MTAT.07.006 Research Seminar in Cryptography
Building a secure aggregation database

Dan Bogdanov
University of Tartu, Institute of Computer Science

22.10.2006

1 Introduction

This paper starts by describing the privacy problem regarding the aggregation of sensitive data. Several general solutions are considered and a secure aggregation scheme based on secret sharing and secure multi-party computation is introduced. We describe our current progress in building a database which provides data privacy, but allows various aggregation algorithms to run and provide correct results on the input data.

This paper is based on joint work with Sven Laur[1] and Taneli Mielikäinen[2].

2 Aggregation of sensitive data

2.1 Problem statement

Databases containing personal, medical or financial information about a person are usually classified as sensitive. Processing such data often requires special licenses from respective authorities as determined by the law. This protection is a problem for research organisations, who can not learn global properties or trends from collected data, because the laws do not allow it .

In this paper we address a simplified version of the problem. Assume that we have asked m people n sensitive questions. By collecting the answers we obtain a matrix with m rows and n columns denoted $\mathcal{D}$) which represents our data. Our goal is to devise a method for calculating aggregate statistics from this matrix without compromising the privacy of a single person.

[1]Helsinki University of Technology
[2]HIIT Basic Research Unit

Let $W_1, \ldots, W_k$ be the participants who gather data and construct the matrix. Let M_1 be the data miner who is interested in global patterns. In the common scenario, all participants give their data to M_1 who constructs $\mathcal{D}$ and runs aggregation and data mining algorithms on it. The participants have to trust that M_1 will not use the data for selfish gains. They have no way of ensuring the privacy of people who have provided the data, because the miner requires control over the complete database to perform calculations.

2.2 General solutions

We want to keep a specific party from having the complete database. This way the participants will not have to trust a single entity.

2.2.1 Distribution of rows

We can divide the $m \times n$ data matrix $\mathcal{D}$ into smaller matrices $\mathcal{D}_1, \ldots, \mathcal{D}_t$ so that each smaller matrix contains some rows from $\mathcal{D}$. We can then distribute matrices $\mathcal{D}_1, \ldots, \mathcal{D}_t$ to independent miners who calculate results based on their part of the data and combine them with results of other miners. Unfortunately this solution does not provide privacy of the data, as for each person who answered the questions some node will have complete data about the person.

2.2.2 Distribution of columns

We can also divide the matrix $\mathcal{D}$ so that matrices $\mathcal{D}'_1, \ldots, \mathcal{D}'_t$ contain columns from $\mathcal{D}$. This allows us to keep the data identifying the person in a separate database from the sensitive data. Such a solution decreases the usability of the data, because one miner has access only to some attribute. For example, this could keep us from finding reliable aggregations or association rules based on all relevant attributes, because some of them might not be available for a given miner.

2.2.3 Distribution of values

It is important to notice, that the two previous solutions are not secure.

Our solution does not alter the dimensions of matrix $\mathcal{D}$. Instead we distribute the values of the matrix between three miners so that no single miner or a pair of them can find the original value given their parts of this value. Participants will need to trust the miners not to co-operate. This can be achieved if the miners are hosted by organisations who do not trust each other or audit each other's work. Examples are competing companies and government organisations protecting people's privacy.

Secret sharing and multiparty computation is used in this scenario. Participants will process values and send each miner only the respective part of the input values. Miners can calculate aggregation results if all three of them work together. The results are based on the complete data, but no miner has a complete row of the input data.

3 Prerequisites

3.1 Secure multi-party computation

3.1.1 Main idea

Assume that we have n participants $p_1, \ldots, p_n$ and each participant i knows an input value x_i [1]. Secure multi-party computation is the calculation of a function $f(x_1, \ldots, x_n) = (y_1, \ldots, y_n)$ in such a way, that the output is correct and the inputs of the nodes are kept private. Each node i will get the value of y_i and nothing else.

One of the methods of achieving secure multi-party computation is verifiable secret sharing. Consider the scenario in which a dealer shares a value s between n nodes. The dealer and the nodes may be malicious. If the dealer is honest, the nodes can gain no information about the value. Honest nodes can reconstruct s even in the presence of malicious nodes.

3.1.2 Modelling the adversary

We use malicious nodes to model the adversary. The adversary may corrupt any number of nodes before the protocol starts. In the beginning of the protocol the honest players do not know, which nodes are corrupted and which are not.

In the case of a passive corruption the adversary can read all the data held, sent and received by the node. If the corruption is active, the adversary has complete control over the node. If the adversary is static, the set of corrupted nodes remains the same for the whole duration of the protocol. The adversary may also be adaptive and corrupt new nodes during the execution of the protocol.

We must limit the adversary to keep secure protocols possible. If all nodes are corrupted, we have no hope to complete the calculation of f. Therefore we restrict the adversary to corrupting only a proper subset of all the nodes.

