Configurable Security Protocols for Multi-party Data Analysis with Malicious Participants

Bradley Malin, Edoardo Airoldi, Samuel Edoho-Eket, and Yiheng Li

September 2004
CMU-ISRI-04-132

Data Privacy Laboratory
Institute for Software Research International
School of Computer Science
Carnegie Mellon University
Pittsburgh, PA 15213

Abstract

Standard multi-party computation models assume semi-honest behavior, where the majority of participants implement protocols according to specification, an assumption not always plausible. In this paper we introduce a multi-party protocol for collaborative data analysis when participants are malicious and fail to follow specification. The protocol incorporates a semi-trusted third party, which analyzes encrypted data and provides honest responses that only intended recipients can successfully decrypt. The protocol incorporates data confidentiality by enabling participants to receive encrypted responses tailored to their own encrypted data submissions without revealing plaintext to other participants, including the third party. As opposed to previous models, trust need only be placed on a single participant with no data at stake. Additionally, the proposed protocol is configurable in a way that security features are controlled by independent subprotocols. Various combinations of subprotocols allow for a flexible security system, appropriate for a number of distributed data applications, such as secure list comparison.

Keywords: multiparty computation, confidentiality, configurable security, secure list comparison, malicious behavior, quasi-commutative cryptography, communication protocols

Contents

1 Introduction

As technologies for collecting information infiltrate society, the ability to record and store personal information about specific individuals continues toward ubiquity. Knowingly and unknowingly, individuals shed data to a number of data collectors both within, as well as beyond, the confines of one's home. The information collection can be overt and apparent to the individual, such as when a consumer visits a retail store and completes a purchase with a personal credit card. Or data gathering can be less discernable, as when an individual's image is captured by an unforeseen video surveillance system. Regardless, the collection, storage, and sharing of personal information is becoming more widespread. [1]

In many instances, it is the interest of disparate data collecting locations to combine their data to learn more robust knowledge. Though locations wish to collaborate, it is preferable not to reveal information which may compromise proprietary or strategic knowledge, overstep the boundaries set forth by legal statutes, or negatively affect individuals from whom the data was derived. Researchers in theoretical [2, 3, 4] and application-based multi-party computation [5, 6] have proposed methods to allow locations to collaborate by communicating only encrypted data. While the current techniques are useful for enabling encrypted data comparisons, they are hindered in their general applicability due to certain assumptions regarding the honesty of participating parties.

Most secure multi-party computation schemes are designed under an expectation that the majority of participating parties are *semi-honest*. In the semi-honest model, participants are expected to follow protocol specifications, but record intermediate values observed during the protocol which can be employed to compromise security. This is a widely used assumption in multi-party system analysis, however, it does not cover the space of adversarial models. When locations are *malicious* or corrupt, they can attempt any number of techniques to gain an advantage over other locations, influence results, or simply wreak havoc. For example, consider multi-party protocols which require all locations to perform some action over every location's dataset [5]. When participating locations receive different data analysis results, a malicious participant can drop out of the protocol once it learns the contents of its results, thus preventing other location's from learning their own results. In previous multi-party models such problems were attended to by limiting the data analysis to a single global result which was broadcast to all participants. Yet, as will be shown, such limitations are unnecessary.

Over the past several years, various multi-party computation schemas have been applied to demonstrate how certain data mining endeavors, such as association rule learning, decision-tree construction, and basic machine learning methods can be achieved in an encrypted setting. [6, 7, 9, 8] The specific type of multi-party computation this research generalizes is based on quasi-commutative cryptography, shown to be applicable for distributed association rule mining. [5, 7] Though encrypted data analysis is achieved, it has been depicted in a proof of concept manner, rather than from a security perspective. Thus, in this paper we develop a protocol to perform distributed data analysis in a manner which adheres to more stringent security requirements. In addition to making the previous multi-party computation more secure, we provide intuition into how such a protocol can be configured for a number of different distributed data analyses. Most importantly, the protocol herein permits for each participating location to receive a differential response, which can be tailored to their data submissions.

In this paper, we extend current methods and introduce a protocol named secure centralized analysis of multi-party data, or SCAMD, to cope with malicious participants. We provide proofs of additional security and integrity features which are not guaranteed in prior multi-party computation methods once semi-honest assumptions are relaxed. From a general perspective, the SCAMD protocol allows for several new security features which garner special attention. First, the protocol is guaranteed to be collusion resistant. No location can collude with another location to bound or learn exactly the plaintext values in another location's dataset. Second, our model protects the integrity of every participating location's dataset. No location can maliciously target another location's dataset and tamper with values without being detected. This is a concern in previous models as will be discussed later. Third, we incorporate a component for a locking mechanism to prevent any location from observing plaintext data until all locations can correctly decrypt their own results. The level of protection afforded in the latter two features are probabilistic, but are specific to each location, such that each participant determines the appropriate amount of security necessary for their own data.

The SCAMD protocol itself is not completely devoid of trust requirements. In order to achieve the aforementioned properties, a semi-trusted third party[1] is incorporated to perform honest data analysis. Yet, the use of a third party requires no more trust than in previous models, and actually permits the protocol to be more trustworthy. In comparison to previous models, where each participant must be viewed with a certain amount of skepticism, the third party model allows for participants to place their trust in a single party. This is especially useful, since the lone trustworthy party has no data of its own at stake.

The remainder of this paper is organized as follows. In the next section, relevant concepts from multi-party computation and encryption are reviewed. In section 3, we present the basic communications and data transfers which comprise the core of the protocol. In section 4, we develop protocol extensions particular to security and integrity, as well as prove their protective properties. Computational and bandwidth requirements of the protocol and its extensions are also addressed. In section 5, we demonstrate how the modular design of the protocol addresses computational concerns and permits different types of data analysis, such as differential encrypted response and centralized broadcasting. As an example, we map previous distributed analyses into the architecture of our protocol. Finally, limitations and possible extensions to this work are discussed.

2 Quasi-commutative Encryption

The protection protocol described below makes use of an interesting concept from secure multi-party computation known as the one way accumulator, or OWA. [10] In related research, OWAs were applied to a variety of distributed secure computations. For example, Zachary [11] demonstrates OWAs provide the necessary features for securely testing membership of nodes in distributed sensor networks. From another perspective, Faldella and Prandini [12] make use of OWAs for certificate authentication in a distributed public-key infrastructure. Most recently, and the work this research is closest to, Kantarcioglu and Clifton [5, 7] apply OWAs for data mining distributed association rules.

The protocol herein also employs OWAs for computation in a distributed environment. With respect to this research, the reader should view an OWA as a function to empower disparate locations, using different encryption keys, with the ability to reveal encrypted information from their local datasets, such that an encrypted piece of data is an equivalent primitive across locations. The OWA applied in this manner permits analysis and protection strategies to be executed over encrypted data. Plaintext information need not be revealed, unless it is desired by the owner of the data.

