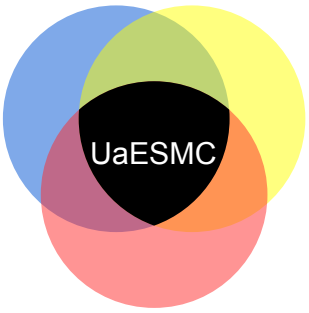

Project N°: **FP7-284731**

Project Acronym: **UaESMC**

Project Title: **Usable and Efficient Secure Multiparty Computation**

Instrument: **Specific Targeted Research Project**

Scheme: **Information & Communication Technologies**

Future and Emerging Technologies (FET-Open)

Deliverable D2.2.1
Advances in Secure Multiparty Protocols

Due date of deliverable: 31st January 2013

Actual submission date: 31st January 2013

Start date of the project: **1st February 2012** Duration: **36 months**

Organisation name of lead contractor for this deliverable: **CYB**

Specific Targeted Research Project supported by the 7th Framework Programme of the EC		
Dissemination level		
PU	Public	✓
PP	Restricted to other programme participants (including Commission Services)	
RE	Restricted to a group specified by the consortium (including Commission Services)	
CO	Confidential, only for members of the consortium (including Commission Services)	

Executive Summary:
Advances in Secure Multiparty Protocols

This document summarizes deliverable D2.2.1 of project FP7-284731 (UaESMC), a Specific Targeted Research Project supported by the 7th Framework Programme of the EC within the FET-Open (Future and Emerging Technologies) scheme. Full information on this project, including the contents of this deliverable, is available online at `http://www.usable-security.eu`.

This report contains a review of the advances in secure multiparty protocols made during the first year of the UaESMC project. We describe our results obtained while searching for a solution for each of the problems we decided to tackle.

First, we look at statistical analysis. We begin by giving possible privacy-preserving solutions for such basic database operations as data collection and storage, selection, join and sorting. These operations are often used especially when statistically analysing data. We go on to talk about the most used statistical measures—five-number summary, mean, variance and standard deviation—that can give the analyst an overview of the data they are not allowed to see. This gives the analyst an opportunity to get information for further studies on the data. We also decided to start looking into statistical tests by tackling the simple two-sample Student's t-test. Analysts have also expressed concern about the quality of entered data and how it is difficult to do data cleaning if they cannot see the data. To address this concern, we also started to look into how outlier detection can be done in a privacy preserving way. We end this chapter by looking into secure data classification on our chosen platform Sharemind.

Second, we look at privacy-preserving optimization. We start by looking at privacy-preserving linear programming and then move on to genetic algorithms (GA) to help us solve the secure subset covering problem and to find the shortest path in a graph. We deduce that GAs are suitable for the first task, but rather less suitable for the second one, at least in its usual linear-programming formulation.

Finally, we look at privacy-preserving network management. We require identifying links among vertices in a network. These links can represent routing paths or intrusion detection events. This problem can be formally phrased as finding the transitive closure of a graph. We propose an algorithm to solve this problem and describe three scenarios that are possible at the end of the algorithm. We intend to compare the performance of these scenarios to find the one that is most feasible.

List of Authors

Dan Bogdanov (CYB)
Liina Kamm (CYB)
Alisa Pankova (CYB)
Jan Willemson (CYB)
Roberto Guanciale (KTH)
Peeter Laud (CYB)
Riivo Talviste (CYB)

Contents

Chapter 1

Introduction

This report contains a review of the advances in secure multiparty protocols made during the first year of the UaESMC project. Deliverable D5.2.1 [4] gives the descriptions of the different problem categories that were chosen as a result of Deliverable D1.2 [21] as Milestone 1. They are the following:

- Privacy-preserving statistical analysis of databases,
- Privacy-preserving optimization,
- Privacy-preserving network management and operation.

A thorough description and the reasoning behind the choices is given in Deliverable D4.1 [11].

We describe our results obtained while searching for a solution for each of the problems we decided to tackle. We give a reasoning why we decided to address the specific concerns, what techniques we tried and what was developed as a result.

In addition, we implement prototypes as proposed solutions for chosen problems. These are described in Deliverable D5.1.1 [5]. Based on the benchmark results of these implementations, we can find shortcomings and word possible improvements.