3.1.3 Models of communication

There are two main communication models used in secure multi-party computation. They are the cryptographic model and the information-theoretic model. In the cryptographic model the adversary is provided with read-only access to all the

messages from the traffic between honest nodes. The adversary may not modify the messages. In the information-theoretic model all nodes have private channels between them.

The cryptographic model is secure if the adversary cannot break the cryptographic problem and read the messages. The information-theoretic model is stronger, as even a computationally unbounded adversary can not read the messages exchanged between honest nodes.

Communication can be synchronous or asynchronous. In the synchronous mode nodes have synchronised clocks which allows us to design protocols with rounds. Messages sent each round will be delivered before the next round begins. An adaptive adversary may decide to corrupt nodes in the adversary structure, but only if they are in the respective adversary structure.

The asynchronous model is more complex as it has no guarantees on message delivery time. If we do not have guarantees for message delivery, we can not demand that the protocol reaches a certain step at all.

3.1.4 Example

A classical problem in multi-party computation is the millionaire problem. Assume that Alice and Bob are millionaires who would like to know, which one of them is richer without revealing their exact wealth.

We have two participants. Let Alice be p_1 and Bob be p_2. Let Alice's wealth be x_1 and Bob's wealth be x_2. The function we need to evaluate is "greater than", that is, we need to find out, if $x_1 > x_2$ without Alice knowing x_2 or Bob knowing x_1.

There are various solutions to this problem. The classic one was presented together with the problem intruduction by Yao [2].

3.2 Secret Sharing

3.2.1 Introduction

Secret sharing is used to keep values such as cryptographic keys secure [3]. An algorithm is used to distribute the value between n nodes, so that each node gets one share and there is another algorithm which will reconstruct the original value when given the shares of all nodes.

3.2.2 Threshold secret sharing scheme

Assume that we have an input value s from a finite set S that we wish to keep secret. Also consider that we have n nodes available for computation.

A threshold secret sharing scheme is a probabilistic algorithm $\mathcal{S}$ which takes s as the input and outputs n bitstrings $s_1, \ldots, s_n$. Values $s_1, \ldots, s_n$ are called shares. The secret sharing scheme has a threshold $t \in \mathbb{N}, 0 < t < n$. The adversary may gain access to up to t shares without learning anything about the value s. If more than t shares are available, s can be calculated.

We will define privacy and correctness as follows.

Privacy: Assume that we have run $\mathcal{S}$ on an input value $s \in S$ and created n nodes $s_1, \ldots, s_n$. The secret sharing scheme is secure if for each subset $K \in P(\{1, \ldots, n\}), |K| \leqslant t$ the probability distribution of $\{s_k | k \in K\}$ is independent of the one of s.

Correctness: Assume that we have run $\mathcal{S}$ on an input value $s \in S$ and created n nodes $s_1, \ldots, s_n$. The secret sharing scheme is correct if for each subset $L \in P(\{1, \ldots, n\}), |L| \geqslant t + 1$ the value s is determined by values $\{s_l | l \in L\}$ and there is an efficient algorithm for calculating s based on these values.

3.2.3 Share conversion

Secret sharing is a technique used for converting shares of the same secret from one sharing scheme to a different one. This can be used to protect a system against malicious attacks by participants who do share calculation. Other participants may convert the shares to another scheme and still retrieve the original value.

3.2.4 Example

Follows a classic implementation by Shamir [4].

Assume that we have an input value s, n nodes and we want a threshold value t. We pick a prime p so that $p > n$. The algorithm $\mathcal{S}$ consists of the following steps:

1. Choose a random polynomial $f(x)$ over $\mathbb{Z}_p$ with a degree at most t so that $f(0) = s$. A suitable polynomial is $f(x) = s + a_1 x + a_2 x^2 + \ldots + a_t x^t$ where $a_1, \ldots, a_t$ are randomly selected elements of $\mathbb{Z}_p$

2. Distribute values $s_i = f(i) \bmod p$ $(i = 1, \ldots, n)$ as shares to nodes $1, \ldots, n$.

Each of the shares can be considered as a point on the curve determined by the polynomial $f(x)$. If we have t or more points of the curve, we can rebuild the polynomial by using Lagrange interpolation. If we have less than t shares available, we will not get a polynomial of the required degree so we can not restore the original secret.

A share conversion solution for the Shamir scheme is descibed by Cramer, Damgård and Ishai [5].

4 A system for secure data aggregation

4.1 General model

Let $M_1, \ldots, M_3$ be the miners. Let $W_1, \ldots, W_k$ be the participants. The participants collect data, send it to the miners and order the miners to perform aggregations on the data. The miners are responsible for storing the received data and calculating aggregations.

Data is shared between the three miners. Distributing data into shares is handled by the participants. The miners receive shares and store them in their database. If we could combine the databases of the miners, we would get the original database.