First, we review the general concepts of OWAs, then their transformation into keyed cryptosystems. Basically, an OWA is a hash function $h : X \times Y \rightarrow X$ that satisfies the *quasi-commutative* property. In equation (1), the following property holds for an arbitrary number and ordering of y_i.

$$h(h(x, y_1), y_2) = h(h(x, y_2), y_1) \tag{1}$$

Benaloh and de Mare note that the modular exponentiation function $e_n(x, y_i) = x^{y_i} mod(n)$, as defined in RSA encryption, is an OWA. [10, 13] For appropriately chosen n, where n is the product of two large prime integers p, q, computing x from $e_n(x, y_i)$ and y can not be accomplished in polynomial time. Since

[1] The third party is trusted to receive and analyze encrypted data only.

repeated use of e_n may reveal hash collisions, values of n are further restricted to be chosen from the set of *rigid integers*, defined as products of two *safe* primes p, q. A prime number p is safe if $p = 2p' + 1$, where p' is an odd prime. Additional information about the features of p and q, such as congruency and collision-resistance requirements, can be found in [10] and [14]. To provide some intuition, a safe prime is a large prime number which makes collisions of hashed values very unlikely to occur.

While other types of accumulators exist [15], with an RSA basis, the quasi-commutative accumulator allows for trapdoor recovery of plaintext values. As a result, OWAs can be converted into asymmetric keyed cryptosystems. In order to do so, each encryption key y_i is paired with a decryption key z_i, where $y_i * z_i = 1 mod(\varphi(n))$[2], for some function $\varphi(\cdot)$. When y_i and z_i are defined in this manner, decryption of an encrypted value v can proceed over m independent locations as

$$x = (h \ldots h(h(v, z_1), z_2), \ldots z_m) \tag{2}$$

Again, the ordering of the decryption keys $z_1, z_2, \ldots, z_m$ is of no consequence. Thus, the encrypted value v can be decrypted in a sequential manner using the same hash function as $h(x, z_i) = x^{z_i} mod(n)$.

3 Basic Communication Protocol

In this section, we introduce a protocol for the secure transfer and analysis of distributed data. The protocol is called SCAMD for secure centralized analysis of multi-party data. As the name implies, the current implementation requires a central authority, which we assume is semi-trusted. More specifically, it is trusted to receive and analyze encrypted data, but not plaintext. The central party will collect encrypted data from each of the data releasing locations and is expected to return honest responses, to known questions and/or analyses, to each location. In previous research, others have proven that the responsibilities of a trusted third party can be distributed among the participants of the protocol. [2, 3, 4] However, when such a feat is achieved, it usually occurs via the sacrifice of computational complexity, such that the protocol may be infeasible to compute given temporal constraints. Moreover, most protocols deficient of a third party assume participants to act semi-honestly, which requires they follow the specifications of the protocol. With the incorporation of a semi-trusted third party, the central authority, the SCAMD protocol can account for any number of malicious locations. Though a certain amount of trust is still necessary with respect the central authority, the protocol shifts trust from each of the participating locations, to a single location with no data of its own at stake in the process.

We begin with a general overview of the protocol. A more in-depth description and formal treatment follows. First, each location encrypts every other location's datasets. Then, the central authority is provided with the encrypted datasets. The central authority performs some function over the submitted datasets and returns a list of encrypted values to each location. The encrypted values are decrypted by the set of locations, such that the final decrypter is the location the list was destined for.

More formally, the SCAMD protocol is defined as follows. Let there exist two types of participants, data locations $L = \{l_1, l_2, \ldots, l_{|L|}\}$ and a single central authority C. Each location $l \in L$ maintains three pairs of encryption-decryption keys, $\langle y_l^b, z_l^b \rangle$, $\langle y_l^r, z_l^r \rangle$, $\langle y_l^m, z_l^m \rangle$, for an agreed upon quasi-commutative hash function h as defined above. The function h is made public, however, all keys are kept private to each location. The first two key pairs are used for blinding purposes only by location l with its own dataset, akin to the blind signature process defined in Chaum's original description of untraceable payment systems. [16]. The first key pair blinds (superscript b) the data so that it can be digitally signed by every location with their multi-party encryption key (superscript m). The second key pair blinds the data after the central authority

[2]The term $\varphi(n)$, Euler's totient function, specifies the number of relatively prime positive integers less than n.

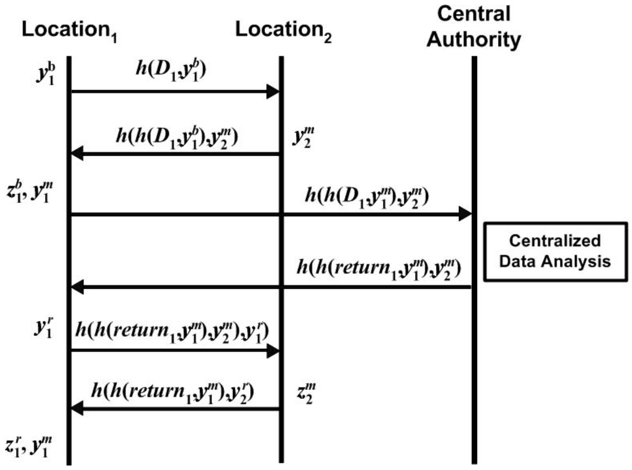

Figure 1: Basic SCAMD protocol as executed by location 1, for scenario with two locations and central authority.

has returned its computation. Thus, this key pair serves for recollection (superscript r) of the plaintext data via decryption with every party's multi-party decryption key.

For simplicity, we represent location l's dataset as D_l and the set of encrypted values as $h(D_l, y)$. In addition, the number of records in a dataset is represented as the cardinality, or $|D_l|$. We now step through the basic protocol. A version of the protocol with two locations is shown in Figure 1.

Step 1. (Blinding for Encryption) Each location l creates a dataset of "dummy" values and adds them to dataset D_l.[3] Then, l encrypts each value in D_l using y_l^b. After this initial encryption, a blinded dataset $h(D_l, y_l^b)$ exists for, and is in the sole possession of, each location.

Step 2. (Full Encryption) Each location $l \in L$ shuffles and encrypts with y_l^m its own blinded dataset and sends it to other locations $x \in L$ in a sequential fashion. Each location x encrypts the received dataset with y_x^m and sends the dataset back to l. Once every location has encrypted the dataset, location l removes the blinding by decrypting with z_l^b. As a result, each location l is in the possession of $h(h(\dots h(h(D_l, y_1^m), y_2^m) \dots, y_{|L|-1}^m), y_{|L|}^m)$.

Step 3. (Encrypted Analysis) Each location sends the resulting dataset to the central authority C, who performs data analysis over the set of datasets. The central authority returns $return_l$ datasets to each l, which specifies values of interest to location l.

Step 4. (Blinding for Decryption) Upon reception, l blinds $return_l$ with the recollection encryption key y_l^r.

Step 5. (Full Decryption: Return) As in Step 2, for each location l, the encrypted $return_l$ datasets are shuffled and sent to each location $x \in L$ (including l). Now, location x decrypts the dataset with z_x^m and sends the dataset back to l. Once every location has decrypted the dataset, location l removes the blinding by decrypting with z_l^r.

