions m in biometrics that rely on the Euclidean distance (such as faces) is significantly lower than m using in the Hamming distance analysis, normally not exceeding 50. Fortunately, our analysis of the scheme for the Euclidean distance shows that lower ℓ can be used for the Euclidean distance than the Hamming distance with comparable sizes of C. This gives us that $\ell = 1$ is sufficient for the Euclidean distance as well, and we obtain that the client can use the same values of n_1 and ℓ as given in Table I and the value of k for $m = 50$ will be lower than in the table.

V. VERIFICATION OF STATISTICS COMPUTATION FOR SET INTERSECTION CARDINALITY

We lastly describe our solution for statistics verification for the set intersection cardinality distance metric. Now each original item is composed of m elements[1] from the range $[0, t]$. The server is to compute the cardinality of set intersection of the elements in x and y, $|x \cap y|$, for each $x \in S_1$ and $y \in S_2$ and compile the distribution of the distances in the form of C.

As before, the client proceeds with inserting n_1 fake items into the sets and k fake elements into the real and fake items to aid verification. Unlike the previous distance metrics, the fake elements are not required to be positioned consistently across all items. All fake elements are set to 0 in real items.

To generate fake items, the client produces $2n_1$ values larger than t and assigns each of them to a single fake item so that each fake item obtains a unique value. We use d_i to denote the value assigned to the ith fake item. To form the ith fake item, the client randomly chooses a value d from the range $[0, k-1]$, and sets d randomly chosen elements of it to 0 and sets the remaining $m + k - d$ elements to d_i. Each resulting real or fake item is now a multiset, and we assume that the distance computation function will work properly on such items. The client also records the number of times each d was used in a fake item in the set S_1 and the set S_2, respectively, and we denote such counts by c_d^j, where $d \in [0, k-1]$ and $j \in [1, 2]$.

This setup gives us that the distances between any two real items fall into the range $[k, m + k]$, the distances between any real and fake items fall into the range $[0, k)$, and the distances between any two fake items fall into the range $[0, k)$. Because of the overlap of the last two ranges, the client needs to precompute the statistics for the distances between

[1]Note that each biometric is not required to be of length exactly m. Both correctness and security of our solution will hold if each item is of length at most m, i.e., m is the upper bound on the size of items.

any two fake items, add it to the expected statistics for the distances between real and fake items, and then compare the result to the statistics returned by the server. The rest of the verification process uses the algorithm in Figure 1 with the inputs: $[l_r, u_r] = [k, m + k]$, $[l_f, u_f] = [0, k - 1]$, and $d_o = 0$.

We note that the value of m is relatively small in biometric types that use this distance metric (e.g., fingerprints). This, based on our analysis, means that higher values of n_1 than what is reported in Section III-C might be necessary for the above scheme. We refer the reader to [7] for additional detail.

VI. Related Work and Conclusions

Research on verifiable or uncheatable computation was initiated in [14], [13] using redundant task execution and insertion of so-called ringers in search for rare events (in particular, performing inversion of one-way function). Consequently, Szajda et al. [21] extended the idea to optimization problems and sequential executions (while still relying on parallel and redundant task execution). Other publications in this direction include [10], [15], [17] that inject chaff sub-tasks or verify portions of the result for computations of certain structure (e.g., NP-complete problems); [18], [23] use redundant scheduling. Du et al. [11] suggest the use of commitment to the result of massively-parallel server's computation using a Merkle hash tree, where the client verifies the computation by challenging the server on a number of individual sub-tasks which must match the commitment. Finally, in [19] distributed checking is used where (possibly malicious) servers perform checks on each other. We note that often the techniques from the literature can achieve a high probability of cheating detection only when p is rather low (i.e., not close to 1). There are also a number domain-specific computation verification techniques that exploit domain knowledge to verify the results efficiently. Such techniques are known for algebraic computations [4], [2] and linear programming [22]. This work is thus can be positions between general and problem-specific techniques, as it assumes a certain structure of the computation, but the developed techniques are applicable to different instantiations of the distance function. We note that the general solutions listed above would not work in the context of this work, as distance computation consists of many elements and should not be treated as an integral function.

