A

Secure and Verifiable Outsourcing of Large-Scale Biometric Computations

MARINA BLANTON and YIHUA ZHANG, University of Notre Dame
KEITH B. FRIKKEN, Miami University

Cloud computing services are becoming more 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 or grid in a way that 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 implementation results that indicate that our solution imposes modest overhead.

Categories and Subject Descriptors: K.6.5 [**Management of Computing and Information Systems**]: Security and Protection; C.2.4 [**Computer-Communication Networks**]: Distributed Systems

General Terms: Security, Design, Algorithms.

Additional Key Words and Phrases: Computation verification, secure outsourcing, all-pairs computation, distance metrics.

1. 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. The cloud enables computational 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 all of the benefits of cloud computing is security and privacy considerations that prevent clients from placing their data or computations on the cloud (see, e.g., survey [Gens 2008]). While in general sensitive data can be protected by the means of encryption, computation using the data encrypted via the traditional means becomes impossible. 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 computation to conserve resources (for financial or other reasons), which is especially true for volunteer-based computational clouds (see, for instance, documented cases of cheating in SETI@home [Kahney 2001]). Furthermore, unintentional data or computation corruption

Portions of this work were supported by grants CNS-0915843 and CNS-1223699 from the National Science Foundation and grant AFOSR-FA9550-09-1-0223 from the Air Force Office of Scientific Research.
Authors' address: M. Blanton and Y. Zhang, Department of Computer Science and Engineering, University of Notre Dame, Notre Dame, IN 46556; K. Frikken, Computer Science and Software Engineering, Miami University, Oxford, OH 45056.

DOI 10.1145/0000000.0000000 http://doi.acm.org/10.1145/0000000.0000000

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 computational task. The computation 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 used in the computation 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.

Secure and verifiable outsourcing 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 DB of biometric images, first each image needs to be processed to extract the features; we refer to the resulting biometric data as the biometric template. Next, the distance between each pair of templates in DB is computed; this 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 and authentic comparisons (images corresponding to different subjects are termed impostor and images corresponding to the same subjects are termed authentic). The accuracy of the result depends on the size and quality of images in DB, 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 cloud or grid. For example, the iris database used in the Computer Vision Research Lab at the University of Notre Dame contains over 100,000 biometrics, and performing all-pairs computation using this database on a single machine takes several days.

To help the users with the task of running such large scale experiments, [Bui et al. 2009] developed a level of abstraction and corresponding end-to-end computing system, called BXGrid, which eliminates the need for users to be effective at configuring and using grid computing systems, relational databases, distributed filesystems, etc. With BXGrid, the computations can be run using four simple abstractions:

- Select(R): select a set of images and metadata from the repository based on requirements R (such as subject gender, location, etc.).
- Transform(S, F): apply transformation function F (such as feature extraction) to each member of a given set S.
- AllPairs(S, F): compare all members of given set S using function F, producing matrix M, where each element $M[i][j] = F(S[i], S[j])$.
- Analyze(M, C): extract statistical (or other quality metric) data from matrix M and store the result in C.

In this work, we extend this framework by integrating security protection in the above computations 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 DB, |DB|, but computation exceeding this linear bound (e.g., quadratic complexity of AllPairs) is beyond the client's capabilities. For that reason, we concentrate on the computation corresponding to AllPairs and Analyze functionalities. Furthermore, because secure comparison of biometric templates has been a subject of prior research (see, e.g., [Erkin et al. 2009; Barni et al. 2010; Blanton and Gasti 2011]), most of this work is dedicated to techniques for computation verification. Our goal is to decouple the verification techniques from the

mechanism for securing the computation, so that any suitable solution for protecting data privacy can be employed. The implementation details for several of our solutions, however, assume that the underlying arithmetic is carried out over a finite group, which makes them suitable for use with data protected via secret sharing or homomorphic encryption. They can also be used with (garbled) Boolean circuits, but the circuits will need to simulate computation in $\mathbb{Z}_q$ for some q. To illustrate how all computation considered in this work can be carried out privately in an outsourced scenario, we describe secure protocols that can be derived from prior literature. For that purpose, we utilize unconditionally secure multi-party computation techniques based on secret sharing to secure the computation and use such techniques in the implementation.

Our contributions. We design a mechanism for verification of outsourced AllPairs computation and provide its rigorous analysis, which allows the client to set the security parameters as to achieve the desired probability of misbehavior detection (Section 3). We also design a mechanism for verifying the distance distribution computation Analyze and likewise provide its rigorous analysis. Our full analysis is for the Hamming distance (Section 4), and consecutive modifications to the strategy and the analysis address the Euclidean distance (Section 5) and the set intersection cardinality (Section 6). Finally, we combine solutions for AllPairs and Analyze functionalities to result in a complete solution for the overall process for each distance metric and report on implementation results for the Hamming distance. The computation is assumed to be carried out on protected data, and we illustrate a way of achieving privacy-preserving outsourcing for all functionalities used in this work.

The basic idea of our constructions – that of inserting pre-computed sub-tasks and verifying them upon task completion – is not new and has been used for verifiable remote computation (e.g., [Golle and Mironov 2001; Szajda et al. 2003]) and storage (e.g., [Ateniese et al. 2007; Juels and Kaliski 2007]). It, however, has never been applied to important classes of computation such as biometric comparisons and statistical computation, which is the subject of this work and lead us to interesting and non-trivial results. That is, despite the simplicity of the high-level idea, our design leads to complex new solutions and novel analysis for verifying not only distance computation, but also probability distributions for a number of standard distance metrics. Furthermore, the ideas used in our solutions for the Analyze functionality are new. Our analysis shows that our verification techniques are suitable for this application and result in reasonable overhead. Our analyses provide valuable insights for development of computation verification techniques for outsourcing computation to servers whom might be incentivized to skip a portion of the computation. To the best of our knowledge, this work is unique in that it assumes a more complex structure of the computation than a large set of parallel, homogeneous, and indivisible tasks (such as one-way function inversion) considered in prior work, and provides a mechanism for exactly determining the values to which security parameters must be set. This work also doesn't simply consider a specific problem in which verification can naturally be performed faster than the computation itself (e.g., matrix multiplication). For that reason, we expect that our techniques will provide value beyond the types of computation treated in this work.

Preliminary version of this work appeared in [Blanton et al. 2011b] as a short paper. Compared to this work, [Blanton et al. 2011b] provided only basic solutions for verification of distribution computation Analyze for fewer distance metrics and without any analysis or evaluation. Such analysis, however, is a major contribution of this work. The full version of this work is also available as a technical report [Blanton et al. 2011a].

2. PROBLEM DESCRIPTION

2.1. 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, the matrix M is partitioned

between multiple computational servers such that each server receives a computational task which can be performed within its memory and computational capacity constraints. Then a server receives 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 $n \times n$ distance matrix as the output. For example, if the client has 80,000 biometric templates to be used in AllPairs computation and sets $n = 1000$, the overall computation will consist of 6,400 individual tasks, which the client sends to a large number of servers and the servers carry out their respective tasks in parallel. When we refer to the distance matrix that each server produces, we will assume that the items from the first set correspond to the rows of the matrix and the items from the second set correspond to the columns.

Each 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. Furthermore, because all computation takes place on protected data, the server will not be able to distinguish the data that the client injects from the original data.

Also, because biometric comparisons amount to computing the distance between two biometric templates which consist of multiple elements, we will assume that each biometric item consists of m elements and each distance is in the range $[0, \sigma]$, where σ is a function of m and depends on the employed distance metric. For instance, $\sigma = m$ for both the Hamming distance and the set intersection cardinality distance metric. In the context of a specific distance metric, each biometric item is either an (ordered) vector or (unordered) set. In the former case, we will use vectors and coordinates to refer to the biometric templates and their components, and in the latter case we will use sets and elements. When the distance metric is not implied by the context, we use generic terms (biometric) items and their 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 its n^2 computed distances (the client can easily merge the data returned by multiple servers in this form by simply adding the counts from the servers for each given distance value). This information fully defines the distribution and will allow the client to compute any necessary statistics from it. To compute the count for each distance, the server will obliviously compare a distance it computed for a given cell to all possible distance values and increment one of them that matched (without knowing which count was incremented). We will 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 $[0, \sigma]$ of distances information about which is being collected. Let $C = \langle c_0, \ldots, c_{v-1} \rangle$, where v could be equal to $\sigma + 1$ or another value depending on the verification method. Then the pseudo-code below shows how C is being updated for each cell after computing the corresponding distance. Below, notation $[x]$ denotes that the value of x is not known by the server, the result of the equality testing operation ($x \stackrel{?}{=} y$) 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 $i = 0, \ldots, v-1$: $\quad [b_i] := ([d] \stackrel{?}{=} [d_i])$; $\;[c_i] := [c_i] + [b_i]$;

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 $[0, v-1]$.

We assume that computation on protected data takes place in a group (or a field) and arithmetic operations are therefore the elementary operations in that representation. Note that this assumption holds for multiple ways of computing on protected data (e.g., using homomorphic encryption or secret sharing).

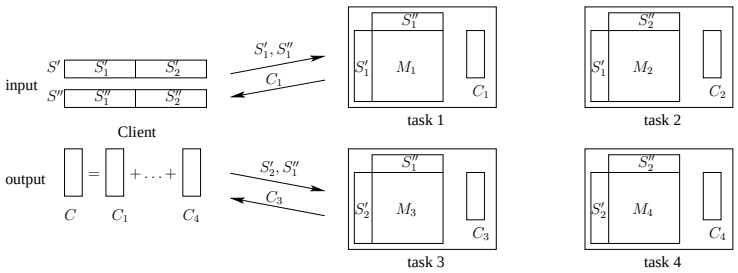


Fig. 1. Overview of computation (without the verification process).

Figure 1 illustrates the setting on a small example that corresponds to combined AllPairs and Analyze functionalities. In the figure, the client partitions its input to form four tasks, and sends each task to a single server. Each server computes an $n \times n$ portion of matrix M, M_i, as well as distribution information C_i corresponding to M_i and returns the result to the client. The client adds values in the C_i's to obtain overall distribution C.

While the abstraction of biometric computation that we use in this paper, namely, AllPairs and Analyze, was initially developed for grid computing, it can be easily adapted to a common cloud computation paradigm such as MapReduce. To implement the AllPairs and Analyze functionalities, the client takes the input datasets, divides them into smaller subproblems, and distributes them to the "map" nodes who perform AllPairs computation. Upon accomplishing the computation, each "map" node sends the distance matrix to a specified "reduce" node (determined by the partition function), which further extracts the statistical data to form a partial distribution corresponding to its inputs, and sends it to the client who combines it with other partial distributions in a meaningful way. Furthermore, if the client would like to perform only AllPairs computation, he himself needs to carry out the reduce functionality, which is straightforward regardless of the underlying infrastructure and implementation of secure computation. [Okcan and Riedewald 2011] also study optimal algorithms for implementing all-pairs functionality using MapReduce and [Afrati et al. 2012] study the trade-offs in implementing all-pairs similarity computation using MapReduce.

2.2. Security model

As mentioned above, our goal is to achieve secure and verifiable computation outsourcing. This means that we wish to (i) protect privacy of the data being outsourced and (ii) verify that the computation was carried out as prescribed. Throughout this work, we use the term "verification" to refer to computation verification (as opposed to biometric verification). Because our focus is on the second goal of computation verification for the types of computations described above, throughout most of this paper we assume that the computation can be carried out in a secure manner. For completeness, we, however, show how the types of computation used in this work can be carried out in a secure manner by the servers.

We therefore obtain that the server performing the computation should be 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 $0 \leq p \leq 1$, and attempts to manipulate the result so as to make the client believe that it computed its task as prescribed. The client's goal, then, is to devise a verification mechanism which detects the server's misbehavior even when the portion of skipped computation, $1-p$, is small. More formally, if the server deviates from the standard semi-honest model in which it follows the prescribed computation, its work is bounded by fraction p of the assigned task. By placing a computational bound on adversary's malicious actions, our goal is to avoid expensive techniques such as zero-knowledge proofs of knowledge [Goldwasser et al. 1985], verifiable secret sharing [Chor

et al. 1985], and fully homomorphic encryption [Gennaro et al. 2010] often used in presence of fully malicious adversaries.

Let D denote the event that the client detects server's cheating (and $\overline{\mathsf{D}}$ the event that the client does not detect it). Then the client's goal is to achieve $\Pr[\mathsf{D}] \geq 1 - \frac{1}{2^\kappa}$ for the desired security parameter κ whenever the fraction of work p that the server performed is below the client-chosen threshold. The data privacy guarantees, however, are required to hold for any value of p.

It is assumed that the server knows the verification procedure and knows (or can sufficiently 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. We also assume that the server might know certain properties of biometric data and use them to its advantage, but while designing verification mechanisms for the client we conservatively do not assume some specific property of biometrics will hold to ensure that the solution can be applied to any type of data.

The above description treated a single task that the client outsources to a server in isolation, while for this application the client is expected to submit many tasks to multiple servers. We then require that exposure to previous tasks (comprising of the same or different computation and on the same or different data) give a server no advantage in passing verification of the current task, even if the server can influence and adaptively ask for specific tasks. In other words, all tasks are independent. Note that a server can perform a different fraction of work p in each task, but $\Pr[\mathsf{D}] \geq 1 - \frac{1}{2^\kappa}$ is required to hold any time p is below the client-specified threshold.

On the security definition. With the above formulation, the client will detect, with the desired probability, instances when the server performs (at most) p fraction of its task. Because a task consists of many sub-tasks, this fraction of work can, for instance, correspond to pn^2 computed distances out of assigned n^2. An alternative formulation of the problem is to suggest that each task is computed with probability p, where the probabilities for each sub-task are independent. (This can be meaningful in the context of data or task corruption, where each sub-task has a small probability of being corrupt.) In that case, the overall number of performed computations is not known, but is characterized by a distribution. We later show that the two definitions lead to almost identical analysis when the task size n is large. For concreteness we base our analysis on the first formulation of server misbehavior.

In prior and our work that uses probabilistic checking (both for computation and storage outsourcing), in order for the client to perform work independent of the overall size of the outsourced job/data, corruption of a constant portion of the job/data (e.g., 1%) can be detected with high probability. If the client wishes to detect smaller corruptions, its verification work grows above constant. In case of storage outsourcing, small errors can also be compensated by using error-correction codes at the expense of extra storage. Such techniques, however, are not applicable to computation outsourcing.