[3]The specifics of the dummy values will be made more clear below. However, for the curious reader, it should be noted that its purpose is for the control of a particular probability.

4 Security and Integrity

In this section we present configurable subprotocols which can be added onto SCAMD for particular guarantees of security and integrity. We present the protocols, as well as several crucial proofs about the afforded protections. The first aspect of SCAMD proven is its ability to prevent any set of independent locations from learning the plaintext information of encrypted data held by honest locations through collusion. This aspect is derived directly from the basic SCAMD protocol presented in the previous section.

Theorem 1 (Collusion Resistant). Given any location $l \in L$, there exists no set of locations $U \subseteq L - \{l\}$, which can collude to determine the plaintext values of D_l.

Proof. In general, there are three ways by which plaintext values of D_l can be revealed. The first case is when D_l is sent to a colluding location. Since plaintext values are only directly revealed when l chooses so, this case never occurs. The second case is when both a hashed version of D_l and the appropriate decryption key is sent to a colluding location. Again, this never arises.

The third case is more subtle. It occurs by exploiting the definition of quasi-commutative encryption. When a colluding location $u \in U$ is in the possession of a hashed version of D_l which has been hashed by the same set of keys as D_u, then it can learn certain features of the data in D_l. The collection of hashed versions of D_l occurs during two points of the protocol: encryption and decryption. The first opportunity is via the encryption process before the dataset is submitted to the central authority. During this process, every version of D_l provided to colluding locations has been hashed with the blinding key y_l^b. Thus, for any colluder $u \in U$ to compare his dataset, it is necessary that l hashes D_u with y_l^b. However this never occurs, since l only uses y_l^b for his own dataset and no one else's. The second opportunity is via the decryption process, when D_l is sent as the $return_l$ list. Yet, as during encryption, the colluding locations only receive versions of $return_l$ which have been hashed with the recollection key, y_l^r, which is only used for l's datasets. □

Now that simple security with respect to semi-honest behavior has been established, we concentrate on problems with respect to malicious adversaries.

4.1 Extensions to Basic SCAMD

The basic protocol prevents locations from making direct inferences about any particular location's dataset. However, the protocol is leaky in security, since colluding, or independently malicious, locations can influence the central authority's analysis and subsequent response, in the form of the $return_l$ datasets. Moreover, a location can perform certain functions that will go undetected. In order to control data representation, the malicious location must be able to make changes to another location's data in a manner that is undetected. Specifically, a malicious location can influence the central authority through several means. First, a location can lie about which values exist in their data collection. While blatant dishonesty regarding one's own data is a concern, the SCAMD protocol does not address how to prevent such malicious acts.[4]

Second, a malicious location can attempt to control how data is represented during the execution of the protocol. In order to do so, the malicious location can employ a different multi-party key pair for another location's dataset. When a malicious location is using more than one multi-party key pair we term this action a *key switch*. We subclassify the key switch attack into two distinct, though related, types. The first type, called a *full* key switch, occurs when the malicious location applies a particular multi-party key pair

[4]Lying about one's dataset exists in the analysis of plaintext data as well. One manner by which dishonesty can be discovered is to validate data with external knowledge regarding the underlying truth. Yet, when dealing with proprietary knowledge, the construction of such a litmus test will be dependent on the data in question and may be impossible. Thus, we consider this problem beyond the scope of the current research.

to every value of a particular location's dataset. The second type, called a *partial* key switch, occurs when the malicious location partitions a location dataset into x parts and each part is encrypted/decrypted with a different multi-party key.

Now we turn to extensions of the basic protocol for integrity checks which detect key switch behavior. As will be proven, several extensions to the basic protocol guarantee that no set of malicious locations (even one location) can tamper with the encrypted data they receive at any stage without being detected. We analyze malicious data corruption in the form of both full and partial key switching. Theorem 2 covers the case of full key switching, which will be detected by the central authority, whereas theorem 3 covers the case of partial key switching, which is more easily detected by the data providing locations. Intuitively, the probability that partial key switching is detected by a single location has a naturally low bound under general conditions, whereas the probability that partial key switching is detected by the central authority requires non-negligible effort (in terms of bandwidth, for example) to be controlled below the same bound.

4.1.1 Checks Performed by the Central Authority

The first type detection for key switching is performed by the central authority. It accounts for the situation of "full" key switching, which occurs when Sally encrypts all of Alice's dataset with the "bad" keys. The extension works as follows. The central authority sends the same "dummy" value v_C to every participating location. Prior to Step 1 of the SCAMD protocol, every location adds the value to their dataset. The subprotocol for dummy data transfers and encryptions is shown in figure 2.

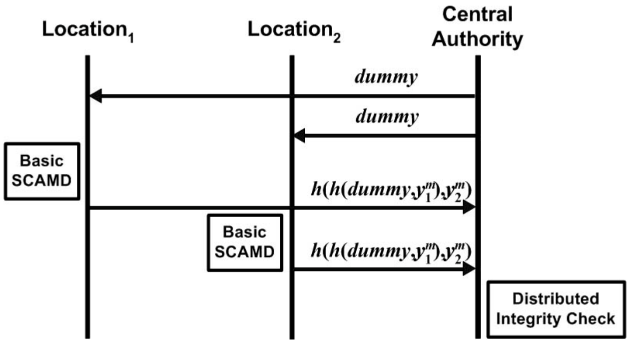

Figure 2: Full key switch detection performed by central authority.

Let us call the switching location Sally and the owner of the switched dataset Alice. We consider the case when Sally uses two multi-key pairs. Instead of $\langle y^m_{Sally}, z^m_{Sally}\rangle$, Sally will use $bad = \langle y^{bad}_{Sally}, z^{bad}_{Sally}\rangle$ and $good = \langle y^{good}_{Sally}, z^{good}_{Sally}\rangle$, respectively. The latter key pair is used with every location's dataset, except for Alice for whom Sally uses the previous.

Theorem 2 (Full Key Switch Integrity). The central authority is guaranteed to detect Sally's full key switch.

Proof. Assume Sally uses y^{bad}_{Sally} for all values in Alice's dataset. The only way that the full encrypted version of v_C, or any other value common to all datasets, will appear the same in all datasets is if Sally uses y^{bad}_{Sally} for every location's dataset including her own. Furthermore, if Sally only used y^{bad}_{Sally} during encryption, she must use z^{bad}_{Sally} with every location's dataset for decryption. However, if the latter is true, then Sally has only used one multi-party key pair and no key switching has occurred. □

If the central authority does not detect a value that is the same at all locations this does not necessarily imply key switching. Rather, it may imply that a location failed to add v_C to its dataset. Regardless, when the latter is true then the central authority still detects that something has gone wrong during the execution of the protocol.

4.1.2 Checks Performed by the Single Locations

In addition to full key switching, Sally can perform "partial" key switching. In a partial key switch, Sally uses the *bad* multi-party key pair with a fraction of Alice's dataset and the *good* multi-party key pair with the remainder. In order to prevent the partial key switch Alice introduces her own dummy data and uses an additional blinding key $\langle y^{check}_{Alice}, z^{check}_{Alice}\rangle$.