In the following chapters we talk about the concrete sub-problems of the problems from Milestone 1. In Chapter 2, we talk about privacy-preserving statistical analysis of databases, in Chapter 3, privacy-preserving optimization and in Chapter 4, privacy-preserving network management and operation. Each of these chapters has subchapters for the specific problem that we addressed.

Chapter 2

Privacy-preserving statistical analysis

The result of this chosen problem category is a statistical suite or API that can help data analysts use secure multiparty computation in their analyses without having to write the code for standard statistics metrics or tests. It is similar to what R provides but meant for computations on shared data. We tackle the application scenarios described in Subsection 2.8 of Deliverable D4.1 [11].

We are building the statistical suite on top of the secure multiparty computation framework Sharemind. This framework, developed over the years in CYB, provides us with a multitude of features, allowing us to concentrate on building the statistics API. The framework provides us with

- a database management system; facilities to load data from persistent storage and write it back again;
- a full-fledged programming language SecreC to express our algorithms in terms of primitive operations;
- protocol suites implementing these primitive operations in secure manner, as well as possibilities to integrate new operations into them.

At the same time, our privacy-preserving statistics operations do not significantly depend on the internal data representations and protocols employed by Sharemind. Hence they could be ported to other frameworks without too much effort.

The main protocol suite of Sharemind stores private values by sharing them additively over $\mathbb{Z}_{2^n}$ among three parties. The framework is flexible in allowing other suites to be plugged in, and used beside the existing one.

2.1 Data collection and sharing

When data is gathered in the secret shared database, metadata must also be added. While the information like data types, attribute count and the amount of data is not hidden in some implementations of secret shared databases, it can be obfuscated to some extent. Unfortunately, to aggregate data, we need to know at least the types of data, as one operations can require different protocols depending on the data type. In the following, we assume that information about the data types and the descriptions of the ontologies, classifiers or domains is available to the analyst. Essentially, this does not leak valuable information.

Sometimes single values are missing from the gathered dataset. One of the first challenges we face is how to deal with missing data. There are two options: either we simply leave the corresponding cells empty, which can leak some information, or we use an extra attribute for each attribute to hold the information. We only need to store one shared bit of extra data per entry. Depending on the implementation of the database, the latter uses extra storage space of $n \cdot a \cdot x$ bits, where n is the number of entries, a is the number of attributes, and x is the smallest data unit that can be stored in the database.

In terms of the representation given in Subsection 2.8.1 of Deliverable D4.1 [11], we store the data of the physical table T as $T' = (T, E, m)$, where m is the identity vector and E contains one axis $\mathsf{ia}(a)$ of

boolean values for each axis a of table T. A value defined by $\mathsf{ia}(a)$ is 0 if the corresponding value in the axis a is missing and 1 otherwise. This information contained in the derived axis $\mathsf{ia}(a)$ will be transferred to the mask vector m.

2.2 Selection

As our aim is to hide the subjects that correspond to a given filter value, we give the result of the selection as a logical table $\sigma(T, p) = T' = (T, E, m)$, where T is the original table, E is empty and m is the mask vector that is 1 if the value is in the selection and 0 otherwise. The mask vector is calculated based on the criterion p and is then multiplied with the existing mask vector of table T that contains information about value availability. This ensures that the availability information is accounted for in later computations. From this, it is possible to compute a logical table $T'' = (S, E, m)$, where E and m are from T', every axis of an observation $o'[i] \in S$ is 0 if $m[i] = 0$ and $o'[i]$ is $o[i] \in T$ if $m[i] = 1$, $i \in range(T)$.

2.3 Join

This Section describes how to perform a privacy-preserving database join operation or to build derived tables according to the introduction of this concept in Section 2.8.3 of Deliverable D4.1 [11]. Let us have two secret-shared tables T_1 and T_2 with m_1 and m_2 rows and n_1 and n_2 columns respectively. In the following, we will obliviously compute $J(T_1, T_2, p)$ and we call the columns used in the predicate p *key columns*. A naive oblivious database join operation on these two tables would first generate full Cartesian product of the two tables (or their key columns) and then apply oblivious comparison for all possible key pairs. Rows with non-matching keys are then removed and the resulting table obliviously shuffled to hide which rows stayed in the joined table. This solution is secure but requires $\Theta(m_1 m_2)$ comparison operations.