If the participants are honest, the miners have no way of retrieving the original value without co-operating with the other two miners. If the miners are honest, the participants have no way of seeing other participants' data. We can also prevent malicious input by dishonest participants by using share conversion.

To run aggregations on the data we implement the necessary share computation operations as three-way protocols between the miners. Aggregation is done by combining these operations. We have implemented two operations — adding a row to the miner database and multiplying values in the database.

4.2 Data storage

The participants have input data where each record is in the form of a vector with n elements. Each miner M_i has a database $\mathcal{D}_{M_i}$ which is a matrix of size $m \times n$. Data elements in the system are members of $\mathbb{Z}_{2^{32}}$.

4.3 Adding a row to the miner database

Assume that we have a random number generator **RNG**. The participant W_t has a record with new values it wants to add to the miners' database. It is represented by a vector R ($|R| = n$). It creates three new vectors R_1, R_2 and R_3 and calculates their values as follows:

$$\forall i = 1, \ldots, n \; R_1[i] \leftarrow \mathbf{RNG}, R_2[i] \leftarrow \mathbf{RNG}, R_3[i] = R[i] - R_1[i] - R_2[i] \quad (1)$$

W_t sends each vector R_i to the respective node M_i ($i = 1, 2, 3$). The miner node M_i adds the values of R_i to its database $\mathcal{D}_{M_i}$ as a new row.

All three vectors are required to restore the original value, because individual miners see only a value that is random to them. Two values are also not enough, because one of them is random and hence the distribution of their sum is also

uniform. Three values are enough, because according to (1)

$$\forall i = 1, \ldots, n \; R_1[i] + R_2[i] + R_3[i] = R[i].$$

4.4 Share multiplication

Let there be miners $M_1, \ldots, M_3$ and their databases $\mathcal{D}_{M_1}, \ldots, \mathcal{D}_{M_3}$. We want to find the product of two values in the database and store it in the same database. Note, that the input values are distributed into three shares and we need the result also in shares.

Assume that the first value is in the k-th row and l-th column and the second value is in the m-th row and n-th column of the database matrix. The values are $x = x_A + x_B + x_C$ and $y = y_A + y_B + y_C$, where x_i = element from the k-th row and l-th column of the matrix $\mathcal{D}_{M_i}$ and y_i = element from the m-th row and n-th column of the matrix $\mathcal{D}_{M_i}$.

To calculate $x \cdot y$ we use the following protocol. We name the nodes M_1, M_2 and M_3 Alice, Bob and Charlie respectively.

Round 1: Sharing randomness

- Alice generates $r_{12}, r_{13}, s_{12}, s_{13} \leftarrow \mathbf{RNG}$
- Bob generates $r_{23}, r_{21}, s_{23}, s_{21} \leftarrow \mathbf{RNG}$
- Charlie generates $r_{31}, r_{32}, s_{31}, s_{32} \leftarrow \mathbf{RNG}$
- All values $*_{ij}$ are sent over a secure channel from M_i to M_j

Round 2: Sharing shares

- Alice computes $\hat{a}_{12} = x_A + r_{31}$, $\hat{b}_{12} = y_A + s_{31}$, $\hat{a}_{13} = x_A + r_{21}$, $\hat{b}_{13} = y_A + s_{21}$
- Bob generates $\hat{a}_{23} = x_B + r_{12}$, $\hat{b}_{23} = y_B + s_{12}$, $\hat{a}_{21} = x_B + r_{32}$, $\hat{b}_{21} = y_B + s_{32}$
- Charlie generates $\hat{a}_{31} = x_C + r_{23}$, $\hat{b}_{31} = y_C + s_{23}$, $\hat{a}_{32} = x_C + r_{13}$, $\hat{b}_{32} = y_C + s_{13}$
- All values $*_{ij}$ are sent over a secure channel from M_i to M_j

Round 3: Local computations

- Alice computes $c_A = x_A\hat{b}_{21} + x_A\hat{b}_{31} + y_A\hat{a}_{21} + y_A\hat{a}_{31} - \hat{a}_{12}\hat{b}_{21} - \hat{b}_{12}\hat{a}_{21} + r_{12}s_{13} + s_{12}r_{13}$

- Bob generates $c_B = x_B\hat{b}_{32} + x_B\hat{b}_{12} + y_B\hat{a}_{32} + y_A\hat{a}_{12} - \hat{a}_{23}\hat{b}_{32} - \hat{b}_{23}\hat{a}_{32} + r_{23}s_{21} + s_{23}r_{21}$
- Charlie generates $c_C = x_C\hat{b}_{13} + x_C\hat{b}_{23} + y_C\hat{a}_{13} + y_A\hat{a}_{23} - \hat{a}_{31}\hat{b}_{13} - \hat{b}_{31}\hat{a}_{13} + r_{31}s_{32} + s_{31}r_{32}$

After running this protocol, the miners have calculated the product

$$xy = (c_A + x_A y_A) + (c_B + x_B y_B) + (c_C + x_C y_C). \quad (2)$$

The correctness of the protocol can be shown by expanding both sides of equation (2) and showing that they are equal. The protocol is secure, since in every round all miners see values with a uniform distribution. The computation requires altogether 24 messages, each miner sends and receives 8 messages.