Prior to Step 1 of the protocol, Alice adds β dummy values to her dataset. After the final location has encrypted her data and prior to submitting the data to the central authority, Alice performs the following integrity check. After decrypting the data with the initial blinding key z^b_{Alice}, she re-encrypts her dataset with the new "check" key y^{check}_{Alice}, and then sends the dataset back to the other locations for decryption. If Sally is not performing a partial key switch, then she can correctly decrypt Alice's dataset without any problems. However, if Sally did perform a partial key switch then the probability she can correctly decrypt Alice's dataset is extremely small. In fact, Theorem 3 proves this happens with a naturally low probability, which can be further reduced by increasing β. Moreover, even if Alice believes that Sally randomly guessed the correct values to change, she can repeat the integrity check an arbitrary number, which we call α, times. For each repetition, Alice uses a new check key pair, again reducing at will the probability that Sally's cheating goes undetected. This process is depicted in figure 3.

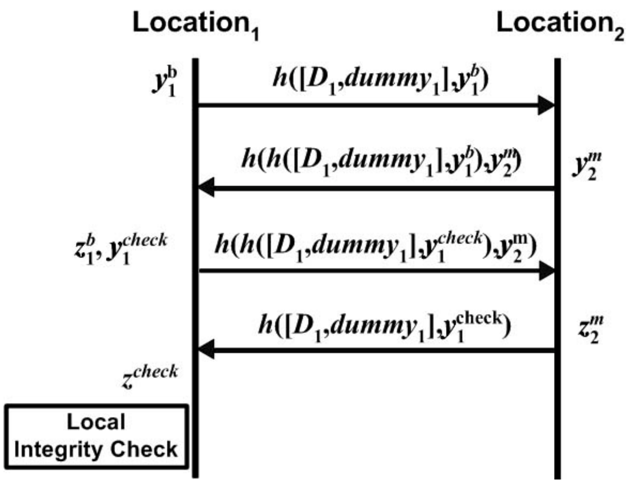

Figure 3: Partial key switch detection as performed by location 1.

Theorem 3 (Partial Key Switch Integrity). The probability Alice does not detect Sally's partial key switch is at most $P_{\alpha,\beta} := 1 - (|D_{Alice}| + \beta)^{-\alpha}$.

Proof. Assume Sally chooses f values in Alice's data to encrypt with y^{bad}_{Sally}. Now that Sally has performed her key switch, she must find those values in Alice's dataset during the decryption process. Yet, when Sally encrypted Alice's dataset, it was blinded by y^b_{Alice}, but now the data is blinded with y^{check}_{Alice}. As a result, unless Sally knows the new blinding key pair, Sally must select the f records which need be decrypted with z^{bad}_{Sally} at random. The probability of f successful guesses in our setting follows a hyper-geometric distribution with parameters $n - f$ (records encrypted by Sally with y^{good}_{Sally}) and f (records encrypted by

Sally with y^{bad}_{Sally}) and can be written as:

$$Pr\begin{pmatrix} \textit{undetected} \\ \textit{key switch} \end{pmatrix} = \frac{\begin{pmatrix} f \\ f \end{pmatrix}\begin{pmatrix} n-f \\ n-f \end{pmatrix}}{\begin{pmatrix} n \\ f \end{pmatrix}} = \frac{f!(n-f)!}{n!}$$

This probability is maximized at $f = 1$ or $f = n - 1$. In Figure 4 this is demonstrated for $n = 25$. As a result, Sally's best probability of remaining undetected is equal to $1/n$.

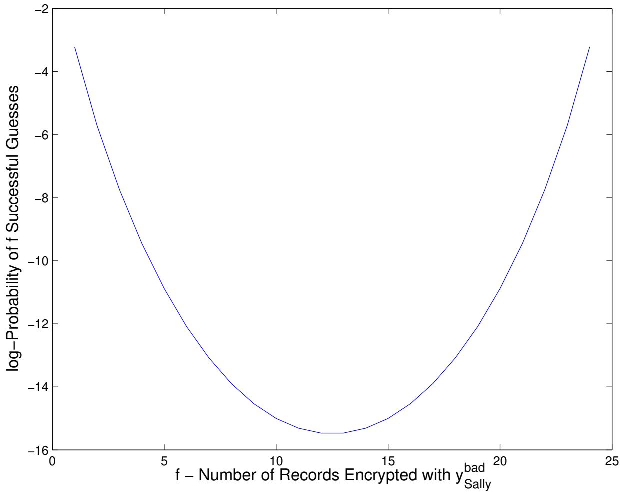

Figure 4: Sample probabilities of exactly f successful guesses for the case of 25 records ($\log(1/25) \approx -3.219$).

While f is chosen by Sally to maximize the probability of a partial key-switch being undetected, Alice can control $n = |D_{Alice}| + \beta$, the size of the dataset, to maximize the probability of detecting partial key switches. In particular, increasing the number of dummy records β, directly increases the probability of Sally's misbehavior being detected. However, Alice may wishes to decrease β to save bandwidth during communication or total time necessary to complete decryption of the dataset. In this case she can still control the probability of detecting Sally's misbehavior by simply increasing the number of times that the decryption check is performed. Each decryption check is performed independently, since Alice uses a different blinding key for each check. Thus, the probability that Sally's partial key switch is detected by Alice is $P_{\alpha,\beta}$ = 1 - $(D_{Alice} + \beta)^{-\alpha}$ or less.[5] $\square$

In combination, the integrity checks performed by both Alice and the central party guarantee that the probability Sally performs a key switch is arbitrarily small. Both Alice and the central party are required to perform key switch detection. Appendix A provides proof that a) Alice can not perform full key switch detection as efficiently as the central authority and b) the central party can not perform partial key switch detection as Alice.

[5]There are two possible scenarios. First, Alice chooses α (the number of checks to be performed) beforehand. In this scenario, the fact that the probability of detection is less than $P_{\alpha,\beta}$ is due to the fact that Sally's misbehavior can be detected before all α checks are performed. Second, Alice keeps on performing checks until the probability that a partial key switch was performed by Sally and was not detected falls below a certain threshold. In this latter scenario, the probability of detection is always equal to $P_{\alpha,\beta}$; in fact, Alice computes $P_{\alpha,\beta}$ after every check is performed and decides to stop when this probability is low enough.

4.1.3 Locking Out Malicious Locations

The implementation of the integrity checks performed in the previous section guarantee that no location can achieve a malicious action in the form of a key switch without being detected. In effect, the integrity of the cryptographic features are guaranteed, such that it is known that every location has both proper encryption and decryption multi-party key pair. However, the existence of such key pairs, does not imply that such key pairs will always be used. Neither the basic SCAMD protocol, nor the extensions for integrity discussed above, prevent Sally from achieving what we term a *grab-and-go*. Basically, Sally can recover the plaintext values of $return_{Sally}$ while simultaneously stopping Alice from recovering the plaintext values in $return_{Alice}$. This occurs when Alice decrypts Sally's dataset (*the grab*), but Sally refuses to decrypt Alice's dataset (*the go*).