2.3. Server instantiation

Depending on how the secure computation is realized, a single task might be carried out by one physical machine (e.g., on encrypted data) or be performed by a number of machines and take the form of secure multi-party computation. To abstract from the implementation details, in both cases we say that the *server performs a task*, and it should be understood that the functionality is carried out by one or more physical machines. When each task is carried out by a number of physical machines, it is expected that the client chooses machines from different service providers in order to achieve proper data protection (e.g., that not all machines participating in a task may collude with each other). In other words, each individual task in Figure 1 is run by machines located at different providers. This makes collusion unlikely, as competing service providers have to actively conspire.

When the task is carried out by a single physical machine in today's cloud computing environments, our security model implies that the client will be able to detect the cases when the machine did not perform all of the computation, when a fraction of the computation was corrupt (e.g., by means of a malicious VM running on the same hardware), or when a fraction of the input or computed data was lost or corrupted (due to malware, service outages, etc.). When a task is executed in the form of an interactive protocol by multiple machines some of which are honest, the client will be able to detect data and computation corruption on any number of participating machines due to similar causes. Furthermore, the client will be able to detect cases when a subset of participating machines skip a portion of or deviate from their computation, while preserving correct communication patterns.

2.4. Achieving privacy of distance and statistics computation

Three distance metrics are treated in this work: the Hamming distance (used for iris), Euclidean distance (used for faces and biometric types), and set intersection cardinality (used for fingerprints). Secure outsourcing of iris code comparisons to one or multiple servers has been recently addressed [Blanton and Aliasgari 2012]. 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 multiple servers setting. Based on that, we also provide a protocol for the Euclidean distance computation. Lastly, [Blanton and Aguiar 2012] developed currently the only multi-party implementation of secure set intersection and its cardinality suitable for use in an outsourced environment. We show how the solution can be adopted to our work. All protocols for secure distance and statistics computation are given in Appendix A.

3. VERIFICATION OF DISTANCE COMPUTATION

The main idea behind verifying AllPairs computation is to insert a number of fake, random items at random positions when forming a computational task $\langle S_1, S_2, \mathsf{dist}\rangle$ for a server. The fake items need to be chosen only once for all tasks and the distances between them are precomputed. Their protected representation, however, is randomized for each new task to ensure that the server cannot identify them. Upon job completion, the client receives and reconstructs the results, and then compares the distances between the fake items computed by the server with the values that it expects. Because the server does not know a priori which values will be checked, it will have to compute the distances honestly.[1] For that reason, to form set S_1, the client uses $n-n_1$ real items and n_1 fake items at random positions to create S_1. Similarly, S_2 consists of $n-n_2$ real items and n_2 fake items at random positions. Here n_1 and n_2 are parameters which will be set to obtain $\Pr[\mathsf{D}] \geq 1-\frac{1}{2^\kappa}$. The exact approaches for assigning values to the fake items and verifying the result can differ. We present two solutions and compare their performance. Both of them result in insertion of n_1 items in S_1 and n_2 items in S_2, but in the first case the client's verification is $O(n_1 n_2)$, while in the second case it is $O(n_1+n_2)$ with a lower probability of cheating detection.

[1]If matrix M is returned to the client, the client can retrieve and reconstruct the necessary values. Most commonly, however, the client does not receive the entire matrix, but rather queries selected cells of interest. In the original BXGrid system, the client might retrieve and manually examine a small number of cells that correspond to biometrics with certain features or otherwise biometrics chosen by the client or might not receive any matrix values at all if the AllPairs computation is followed by the Analyze computation. In our case, the client will need to retrieve at least the cells that it needs for computation verification. Thus, when the entire matrix M is not returned to the client, the server sends a commitment to all computed values in M using, e.g., the Merkle hash tree approach of [Du et al. 2004]. The client then challenges the server on certain cells of the matrix and the server must show that correct distances were included in the commitment.

3.1. First approach

The solution is as follows: The client generates n_1 fake items, the set of which is denoted by F_1, and n_2 fake items, the set of which is denoted by F_2, such that the client knows or computes the pairwise distances between each $\hat{x} \in F_1$ and $\hat{y} \in F_2$. The $\hat{x}$'s ($\hat{y}$'s) are placed at random locations in S_1 (resp., S_2). After the server performs the computation, the client verifies all $n_1 n_2$ distances between each $\hat{x}$ and $\hat{y}$. We note that the client must use fake items because any $n_1 n_2$ distances between original biometrics in the task are not guaranteed to have a certain amount of uncertainty each. Fortunately, the client can reuse the same sets F_1 and F_2 (after proper re-randomization of their representation) for all tasks.

Per our assumption, a lazy server honestly computes fraction p of n^2 distances, and creates the distances for the rest using any desired strategy of significantly lower work (guessing, copying computed distances, etc.). If the server computed all $n_1 n_2$ distances checked by the client correctly, its misbehavior will not be detected.

In what follows, we analyze the probability of server's misbehavior to be detected for a fixed value of p. Let $n_1 \ll n$ and $n_2 \ll n$ (which holds under our assumption that n is large). The manner in which the server computes the pn^2 distances can affect the probability of its cheating being detected. In order to determine what server's strategy leads to the minimal probability of misbehavior detection for a given p, we analyze three server's strategies:

(1) *The server computes the distances in the entire matrix row or column.* That is, if the server computes a distance in any given row, it will compute all other distances in that row. Thus, the number of computed rows is pn. Suppose for the sake of the current analysis that the server's cheating is detected with probability 1 if it skipped the computation of a cell that the client checks. Then the probability that the server's behavior is undetected corresponds to the probability that all n_1 fake items from F_1 fell into the pn computed rows:

$$\Pr[\mathsf{D}] = (1-p) + p\frac{(1-p)n}{n-1} + p\frac{pn}{n-1}\frac{(1-p)n}{n-2} + \cdots + p\frac{pn}{n-1}\cdots\frac{pn}{n-n_1+2}\frac{(1-p)n}{n-n_1+1}$$

The above probability follows hyper-geometric distribution, where elements are drawn from a set without replacement. Using the properties of hyper-geometric distribution, the above can be rewritten as:

$$\Pr[\mathsf{D}] = 1 - \binom{np}{n_1} / \binom{n}{n_1} = 1 - \frac{(np)!(n-n_1)!}{(np-n_1)!n!}. \quad (1)$$

Given that $n_1 \ll n$ and $n_2 \ll n$, we can instead use binomial distribution for computing $\Pr[\mathsf{D}]$ which assumes that elements are drawn with replacement. We then obtain:

$$\Pr[\mathsf{D}] = (1-p) + p(1-p) + p^2(1-p) + \cdots + p^{n_1-1}(1-p) = 1 - p^{n_1} = 1 - \Pr[\overline{\mathsf{D}}]. \quad (2)$$

While equation 1 more accurately determines the probability, for the purposes of this work it is sufficient to consider binomial distribution as a close approximation that simplifies the analysis and we thus use the expression in equation 2.

Next, we modify the analysis to take into account the fact that the server can avoid detection by successfully guessing the correct distance for a cell that the client checks. Recall that the computed distances lie in the range $[0, \sigma]$. Because the client has full control over the fake items that it injects in the computation, it will want to minimize the probability of the server guessing any distance it checks. For that reason, the client ideally would like to make the distances distributed uniformly at random (to have the probability of guessing a distance be $1/(\sigma+1)$), but unfortunately this is not possible with the way the distances are created and checked. That is, the client assigns $n_1 + n_2$ values, and due to triangle inequality $\mathsf{dist}(A, B) \leq \mathsf{dist}(A, C) + \mathsf{dist}(C, B)$ all possible combinations of $n_1 n_2$ distances could not be achieved. Therefore, for the current discussion we denote the

maximum probability with which the server can guess any given distance by $0 < \alpha < 1$, and later instantiate it with a specific value.
Going back to our analysis, if the server did not compute the cells in a particular row, it must guess all n_2 checked distances in that row in order to avoid being detected. The server can do so with probability α^{n_2}. This means that the probability that the server is not detected after checking the cells in a single row is $p + (1-p)\alpha^{n_2}$, and the probability that the server is detected is $1 - (p + (1-p)\alpha^{n_2}) = (1-\alpha^{n_2})(1-p)$. This tells us that the probability that the server's cheating is detected after checking n_1 rows is when the detection was successful at least for one row:

$$\Pr[\mathsf{D}] = 1 - (p + (1-p)\alpha^{n_2})^{n_1} \tag{3}$$

Similarly, if instead of the rows, the server computes all distances in a number of matrix columns, the probability of detection becomes:

$$\Pr[\mathsf{D}] = 1 - (p + (1-p)\alpha^{n_1})^{n_2} \tag{4}$$

The above tells us that to maximize $\Pr[\mathsf{D}]$, it is to the client's advantage to set $n_1 = n_2$. This is due to the fact $\Pr[\mathsf{D}]$ will be close to 1 in equation 3 only when n_1 is significant, but in equation 4 the value of n_2 needs to be maximized for the same reason.

(2) *The server computes partial rows and columns.* This strategy generalizes the first one, and the server now computes cells corresponding to $p_r n$ partial rows and $p_c n$ partial columns in a consistent way, where $p = p_r p_c$, and a cell is computed if both its row and column are among the computed $p_r n$ and $p_c n$, respectively. With this server's strategy, the client will not be able to detect cheating if all n_1 fake items fall within the $p_r n$ computed rows and all n_2 fake items fall within the $p_c n$ computed columns or if the server did not compute all checked distances, but guesses their value correctly.
For any given row out of n_1, the server's behavior is not detected if either (i) the row was among partially computed rows and the server either computed n_2 checked cells in that row or guessed their value correctly and (ii) the row was not among partially computed rows and the server guessed all n_2 checked cells in that row. The probability of (ii) is $(1-p_r)\alpha^{n_2}$. The probability of (i) consist of the multiplicative term p_r and the probability that all n_2 cells were among computed or guessed. The latter can be expressed in terms of the probability that a single cell is among computed or guessed, which is $p_c + \alpha(1-p_c)$, and the probability that all of them were computed or guessed is therefore $(p_c + \alpha(1-p_c))^{n_2}$. Thus, the probability that checking a single row does not result in detection is $p_r\left(p_c + \alpha\left(1-p_c\right)\right)^{n_2}$ and the overall formula for client's success after checking all n_1 rows is:

$$\Pr[\mathsf{D}] = 1 - \left(p_r\left(p_c + \alpha\left(1-p_c\right)\right)^{n_2} + \alpha^{n_2}\left(1-p_r\right)\right)^{n_1}. \tag{5}$$

As before, we use binomial distribution to approximate the probabilities. We can also reverse the role of rows and columns to obtain an expression similar to the above. Furthermore, since based on the previous analysis we know that the client sets $n_1 = n_2$, the above equation can be simplified to be a function of n_1 only.

(3) *The server computes distances at random cells.* The server chooses pn^2 matrix cells at random and computes their values. The server's behavior is undetected when all $n_1 n_2$ checked by the client cells fall within the computed pn^2 cells or the server guesses correctly the checked distances that it did not compute. In this case the probability that the cheating is undetected after checking a single cell is $p + (1-p)\alpha$. As before, the cheating is detected when it is detected in a least one of the checks and we obtain:

$$\Pr[\mathsf{D}] = 1 - (p + (1-p)\alpha)^{n_1 n_2}. \tag{6}$$

In Appendix B we show that, of all above strategies and their combinations, under certain parameters strategy 1 minimizes the probability of the server's misbehavior to be detected, while in other cases it is strategy 3. Such a strategy will thus be chosen by the server.

3.2. Second approach

Now instead of verifying all $n_1 n_2$ cells corresponding to the distances between a pair of fake items, the client will verify $\max(n_1, n_2)$ cells that cover all n_1 rows and all n_2 columns. The solution itself is as follows: For a task of square size $n \times n$ let $n_1 = n_2$. The client generates n_1 fakes items F_1 and $n_2 (= n_1)$ fake items F_2 such that the distance between each ith item $\hat{x}$ from F_1 and ith item $\hat{y}$ from F_2 is chosen uniformly from the range $[0, \sigma]$ and then items $\hat{x}, \hat{y}$ are created with that distance[2]. The $\hat{x}$'s ($\hat{y}$'s) are placed at random locations in S_1 (resp., S_2). To verify the result, the client extracts the distances corresponding to ith items from F_1 and F_2 for each i from the matrix returned by the server and compares the result to the values it generated.

As before, we analyze three server's strategies assuming that elements are drawn from a set with replacement which holds when $n_1 = n_2 \ll n$. The probability that the server's misbehavior is not detected after verifying a single cell is:

The server computes the distances in the entire row or column	$p + (1-p)\frac{1}{\sigma+1}$
The server computes partial rows and columns	$p_r p_c + (1 - p_r p_c)\frac{1}{\sigma+1}$
The server computes distances at random cells	$p + (1-p)\frac{1}{\sigma+1}$

In each case the probability is computed as the server either computed the corresponding distance or did not compute it, but guessed it correctly. This gives us that in any of the above cases the detection probability after checking n_1 cells, at distinct rows and columns unpredictable to the server, is:

$$\Pr[\mathsf{D}] = 1 - \left(\frac{p\sigma + 1}{\sigma + 1}\right)^{n_1}. \tag{7}$$

The same applies to the case when role of columns and rows is reversed since $n_1 = n_2$. The above makes it easy for setting the value of n_1. In particular, if we set $n_1 \geq \frac{\kappa(\sigma+1)}{(1-p)\sigma}$, then the adversary will be undetected with probability

$$\Pr[\overline{\mathsf{D}}] = \left(\frac{p\sigma + 1}{\sigma + 1}\right)^{n_1} = \left(1 - \frac{(1-p)\sigma}{\sigma + 1}\right)^{n_1} \leq e^{-\frac{n_1(1-p)\sigma}{\sigma+1}} = e^{-\kappa}.$$

3.3. Discussion

Before we proceed with further description, we would like to compare the two proposed solutions with respect to their probability of detecting server's misbehavior and the overhead they incur on the client. Let $n_1 = n_2$ according to our prior analysis, and recall that in the first solution the client performs $O(n_1^2)$ work but might achieve better detection probability than using the second solution with $O(n_1)$ client's work. Higher detection probability in turn leads to the ability to set n_1 to a lower value which decreases the server's overhead.

Let us consider the first server's cheating strategy discussed in sections 3.1 and 3.2, i.e., computing entire rows or columns. The equations for the two proposed solutions are given in equations 3 (or 4) and 7, respectively. Because equation 3 uses parameter α, we need to show that it is possible to produce fake items so that α can be guaranteed to have a reasonably lower value such as $1/2$ for any pair of fake items.