2.3.1 Oblivious database join with unique keys

To come up with a more efficient database join algorithm, we consider a specific kind of join, *equi-join*, where the join predicate consists only of equality operation(s), combined using propositional connectives. Let us have a setting where the computing parties obliviously apply pseudorandom permutation π_s to encrypt the key column. As π_s is a pseudorandom permutation (a block cipher depending on an unknown key s) and all the values in the key column are unique, the resulting values look completely random if none of the miners knows π_s. Hence, it is secure to publish all the encryptions of key columns. Moreover, the tables can be correctly joined using the encryptions instead of key values.

However, such a join still leaks some information – parties learn which database table rows in the first table correspond to the database rows in the second table. By obliviously shuffling the rows of initial tables, this linking information is destroyed. The resulting algorithm is depicted as Algorithm 1. Note that in each step all the tables are in secret-shared form. In particular, each computing party performs the actual join operation with its local shares and thus the joined table is created in a secret-shared form.

Note that the last optional step in the algorithm is necessary if there exist some duplicate keys in the key columns. In this case the structure graph of matching keys in tables still leaks, but only with the precision of its isomorphism.

As the actual join operation is performed on public (encrypted) values, the construction works not only for *inner joins* but also for the *left* and *right outer joins*, where either the left or right table retains all its rows, whether a row with a matching key exists in the other table or not. These outer joins are common in data analysis. However, in this case parties must agree on predefined constants to use instead of real shares if the encrypted key is missing.

By combining the secure oblivious AES evaluation[1] and the oblivious shuffle from [17], we get an efficient

[1] We have a yet unpublished implementation of the AES block cipher that works on bitwise shared values and is highly vectorized.

Algorithm 1: Algorithm for performing an equi-join on two tables.

Data: Two secret shared tables T_1 and T_2 with key columns $\boldsymbol{k}_1$ and $\boldsymbol{k}_2$.
Result: Joined table T^*.

1 Parties obliviously shuffle each database table T_i resulting in a new shuffled table T_i^* with a key column $\boldsymbol{k}_i^*$.
2 Parties choose a pseudorandom permutation π_s by generating a shared key s.
3 Parties obliviously evaluate π_s on all shared key columns $\boldsymbol{k}_i^*$.
4 Parties publish all values $\pi_s(k_{ij}^*)$ and use standard database join to merge the tables based on columns $\pi_s(\boldsymbol{k}_i^*)$. Let T^* be the resulting table.
5 If there are some non-unique keys in some key column $\pi_s(\boldsymbol{k}_i^*)$, parties should perform additional oblivious shuffle on the secret-shared table T^*.

instantiation of the Algorithm 1. For all database sizes, the resulting protocol does $\Theta(m_1 + m_2)$ share-computing operations and $\Theta(m_1 \log m_1 + m_2 \log m_2)$ public computation operations.

2.4 Sorting

2.4.1 Principles of privacy-preserving sorting

Privacy-preserving sorting is both a useful tool in statistical processing and an important primitive in other data analysis algorithms. For example, in Section 3.2 we show how sorting is required for implementing certain kinds of genetic algorithms.

We require that sorting is *oblivious* of the data. This means that the sorting algorithm must rearrange the input data into the desired order without being able to learn the values or even their relations with one another.

The insight that helps us solve this problem comes from the theory of sorting networks. A sorting network is an abstract structure that consists of several layers of comparators that change the positions of incoming values. These comparators are also called CompEx (compare-and-exchange) functions. A CompEx function takes two inputs, compares them according to the required condition and exchanges them, if the comparison result is true. The following mathematical representation shows a CompEx function for sorting numeric values in the ascending order.

$$\mathsf{CompEx}(x, y) = \begin{cases} (y, x), & \text{if } x > y \\ (x, y), & \text{otherwise.} \end{cases}$$

Basically, a sorting network is an arrangement of CompEx functions so that if the comparators of all the layers of the sorting network are applied on the input data array, the output data array will be sorted according to the desired condition. For a more detailed explanation of sorting networks with examples, see [15].