In this section, we introduce a security feature that functions as a locking mechanism to prevent the grab-and-go. Basically, the central authority will guarantee that no location can recover their own values without acting honestly on behalf of all other locations datasets. Furthermore, the central authority will perform this validation without inspecting the plaintext values of any location's dataset.

The protection manifests in the form of a locking mechanism as follows. The central authority C creates a dummy dataset D_C at the very beginning. He then acts like an extra location performing Steps 1 and 2 of the SCAMD protocol, in other words, C sends D_C around through every location in L for full encryption, while all actual locations still perform the original Steps 1 and 2 with their own datasets. The lockout subprotocol is depicted in figure 5.

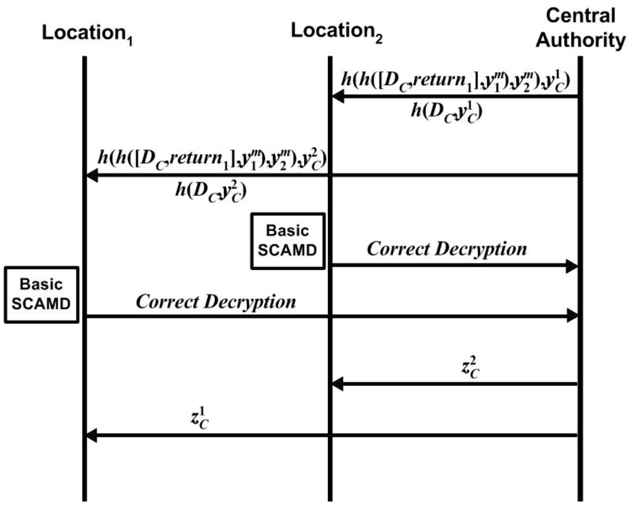

Figure 5: Locking protocol.

For each location $l \in L$, C chooses a blinding key pair $\langle y_C^l, z_C^l \rangle$, blinds the mixture with y_C^l, and sends $h(\ldots h(h([D_C,\ return_l],\ y_C^b)y_1^m)\ \ldots, y_{|L|}^m)$ back to location l. In addition, C sends the blinded plaintext dummy dataset values, $h(D_C, y_C^l)$, to l. Next, each location performs full decryption as specified in SCAMD, and each location now possesses the central authority's blinded dataset $h([D_C, return_l], y_C^l)$. If the blinded dataset includes $h(D_C, y_C^l)$, then l tells C that decryption was performed honestly. Once all location report honest decryptions, the central authority sends each $l \in L$ the appropriate blinding decryption key z_C^l. With the decryption key in hand, l decrypts the returned mixture and removes D_C.

Theorem 4 (Honest Decryption) The probability Sally achieves a *grab-and-go* against Alice (A) is at most $|return_A|$ / $(|return_A| + |D_C|)$.

Proof. We assume that both Alice (A) and the central authority (C) have verified that no key switching behavior exists. Since Alice does not return the full decrypted dataset to C, the detection of an attempted grab-and-go is the sole responsibility of Alice. When Sally performs a grab-and-go, she needs to correctly decrypt D_C, while leaving $return_A$ in an encrypted state. Let f be the number of values Sally selects for a false decryption. Assuming Sally chooses to correctly decrypt a minimum of $|D_C|$ values (otherwise it is guaranteed her malicious behavior is detected with probability 1), the probability of an undetected grab-and-go is the equal to the probability all f values are selected from $return_A$:

$$Pr\begin{pmatrix} \textit{undetected} \\ \textit{grab-and-go} \end{pmatrix} = \frac{\begin{pmatrix} |D_C| \\ 0 \end{pmatrix}\begin{pmatrix} |return_A| \\ f \end{pmatrix}}{\begin{pmatrix} |D_C| + |return_A| \\ f \end{pmatrix}}$$

$$= \frac{|return_A|!(|D_C| + |return_A| - f)!}{(|return_A| - f)!(|D_C| + |return_A|)!}$$

The probability is monotonic and is maximized when $f = 1$. An example of this is shown in Figure 6 for $|return_A| = 10$. When maximized, the probability Sally's action is undetected becomes $|return_A|$ / $(|return_A| + |D_C|)$. $\Box$

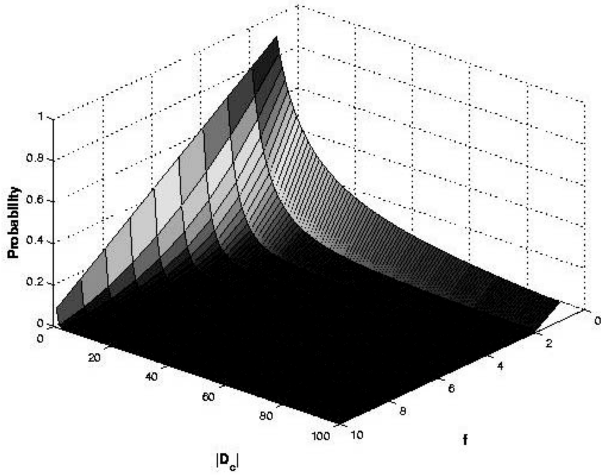

Figure 6: Probability Sally's grab-and-go attack against Alice is successful; $|return_{Alice}|$ = 10. D_C is the size of dummy dataset used by the central authority. f is the number of values Sally targeted.

In this locking mechanism, the central authority could initiate an extra integrity check for D_C in order to detect partial key switch attack by a malicious location. However, this type of attack (even a full key switch against the whole D_C) can be detected in later comparisons between what is supposed to be $h([D_C, return_l], y_l^C)$, and $h(D_C, y_l^C)$, by each location $l \in L$. As a result, a standalone integrity check for D_C is not necessary.

4.2 Computational Overhead

4.2.1 Encryptions/Decryptions

For the following analyses, we assume each encryption or decryption operation on a dataset costs constant time (or can be bounded by it).

As described in section 3, in the basic protocol, each location $l \in L$ maintains three pairs of encryption and decryption keys. The first two pairs are only applied to l's own dataset, while the third pair is applied to every location's dataset.

The number of encryptions/decryptions that a specific location l needs to perform is:

$$\text{O}(\textit{encryption - basic}) = 2(1 + 1 + |L|) = \text{O}(|L|)$$

The total number of encryptions/decryptions performed by the whole system is $\text{O}(|L|^2)$. However, the encryption/de-cryption process for all locations is done in parallel, such that the total time necessary to complete this process remains $\text{O}(|L|)$.

In the protocol with integrity check, each location $l \in L$ is asked to perform α decryptions for integrity check initiated by each $l' \in L$ respectively, i.e., totally $\alpha|L|$ decryptions. l would also have α pair of keys applying to its own dataset for encryption and decryption. Now, the number of encryptions/decryptions that l needs to perform becomes:

$$\text{O}(\textit{encryption - with integrity}) \quad = \quad 2(1 + 1 + |L|) + \alpha_l|L| + 2\alpha_l = \text{O}(\alpha_l|L|)$$

As a result, the total number of encryptions and decryptions for the whole system becomes $\text{O}(\substack{max \\ l \in L}\alpha_l|L|^2)$. Yet, akin to the basic protocol, the integrity checks are performed in parallel, so the total time necessary for completion is $\text{O}(\substack{max \\ l \in L}\alpha_l|L|)$.