[2]We assume that any distance in the range $[0, \sigma]$ can be generated, which is true for all of the distance metrics considered in this work.

Let $m = 2n_1$ and $n_1 \leq n_2$.[3]
(1) Set ith bit of the ith item in F_1 to 1 and all other bits to 0.
(2) To set an item $\hat{y}$ in F_2, choose $m/2$ random bits $j_1, \ldots, j_{m/2}$. Let $s = \sum_{k=1}^{m/2} j_k$ and let $\hat{y}'$ be the $m/2$-bit string that starts with $m/2 - s$ 1s and then has s 0s. The vector $\hat{y}$ is then set to $j_1||j_2|| \ldots ||j_{m/2}||\hat{y}'$, where "||" denotes concatenation. A key property of this assignment is that there is always $m/2$ 1s in any $\hat{y}$.
(3) When computing $\mathsf{dist}(\hat{x}, \hat{y})$ for each $\hat{x} \in F_1$ and $\hat{y} \in F_2$, there are two cases: (i) $j_i = 0$ in $\hat{y}$ and (ii) $j_i = 1$ in $\hat{y}$, where i is the position of the only bit set to 1 in $\hat{x}$. In case of (i), the ith bit of $\hat{y}$ is 0 and there are m other locations in $\hat{y}$ set to 1, which gives us $\mathsf{dist}(\hat{x}, \hat{y}) = m/2 + 1$. In case of (ii), $\hat{x}$ and $\hat{y}$ match at the ith bit, but differ in $m - 1$ other locations, which gives us $\mathsf{dist}(\hat{x}, \hat{y}) = m/2 - 1$.

For example, if $m = 6$ and $n_1 = 3$, then $F_1 = \{100000, 010000, 001000\}$. The eight possible values for $\hat{y}$ are $\{000111, 001110, 010110, 011100, 100110, 101100, 110100, 111000\}$. If we let $\hat{y} = 110100$, then $\mathsf{dist}(100000, 110100) = 2$, $\mathsf{dist}(010000, 110100) = 2$, and $\mathsf{dist}(001000, 110100) = 4$.
It is important to note that $\mathsf{dist}(\hat{x}, \hat{y})$ depends only on the value of the bit j_i, and thus all distances are independent from each other. Hence, the probability that an adversary could guess L distances computed in this way is $\leq \frac{1}{2^L}$.

Fig. 2. Example: Generating fake items for Hamming distance.

Specifically, the problem consists of generating item sets F_1 and F_2, such that $\mathsf{dist}(\hat{x}, \hat{y})$ for each $\hat{x} \in F_1$ and $\hat{y} \in F_2$ has as much uncertainty as possible. Ideally, we would want to choose all $n_1 \times n_2$ distances independently from $[0, \sigma]$ and then compute fake items sets that produce these distances. However, this is usually not possible, because the distances must satisfy the triangle inequality. Furthermore, even if a chosen set of distances satisfies the triangle inequality, it is non-trivial to generate the fake items that produce them. We present a solution for achieving $\alpha \leq 1/2$ on the example of the Hamming distance in Figure 2. A similar approach can be used for other distance metrics as well.

We obtain that $\alpha \leq 1/2$ is achievable and as a result α^{n_2} can be sufficiently close to 0. This will make equation 3 of the first solution equal to $1 - (p + \epsilon_1)^{n_1}$ for a small quantity $\epsilon_1 = (1 - p)\alpha^{n_2}$, while equation 7 of the second solution has $1 - (p + \epsilon_2)^{n_1}$, where $\epsilon_2 = (1 - p)/(\sigma + 1)$. While for common choices of parameters $\epsilon_1 < \epsilon_2$ and therefore the probability of detection is higher in the first solution, contrary to the expectations, this gives us that by checking $O((n_1)^2)$ values in the returned matrix, the client does not gain a noticeable advantage in detecting server's misbehavior compared to the solution where the client checks only $O(n_1)$ values. This is not what one would expect because in a more traditional analysis where an adversary guesses all values (instead of computing a significant portion of them honestly) this is not the case. In particular, with a traditional analysis the detection probabilities for similar strategies would be $1 - (1/(\sigma + 1))^{n_1}$ for checking n_1 values and $1 - (\alpha^{n_1})^{n_1}$ for checking $(n_1)^2$ values – a significant difference for any choice of n_1 that can provide reasonable security guarantees. We therefore conclude that, unlike other settings, the solution presented in section 3.1 would not provide a noticeable advantage over the solution of section 3.2, which, in addition to having a lower overhead for the client, is easier to analyze. (Note that we can make this conclusion based on the analysis of a single server's strategy since the server will follow the strategy with the lowest probability of detection.) In the rest of this work, we therefore use the solution of section 3.2.

Before we conclude this section, we note that the analysis presented so far is applicable to any type of tasks when pair-wise computations are performed $n \times n$ times. Biometric comparisons, however, consists of computing a distance over multiple elements of biometrics.

[3]That is, $m \geq 2\min\{n_1, n_2\}$ will be true for biometrics such as iris codes that use the Hamming distance. For simplicity, we also let $m = 2n_1$ with $n_1 \leq n_2$, which means that the extra bits are ignored, but their use can result in a better scheme.

Thus, the server might attempt to reduce its workload by computing partial distances using a subset of the elements and then adjust the results in the attempt to avoid detection. We next show that our solution (from section 3.2) also effectively combats this server's strategy. Suppose that the server computes p fraction of each distance for each cell of the matrix. That is, it computes the distance corresponding to pm elements of its choice and needs to guess the remaining portion of the distance. Because the distances that the client checks are distributed uniformly at random in the range $[0, \sigma]$ and with each distance metric considered in this work one element of a biometric can contribute distance σ/m, the distances corresponding to the $(1-p)m$ elements that the server did not compute in the checked cells will be distributed uniformly at random in the range $[0, (1-p)\sigma]$. This means that the probability that the server's behavior is not detected after checking a single cell is $\frac{1}{(1-p)\sigma+1}$ and the detection probability after checking all n_1 cells is $\Pr[\mathsf{D}] = 1-\left(\frac{1}{(1-p)\sigma+1}\right)^{n_1}$. This quantity is at least as large as the value computed in equation 7 for any $p \leq 1$ and σ (and is strictly larger when $p < 1$), and therefore is not attractive for the adversary.

In addition, if the server combines this strategy with one of the three possibilities listed above, it still will not be able to reduce the probability of detection below the value listed in equation 7. For instance, if the server computes partial distances in $p'n$ rows using $p''m$ elements, such that $p' \cdot p'' = p$, the probability that misbehavior is not detected after checking a single cell is when the server either computed the partial distance and guessed the remaining portion or did not compute the distance and guessed it entirely: $p'\frac{1}{(1-p'')\sigma+1} + (1-p')\frac{1}{\sigma+1}$. It is clear that the success probability of this hybrid strategy lies between the computed probabilities for the individual misbehavior strategies and becomes equal to the probabilities of one of the individual strategies when $p' = 1$ or $p'' = 1$ (while maintaining $p' \cdot p'' = p$). Therefore, equation 7 provides the lower bound for successful cheating detection for given p, σ, and n_1.

4. VERIFICATION OF STATISTICS COMPUTATION FOR HAMMING DISTANCE

We are now interested in providing a verifiable solution to the Analyze functionality. To accomplish that, the client employs a different method for verifying that the distribution computation was performed correctly. In the following 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 provided in the next sections.

4.1. First attempt

As before, the server receives a job of size $n \times n$. The client inserts fake items in the computation to aid the verification process, but it also inserts an additional fake element into both real and fake items. The goal of the additional element is to separate the ranges of distances between a pair of real items and distances between real and fake items. More precisely, the distances between two real items remain in the range $[0, \sigma]$, while distances between a real and fake items now fall into the range $[\sigma+1, 2\sigma+1]$. The server's job consists of compiling distribution information $C = \langle c_0, \ldots, c_{2\sigma+1} \rangle$ by computing the distance between two biometrics and comparing it to $2\sigma + 2$ possible distances in C in such a way that the count for the distance that matched is obliviously incremented. This allows the client to verify correctness of the computation. That is, the client adds the counts corresponding to the distances in the range $[0, \sigma]$ and the counts corresponding to the distances in the range $[\sigma + 1, 2\sigma + 1]$. and compares the aggregate counts to their expected values (i.e., $(n-n_1)(n-n_2)$ and $(n-n_1)n_2+n_1(n-n_2)$, resp.). If both match, the server's computation is considered correct.

We show how the above can be accomplished for the Hamming distance, as it will be useful for building our final solution. Each biometric is represented as an m-bit vector, and $\mathsf{dist}(x, y) = \mathsf{dist}(\langle x_1, \ldots, x_m \rangle, \langle y_1, \ldots, y_m \rangle) = \sum_{i=1}^{m} x_i \oplus y_i \in [0, m]$. The client inserts

coordinate x_{m+1} into each vector x, where $x_{m+1} = 0$ if x is a real biometric and $x_{m+1} = m+1$ otherwise (the remaining coordinates of a fake biometric are chosen as bits according to any desired distribution). Note that we are somewhat abusing the computation because $y_{m+1} = m+1$ is not a bit, but the server will be unable to distinguish the two types of values because their protected values are represented in the same way.[4] This gives us distances in $[0, m]$ for two real items and in $[m+1, 2m+1]$ for real and fake items. Also, if during the computation the XOR is applied directly to a pair of coordinates, the distance between two fake vectors will lie in the range $[0, m]$. When, however, arithmetic operations are used to implement $a \oplus b$ as $a + b - 2ab$, the distance between two fake items will be in $[1 - 4m^2, 1 - m^2]$. We can setup secure computation over non-negative integers in such a way that this range has no overlap with $[0, 2m + 1]$.

Now if the server computes pn^2 distances and would like to avoid being detected, instead of trying to guess the locations of fake items, it can simply return $2m+2$ counts, such that c_0 through c_m add to $(n-n_1)(n-n_2)$ and c_{m+1} through c_{2m+1} add to $(n-n_1)n_2+n_1(n-n_2)$. Because the server knows (or can guess sufficiently well) the values of n_1 and n_2, it can always be successful in avoiding the detection. This means that a different solution is needed.

4.2. Improved solution

To improve the security properties of the above solution, we employ two ideas: (i) (protected) distances used for computing statistics are given to the server in a 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. To combat certain attacks, the client also inserts a number of fake elements in each item instead of only one.

The idea is as follows: before the computation is sent to the server, the client adds extra k elements to each real item (with m original elements), and creates fake items consisting of $m + k$ elements. The $m + k$ elements are randomly permuted, but consistently across all items. The client chooses a small integer ℓ, which will be used as a security parameter and also chooses ℓ values larger than the maximum distance σ. Each value will be used to increase the distance between certain fake items and real items, and for concreteness, and without loss of generality, let these ℓ values be $\sigma + 1, \ldots, \sigma + \ell$. The client forms real and fake items in such a way that the distance between two real items is in the range $[0, \sigma]$, the distance between any given fake item and a real item is in the range $[d, d + \sigma]$ for a fixed $d \in [\sigma + 1, \sigma + \ell]$, and the distances between real items and all fake items fall in the range $[\sigma+1, 2\sigma+\ell]$. For each distance $d \in [\sigma+1, 2\sigma+\ell]$, the client compiles statistics and records the expected counts. At the time of computation verification the client will then need to compare the computed counts for distances in $[\sigma + 1, 2\sigma + \ell]$ to their expected values and compare the aggregate count for distances in $[0, \sigma]$ to $(n - n_1)(n - n_2)$.

In the context of Hamming distance, this solution is realized as follows. The client sets all extra coordinates to 0 in real biometric vectors. Let $i_1, \ldots, i_k$ denote positions of these extra coordinates. To create a fake vector, the client chooses a distance d at random from $[m + 1, m + \ell]$ and k values $d_1, \ldots, d_k$ such that $\sum_{i=1}^{k} d_i = d$. The client sets the i_jth coordinate in the fake vector to d_j for $j = 1, \ldots, k$ and all original coordinates to 0. This gives us that the distance between this fake vector and any real vector will be in the range $[d, d+m]$. The client records the number of times each d was used in a fake vector in the set S_1 and the set S_2, respectively (both formed in the above described way). Let the counts be denoted by c_i^j, where $i \in [m + 1, m + \ell]$ and $j \in [1, 2]$.

[4]This holds when the secure computation is realized using homomorphic encryption or secret sharing techniques. If the client, however, wishes to employ two-party techniques based on garbled Boolean circuits, the size of coordinate representation will have to be increased to make values of different size indistinguishable to the computational servers.

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 corresponding to 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. 3. Description of client's computation of the expected statistics.

Before the client is able to verify the results of server's computation, the client needs to compute additional information as follows: For each real vector in S_1, the client computes the number of coordinates set to 1 in that biometric, i.e., its Hamming weight, and counts the number of instances of each Hamming weight across all real vectors. Let $s_0^1, \ldots, s_m^1$ denote the distribution of the Hamming weights, where s_i^1 indicates the number of real vectors in S_1 with the Hamming weight of i. Similarly, the client produces the same distribution information for the real vector in S_2, and we denote the computed values by $s_0^2, \ldots, s_m^2$. The client computes the expected distribution of the distances above m as follows: for each $i = m+1, \ldots, 2m+\ell$, set $c_i = \sum_{j=\max(m+1, i-m)}^{\min(m+\ell, i)} (c_j^1 s_{i-j}^2 + c_j^2 s_{i-j}^1)$. The above represents the number of instances contributed by the distances between fake vectors in S_1 and real vectors in S_2 and fake vectors in S_2 and real vectors 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.

After the server receives sets S_1 and S_2, it computes the distances between all pairs and produces statistics by comparing each computed distance to the values in the range $[0, 2m + \ell]$. Since the comparisons are performed without the server knowing to what value a distance is being compared, the client randomizes the order of the distances. This means that the server will not be able to know what positions within the set of $2m + \ell$ values correspond to original distances from 0 to m and which correspond to distances above m (this is applicable to both the distances used during comparisons and computed counts in C). After the server returns the distribution data to the client, the client (i) adds the counts corresponding to distances between 0 and m and compares the result to $(n - n_1)(n - n_2)$ and (ii) compares each c_i with the value returned by the server. If all above checks succeed, the client treats the obtained statistics data as correct.