2.4.2 Implementing sorting networks obliviously

A sorting network is suitable for oblivious sorting, because it is static and independent of the incoming data. One network will sort all possible input arrays, making the approach inherently oblivious. Furthermore, sorting networks are relatively straightforward to implement using secure multiparty computation, as we only need a Min (minimum of two values) and Max (maximum of two values) operation to sort numeric data. Using these two operators, we can easily implement the CompEx function as a straight line program (one with no conditional branches). For an example of a CompEx function that sorts an array of numbers in an ascending order, see the following formula.

$$\mathsf{CompEx}(x, y) = (\mathsf{Min}(x, y), \mathsf{Max}(x, y))$$

Algorithm 2: Algorithm for sorting an array of integers.

Data: Input array $\mathcal{D} \in \mathbb{Z}_n^k$ and a sorting network $\mathcal{N} \in \mathbb{L}^m$.
Result: Sorted output array $\mathcal{D}' \in \mathbb{Z}_n^k$.

```
foreach 𝕃_i ∈ 𝒩 do
    foreach (x, y) ∈ 𝕃_i do
        (𝒟_x, 𝒟_y) ← CompEx(𝒟_x, 𝒟_y)
    end
end
```

We express a k-element array of n-bit integers as $\mathbb{Z}_n^k$. We will assume that the input and the output of the sorting network are in this form. We also need to represent the structure of that network. Intuitively, a sorting network consists of several layers of CompEx functions. The inputs of each CompEx function can be encoded with their indices in the array. Therefore, we will represent a layer $\mathbb{L}_i$ consisting of ℓ_i CompEx functions as follows.

$$\mathbb{L}_i = (\mathbb{N} \times \mathbb{N})^{\ell_i}$$

The complete sorting network, consisting of m layers, will then be written as $\mathbb{L}^m$. We also add one restriction to the network for efficiency and simplicity. We assume that no index appears more than once in each individual layer of the sorting network.

Algorithm 2 presents a general algorithm for evaluating a sorting network in this representation. Note that we can use the same array $\mathcal{D}$ for storing the results of the compare-exchange operation, because according to our assumption above, as a single layer does not use the same array index twice.

This algorithm is easy to implement securely, because we only have to provide a secure implementation of the CompEx operation. The rest of the algorithm is independent of the data, and can be implemented with public operations.

We intentionally omit guidance on how to implement a generator for sorting networks, as this is a well-researched area. We refer the reader to the classical work of Knuth as a starting point [15].

2.4.3 Sorting more complex data structures

We often need to sort data in other form as arrays. For example, we may want to sort a table of values according to a certain key column. In this case, we need to redefine the CompEx operation to work on the full data structure.

Let's consider the case where we need to sort a table of integer values according to a certain column. Algorithm 2 still works perfectly, but we need to design a new kind of a compare-exchange function that evaluates the comparison condition on the value from the respective column, but performs the exchange on the whole table row.

Assume that our input data table is in the form of a matrix $\mathcal{D}_{i,j}$ where $i = 1 \ldots k$ and $j = 1 \ldots l$. Then, the CompEx needs to compare and exchange two input arrays $\mathcal{A}, \mathcal{B} \in \mathbb{Z}_n^k$ according to the comparison result from column c. Equation (2.1) shows the definition of such a function.

$$\mathsf{CompEx}(\mathcal{A}, \mathcal{B}, c) = \begin{cases} (\mathcal{B}, \mathcal{A}), & \text{if } \mathcal{A}_c > \mathcal{B}_c \\ (\mathcal{A}, \mathcal{B}), & \text{otherwise.} \end{cases} \tag{2.1}$$

A suitable oblivious implementation for a CompEx shown in Equation (2.1) on this structure is given in Algorithm 3. The algorithm uses two steps. First, it performs an oblivious comparison part of CompEx. In the given example, it evaluates a greater-than comparison. The main important thing here is that the result should be expressible as either 0 or 1 so that it can later be used in the oblivious exchange. The second step is to obliviously exchange the input data based on the result of the comparison.

This algorithm has the following assumptions for oblivious implementation.

Algorithm 3: Algorithm for obliviously comparing and exchanging two rows in a matrix.