When taking into account the locking mechanism to prevent the grab-and-go, the following additional encryptions and decryptions are needed due to dummy dataset D_C:

1. Each location needs to perform related encryption one time to create the fully encrypted version of D_C;

2. The central authority needs to perform blinding encryptions for each location, totally $|L|$ times;

3. Each location needs to apply a decryption key provided by the central authority, to restore its returning dataset.

However, the encryptions/decryptions associated with recollection keys are spared at each location. Thus, the total time necessary is still $\text{O}(\substack{max \\ l \in L}\alpha_l|L|)$ because of parallelism of the protocol.

4.2.2 Bandwidth

Each location l provides a dataset of size s_l. To be encrypted by another location, the dataset needs to be sent from the originating location l to the destination location l', and sent back to l after encryption. A similar process is performed during the decryption phase. In addition, all datasets must be sent to central authority for analysis; then a corresponding returning dataset of size r_l is sent back to the appropriate l.

In the basic protocol, each dataset proceeds through ($|L|$ - 1) encryptions and decryptions by locations other than the originating one, and communicates with the central authority once, so the total bandwidth required for a specific location l is:

$$\text{O}(\textit{bandwidth - basic}) \quad = \quad 2s_l(|L| - 1) + 2r_l(|L| - 1) + s_l + r_l = \text{O}((s_l + r_l)|L|),$$

where s_l = $|D_l|$ and r_l = $|return_l|$. Because of parallelism, the total bandwidth required to finish the protocol is $\text{O}(\substack{max \\ l \in L}(s_l + r_l)|L|)$.

In the protocol with integrity check, s_l equals ($|D_l|$ + β_l + 1), where β_l is the number of dummy value added by l and "1" accounts for the dummy value provided by the central authority. Each location needs extra α_l rounds of decryption from other locations, so the total bandwidth becomes:

$$\text{O}(\textit{bandwidth - with integrity}) = 2s_l(|L|-1) + 2r_l(|L|-1) + s_l + r_l + 2\alpha_l s_l(|L|-1), \quad (3)$$

where s_l = $|D_l|$ + β_l + 1 and r_l = $|return_l|$. Thus, an upper bound on the bandwidth is $\text{O}(\max_{l\in L} \alpha_l(s_l+r_l)|L|)$.

When we consider the additional locking mechanism, r_l equals ($|D_C|$ + $|return_l|$) and the central authority sends an extra version of blinded D_C to l. Assuming the full encryption of D_C is performed in parallel, the total bandwidth required for a specific location l is:

$$\text{O}(\textit{bandwidth - with integrity \& locking}) = 2s_l(|L|-1)+2r_l(|L|-1)+s_l+r_l+|D_C|+2\alpha_l s_l(|L|-1), \quad (4)$$

where s_l = $|D_l|$ + β_l + 1 and r_l = $|D_C|$ + $|return_l|$. Similarly, an upper bound is $\text{O}(\max_{l\in L} \alpha_l(s_l + r_l)|L|)$.

4.2.3 Integrity Check vs. Bandwidth

According to Theorem 3, the lower bound of the probability of location l detecting a partial key switch attack is $1 - (|D_l| + \beta_l + 1)^{-\alpha}$ (1 dummy value provided by the central authority considered). Assume each location l requires the protocol to satisfy a certain confidence requirement λ_l on this lower bound. In order to achieve the lowest bandwidth cost, we are actually solving a special non-linear optimization problem. Specifically, we need to solve for α_l's and β_l's as integers, when minimizing equation (3) and satisfying constraints on confidence in honesty for each $l \in L$:

$$1 - (|D_l| + \beta_l + 1)^{-\alpha_l} > \lambda_l,$$

as well as bandwidth and computational constraints.

Similarly, if the locking mechanism is also required, we need to simultaneously solve for α_l's and β_l's, as well as $|D_C|$ as integers, while minimizing equation (4) and satisfying additional constraints according to Theorem 4.

5 Protocol Application

The distributed operations the SCAMD protocol enable us to carry out in a secure manner are very different in nature. This has an impact on the format of the data needed for centralized data analysis, for example, a certain application may require plaintext data to be broadcasted by the *semi-trusted* third-party to the participating locations, whereas another application may require an analysis on the union of the encrypted data sets and only few sensitive records, still encrypted, may need be returned to the single locations. Thus, different scenarios require slightly different definitions of *semi-trusted*, which we now discuss. In the original definition, semi-trusted requires the central authority is never permitted to know the plaintext values it was analyzes. However, if plaintext values need to be broadcast, the definition of semi-trusted can be relaxed. In the relaxed definition, the third-party is not allowed to know which participating location submitted which specific values, though it is permitted to see the plaintext values of the records it is going to broadcast.

In addition, other aspects must change to fit the SCAMD protocol to different scenarios. For example, in order to allow the central authority to have broadcasting powers, we would modify the *full decryption* section of the protocol. Instead of sending a $return_l$ dataset to each location l, the central authority only needs to send a single dataset, which we refer to as $return_C$, around for decryption. Briefly, the central authority performs Steps 4 and 5 of SCAMD with it's own recollection key pair $\langle y^r_C, z^r_C\rangle$. The central authority blinds and sends $h(return_C, y^r_C)$ to each of the locations, for full decryption. Once fully decrypted, the central authority removes its blinding and broadcasts the plaintext data to all participating locations.

5.1 Configurability of the SCAMD Protocol

The SCAMD protocol provides security with provable probabilistic guarantees in carrying out distributed data mining tasks. The main ideas that enable functionality in a diversity of applications include: (a) the incorporation of dummy data, which allows for control over the detection of integrity tampering, (b) data shuffling, along with (c) forcing a malicious location to compete against its own behavior by decrypting its own encryption, and (d) a locking mechanism, which prevents malicious locations from learning information the protocol does not prescribe. Given a variety of practical tasks, we believe it was always possible to combine these basic security enabling addenda to fit the SCAMD protocol to the specific scenario at hand. This is achieved without any assumptions about semi-honest behavior on the part of the participating locations. Hence, we say that SCAMD protocol is *configurable* to fit an ample spectrum of distributed data mining computations.

In this light, removing certain parts of the protocol harms neither the functionality, nor the security of our protocol. For example, when the definition of semi-trusted is relaxed to provide the central authority with plaintext broadcasting capabilities the locking mechanism is not needed. It can be validated that removal of the locking mechanism security component does not affect the overall security of the protocol. Rather, it is not necessary to carry out the specific distributed computation task.

The SCAMD protocol is the first step towards a formal modular architecture for secure, distributed data mining, with provable guarantees in environments where semi-honest behavior on the part of participating locations can not safely be made. The next step in the development of a modular protocol is to feature a description of distributed data mining tasks along relevant dimensions, and map them into a sequence of primitive sub-tasks, which the various modules of our protocol can address in a secure way. A major portion of the single modules will address primitive sub-tasks, whereas others will provide provable security guarantees that the modules will produce the expected results, even in the presence of malicious participating locations.