We generalize the above procedure (which also will be used for other distance metrics) in Figure 3. We use notation $C[i]$ to denote the count returned by the servers for distance i. For the current solution, we have $[l_r, l_u] = [0, m]$, $[l_f, u_f] = [m + 1, 2m + \ell]$, and $d_o = m$. Note that the last parameter d_o indicates that the original coordinates of real vectors can contribute distances between 0 and m to the distances between real and fake vectors.

In order for the distances between two fake vectors 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 $[0, 2m + \ell]$ and suggest the following: Because we assume that computation takes place in a group, the client can choose d_i's uniformly at random from $\mathbb{Z}_q$, where q is the group size, 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 vectors will fall outside of the range $[0, 2m + \ell]$, while other distances remain unaffected. 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 without having to increase the server's load.

4.3. Analysis of misbehavior detection

We next analyze the server's success in avoiding being detected when it performs the fraction p of its task. We treat two options when (i) the server computes all distances between real and fake vectors correctly (by guessing the locations of fake vectors) and increases the count(s) for distances between real and real values, and (ii) the server does not compute all distances between real and fake vectors and instead increases the counts for both distances between real and real and distances between real and fake vectors.

Correct computation of statistics for distances between real and fake vectors. The server's goal is to identify locations of vectors that are fake (both for columns and rows) and compute their distance to real vectors, as the client distinguishes between distances between real-real and real-fake vectors. Suppose that, to maximize the coverage of fake vectors, the server selects a number of columns and a number of rows and computes a cell if its row or column (or both) is among the selected rows or columns, respectively, and the total number of cells computed is pn^2. While the server needs to compute only $n - n_1$ (or $n - n_2$) cells in each chosen column (row) instead of all n, the server does not know which locations to skip as it is trying to guess their positions. Therefore, it is to the server's advantage to compute entire rows and columns. We have that $pn^2 = z_1 n + z_2(n - z_1)$, where z_1 is the number of chosen rows, and z_2 is the number of chosen columns. As before, let $n_1 = n_2$ and also let $z_1 = z_2$ since this is the best server's strategy when $n_1 = n_2$. Now, given the value of p, we can compute $z_1 = z_2$ and then compute the probability that all n_1 rows/columns will appear among the computed z_1.

Let $p' = z_1/n = 1 - \sqrt{1-p}$. The probability that the rows that the server computes include all n_1 inserted rows is $1 - (p')^{n_1}$ and the probability that the columns that the server computes include all n_1 inserted columns is also $1 - (p')^{n_1}$. Furthermore, the server's behavior is not detected only when it computed all distances between real and fake vectors and also guessed the location of a cell corresponding to a value between 0 and m among the counts in C; the success probability of the latter is $(m+1)/(2m+1+\ell)$. We obtain:

$$\Pr[\mathsf{D}] = 1 - \frac{m+1}{2m+1+\ell}(p')^{n_1+n_2} = 1 - \frac{m+1}{2m+1+\ell}\left(1-\sqrt{1-p}\right)^{2n_1} \tag{8}$$

Now notice that the server might also be able to compute correct distances between the vectors using only a fraction of coordinates during the computation of each distance. In particular, recall that each distance is computed using $m+k$ coordinates, where m of them correspond to the original data in real vectors, and the server can compute n^2 partial distances using $p(m+k)$ coordinates. When computing a distance between two real vectors, using any number of coordinates will result in the outcome falling into the correct range $[0, m]$. For computing distances between real and fake values, however, notice that all artificial k coordinates must be used in order to pass the client's check. That is, all k artificial coordinates should fall within $p(m+k)$ coordinates used by server. Because k can be comparable to m, we need to assume that the coordinates are drawn without replacement, and using the properties of hyper-geometric distribution we obtain the probability of including all k coordinates in $p(m+k)$ coordinates:

$$\Pr[\overline{\mathsf{D}}] = \binom{m}{p(m+k)-k} \Big/ \binom{m+k}{p(m+k)} = \prod_{i=0}^{k-1} \frac{p(m+k)-i}{m+k-i} \tag{9}$$

Second, in order to pass the client's verification, the server needs to obtain the exact distances between real and fake vectors, which means that it also needs to have the coordinates

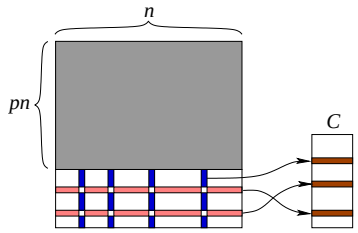


Fig. 4. Illustration of incomplete computation while creating statistical data.

set to 1 to be within the computed coordinates. Assuming that at least $\beta \cdot m$ of the original coordinates are set to 1 in each biometric for some fixed β and the coordinates are randomly (and consistently) permuted to eliminate any patterns, the above probability decreases to:

$$\Pr[\overline{\mathsf{D}}] = \binom{m-\beta m}{p(m+k)-\beta m-k} \Big/ \binom{m+k}{p(m+k)} = \prod_{i=0}^{\beta m+k-1} \frac{p(m+k)-i}{m+k-i} \tag{10}$$

The above assumes that $p(m+k) \geq \beta m + k$; otherwise, $\Pr[\overline{\mathsf{D}}] = 0$. We also note that while equation 8 tells us the minimum value to which n_1 should be set for a desired bound on p and desired probability of detecting cheating, equations 9 and 10 tell us the minimum value to which the security parameter k should be set for a given p and $\Pr[\mathsf{D}]$.

Incomplete computation of statistics for distances between real and fake vectors. Now suppose that the server does not try to guess the locations of fake vectors, but rather attempts to adjust statistics information to avoid detection. In this case, the server might have higher chances of being undetected and its success depends on the values of m and ℓ, as well as on the distribution of distances between real and fake vectors.

Because there is a structure to the distances in the matrix, we first examine the case when the server fully computes distances and statistics information for a number pn of rows and modifies the statistics information to compensate for the remaining distances. First, note that if the server computes pn rows correctly, it will need to guess (i) the number of fake rows among the remaining rows (horizontal red matrix rows in Figure 4), (ii) the locations of the counts in C associated with distances between each of not computed fake vector in S_1 and real vectors in S_2 (i.e., locations in C associated with red cells in Figure 4), (iii) the locations of the counts in C associated with the distances between n_2 fake vectors in S_2 and not computed real vectors in S_1 (locations in C associated with vertical blue columns in Figure 4), and (iv) at least one location in C corresponding to a distance between 0 and m (for increasing the count corresponding to white matrix cells in Figure 4).

Next, because the server might be able to estimate (i) with good accuracy and the probability of guessing (iv) is high, we have to rely on (ii) and/or (iii) being difficult. Suppose that the server guessed (i) correctly, and its value is $\hat{w} = (1-p)n_1$. A naive strategy for the server would be to guess one location in C for a distance in the range $[0, m]$ and increment the corresponding count by $((1-p)n - \hat{w})(n - n_2)$ (thus covering all white cells) and then guess the locations in C corresponding to all red and blue cells correctly and update the counts. We, however, notice that the server can reduce the amount of information that needs to be guessed by reusing information associated with some cells. For instance, the server can take one of the computed rows (which with high probability corresponds to a real vector in S_1) and copy it $(1-p)n - \hat{w}$ times into not computed rows (or just replacing the value of $[b]$ with $((1-p)n - \hat{w} + 1)[b]$ when updating the counts for that row). This would remove the need for the server to guess the locations in C associated with the blue cells in Figure 4. Alternatively, the server can compute the distances in the first cell of each row (which with high probability correspond to a real vector in S_2) and copy the dis-

tance contained in that cell to all distances in the row. This will remove the need to guess locations of the distances contained in the red cells, but all locations associated with the blue cells need to be guessed, as well as at least one count associated with a distance in the range $[0, m]$ needs to be decremented to compensate for incorrectly updated blue cells distances. The previous, row copying, strategy is superior for the server in that it minimizes the number of values to be guessed. In particular, for each fake vector x in S_1 or S_2, the distance between that vector and a real vector y in S_2 or S_1, respectively, will be equal to $s_x + s_y$, where s_x is the sum of the coordinates of x and s_y is the sum of the coordinates in y. In the worst-case for the client, all original biometrics have the same Hamming weight, which means that the distances between a single fake coordinate and all real coordinates will have the same value. This also means that all distances corresponding to the cells in a single blue column or a single red row will be stored in a single location in C. And, in order to update statistics information about a row or column that has not been computed, the server will need to guess a single location in C. Therefore, our goal is to set the security parameters in such a way that, regardless of what strategy the server employs, it will have to guess enough locations in C so that its probability of success is sufficiently low.

Because the server's success is maximized when it has to guess only the locations in C associated with the distances in red rows, we perform the analysis based on this case. In what follows, let γ be a confidence security parameter, which will guide the values of other security parameters to ensure that the client's security guarantees hold with probability at least $1 - \gamma$. Let us look at a hyper-geometric experiment that models the number of fake (red) rows falling within the not computed region of $(1 - p)n$ rows. If we denote a random variable that follows this distribution by X and its outcome by w, we obtain: $\Pr[X = w] = \binom{(1-p)n}{w}\binom{pn}{n_1-w}/\binom{n}{n_1}$, where $\hat{w} = (1 - p)n_1$ is the mean value. Because the client will want the outcome of X to be above a certain threshold, we obtain:

$$\Pr[X \geq w] = \sum_{i=w}^{n_1} \binom{(1-p)n}{i}\binom{pn}{n_1 - i}/\binom{n}{n_1} \geq 1 - \gamma \tag{11}$$

Then given the value of γ and w, the client will be able to compute the necessary value of parameter n_1 from equation 11. To determine the value of w, we first need to compute how many distinct locations in C the server must guess to achieve the desired probability of cheating detection $\Pr[\mathsf{D}]$, after which we will use that value, s, to compute w.

The server can guess s out of $2m + \ell + 1$ locations in C correctly with probability:

$$\Pr[\overline{\mathsf{D}}] = \frac{s}{2m + 1 + \ell} \cdot \frac{s - 1}{2m + \ell} \cdots \frac{1}{2m + 1 + \ell - (s - 1)} = 1/\binom{2m + 1 + \ell}{s}. \tag{12}$$

Therefore, given $\Pr[\overline{\mathsf{D}}] = 1 - \Pr[\mathsf{D}]$, the client computes s and will need to set $\ell \geq s$, so that the expression in equation 12 evaluates to a sufficiently low value which does not exceed the desired probability $\Pr[\overline{\mathsf{D}}]$. Next, we determine how many fake vectors from S_1 (red rows) need to fall within the not computed region to result in s distinct distance values (where a row is conservatively assumed to contribute one distance) with the probability of success that the client wants to achieve. That is, we express the probability that w (fake) vectors, each with a randomly chosen distance d in the range $[1, \ell]$, will result in s out of ℓ distinct values. By increasing the value of w, this probability increases, and the client will choose the minimum w that gives probability at least $1 - \gamma$.

In what follows, let Y denote a random variable associated with an experiment of throwing w balls into ℓ bins and follow the distributions of the number of bins with at least one ball in them. First notice that when $w = s$, each ball must land in a new bin:

$$\Pr[Y \geq s] = 1 \cdot \frac{\ell - 1}{\ell} \cdot \frac{\ell - 2}{\ell} \cdots \frac{\ell - s + 1}{\ell} = \frac{\prod_{i=1}^{s-1}(\ell - i)}{\ell^{s-1}}. \tag{13}$$

When $w = s + 1$, the first s balls can still land in unique bins, but now one ball can also land in a previously occupied bin. We therefore obtain:

$$\Pr[Y \geq s] = 1 \cdot \frac{\ell-1}{\ell} \cdot \frac{\ell-2}{\ell} \cdots \frac{\ell-s+1}{\ell} \cdot 1 + 1 \cdot \frac{1}{\ell} \cdot \frac{\ell-1}{\ell} \cdots \frac{\ell-s+1}{\ell} + \cdots + \quad (14)$$
$$+1 \cdot \frac{\ell-1}{\ell} \cdots \frac{\ell-s+2}{\ell} \cdot \frac{s-1}{\ell} \cdot \frac{\ell-s+1}{\ell} = \frac{\prod_{i=1}^{s-1}(\ell-i)}{\ell^{s-1}} \left(1 + \sum_{i=1}^{s-1} \frac{i}{\ell}\right)$$

Similarly, for $w = s + 2$, we have:

$$\Pr[Y \geq s] = \frac{\prod_{i=1}^{s-1}(\ell-i)}{\ell^{s-1}} \left(1 + \frac{1}{\ell} \sum_{i=1}^{s-1} i + \frac{1}{\ell^2} \sum_{i=1}^{s-1} \sum_{j=1}^{s-1} (i \cdot j)\right). \quad (15)$$

Therefore, by increasing the value of w, the client will find a value that satisfies $\Pr[Y \geq s] \geq 1-\gamma$ and will further use that value of w to compute the value of n_1 using equation 11. Note that by increasing the value of parameter ℓ, the value of n_1 will be reduced.

Now let us analyze the case when the server computes partial distances using $p(m + k)$ coordinates for all cells and then the corresponding distribution using the computed distances. We next show that if in doing so the server misses at least one artificial coordinate, its probability in avoiding detection will be low (and can be controlled by appropriately setting the parameter ℓ). The client should set the value of k as to obtain

$$1 - \binom{m}{(1-p)(m+k)} \Big/ \binom{m+k}{(1-p)(m+k)} = 1 - \frac{m!(p(m+k))!}{(m+k)!(p(m+k)-k)!} \geq 1 - \gamma. \quad (16)$$

This expression corresponds to the probability that at least one of the k coordinates will fall within the skipped $(1-p)(m+k)$ coordinates. It allows the client to compute the security parameter k for the desired value of γ.