In this work we develop techniques for verifiable outsourcing of large-scale biometric computations for statistical data computation and provide their rigorous security analysis in the full version [7]. We treat several popular distance metrics such as the Hamming distance, Euclidean distance, and set intersection cardinality and show that our techniques introduce reasonable overhead.

Acknowledgments

Portions of this work were supported by grant CNS-0915843 from the National Science Foundation and grant AFOSR-FA9550-09-1-0223 from the Air Force Office of Scientific Research.

References

[1] IT cloud services user survey, pt. 2: Top benefits & challenges. http://blogs.idc.com/ie/?p=210.

[2] M. Atallah and K. Frikken. Securely outsourcing linear algebra computations. In *ACM Symposium on Information, Computer and Communications Security*, pages 48–59, 2010.

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

[4] D. Benjamin and M. Atallah. Private and cheating-free outsourcing of algebraic computations. In *Annual Conference on Privacy, Security, and Trust (PST)*, pages 240–245, 2008.

[5] M. Blanton and E. Aguiar. Private and oblivious set and multiset operations. ePrint Cryptology Archive 2011/464, 2011.

[6] M. Blanton and M. Aliasgari. Secure outsourced computation of iris matching. ePrint Cryptology Archive 2011/462, 2011.

[7] M. Blanton, Y. Zhang, and K. Frikken. Secure and verifiable outsourcing of large-scale biomeric computations. Technical Report 2011-04, Department of Computer Science and Engineering, University of Notre Dame, 2011.

[8] 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*, 12(4):373–386, 2009.

[9] O. Catrina and S. de Hoogh. Improved primitives for secure multiparty integer computation. In *International Conference on Security and Cryptography in Networks (SCN)*, pages 182–199, 2010.

[10] W. Du and M. Goodrich. Searching for high-value rare events with uncheatable grid computing. In *Applied Cryptography and Network Security*, pages 122–137, 2005.

[11] W. Du, J. Jia, M. Mangal, and M. Murugesan. Uncheatable grid computing. In *International Conference on Distributed Computing Systems*, pages 4–11, 2004.

[12] 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.

[13] P. Golle and I. Mironov. Uncheatable distributed computations. In *RSA Conference*, pages 425–440, 2001.

[14] P. Golle and S. Stubblebine. Secure distributed computing in a commercial environment. In *International Conference on Financial Cryptography*, pages 289–304, 2001.

[15] M. Goodrich. Pipelined algorithms to detect cheating in long-term grid computations. *Theoretical Computer Science*, 408(2–3):199–207, 2008.

[16] M. Goodrich. Randomized Shellsort: A simple oblivious sorting algorithm. In *ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 1262–1277, 2010.

[17] G. Karame, M. Strasser, and S. Capkun. Secure remote execution of sequential computations. In *International Conference on Information and Communications Security (ICICS)*, pages 181–197, 2009.

[18] H. Kim, J. Gil, C. Hwang, H. Yu, and S. Joung. Agent-based autonomous result verification mechanism in desktop grid systems. In *Agents and Peer-to-Peer Computing (AP2PC)*, pages 72–84, 2007.

[19] M. Kuhn, S. Schmid, and R. Watterhofer. Distributed asymmetric verification in computational grids. In *IEEE International Symposium on Parallel and Distributed Processing*, pages 1–10, 2008.

[20] A. Shamir. How to share a secret. *Communications of the ACM*, 22(11):612–613, 1979.

[21] D. Szajda, B. Lawson, and J. Owen. Hardening functions for large scale distributed computations. In *IEEE Symposium on Security and Privacy*, pages 216–224, 2003.

[22] C. Wang, K. Ren, and J. Wang. Secure and practical outsourcing of linear programming in cloud computing. In *INFOCOM*, 2011.

[23] K. Watanabe, M. Fukushi, and S. Horiguchi. Collusion-resistant sabotage-tolerance mechanisms for volunteer computing systems. In *IEEE International Conference on e-Business Engineering (ICEBE)*, pages 213–218, 2009.

Appendix A
Private Distance and Statistics Computation

Here we describe solutions for private distance and statistics computation in the multi-server outsourced context, where the computation takes the form of secure multi-party computation. Let servers $P_1, \ldots, P_n$ carry out the computation. The client secret-shares all data items among the servers using a (n, t) linear threshold secret sharing scheme as such as [20], and the computation proceeds on their shares. This means that $t < n/2$ participants information-theoretically learn no information about shared values, and the computation is secure in presence of collusion of size at most t. We use $[x]$ to denote that value x is secret-shared among the servers. Any linear combination of shared values can be computed locally by each server, but multiplication $[x][y]$ requires interaction. Complexity is measured in the number of interactive operations.