5 Software specification

5.1 Overview

Nodes run two kinds of software. Mining nodes run the miner software and participants run the controller software. The miner software consists of algorithms and protocols which perform the multi-party computation and data mining tasks. The controller software is used to send data and commands to the miner software.

We consider two separate implementations of the system. The first is a framework for algorithm testing and development and the second is a prototype of a database engine. The systems differ in purpose, security features and performance.

5.2 Development framework

5.2.1 Overview

The development framework provides the user with tools for implementing multi-party computation solutions in our model. It is essentially a distributed virtual processor which has a memory and instruction scheduler. This processor is wrapped into a function library, which can be used by client programs.

The system is self-organising. The miners are generic processes which are configured by the client program during system initialisation. This kind of a self-organising network makes developing easier, because running the system requires less manual configuration.

The miners can send any data back to the client program. This includes the contents of the database. This helps the programmer develop and debug algorithms. Automated tests are possible, because the client can ask the miners for intermediate results.

The software is implemented in the C++ programming language. It makes use of the RakNet network library for communication[1]. The system is designed to be cross-platform. Development is done on Apple Mac OS X and Microsoft Windows. Linux/UNIX versions are planned.

5.2.2 Communication

The communication between nodes is message-based. During system setup the client application locates all three miners and assigns node numbers 1, 2 and 3 to them. After that the miners connect to each other. At the end of start-up all nodes have communication channels to other nodes in the system.

The miners in the development framework are not designed to support multiple client applications at the same time.

5.2.3 Storage

The client programs have access to three kinds of storage. Each miner M_i has a database $\mathcal{D}_i$, a stack $\mathcal{S}_\rangle$ and a heap $\mathcal{H}_\rangle$. The database is used for persistent storage. The stack and heap are used during runtime for temporary storage. Contents of the stack and the heap are forgotten when miners stop or restart.

The stack is used for passing parameters to instructions. For example, the share multiplication operation could pop two values from the stack, multiply them, and push the result back onto the stack top. The stack provides standard methods for pushing, peeking and popping. It also provides random access to its data so that vector operations will find the start of their parameter list. The heap is used to store intermediate results in algorithms. The heap is implemented as an index-addressed array of elements.

Protocols also have internal storage for intermediate results.

5.2.4 Instruction scheduler

The miners recognise a number of pre-defined commands. The commands are divided into the following categories:

1. system operations — managing the database, modifying miner configuration
2. data transfer — exchanging data between the client program and the miners
3. computation — performing calculations with the distributed database

[1] RakNet — a reliable cross-platform network library. URL: http://www.rakkarsoft.com/

The scheduler processes one instruction at a time. The order is determined by the arrival of messages containing the instruction code. Parameter passing is handled by a simple and generic scheme. Parameters are not sent together with the instruction code but rather as data values from client node to miner node. The client sends the instruction code in one message and the parameters in the others. The miner, when processing the instruction, first waits for the parameters to arrive.

5.3 Prototype database

In the future work we will investigate the feasibility of building an actual prototype database platform which could be used to process sensitive data. Such a system will have much stricter security requirements than the development framework.

To avoid collusion by the miners, they will have to be controlled, configured and hosted by different organisations. This means that the miners will be a lot more independent and clients will have almost no control over their operation.

In a production environment the miners can not give out any information which could compromise the sensitive data. This means that under no circumstances can miners send raw shares to the clients. The miners will also need some logic which would determine, whether the results of an aggregation query reveal too much about the source data. In that case the miners will refuse to process the query.

The miners will have to work as a standard database server. They will have to support and distinguish multiple simultaneous clients at the same time. The miners' protocols will be more specific and optimised to perform the queries as fast as possible. Advanced scheduling and caching techniques have to be considered.

References

[1] Ronald Cramer and Ivan Damgård. Multiparty computation, an introduction. Course Notes, 2002.

[2] Ronald Cramer, Ivan Damgård, and Yuval Ishai. Share conversion, pseudorandom secret-sharing and applications to secure computation. *Lecture Notes in Computer Science*, 3378:342–362, 2005.

[3] Ivan Damgård. Secret sharing. Course notes, 2002.

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

[5] Andrew Chi-Chih Yao. Protocols for secure computations (extended abstract). In *Proceedings of the 21st Annual IEEE Symposium on the Foundations of Computer Science*, pages 160–164, 1982.