Now suppose that at least one out of k artificial coordinate was not used during distance computation. This means that the computed distances between two real vectors will be in the correct range and will pass verification. Similarly, the distances between two fake vectors with high probability will fall outside of the checked range, as intended. The distances between real and fake vectors, however, with high probability will also fall outside of the checked range and the counts corresponding to their true values will need to be updated.[5] If the server knows the Hamming weight distribution in a biometric and sufficiently well estimates the number of fake vectors with the sum of its coordinates equal to any given d, the server will be able to update the counts correctly if it can guess the location of the appropriate counts in C. This is, however, difficult because a large number of locations needs to be guessed. Furthermore, for each unique count difference, the exact locations in C must be determined. In the worst-case for the client, the server needs to update all locations with the same information, in which case the server has to guess $u + \ell - 1$ locations (out of $2m + \ell + 1$) in C, where $u = \max_{x \in S_1 \cup S_2}(||x||) - \min_{x \in S_1 \cup S_2}(||x||) + 1$ corresponds to the difference between the smallest and the largest Hamming weight in a biometric. Because in the best-case for the server, it does not need to distinguish between the $u + \ell - 1$ locations, the probability of detecting server's misbehavior is:

$$\Pr[\mathsf{D}] \geq 1 - \prod_{i=0}^{u+\ell-2} \frac{u+\ell-1-i}{2m+\ell+1-i} = 1 - \prod_{i=1}^{u+\ell-1} \frac{i}{2m-u+2+i}. \quad (17)$$

[5]Note that if some partial distances between fake and real vectors fall within the checked range, the server will have to additionally guess the locations of the counts in which erroneous distances fell and decrease the counts. We therefore conservatively assume that computed distances between fake and real vectors fall outside the checked range, and the client's detection probability will be at least as large as what we compute based on our analysis.

If for a particular biometric representation u is small, then the value of ℓ can be increased to achieve the desired Pr[D] using equation 17.

To summarize, to detecting cheating with the desired probability, the client computes a set of parameters as follows: (i) n_1 using equation 8, (ii) k using equation 10, (iii) n_1 and ℓ using equation 11 and the consecutive analysis (equations 12 through 15), (iv) k using equation 16, and (v) ℓ using equation 17. For all of n_1, k, and ℓ the highest computed value should be used. The reason for computing multiple values of n_1, k, ℓ is to deter different ways of avoiding cheating detection by the server. In section 4.5 we also show how, based on our analysis, the values of n_1, ℓ, and k can be computed using sample client-supplied n, m, Pr[D], γ, and a bound on p.

4.4. Combining verification of distances and statistics

When the client needs to verify computation of both the distances and statistics, we next describe how it can combine the techniques of sections 3 and 4.2: The client adds fake vectors and artificial coordinates to both real and fake vectors as described in section 4.2 with the difference that, instead of setting the original coordinates of fake vectors to 0, they are set according to the description in section 3 (to maintain the uniform distribution of the distances in the cells checked by the client). Then to verify the computation of the distances (i.e., cells of the matrix), the client as before checks $n_1(= n_2)$ distances between pairs of fake vectors. The verification procedure now needs to take into account the offset introduced by the values assigned to the artificial coordinates. To verify computed statistics data, the client proceeds as before, with the exception that now to produce expected counts for the distances in the range $[m+1, 2m+\ell]$ the client needs to perform work dominated by $2n_1(n-n_1)m$ operations instead of work on the order of nm. As before, the distances between real and fake vectors fall into the range $[m+1, 2m+\ell]$ and the distances between two fake vectors fall outside of the range $[0, 2m+\ell]$ with high probability. The computation is summarized in Figure 5. Steps 1, 2, and 5 of task creation are performed offline once for all tasks. Step 1 of task verification can be performed during task execution.

To compute the security parameters n_1, k, and ℓ, we can use the previously established bounds. In particular, the client's success probability in detecting cheating in computing the distances is at least the probability in equation 7, which on input κ gives us the value of $n_1 = n_2$. To detect cheating in computing statistics with the desired probability, the client computes n_1, k, and ℓ as specified in section 4.3 and takes their maximum values.

4.5. Implementation

To evaluate the performance of our solution, we conducted experiments by implementing secure and verifiable AllPairs (section 3) and Analyze (section 4.2) functionalities for the Hamming distance. We first present the results for AllPairs computation outsourcing.

Recall that the client first partitions its job into individual tasks of manageable size $n \times n$. To securely outsource an individual task, the client sends shares of S_1 and S_2 to a number of computational servers N who jointly compute the result using secure multi-party computation (SMC) techniques based on Shamir secret sharing [Shamir 1979]. Due to simplicity of function $\mathsf{dist}(x, y) = \sum_{i=1}^{m}(x_i + y_i - 2x_i y_i)$, and properties of this type of SMC techniques, each server can compute its share of $\mathsf{dist}(x, y)$ locally and return the result to the client. In particular, with (N, t)-linear secret sharing, there are N participants, each of which receives a share of a secret. Then any $t+1$ shares can be used to reconstruct it, while combining t or less shares information-theoretically reveals no information about the secret. For parties that follow the computation, it is required that $t < N/2$. Each secret s is represented by a random polynomial $f_s(x)$ of degree t, where $f_s(0) = s$. Party P_i, for $i = 1, \ldots, N$, obtains its share $f_s(i)$. With this representation, any linear combination of secret shared values is carried out locally with no communication. Multiplication of two secret shared values, on the other hand, is interactive and involves multiplying two shares

Given n, m, p, κ, and γ, compute $n_1 = n_2$, k, and ℓ.
To create an outsourced task:

(1) Create n_1 pairs of m-dimensional fake vectors $\langle \hat{x}, \hat{y} \rangle$ such that within each pair $\mathsf{dist}(\hat{x}, \hat{y})$ is a random value from $[0, m]$. Let F_1 and F_2 denote the (ordered) sets of all $\hat{x}$'s and $\hat{y}$'s, resp.
(2) Append k coordinates to each fake vector, where the coordinate values are chosen as follows: for each fake vector, choose d at random from $[m+1, m+\ell]$ and set the values of the artificial coordinates to randomly chosen d_j's such that $\sum_{j=1}^{k} d_j = d$. Compute $\mathsf{dist}(x, y)$ for each $x \in F_1$ and $y \in F_2$; if some value falls in the range $[0, 2m+\ell]$, choose new d_j's for x or y. Store n_1 values $\mathsf{dist}(\hat{x}, \hat{y}) = \mathsf{dist}(F_1[i], F_2[i])$ for each i. Permute the coordinates in each vector using a random permutation π_0.
(3) Given two sets S_1' and S_2' of size $n - n_1$ comprised of m-dimensional biometrics, append k artificial coordinates to each vector, which are set to 0. Permute the coordinates in each vector using π_0.
(4) Form S_1 of size n by appending the elements of F_1 to the set S_1' and permuting the result using random permutation π_1. Similarly, form S_2 using S_2' and F_2 and permuting the resulting set using random permutation π_2. Save S_1 and S_2.
(5) Create (protected) distances $[i]$, $i = 0, \ldots, 2m+\ell$, and permute them using random permutation π_3.
(6) Send S_1, S_2, and $\pi_3([0]), \ldots, \pi_3([2m+\ell])$ to a server for secure AllPairs and Analyze computation, where the distances $\pi_3([0]), \ldots, \pi_3([2m+\ell])$ will be used to compute C.

To verify an outsourced task:

(1) Compute expected C as follows:
  (a) for $i = m+1, \ldots, 2m+\ell$ set $C[i] = 0$ and also set $C[0\text{–}m] = (n - n_1)^2$.
  (b) for $i = 1, \ldots, n_1$, compute $w_1 = \mathsf{dist}(F_1[i], z)$ and increment $C[w_1]$ by 1 for each $z \in S_1'$; compute $w_2 = \mathsf{dist}(z, F_2[i])$ and increment $C[w_2]$ by 1 for each $z \in S_2'$.
(2) For each $i = 1, \ldots, n_1$ compare stored $\mathsf{dist}(F_1[i], F_2[i])$ to the returned $M[\pi_1(F_1[i])][\pi_2(F_2[i])]$ and output failure if at least one value disagrees.
(3) Add returned counts $C[\pi_3^{-1}(0)]$ through $C[\pi_3^{-1}(m)]$ and compare them to computed $C[0\text{–}m]$. For each $i = m+1, \ldots, 2m+\ell$ compare returned $C[\pi_3^{-1}(i)]$ to the expected $C[i]$. If at least one check fails, output failure.

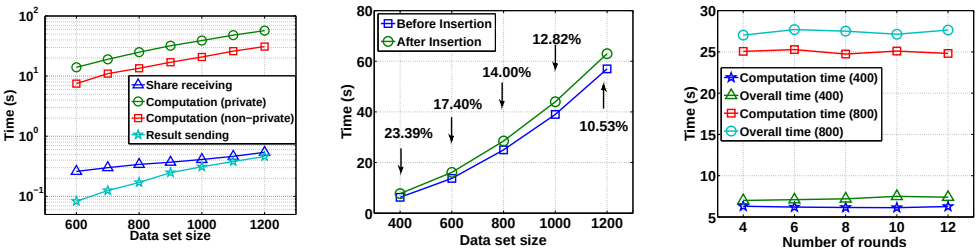
Fig. 5. Description of preparation and verification procedures for an outsourced task.

locally, after which they are randomized and re-shared. This temporarily raises the degree of the polynomial representation to $2t$, after which it is reduced back to t, and this is the reason for $2t < N$ requirement [Ben-Or et al. 1988; Gennaro et al. 1998]. It is important to notice that a (possibly multi-variate) polynomial of degree k can also be evaluated locally, as long as $kt < N$. This fact is exploited in our implementation, where computation of $\mathsf{dist}(x, y)$ is represented as a polynomial of degree 2 over variables x_i, y_i for $i = 1, \ldots, m$ and $t < N/2$. This means that the parties compute shares of $\mathsf{dist}(x, y)$ without any interaction (and the result is represented by a polynomial of degree $2t$), and the client uses $N = 2t+1$ shares it receives from them to reconstruct the result. Appendix A describes how this and other relevant computation can be carried out in this framework.

Our implementation used Java-based SEPIA library [Burkhart et al. 2010] for the underlying communication and elementary operations on secret shares with one client and three servers for a singe task. In particular, SEPIA handled establishment of SSL connections between the client and the servers and between each pair of servers. We used $m = 1000$ with a varying number n of vectors per task and arithmetic modulo a 15-bit prime. The value of m was chosen to be comparable with the length of iris codes in practice.[6] The value of n was chosen so that $n^2 m$ shares would fit the machines' memory for performance reasons. The client and each server were 2.4 GHz Linux machines with 12GB of memory on a 1Gbps LAN. We used (3, 1)-Shamir secret sharing with $N = 3$.

Figure 6 reports the results of our experiments. Figure 6(a) measures the time to transmit $2nm$ shares to a single party, the computation taken by a single party to compute its shares of n^2 distances, and the time for a party to send all n^2 shares back to the client. For com-

[6]For the purposes of the analysis in this work, the exact value of m plays insignificant role assuming that it is large enough. This will become clear from the discussion below.



(a) task computation and communication time

(b) server's task computation time

(c) server's performance for a large task

Fig. 6. Performance of secure and verifiable outsourcing of Hamming distance computations.

parison, we also plot the time taken to execute the same task without privacy protection in a similar environment on shorter representations of the coordinates and implementing XOR operations directly. (Note that in secure execution, a single task is run by N machines, while in insecure computation only a single server is needed.) The plot shows notably efficient performance for secure function evaluation techniques. Also, while our LAN experiment represents a best-case scenario for communication delay, for this computation communication is not expected to be a dominating factor even for significantly slower networks.

Figure 6(b) reports on the servers' overhead caused by the addition of fake vectors to make the computation verifiable. The curves show the time to compute tasks of size n and $n + n_1$ for fixed $n_1 = 50$ and variable n. We compute n_1 according to equation 7. For a sample setup of $\Pr[\mathsf{D}] \geq 0.99$ when $p \leq 0.95$, we obtain $n_1 = 90$, and for $\Pr[\mathsf{D}] \geq 0.95$ with $p \leq 0.9$ we obtain $n_1 = 29$; we then choose $n_1 = 50$ as a medium value. As the figure indicates, the overhead is only 10–20% for the plotted data set sizes. Furthermore, because the overhead consists of $2nn_1 + n_1^2$ distance computations, it is clear that it will constitute a smaller fraction of the task as the value of n increases.

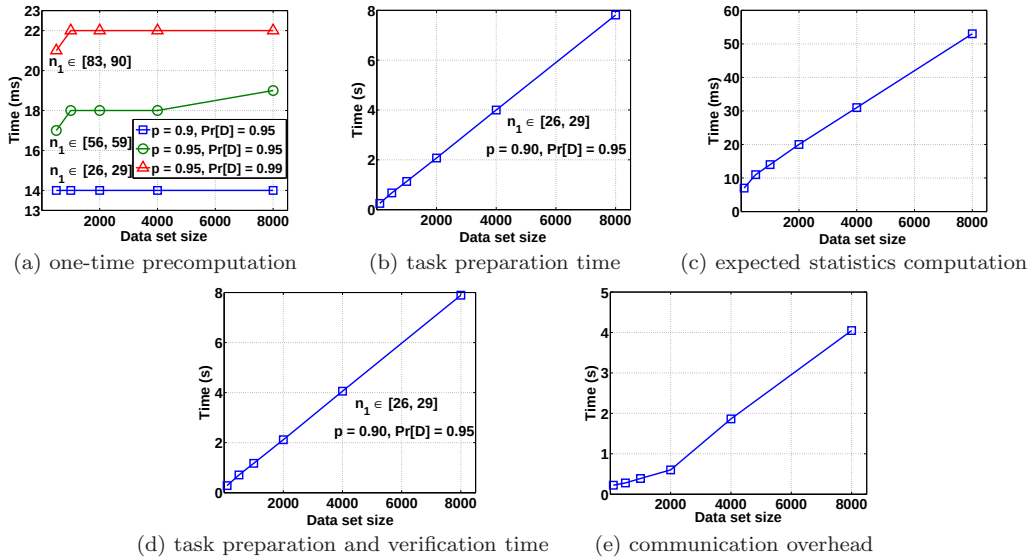
Finally, Figure 6(c) reports on the time for securely computing a large task. The plot shows average time per sub-task when the overall task is computed using sub-tasks of size $n = 400$ and $n = 800$. It is clear that the time per sub-task is constant. The overall time is slightly higher than the computation time and includes overhead such as key establishment and also depends on the worst out of N (vs. average) communication and computation. From all of the plots, we see that the techniques are efficient and do not substantially exceed the computation time without security protection or correctness verification.

To evaluate the performance of secure and verifiable Analyze, we start with computing all necessary parameters. As summarized in section 4.3, we need to compute parameters n_1, k, and ℓ using $m = 1000$, variable n, and desired $\Pr[\mathsf{D}]$, γ, and p. First, notice that computation of k (parts (ii) and (iv)) is independent of either n_1 or ℓ. Furthermore, equation 16 will result in strictly higher value of k than equation 10 even for very conservative values of β. Therefore, it is sufficient to consider only equation 16 and compute k as a function of p, m, and γ. We also note that the value of m has a low impact on k (i.e., similar to computing the number of fake vectors n_1, approximation by binomial distribution can be used to compute the number of artificial coordinates k when m is large, in which case k is independent of m). We give the value of k and other parameters for three distinct settings of security parameters $\Pr[\mathsf{D}]$, γ, and p in Table I.