Our protocols rely on the following building blocks:

- $[c] \leftarrow \mathsf{Inner}([a_1], \ldots, [a_m], [b_1], \ldots, [b_m])$, on input of two vectors A and B of equal size, returns the inner product of the elements of A and B. The cost is one interactive operation (multiplication).
- $[b] \leftarrow \mathsf{Eq}([x], [y])$, on input two integers x and y, outputs a bit which is set to 1 iff $x = y$. Existing implementations use a linear or even sublinear in the length of x and y number of interactive operations in constant round (see, e.g., [9]).
- $[b_1], \ldots, [b_m] \leftarrow \mathsf{Sort}([a_1], \ldots, [a_m])$, given a set of elements, outputs the values in a sorted order. The sorting must be data-oblivious (i.e., the same sequence of comparisons is executed regardless of the input) to preserve privacy of values. Existing algorithms achieve this using $O(m \log m)$ comparisons (see, e.g., [16]).

All of the protocols we provide are information-theoretically secure in presence of servers that follow the computation (or do not maliciously change the computation) and achieve perfect secrecy (unless the invoked building blocks are only statistically secure). We present protocols for computing the distance between two biometrics separately from computing statistics based on computed distances since the statistics computation is the same for each distance metric.

The protocol for the Hamming distance is very simple:

Protocol 1. $[d] \leftarrow \mathsf{HD}([x_1], \ldots, [x_m], [y_1], \ldots, [y_m])$

```
1) $[d] \leftarrow \mathsf{Inner}([x_1], \ldots, [x_m], [y_1], \ldots, [y_m])$;
2) return $[d]$;
```

The Euclidean distance can be securely computed similarly:

Protocol 2. $[d] \leftarrow \mathsf{ED}([x_1], \ldots, [x_m], [y_1], \ldots, [y_m])$

```
1) for $i = 1$ to $m$ do in parallel $[z_i] \leftarrow [x_i] - [y_i]$;
2) $[d] \leftarrow \mathsf{Inner}([z_1], \ldots, [z_m], [z_1], \ldots, [z_m])$;
3) return $[d]$;
```

The cost of both of the above protocols is equivalent to one multiplication. The set intersection cardinality of X and Y of size m_1 and m_2, respectively, is computed according to [5].

Protocol 3. $[d] \leftarrow \mathsf{SIC}([x_1], \ldots, [x_{m_1}], [y_1], \ldots, [y_{m_2}])$

```
1) $[z_1], \ldots, [z_{m_1+m_2}] \leftarrow \mathsf{Sort}([x_1], \ldots, [x_{m_1}], [y_1], \ldots, [y_{m_2}])$;
2) for $i = 1$ to $m_1 + m_2 - 1$ do in parallel $[u_i] \leftarrow \mathsf{Eq}([z_i], [z_{i+1}])$;
3) $[d] \leftarrow \sum_{i=1}^{m_1+m_2-1} [u_i]$;
4) return $[d]$;
```

The protocol above works correctly only when all elements of each input set is unique (i.e., the inputs are sets rather than multisets). For the (more complex) protocol that correctly works on multisets, we refer the reader to [5]. The cost of both the set and multiset intersection cardinality protocols is dominated by $O((m_1 + m_2) \log(m_1 + m_2))$ invocations of comparison protocol, which uses the number of multiplications linear in the length of its operands.

Finally, given computed $[d]$, the servers update C as follows:

Protocol 4. $[c_0], \ldots, [c_{v-1}] \leftarrow \mathsf{Stat}([d], [c_0], \ldots, [c_{v-1}], [d_0], \ldots, [d_{v-1}])$

```
1) for $i = 0$ to $v - 1$ do
2)     $[b_i] \leftarrow \mathsf{Eq}([d], [d_i])$;
3)     $[c_i] \leftarrow [c_i] + [b_i]$;
4) return $[c_0], \ldots, [c_{v-1}]$;
```