5.2 Example: Distributed Data Union

To illustrate SCAMD's flexibility and security, we map a distributed association rule learning algorithm [5, 7] into the SCAMD architecture. The previously defined algorithm is based on semi-honest assumptions, which we refer to as *SemiSecureUnion*, or *SSU*, is presented to find the secure union of distributed data without revealing which itemsets belong to a given location. The underlying mechanism *SSU* is as follows. Each location l sends D_l to all locations in L for encryption, such that another location $x \neq l$, receives the full encrypted D_l. Once completed, each location holds another location's full encrypted dataset. Next, two locations collect and union half the full encrypted datasets. Then, one location sends its union to the second location, who again performs a unions. Finally, the locations, send the dataset around for full decryption, such that a third location, not one of the previous two, broadcasts the plaintext union.

The *SSU* protocol is susceptible to collusion, which suggests it may not be practical in real-world settings - even in a semi-honest environment. Consider, for example, a situation where a regional association of large-size retail stores wants to provide some aggregate statistics about the market, in the form of large itemsets. Collusion is more likely to occur in such a network when several sites belong to the same umbrella company or to the same chain (*e.g.* Wal-Mart comprises 70% of the participating sites).

We present the *SCAMD-SemiSecureUnion* (*SCAMD-SSU*), pseudocode provided in Algorithm 1, which maps *SSU* into the SCAMD architecture. *SCAMD-SSU* achieves the same goal as *SSU* and provides and in addition it 1) prevents each location from seeing other locations' fully encrypted data, 2) prevents two participating locations from performing the union of all fully encrypted data sets, and 3) prescribes for the data to be broadcasted from a third party, not present in *SSU*, that has no data at stake. In the semi-honest

Algorithm 1 SCAMD-SemiSecureUnion: Secure union of itemsets with SCAMD

Phase 0: *Local plaintext association rule learning*
for each $l \in L$ **do**
Generate set of local association rules $rule_l$ as defined in distributed association rule mining algorithm (FDM) defined in [17]
end for

Phase 1: *Encryption by all sites*
Each $l \in L$ executes SCAMD encryption phase (Steps 1-2) {*Each location possesses full encrypted* $rule_l$, *denoted* $frule_l$}

Phase 2: *Central itemset merge (Step 3 of SCAMD)*
All $l \in L$ send $frule_l$ to C
C creates $RuleSet = \bigcup_{l \in L} frule_l$

Phase 3: *Central Broadcast*
C sets $return_C = RuleSet$
C executes SCAMD decryption phase (Steps 4-5)
C broadcasts full decrypted dataset $return_C$

environment, the main concern is collusion. Since *SCAMD-SSU* is an implementation of the SCAMD protocol, collusion among participating locations is no longer a concern (as shown in Theorem 1). In addition, *SCAMD-SSU* solves problems which *SSU* is susceptible to once implemented with malicious locations. It is interesting to note that while *SSU* is susceptible to collusion, it is partially protective against a key switch attack. Again, consider the situation where Sally performs a key switch against Alice. When all of Alice's switched values are in any other location's dataset, then the key switch has no real influence on the union. In the alternative situation, if Alice's switched values are not within another location's submission, Alice can claim the integrity of her data was tampered with. However, this is why *SSU* provides only partial protection. Once the plaintext union is broadcast, Alice can not add her dataset to the union without every other location learning the unique values of her dataset. This problem is solved by *SCAMD-SSU* once the integrity checking is integrated.

5.3 Security Concerns and Future Research

In this section, we briefly discuss several concerns and challenges regarding the current design of the SCAMD protocol. The first concern corresponds to the difficulty in detecting which location is malicious. The second concern addresses assumptions regarding the central authority.

5.3.1 Traitors Lost in the Crowd

Theorems 2-4 prove it is possible to control the probability of malicious actions going undetected. Yet despite these controls, we acknowledge it is not possible to detect the source of the irregularity. This is mainly due to the usage of an accumulator based on quasi-commutative encryption. It is possible that more complex schemas may be able to detect both mishaps, as well as their sources, however, this is beyond the scope of the current research. We expect to look into such extensions in future research.

5.3.2 The Semi-Trusted Assumption

With respect to the central authority, many of the protections afforded by the SCAMD protocol are dependent on the assumption that the central authority is honest; it neither collaborates with participating locations nor answers locations dishonestly. First, consider a central authority which is merely semi-honest. In this case, collusion resistance as proven in Theorem 1, no longer holds true. Specifically, locations which collude with the central authority will be able to compare their full encrypted dataset against the full encrypted datasets of every non-colluding location. When an encrypted value v is found to be equivalent between the colluding and non- colluding datasets, then the colluder can bound the set of plaintext values for v. When multiple locations are colluding, the possibility exists that the colluder can learn the exact plaintext value for v. This occurs when the number the values in common between a set of colluders datasets is equivalent to the number of values in the non-colluders dataset.

Second, and of more grave concern, we consider a central authority which is actually malicious. When such an event occurs, Theorem 4 can nullified in such a way that non-colluding locations fail to detect malicious behavior. Basically, the central authority can supply corrupt locations with its blinding decryption key regardless of if corrupt locations used the correct multiparty decryption keys with other locations. Moreover, the central party can violate Theorem 1 in such a manner that a location $x \in L$ colluding with the central authority can learn all plaintext values of a dataset for any arbitrary location $l \in L$ without being detected. This would occur if the central authority was to provide a colluder with $return_x = D_l$ and Steps 4 and 5 of SCAMD proceed as specified.

Recent research in theoretical multi- computation proves that third parties can be removed from the protocol while maintaining the same level of security. [18] However, the design of these systems are most often inefficient to the point of being intractable for practical application. Thus, in future research we expect to continue investigating models which incorporate third parties, but reduce the requirement of the semi-trusted model.

6 Conclusions

This work introduced a novel protocol, termed secure centralized analysis of multi-party data, or SCAMD. The protocol allows for multiple locations to conduct analyses over distributed data in a secure manner in the face of malicious behavior. The protocol supports location-specific responses, such that each location can learn information, of which other locations can not ascertain the contents. Moreover, parallelism of the protocol allows for execution in linear time and bandwidth. The protocol is amenable to different types of encrypted data analysis, of which we demonstrated how secure unioning can be made more secure. In future research, we expect to demonstrate how the protocol can be used for a range of distributed computations in malicious environments.

Acknowledgements

The authors thank the members of the Data Privacy Laboratory at Carnegie Mellon University for support and encouragement, and Alessandro Acquisti for helpful discussions. This work was supported by the Data Privacy Laboratory.

References

[1] L. Sweeney. Information explosion. In: L. Zayatz, P. Doyle, J. Theeuwes, and J. Lane (eds): Confidentiality, disclosure, and data access: theory and practical applications for statistical agencies. Urban Institute, Washington, DC, 2001.