Next, we consider parameters ℓ and n_1 computed in parts (i), (iii), and (v). First, notice that the value $\ell = 1$ satisfies all analyses with $m = 1000$. Namely, the probability in equation 17 is very high even with the lowest $u = 1$, the probability in equation 12 is very low with $\ell = s = 1$, and the probability in equation 13 is 1 with $\ell = s = t = 1$. This gives us $t = 1$ for the purposes of equation 11, and all that remains is to compute the higher value of n_1 using equations 8 and 11. Such values of n_1 are shown in Table I as a function of data

Table I. Values of parameters k, ℓ, and n_1 for verification of Hamming distance-based statistics with $m = 1000$.

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



(a) one-time precomputation (b) task preparation time (c) expected statistics computation

(d) task preparation and verification time (e) communication overhead

Fig. 7. Client's performance of secure and verifiable outsourcing of Hamming distance-based statistics computation and communication.

set size n. As can be seen from the table, n_1 increases slowly with n and approaches values independent of n that were computed for the purpose of AllPairs verification experiments. Observe that the values of k with $m = 1000$ are very similar to values of n_1 with $n = 1000$.

The results of our experiments for Analyze functionality are given in Figures 7 and 8, where the former reports on the client's computation and communication overhead and the latter shows the server's performance. For the client, we measured all components necessary for preparation and verification of an individual task. Figure 7(a) measures the client's preparation time for generating the fake vectors to be inserted into a data set. This is a one-time cost for all possible tasks to be outsourced. We present the client's overhead for the three security settings from Table I. Note that the time is very low and grows slowly with the size of the task, as the value of n_1 increases before reaching the ceiling for large n.

Figure 7(b) reports on the client's time for preparing a task for outsourcing. It includes reading the input from a locally stored file and inserting fake vectors into random locations of the data set, where the former amounts for the majority of that time. The curve in the plot corresponds to the security setting with $p = 0.9$, $\Pr[\mathsf{D}] = 0.95$, and $\gamma = 0.95$ and therefore n_1 in the range $[26, 29]$ which increases with the value of n. It is clear that the time is linear in the data set size, and was the same in our experiments for the three security setting (i.e., the value of n_1 does not play a major role in the task preparation time).

Figure 7(c) reports on the client's overhead for computing the expected statistics for task verification. This can be carried out during task computation. Recall that to compute the expected statistics, the client needs to compute the Hamming weight of each (real) biometric in its data sets, which necessitates a single round of traversing of all of the vectors (which

are already in the memory after task preparation). Thus, the time in Figure 7(c) grows linearly with the size of the data set. Once again, we report the results for the first security setting in Table I and performance for other security settings is the same.

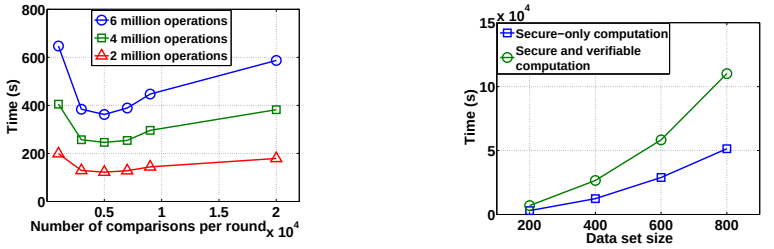
The next Figure 7(d) presents the client's overall time for preparing a task and verifying the returned result, which combines the times in Figures 7(a)–(c) with task verification time. The task verification time includes (i) reconstruction of the secret shares received from the servers and (ii) comparing the result with the expected statistics. Note that both (i) and (ii) depend only on the size of C and are independent of the data set size n. Therefore, we observed a constant task verification time around 15 msec. From the figures, we can conclude that the client's overhead is dominated by the task preparation time, which is linear in n and takes about a couple of seconds for data sets of a few thousands of vectors.

Finally, Figure 7(e) measures the time that communication between the client and the servers takes, which consists of sending shares of the data sets and receiving shares of the resulting C. Since the size of C is independent of n, the time for receiving shares of the result is also constant around 10–15 msec. Thus, the communication overhead is dominated by input transmission and therefore grows linearly with the data set size.

Unlike AllPairs computation, to produce distribution data C the servers need to engage in interactive computation. In the implementation, we compute the distances using the same approach as before, after which the servers reduce the degree of each distance representation from $2t$ to t and engage in comparison operations as described in section 2. Secure implementation of this functionality is also given in Appendix A. The fastest known realization of equality testing in this framework from [Catrina and de Hoogh 2010], but because we build our prototype using the SEPIA library, our implementation relies on the equality testing available in SEPIA. This means that faster implementations are possible today.

Performance of an interactive SMC protocol can be improved if the computation can be parallelized. In our context, all cells in the $n \times n$ matrix can be processed in parallel. In addition, all $|C| = 2m + \ell + 1$ comparison operations per cell in the matrix can also be carried out in parallel. We therefore utilize SEPIA's limited ability to carry out operations in parallel: operations are performed in rounds (which differ from the traditional SMC definition of elementary sequential interactive operations) and all computation and communication within a single round are synchronized. That is, each party waits until it receives all intermediate results necessary for the next step for all operations within the same synchronization round, and then proceeds with the computation for the next step. In other words, synchronization after each round is mandatory even if it is not required by the computation itself. We experimented with SEPIA to find out the number of operations per synchronization round that would minimize the overall runtime. The experiments were performed using a few million comparison operations, all of which could be carried out in parallel. Performing 10^7 comparisons is roughly equivalent to computing statistics C for $n = 100$ and $|C| = 1000$. The results of our experiments are given in Figure 8(a) for $N = 3$. As can be seen from the figure, the computation time is the lowest when 5000 comparison operations are used per synchronization round. We, however, would like to note that this result is specific to the SEPIA library and faster performance can be achieved by using tools with more flexible parallelization options.

The time that it takes the servers to carry out an outsourced task is shown in Figure 8(b). We compare two settings: secure computation of C and secure and verifiable computation of C. The former guarantees that all information is processed privately, and the latter additionally ensures that correctness of the computation can be verified by the client. We use the parameters for the first security setting with $p \leq 0.9$, $\Pr[\mathsf{D}] \geq 0.95$, and $\gamma \leq 0.05$ from Table I. The overhead introduced by the addition of fake coordinates to each vector is k/m for tasks of all sizes n. The overhead due to the addition of fake vectors is n_1/n, which decreases as n grows. Finally, expanding the size of C from $m+1$ to $2m+\ell+1$ doubles the work associated with processing each pair of vectors.



(a) performance with varying number of operations per synchronization round

(b) server's task execution time

Fig. 8. Server's performance of secure and verifiable outsourcing of statistics computation for Hamming distance.

5. VERIFICATION OF STATISTICS COMPUTATION FOR EUCLIDEAN DISTANCE

In this section, we use the Euclidean distance as the distance metric for computing distances between each pair of biometric items. Because the verification mechanism for distance computation in section 3 works for all distance metrics, we concentrate only on verification of statistics computation. This time, each biometric item x is represented as a vector $\langle x_1, x_2, \ldots x_m \rangle$, which is treated as a point in m-dimensional space and each coordinate $x_i \in [0, h]$ for $1 \leq i \leq m$. Unlike the Hamming distance computation which is carried out following the formula exactly, in this metric, we have the server compute the distribution of squared distance 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 then either will produce mapping between regular and squared distances as it forms a task assignment for the server (recall that the client supplies protected distances used to collect distribution information), take the square root of each returned result, or operate directly on squared distances.

5.1. Preliminary solution

Our first solution is very similar to that used for the Hamming distance. To aid the verification process, the client, as before, inserts fake vectors into the computation and inserts fake coordinates into both real and fake vectors. In detail, before outsourcing a task, the client inserts k fake coordinates into each real vector and the resulting $m+k$ coordinates are randomly permuted, but consistently across all vectors. As before, we denote the positions of these extra coordinates by $i_1, \ldots, i_k$. All artificial coordinates are set to 0 in real vectors.

To form fake vectors, the client chooses a small integer ℓ, which plays role of a security parameter, and ℓ values larger than mh^2 with each of them being used to increase the distance between real and fake vectors. For concreteness, we set these values to $mh^2 + 1$, $mh^2 + 2$, $\ldots$, $mh^2 + \ell$. To form a fake vector, the client first randomly chooses a distance d out of these ℓ candidates. Next, the client chooses k fake coordinates at random from $\mathbb{Z}_q$, denoted by d_i for $1 \leq i \leq k$, so that the constraint $\sum_{i=1}^{k} {d_i}^2 = d$ is satisfied. In more detail, the client chooses the first $k-1$ values of 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 the 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) coordinates in that vector to 0. To aid producing the expected statistics for computation verification purposes, the client records the number of times each d was used in a fake vector in the set S_1 and S_2, respectively. Let the counts be denoted by c_i^j, where $i \in [mh^2 + 1, mh^2 + \ell]$ and $j \in [1, 2]$.

By forming the vectors in S_1 and S_2 as described above, the distances between two real vectors will fall into the range $[0, mh^2]$, the distances between a real and a fake vectors will fall into the range $[mh^2 + 1, 2mh^2 + \ell]$, and the distances between two fake vectors

could be anywhere in $\mathbb{Z}_q$. Because the range of distances between two fake vectors might now overlap with the range of distances between two real (or real and fake) vectors, for verification purposes, the client needs to precompute the distribution of distances between two fake vectors that fall into the range $[0, 2mh^2 + \ell]$, and subtract it from the statistics returned by the server. Notice that the fake vectors could be reused for each task without lowering the detection probability, thus the overhead is one-time.

Before the client is able to verify the result of returned computation using S_1 and S_2, the client needs to compute additional information as follows: for each real vector x in S_1 and S_2, the client computes the sum of squares of its coordinates $\sum_{i=1}^{m} x_i^2$, and counts the number of instances of each occurred value across all vectors. Let s_i^1 (s_i^2) denote the number of vectors with computed value i in S_1 (resp., S_2). After acquiring this distribution, the client computes the expected statistics and verifies the results returned by the servers using the algorithm in Figure 3 with the following inputs:

$$[l_r, u_r] = [0, mh^2], \;\; [l_f, u_f] = [mh^2 + 1, 2mh^2 + \ell], \;\; d_o = mh^2$$

If all checks in the algorithm succeed, the client treats the obtained distribution as correct.

Next, notice that if we now compute the security parameters using the analyses in section 4.3, the necessary security guarantees will hold. (The only obvious exception is that we replace the total number of distances $2m + \ell + 1$, i.e., the size of C, with $2mh^2 + \ell + 1$ in equations 8, 12 and 17.) In particular, for given p, $\Pr[\mathsf{D}]$ and γ, the values of parameters n_1, ℓ, and k are determined from equations 11–17. There are, however, two major differences from the setting for the Hamming distance that influence the values of these parameters:

(1) The number of dimensions m in biometrics that rely on the Euclidean distance (such as faces) is often significantly lower, e.g., not exceeding 50.
(2) If the server misses at least a single fake vector in the computation (either from S_1 or S_2), it will have to correctly adjust several counts in C. Recall that in the case of the Hamming distance it was realistic to assume that all original biometric vectors might have the same Hamming weight (e.g., $m/2$), in which case all distances between a given fake vector and all real vectors from the other set would be the same. In the case of the Euclidean distance, however, it is not realistic to assume that all $\sum_{i=1}^{m} x_i^2$ for each x would result in exactly the same value.

A direct consequence of item 2 above is that $u > 1$ for the purposes of equation 17, where u is now the number of unique values $\sum_{i=1}^{m} x_i^2$ across all original x in S_1 and S_2. Furthermore, when the server needs to guess locations in C corresponding to s fake rows (red rows in Figure 4), the number of locations to guess now is at least $s + u$. This changes equation 12 to $\Pr[\overline{\mathsf{D}}] \leq 1/\binom{2mh^2+\ell+1}{s+u}$, which means that lower s and ℓ can be used for the Euclidean distance than the Hamming distance for comparable sizes of C. This gives us that $\ell = 1$ will be sufficient for the Euclidean distance as well, and we obtain that the client will be able to use the same values of n_1 and ℓ as given in section 4.5 and compute k from equation 16.

Also note that when item 1 does not hold, i.e., m is large, the analysis of this solution is closer to that of the Hamming distance and will result in lower security parameters (and thus lower overhead) compared to when m is not large.

5.2. Improved solution

Now notice that, while the above solution meets the security goals, it can be become inefficient due to the large size of C and therefore a large number of comparisons per pair of vectors. In particular, this happens when m and/or h are not small. Under such circumstances, the client might be interested in learning aggregate statistics, where the computed distances are rounded with the desired precision or, more generally, the distances are placed in specific ranges and the aggregate count for the entire range is reported instead of individual counts for each distance in the range. While the client can clearly compute this

information using the current solution, having the server compile the necessary information directly will result in significant reduction of its computational load and therefore the speed with which a task is performed. In what follows, we describe a modified solution that allows us to improve the computational load of the servers while still maintaining security.

When the client sends to the server a task in the form of S_1 and S_2, it now supplies a list of ranges $[l_i, u_i]$ for $0 \leq i < v$ which are not known to the server. The server's task becomes to compute the number of distances between the vectors in S_1 and S_2 that fall within each range. The sets S_1 and S_2 themselves are formed as previously described. Therefore, within the v different ranges, some ranges will correspond to the distances between real and real vectors, and others will corresponds to the distances between real and fake vectors.

To compute the count for each range, the server obliviously compares a computed distance to all possible ranges and increments the one that matched. As before, the counts are stored in $C = \langle c_0, c_1, \ldots, c_{v-1} \rangle$, where c_i corresponds to the number of distances in the range $[l_i, u_i]$. Initially, all c_i's are set to 0 and are updated as follows, where b_1 and b_2 are bits:

$[d] := \mathsf{dist}([x], [y])$;
for $i = 0, \ldots, v-1$: $[b_1] := ([l_i] \overset{?}{\leq} [d])$; $[b_2] := ([d] \overset{?}{\leq} [u_i])$; $[c_i] := [c_i] + [b_1][b_2]$;

By adjusting the granularity of the ranges, the client has a lot of flexibility to express its preference. That is, the ranges can aggregate a different number of distances, and the client can use fine granularity for the regions that convey a lot of information and coarse granularity for other regions.

Note that now processing each distance requires performing two comparisons for each range instead of a single equality test for each possible distance. This means that this solution results in computational savings for the server only when the number of ranges is less than one half of the number of original distances. Furthermore, because this approach can lead to accuracy reduction for the client, the client should choose a level of granularity that is guaranteed to preserve the utility of the data.

The above solution reduces both the server's work and the size of C. This in particular means that the security analysis must be revisited to ensure that the parameters provide the necessary guarantees. As described in section 5.1, however, despite smaller $|C|$ the same security parameters are sufficient for the Euclidean distance as for the Hamming distance. That is, because u (which now corresponds to the number of ranges in which values $\sum_{i=1}^{m} x_i^2$ fall for all original x) will still be larger than 1, the same security parameters should be sufficient, which the client will need to verify.

This setting also provides new opportunities for security enhancements. In particular, the client can use overlapping ranges, which can be valuable for setting distances between real and fake vectors. In this case, the need to update a single count by a cheating server translates into the need to update (i.e., correctly guess) multiple locations in C.

We combine verification of distance and statistics computation in Appendix C.1.

6. VERIFICATION OF STATISTICS COMPUTATION FOR SET INTERSECTION CARDINALITY

We next proceed with the description of our solution for statistics verification when the set intersection cardinality is used as the distance metric (and, as before, the solution from section 3 is used for verification of distance computation). Now each original item is a set composed of m elements[7] from the range $[0, h]$. Given S_1 and S_2, the server is to compute the cardinality of set intersection of 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. Note that this metric is

[7]Note that biometric representations that rely on the set intersection cardinality can have a variable length. For our purposes, each biometric is not required to be of length exactly m. Both correctness and security of our solution hold if each item is of length at most m, i.e., m is the upper bound on the size of items.

equivalent to symmetric set difference, which can be expressed in terms of set intersection as $\mathsf{dist}(x, y) = 2m - |x \cap y|$.

6.1. Preliminary solution

Similar to prior solutions, the client inserts n_1 fake biometric sets into both S_1 and S_2 and k fake elements into real and fake sets to aid the verification process. Note that unlike the previous distance metrics, the fake elements are not required to be positioned consistently across all sets. All artificial elements are set to 0 in the real sets.

To generate fake sets, the client produces $2n_1$ values larger than t and assigns one of them to a single fake set. We use d_i to denote the value assigned to the ith fake set. To form the ith fake set, 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 set is now a multiset, and we assume that the distance computation function will produce correct output when the inputs are multisets. The client also records the number of times each d was used in a fake set in S_1 and 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 sets fall into the range $[k, m+k]$ due to the use of k zero elements, the distances between real and fake sets into the range $[0, k)$, and the distances between any two fake sets into the range $[0, k)$. Because of the overlap of the last two ranges, the client will need to precompute the statistics for the distances between any two fake sets, add it to the expected statistics for the distances between real and fake sets, and then compare the result to the statistics returned by the server. The rest of the verification process uses the algorithm in Figure 3 with inputs:

$$[l_r, u_r] = [k, m+k], \;\; [l_f, u_f] = [0, k-1], \;\; d_o = 0$$

As far as security analysis goes, note that parameter k now serves the role of both k and ℓ in section 4.3, and the size of C is $m+k$. Also, the value of m is relatively small in biometric types that use this distance metric (e.g., fingerprints). Furthermore, the distances between a single fake set and all real sets are always the same depending merely on the value of d for that fake set, which means that equation 12 does not change. Then for given m and k, the quantity in equation 12 might not be sufficiently low when $s = 1$, which implies that higher s and therefore higher n_1 than what is reported in section 4.5 might be necessary.

With respect to the analysis for the value of k (equation 17), we have that when the server skips at least one artificial element, the verification associated with the distances between real sets will be successful, but the counts associated with the distances in the range $[0, k-1]$ will need to be updated by the server to pass verification. Skipping one artificial element will cause some, but not all, of the distances in that range to decrease depending on the value of the element at that position (this affects distances between real and fake sets as well as between two fake sets). This will result in at least two incorrect counts due to the distances between real and fake sets (the count for distance $k-1$ will need to be increased and the count for distance 0 will need to be decreased in the best-case for the server when all counts consistently shift down), the locations of which must be guessed correctly. This will also invalidate any number of counts from this range due to the distances between two fake sets, and we should expect the server to have to guess the locations and update at least half of them with the exact differences. Equation 17 then becomes $\Pr[\mathsf{D}] \geq 1 - \prod_{i=0}^{u-1} \frac{1}{m+k-i}$, where u can be set to $k/2$ or to a more conservative lower value.

6.2. Improved solution

As shown above, when the server misses distance computation for a single fake set (either a row or column), to remain undetected it only needs to correctly guess and update one location in C. When the size of C is not large, to guarantee that the probability of misbehavior

detection is sufficiently high, the parameters will have to be increased resulting in larger overheads. For that reason, we propose an improved solution that remedies this problem.

The modification to the current solution is as follows: the client defines a small integer ℓ as the security parameter which will correspond to the number of additional elements which in fake sets will be from the range $[0, h]$. The ith fake set now consists of three components: (i) d elements with value 0, where d is now from the range $[0, k - \ell - 1]$; (ii) ℓ elements with the values in the range $[0, h]$, which we term offset elements; and (iii) $m + k - \ell - d$ elements with values d_i greater than h. The purpose of these ℓ offset elements is to introduce differences in the distances between a single fake set and a number of real sets. In particular, the values of the ℓ elements in the fake sets will be such that they will overlap with certain elements in real sets, causing the distances between a single fake set and real sets to be in the range $[d, d + \ell]$ instead of always d. The values of the ℓ offset elements are set to 0 in all real sets. For simplicity, we will use the same set of ℓ offset elements for each fake set.

Notice that the above setup does not change the range of distances between any pair of real sets or the range of distances between a fake set and either fake or real set. The difference is that one fake set can now affect the counts corresponding to $\ell + 1$ distances in C. To guarantee that a single fake set indeed affects $\ell + 1$ locations in C, the values for the ℓ offset elements must be properly chosen. That is, in the worst case, the values selected for the ℓ offset elements might not share any elements with any real set in S_1 and S_2, resulting in no benefit from this approach. For that reason, when the client chooses candidate values for the offset elements, it should scan S_1 and S_2 to ensure that there are real sets that overlap with the candidate offset elements by everything between 0 and ℓ elements.

Before the client is able to verify the computations performed by the server using S_1 and S_2, the client needs to compute additional information as follows: for all real sets in S_1 and S_2, compute the number of sets that share i values in common with the ℓ offset elements. Let s_i^1 (s_i^2) denote this number in S_1 (resp., S_2) for $0 \leq i \leq \ell$. Now the client can produce the expected statistics by executing the algorithm in Figure 3 with the inputs:

$$[l_r, u_r] = [k, m + k], \;\; [l_f, u_f] = [1, k - 1], \;\; d_o = \ell$$

Finally, the client needs to incorporate the precomputed statistics for the distances between each pair of fake sets into the expected statistics, and compare it with the results returned by the server. If all checks succeed, the client treats the obtained result as correct.

With this modified solution, certain portions of the security analysis change and we obtain that now equation 12 becomes $\Pr[\overline{\mathsf{D}}] \leq 1/\binom{m+k}{s+\ell}$. This means that even by setting ℓ to a low value such as 1 or 2, having $s = 1$ will be sufficient to meet the necessary security guarantees even when m is not large. This will imply that the value of parameter n_1 will not increase over the values reported in section 4.5 for a range of security settings.

We describe the strategy for combined verification of distance and statistics computation in Appendix C.2.

7. RELATED WORK AND CONCLUSIONS

Research on verifiable or uncheatable computation was initiated in [Golle and Stubblebine 2001; Golle and Mironov 2001] using techniques such as redundant task execution and insertion of so-called ringers in search for rare events (in particular, performing inversion of a one-way function). In this context, it is crucial that the parties carrying out the computation are unable to distinguish ringers from other components of a task. Consequently, Szajda et al. [Szajda et al. 2003] extended the idea to optimization problems and sequential executions (while still relying on parallel and redundant task execution). Other publications in this direction include [Du and Goodrich 2005; Goodrich 2008; Karame et al. 2009] that inject chaff sub-tasks or verify portions of the result for computations of certain structure (e.g., NP-complete problems); [Kim et al. 2007; Watanabe et al. 2009] use redundant scheduling. [Du et al. 2004] 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 [Kuhn et al. 2008] distributed checking is used where (possibly malicious) servers perform checks on each other. We note that often the existing techniques can achieve a high probability of cheating detection only when p is rather low (i.e., not close to 1). While redundant task execution can be used to detect cheating by lazy servers, there is an extra computational overhead compared to our solution still remains. Also, one has to assume that one of the chosen servers always returns correct results (which is a stronger assumption than what we make).

There are also a number domain-specific computation verification techniques [Benjamin and Atallah 2008; Atallah and Frikken 2010; Wang et al. 2011]. Such techniques are known for algebraic computations [Benjamin and Atallah 2008; Atallah and Frikken 2010] and linear programming [Wang et al. 2011], where verification can be performed faster than the computation itself. This work is thus unique in its scope, as it assumes a certain structure of the computation, but the developed techniques are applicable to different instantiations of the distance function. In particular, the general solutions listed above would not work in the context of this work even for verifying AllPairs computation, as distance computation consists of many elements and should not be treated as an integral function.

Lastly, the line of work on integrity verification of remote storage that goes under the name of Proofs of Retrievability (POR) or Provable Data Possession (PDP) is related to this work ([Ateniese et al. 2007; Juels and Kaliski 2007] and others). At high level, a client partitions its data into data blocks and stores them together with meta-data at a remote server. Periodically, the client issues integrity verification queries and checks a number of data blocks at unpredictable to the server indices using the meta-data. The probability that the client can detect problems is computed as $1-((n-r)/n)^c$, where n is the total number of stored blocks, r is the number of corrupted blocks, and c is the number of verified blocks. We achieve similar guarantees for distance computation verification.

In this work we develop techniques for verifiable outsourcing of large-scale biometric computations on protected data consisting of distance and statistical data computation and provide their rigorous security analysis. Our techniques for the distance computation are general and can be applied to any distance metric, while the techniques for statistics computation are distance-metric dependent and we treat several popular distance metrics such as the Hamming distance, Euclidean distance, and set intersection cardinality. We also provide experimental results using linear secret sharing as the underlying data protection mechanism that show that the overhead introduced by our techniques is reasonable. We expect that the design insights that emerged as a result of studying this problem will inform design decisions for verifiable computation in other domains as well.

REFERENCES

Afrati, F., Das Sarma, A., Menestrina, D., Parameswaran, A., and Ullman, J. 2012. Fuzzy joins using MapReduce. In *IEEE International Conference on Data Engineering*. 498–509.

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

Ateniese, G., Burns, R., Curtmola, R., Herring, J., Kissner, L., Peterson, Z., and Song, D. 2007. Provable data possession at untrusted stores. In *CCS*. 598–609.

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

Ben-Or, M., Goldwasser, S., and Wigderson, A. 1988. Completeness theorems for non-cryptographic fault-tolerant distributed computation. In *ACM Symposium on Theory of Computing (STOC)*. 1–10.

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

Blanton, M. and Aguiar, E. 2012. Private and oblivious set and multiset operations. In *ACM Symposium on Information, Computer and Communications Security*.

Blanton, M. and Aliasgari, M. 2012. Secure outsourced computation of iris matching. *Journal of Computer Security 20,* 2–3, 259–305.

Blanton, M. and Gasti, P. 2011. Secure and efficient protocols for iris and fingerprint identification. In *European Symposium on Research in Computer Security (ESORICS)*. 190–209.

Blanton, M., Zhang, Y., and Frikken, K. 2011a. Secure and verifiable outsourcing of large-scale biomeric computations. Tech. Rep. 2011-04, Department of Computer Science and Engineering, University of Notre Dame.

Blanton, M., Zhang, Y., and Frikken, K. 2011b. Secure and verifiable outsourcing of large-scale biometric computations. In *IEEE International Conference on Information Privacy, Security, Risk and Trust (PASSAT)*. 1085–1091.

Bui, H., Kelly, M., Lyon, C., Pasquier, M., Thomas, D., Flynn, P., and Thain, D. 2009. Experience with BXGrid: A data repository and computing grid for biometrics research. *Journal of Cluster Computing 12,* 4, 373–386.

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

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

Chor, B., Goldwasser, S., Micali, S., and Awerbuch, B. 1985. Verifiable secret sharing and achieving simultaneity in the presence of faults. In *FOCS*. 383–395.

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

Du, W., Jia, J., Mangal, M., and Murugesan, M. 2004. Uncheatable grid computing. In *ICDCS*. 4–11.

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

Gennaro, R., Gentry, C., and Parno, B. 2010. Non-interactive verifiable computing: Outsourcing computation to untrusted workers. In *Advances in Cryptography – CRYPTO*. 465–482.

Gennaro, R., Rabin, M., and Rabin, T. 1998. Simplified VSS and fast-track multiparty computations with applications to threshold cryptography. In *PODC*. 101–111.

Gens, F. 2008. IT cloud services user survey, pt. 2: Top benefits & challenges. http://bit.ly/oUCbY.

Goldwasser, S., Micali, S., and Rackoff, C. 1985. Knowledge complexity of interactive proof systems. In *STOC*. 291–304.

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

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

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

Goodrich, M. 2010. Randomized Shellsort: A simple oblivious sorting algorithm. In *SODA*. 1262–1277.

Juels, A. and Kaliski, B. 2007. PORs: Proofs of retrievability for large files. In *CCS*. 584–597.

Kahney, L. 2001. Cheaters bow to peer pressure. WIRED Magazine. http://bit.ly/ZVuXCJ.

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

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

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

Okcan, A. and Riedewald, M. 2011. Processing theta-joins using MapReduce. In *ACM SIGMOD International Conference on Management of Data*. 949–960.

Shamir, A. 1979. How to share a secret. *Communications of the ACM 22,* 11, 612–613.

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

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

Watanabe, K., Fukushi, M., and Horiguchi, S. 2009. Collusion-resistant sabotage-tolerance mechanisms for volunteer computing systems. In *ICEBE*. 213–218.

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 an (N, t) linear threshold secret sharing scheme as such as [Shamir 1979], 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 $x = y$. The most efficient existing implementations we are aware of [Catrina and de Hoogh 2010] uses $\ell + 4 \log \ell$ interactive operations, where ℓ is the bitlength of x and y, operations in 4 rounds (where some of the computation is input-independent and can be performed in advance).