[2] A. Yao. How to generate and exchange secrets. In 27^{th} *IEEE Symposium on Foundations of Computer Science*. 1986, pp. 162-167.

[3] O. Goldreich, S. Micali, and A. Wigderson. How to play any mental game - or - a completeness theorem for protocols with honest majority. In 19^{th} *Symposium on Theory of Computing*. New York, NY, 1987, pp. 218-229.

[4] R. Canetti, U. Feige, O. Goldreich and M. Naor. Adaptively secure multi-party computation. In 28^{th} *ACM Symposium on Theory of Computing*. 1996, pp. 639-648.

[5] M. Kantarcioglu and C. Clifton. Privacy-preserving data mining of association rules on horizontally partitioned data. *IEEE Transactions on Knowledge and Data Engineering*. Forthcoming.

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

[7] M. Kantarcioglu and C. Clifton. Privacy-preserving distributed data mining of association rules on horizontally partitioned data. In *ACM SIGMOD Workshop on Research Issues in Data Mining and Knowledge Discovery* . Madison, WI, 2002.

[8] J. Canny. Collaborative filtering with privacy. In *IEEE Conference on Security and Privacy.* Oakland, CA, 2002, pp. 238-245.

[9] J. Vaidya and C. Clifton. Privacy-preserving k-means clustering over vertically partitioned data. In 9^{th} *ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. Washington, DC, 2003.

[10] J. Benaloh and M. deMare. One-way accumulators: a decentralized alternative to digital signatures (Extended Abstract). In: Hellsuth, T. (ed.): Advances in Cryptology (EUROCRYPT '93). LNCS 765. Springer-Verlag New York 1994, pp. 274-285.

[11] J. Zachary. A decentralized approach to secure group membership testing in distributed sensor networks. In *Military Communications Conference*. Boston, MA, Oct 2003.

[12] E. Faldella and M. Prandini. A novel approach to on-line status authentication of public-key certificates. In *16thAnnual Computer Security Applications Conference*. New Orleans, LA, Dec 2000.

[13] R. Rivest, A. Shamir, and L. Adleman. A method for obtaining digital signatures and public-key cryptosystems. Communications of the ACM, 1978; 21(2): pp. 120-126.

[14] N. Baric. and B. Pfitzmann. Collision-free accumulators and fail-stop signature schemes without trees. In *Advances in Cryptology: Proc. EUROCRYPT*. LNCS 1233. Springer-Verlag, New York 1997, pp. 480-494.

[15] T. Sander. Efficient accumulators without trapdoor. In: Varadharajan, V.and Mu, Y. (eds.) 2^{nd} *International Conference on Information and Communications Security - ICICS '99*. LNCS 1726. Springer-Verlag, New York, 1999, pp. 252-262.

[16] D. Chaum. Blind signatures for untraceable payments. Advances in Cryptography, Crypto 1982. Plenum Press. 1983, pp. 199-203.

[17] D. Cheung, J. Han, V. Ng, W. Fu, and Y. Fu. A fast distributed algorithm for mining association rules. In *International Conference on Parallel and Distributed Information Systems*. Miami Beach, FL, 1996, pp. 31-42.

[18] R. Canetti, Y. Lindell, R. Ostrovsky, and A. Sahai. Adaptively secure multi-party computation. In 34^{th} *Symposium on Theory of Computing*. New York, NY, 2002, 494503.

Appendix A: Necessity of Multiple Key Switch Detectors

Here we prove two facts: a) Alice can not perform full key switch detection as efficiently as the central authority and b) the central party can not perform partial key switch detection as efficiently as Alice.

Proof. For (a) we show that Alice, a honest location, cannot detect full key switch performed by Sally, a malicious location. In fact, if Sally performs a full key switch on Alice's data prior to submission to the central authority. No matter how many times Alice performs her own integrity check, Sally correctly decrypts all of Alice's dataset with probability 1.

For (b) recall that in section 4.1.1 the central authority requires Alice (and all other locations) to add one dummy record to its dataset to detect full key switching. The mechanism that allows the central authority to detect partial key switching follows the same logic: the central authority requires Alice (and all other locations) to add additional, say, γ dummy records, and then counts the number of identical records shared by the fully encrypted datasets it receives in step 3 of the SCAMD protocol. Now, the general idea that drives the proof is that the probability that partial key switching is detected by Alice has a naturally low bound under general conditions, whereas the probability that partial key switching is detected by the central authority requires non-negligible effort (in terms of bandwidth, for example) to be controlled below the same low bound. This happens since Alice starts and ends with its own fully decrypted dataset, whereas the central authority starts by imposing *a minimum* number of identical records (γ) and ends observing a number of identical encrypted values which may include some real records that appear multiple times in all datasets.

Recall from section 4.1.2 that Alice (and similarly for other locations) adds β dummy records to its dataset so that the probability of successfully detecting partial key switching equals[6] $P_{\alpha,\beta} = 1-(|D_{Alice}|+\beta)^{-1}$. Assume that there are r identical records which appear in the dataset of each location (not including those contributed by the central authority), and that Sally decides to encrypt f records with the multi-party key y^{bad}_{Sally} and $|D_{Alice}|+\gamma-f$ with the multi-party key y^{good}_{Sally}. The central authority is able to detect that a partial key switching has occurred only when the number of encrypted identical records shared by the datasets of all locations is less than γ. In order for this to happen Sally must pick $f > r$ records that it encrypts with y^{bad}_{Sally} among the $r+\gamma$ identical records. The probability of successfully detecting a partial key switch can be computed, again, using a hypergeometric distribution with parameters $\gamma+r$ (the total number of identical records shared by the datasets of all locations) and $|D_{Alice}|-r$ (the number of different records in Alice's dataset), and is equal to:

$$\begin{cases} 0 & \text{if } f \le r \\ 1-\displaystyle\sum_{i=0}^{r}\frac{\binom{\gamma+r}{i}\binom{|D_{Alice}|-r}{f-i}}{\binom{|D_{Alice}|+\gamma}{f}} & \text{o.w.} \end{cases}$$

where the summation takes into account the fact that Sally must pick at most r records (of the f she is going to encrypt with y^{bad}_{Sally}) among the $\gamma+r$ identical ones in order to fool the central authority.

[6] Assume for simplicity $\alpha = 1$.

According to the SCAMD protocol the central authority fixes γ before step 1 is performed, then is Sally's turn to choose how may records to encrypt (f) with the bad multi-party key. For $r = 0$ the summation reduces to:

$$\frac{\binom{|D_{Alice}|}{f}}{\binom{|D_{Alice}|+\gamma}{f}}$$

which is maximized at $f = 1$ and entails that when Alice uses $\beta = \gamma$ dummy records, Sally is more likely to fool the central authority than Alice since

$$\frac{|D_{Alice}|}{|D_{Alice}|+\gamma} > \frac{1}{|D_{Alice}|+\gamma}.$$

It is easy to verify that, as r increases, Sally is still more likely to fool the central authority than Alice, when Alice uses $\beta = \gamma$ dummy records as the central authority. $\square$