— $[b_1], \ldots, [b_m] \leftarrow \mathsf{Sort}([a_1], \ldots, [a_m])$, 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., [Goodrich 2010]). A comparison operation, for example, "less than" $\mathsf{LT}([x], [y])$, can be implemented using $4\ell - 2$ interactive operations in 4 rounds, where, as before, ℓ is the length of x and y and some of the computation is input-independent.

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 the 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 [Blanton and Aguiar 2012] as follows:

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 [Blanton and Aguiar 2012]. 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 a comparison protocol.

Finally, for each computed distance $[d]$, the servers update counts in 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) $\quad [b_i] \leftarrow \mathsf{Eq}([d], [d_i])$;
(3) $\quad [c_i] \leftarrow [c_i] + [b_i]$;
(4) return $[c_0], \ldots, [c_{v-1}]$;

B. COMPARISON OF SERVER'S STRATEGIES FOR ALLPAIRS COMPUTATION

In this section, we provide additional analysis of the server's strategies for AllPairs computation from Section 3.1 with the goal of determining the best one (i.e., with the lowest detection probability) for the adversary. Our analysis covers (i) determining detection probability for a server's strategy that combines strategies 2 and 3 in Section 3.1 (recall that strategy 1 is a special cases of strategy 2 and does not need to be considered separately); (ii) determining parameters under which the probability of detection for strategy 2 is minimized; and (iii) comparing all strategies to determine the best one for the adversary.

B.1. Analysis of combined server's strategy

To complete the analysis of the server's strategies in avoiding detection during dishonest AllPairs computation, we analyze a combination of server's strategies 2 and 3 from Section 3.1. This strategy combines partial rows with randomly chosen cells within each selected row. Now the server first chooses $p_r n$ rows, and within each chosen row it chooses $p_c n$ cells at random positions, independently for each row. As before, we have $p_c p_r = p$. Then for any given row that the client checks, the server's behavior is not detected if either (i) the row was among partially computed and the server either computed or guessed correctly values of n_2 checked cells in that row or (ii) the row was not among partially computed rows and the server guessed all n_2 checked cells in that row. The probability of (i) equals to $p_r \left(p_c + \alpha \left(1 - p_c\right)\right)^{n_2}$ (where $p_c + \alpha \left(1 - p_c\right)$ is the probability that one cell was computed or its value was guessed correctly), and the probability of (ii) equals to $(1 - p_r)\alpha^{n_2}$. We obtain that the probability of detection after checking all n_1 rows is:

$$\Pr[\mathsf{D}] = 1 - \left(p_r \left(p_c + \alpha \left(1 - p_c\right)\right)^{n_2} + \alpha^{n_2} \left(1 - p_r\right)\right)^{n_1}$$

i.e., the same as for strategy 2.

B.2. Analysis of strategy 2

The goal of this section is to analyze server's second strategy to determine under which set of parameters p_c and p_r its detection probability is the lowest. Because strategy 1 is a special case of strategy 2, this will also answer the question of how important strategy 1 is for further consideration.

In the current analysis, we treat p_c and p_r as variables, constrained by the condition $p_c p_r = p$, and the rest of the parameters as fixed. The detection probability for strategy 2

is given in equation 5, and we start the analysis by substituting p_r with p/p_c, which gives us a function of a single variable p_c, whose values range from p to 1. We obtain:

$$\Pr[\mathsf{D}] = 1 - \left(\left(p/p_c\right)\left(p_c + \alpha\left(1-p_c\right)\right)^{n_2} + \alpha^{n_2}\left(1-p/p_c\right)\right)^{n_1}$$

To determine the function's critical point, we take its derivative and after rearranging the terms obtain:

$$\begin{aligned} f'(p_c) \;=\; & -\frac{n_1 p}{(p_c)^2}\left(\left(p/p_c\right)\left(p_c+\alpha\left(1-p_c\right)\right)^{n_2} + \alpha^{n_2}\left(1-p/p_c\right)\right)^{n_1-1} \times \\ & \times\left(\left(p_c+\alpha(1-p_c)\right)^{n_2-1}\left(p_c(n_2-1)(1-\alpha)-\alpha\right)+\alpha^{n_2}\right) \end{aligned}$$

Now notice that the sign of this expression is determined by the term $p_c(n_2-1)(1-\alpha)-\alpha$, and the remaining terms are all positive when $p_c \in [p,1]$. We thus consider two cases:

(1) If $p \geq \frac{\alpha}{(n_2-1)(1-\alpha)}$, the term $p_c(n_2-1)(1-\alpha)-\alpha$ is non-negative, which results in a negative value for $f'(p_c)$. This means that the function $\Pr[\mathsf{D}]$ is strictly monotonically decreasing and its minimum value is at the endpoint $p_c = 1$, which corresponds to strategy 1.

(2) If $p < \frac{\alpha}{(n_2-1)(1-\alpha)}$, the term $p_c(n_2-1)(1-\alpha)-\alpha$ is either always negative for $p_c \in [p,1]$ or changes its sign from negative to positive within that interval (i.e., it is a linearly increasing function). This means that the factor $\left(\left(p_c+\alpha(1-p_c)\right)^{n_2-1}\left(p_c(n_2-1)(1-\alpha)-\alpha\right)+\alpha^{n_2}\right)$ may also change its sign, which implies that there exists a critical point within the interval $[p,1]$. Since the sign of $f'(p_c)$ can only change from positive to negative within $[p,1]$, in that case the function for $\Pr[\mathsf{D}]$ has a relative maximum and no other extrema, which means that the minimum value of $\Pr[\mathsf{D}]$ is either at $p_c = p$ or $p_c = 1$, which correspond to strategy 1. If the sign of $f'(p_c)$, however, does not change, the function is strictly monotone and likewise we obtain that the minimum value of $\Pr[\mathsf{D}]$ is at either $p_c = p$ or $p_c = 1$.

We obtain that, regardless of the parameter values, it is advantageous for an adversary who wishes to minimize the detection probability to use the special case of strategy 2 that corresponds to strategy 1.

B.3. Comparison of strategies 1 and 3

What remains is to compare server's strategies 1 and 3 to determine if one of them always results in a lower probability of detection than the other and thus will be preferred by an adversary. To ensure that the result holds for any values of security parameters n_1 and n_2 that the client chooses, we treat the detection probabilities (given in equations 3 and 6, respectively) as functions of variables n_1 and n_2 with fixed α and p. Because both of these strategies have the same format, namely, $1 - X^{n_1}$ for some X, we need to compare only the expressions corresponding to X in the two strategies, which we denote by X_1 and X_3 in strategies 1 and 3, respectively. From equations 4 and 6 we obtain $X_1 = p + (1-p)\alpha^{n_2}$ and $X_3 = (p+\alpha(1-p))^{n_2}$.

To compare the values of X_1 and X_3, we compute their difference

$$f(n_2) = X_1 - X_3 = p + (1-p)\alpha^{n_2} - (p+\alpha(1-p))^{n_2} \tag{18}$$

and then take its derivative to obtain

$$f'(n_2) = (1-p)(\log\alpha)\alpha^{n_2} - \log(p+\alpha(1-p))(p+\alpha(1-p))^{n_2} \tag{19}$$

Setting $f'(n_2) = 0$ allows us to obtain the function's critical point at value $n_2' = \log\frac{(1-p)\log\alpha}{\log(p+\alpha(1-p))} / \log\frac{p+\alpha(1-p)}{\alpha}$. Because $p+\alpha(1-p) > \alpha$ for any $0 < \alpha < 1$, $\log\frac{p+\alpha(1-p)}{\alpha}$ is always positive, and the sign of the value that n_2' takes is determined by the dividend $\log\frac{(1-p)\log\alpha}{\log(p+\alpha(1-p))}$.

In our application, n_2 is always positive with its values lying in the range $[0, +\infty]$. At the endpoints 0 and $+\infty$, the expression in equation 18 evaluates to 0 and p, respectively (as both α and $p + (1-p)\alpha$ are less than 1). Therefore, if the critical point at n_2' does not correspond to a relative extremum, we know that within the interval $[0, +\infty]$ $X_1 - X_3$ will always take a positive value. This means that strategy 1 leads to a lower detection probability. If, on the other hand, n_2' corresponds to a local extremum, based on the sign of n_2', two situations may occur:

(1) $n_2' < 0$: To determine whether n_2' corresponds to the function's minimum or maximum value, we need to evaluate the sign of $f'(n_2)$ near the point n_2'. From equation 19, we have that $(\frac{p+\alpha(1-p)}{\alpha})^{n_2'} = \frac{(1-p)\log\alpha}{\log(p+\alpha(1-p))}$. For any $n_2 < n_2'$, we also have that $(\frac{p+\alpha(1-p)}{\alpha})^{n_2} < (\frac{p+\alpha(1-p)}{\alpha})^{n_2'}$ as $\frac{p+\alpha(1-p)}{\alpha} > 1$ and therefore $(\frac{p+\alpha(1-p)}{\alpha})^{n_2} < \frac{(1-p)\log\alpha}{\log(p+\alpha(1-p))}$. After multiplying both sides of this inequality by negative $\alpha^{n_2}\log(p+\alpha(1-p))$, we obtain that $f'(n_2) < 0$. Using the same approach, we obtain that $f'(n_2) > 0$ for $n_2 > n_2'$, and thus n_2' corresponds to a local minimum. Based on the fact that there is a single extremum, this analysis gives us that $f(n_2)$ is strictly monotonically increasing on the interval $[0, +\infty]$. This function's behavior is also in compliance with the fact that equation 18 evaluates to 0 at the endpoint 0 and to p at the endpoint $+\infty$. We obtain that when $n_2' < 0$, strategy 1 always leads to a smaller detection probability than strategy 3. What remains to determine is under what circumstances $n_2' < 0$. In order for n_2' to have a negative value, we must have $\log\frac{(1-p)\log\alpha}{\log(p+\alpha(1-p))} < 0$ and thus $\frac{(1-p)\log\alpha}{\log(p+\alpha(1-p))} < 1$. This is true when α and p satisfy $\alpha^{1-p} > p + \alpha(1-p)$.

(2) $n_2' \geq 0$: Applying the same analysis as in case 1, we obtain that the critical point at n_2' corresponds to a local minimum. This implies that $X_1 - X_3 = 0$ at both $n_2 = 0$ and another point greater than n_2' which we denote by n_2''. We obtain that the value of $f(n_2)$ is negative on the interval $(0, n_2'')$, and it is positive on the interval $(n_2'', +\infty]$. This gives us that under a certain set of parameters strategy 3 will result in lower detection probability than strategy 1 and therefore will be superior for the adversary.

To summarize:

— When $\alpha^{1-p} > p + \alpha(1-p)$, strategy 1 is superior for an adversary to use to strategy 3.
— If $\alpha^{1-p} < p + \alpha(1-p)$ and $n_2 > n_2''$, strategy 1 is also superior for an adversary to use to strategy 3.
— If $\alpha^{1-p} < p + \alpha(1-p)$ and $n_2 < n_2''$, strategy 3 is superior for an adversary to use to strategy 1.

C. COMBINING VERIFICATION OF DISTANCES AND STATISTICS

Here we describe how the verification of distances and statistics can be combined for the edit distance and set intersection cardinality (this was done for the Hamming distance in section 4.4).

C.1. Edit Distance

The techniques for combining verification of AllPairs and Analyze computation remain largely unchanged from the techniques for the Hamming distance. That is, the algorithm in Figure 5 can be used with minimal changes. In particular, during creation of an outsourced task, the client generates n_1 pairs of m-dimensional fake vectors with the distance uniformly distributed in the range $[0, mh^2]$. It then chooses the values of d in step 2 from the range $[mh^2+1, mh^2+\ell]$ and creates protected distances in step 5 for $i = 0, \ldots, 2mh^2+\ell$. During verification of an outsourced task, the range of distances between two real vectors in steps 1(a) and 3 changes from $[0, m]$ to $[0, mh^2]$ and the range of distances between real and fake vectors in steps 1(a) and 3 changes from $[m+1, 2m+\ell]$ to $[mh^2+1, 2mh^2+\ell]$.

C.2. Set Intersection Cardinality

The technique for combining verification procedures for distance and statistics computation for the set intersection cardinality metric differs from those for the Hamming and Euclidean distances. This time, the real sets are generated in the same way as for verification of statistics computation, but there are small changes in the way the fake sets are produced. In particular, instead of setting d elements to 0 in a given fake set, where d is chosen uniformly from a range $[0, k-\ell-1]$, we coordinate the values of d in a pair of fake elements. That is, we create n_1 pairs of fake elements $\langle \hat{x}, \hat{y} \rangle$, where the distance between them is a randomly chosen value from $[0, k-\ell-1]$. This is accomplished by having $k-\ell-1-\mathsf{dist}(\hat{x}, \hat{y})$ 0 elements in common between $\hat{x}$ and $\hat{y}$. All other elements in both $\hat{x}$ and $\hat{y}$ are set as before. In other words, the values of d for $\hat{x}$ and $\hat{y}$ are chosen in a coordinated manner, but otherwise the process is the same as before. This gives us that the number of 0 elements among the fake sets in either S_1 or S_2 is no longer guaranteed to be uniform in the range $[0, k-\ell-1]$, but this does not pose a security problem for the following reason: Because the solution is designed in such a way that by missing the computation associated with a single fake set the server will have to guess multiple locations in C and this is what guarantees the necessary probability of detection, this will be true if any fake set has been missed, regardless of the overall distribution of the distances associated with the set of fake sets in general. The procedure for verifying correctness of an outsourced task is therefore the same as for verifying the statistics computation as described in section 6.2, followed by verification of the distances $\mathsf{dist}(\hat{x}, \hat{y})$ for n_1 pairs of fake elements $\langle \hat{x}, \hat{y} \rangle$.

This modification does influence a portion of the security analysis. In particular, because the distances between two fake sets are now in the range $[\ell, k-1]$, the probability of detection of incorrect computation of distances in equation 7 now becomes $\Pr[\mathsf{D}] = 1 - \left(\frac{p(k-\ell-1)+1}{k-\ell}\right)^{n_1}$. If $(k-\ell-1) < m$ and $\Pr[\mathsf{D}]$ is insufficiently high, either the value of n_1 or k can be increased until a desired probability of detection is achieved, where changing n_1 will have a significantly larger impact on the value of $\Pr[\mathsf{D}]$ than changing k.