Data: Two input arrays $\mathcal{A}, \mathcal{B} \in \mathbb{Z}_n^k$, column index $c \in \{1 \dots k\}$.
Result: Pair of arrays $(\mathcal{A}', \mathcal{B}') = \mathsf{CompEx}(\mathcal{A}, \mathcal{B}, c)$.
`/* Compute result of the condition */`
1 $b \leftarrow \begin{cases} 1, & \text{if } \mathcal{A}_c > \mathcal{B}_c \\ 0, & \text{otherwise.} \end{cases}$
`/* Exchange the vectors based on the condition */`
2 **foreach** $i \in 1 \dots k$ **do**
3 $\quad \mathcal{A}'_i = (1-b)\mathcal{A}_i + b\mathcal{B}_i$
4 $\quad \mathcal{B}'_i = b\mathcal{A}_i + (1-b)\mathcal{B}_i$
5 **end**

1. We can obliviously implement the comparison operation on input data so that the result is represented as a numeric zero or one.
2. We can subtract the comparison result from the constant 1.
3. We can cast the numeric zero-one result (or the subtraction result) to a type that can be multiplied with the input data type.
4. We can add two values of the input data type.

Fortunately, these assumptions hold for different secure computation paradigms. It is relatively easy to implement such an oblivious CompEx function with secure multiparty computation on different integer sizes.

2.4.4 Optimization using parallel operations

We now show how to optimize the implementation of the proposed algorithms using parallel operations on multiple values. Such SIMD (single instruction, multiple data) operations are very efficient on most secure multiparty computation paradigms. For example, secure multiparty computation protocols based on secret sharing can put the messages of many parallel operations into a single network message, saving on networking overhead.

If we observe the Algorithms 2 and 3, we see that it is trivial to make them use vector operations. First, let's consider the general sorting algorithm given in Algorithm 2. Note that the outer loop of that algorithm can not be vectorized by replacing it with parallel operations. The intuitive reason is that every layer of the sorting network is directly dependent on the output of the previous layer and this does not allow multiple layers to be processed in parallel.

However, thanks to the assumption on the uniqueness of indices we made while describing the structure of the sorting network, we can trivially vectorize the inner loop. Indeed, we can replace the entire inner loop with two operations. One computes the maximum values of all input pairs and the other computes the minimal values.

The same approach works also for if we want to implement sorting on matrices with the CompEx function in Algorithm 3. The comparison operator can again be performed as a single parallel comparison, assuming that the secure multiparty computation system provides such a protocol. The flipping of the comparison results, the multiplications and the additions can all be performed in parallel, allowing also this function to be efficiently obliviously implemented.

2.5 Simple statistical measures

Many interviewees (see [21]) emphasized the need to see the data before analyzing it in order to gain an understanding of the cohort and to find out which tests would be interesting to conduct. With privacy-

Algorithm 4: Algorithm for finding the median and quartiles of an axis.

Data: Input axis a of table T and corresponding mask vector m.
Result: Minimum $min(T, a)$, lower quartile $lq(T, a)$, median $me(T, a)$, upper quartile $uq(T, a)$ and maximum $max(T, a)$

1 Parties obliviously shuffle the pair of vectors (a, m)
2 $\mathsf{Declassify}(m)$
3 $a' = (a[i] \mid m(i) = 1, i \in range(T))$
4 $n \leftarrow |a'|$
5 $b \leftarrow \mathsf{Sort}(a')$
6 $min = b[1]$
7 $max = b[n]$
8 $L_{lq} \leftarrow n * 0.25$
9 $L_{me} \leftarrow n * 0.5$
10 $L_{uq} \leftarrow n * 0.75$
11 **foreach** $\alpha \in \{lq, me, uq\}$ **do**
12 $\quad \alpha \leftarrow \begin{cases} \frac{b[L_\alpha]+b[L_\alpha+1]}{2}, & \text{if } L_\alpha \in \mathbb{N} \\ b[\lceil L_\alpha \rfloor], & \text{otherwise.} \end{cases}$
13 **end**

preserving computations and datasets, the aim is to keep the data secret so that no-one has access to individual values. Therefore, we cannot give an analyst direct access to the data.

However, if dealing with large datasets, the analyst cannot grasp the data by looking at the individual values anyway. They will perform simple measurements on the data and draw plots and graphs to get an overview of the characteristics. We claim, that given access to common aggregate values, there is no real need to see the individual values. But in order to support the analyst, we have to provide secure implementations of the simple statistics he/she might be interested in.

Count

Based on the data representation described in Subsection 2.1, it follows that the count of values in an axis of table T' is the sum of elements in the corresponding boolean value axis defined by E.

Sum

The sum (Sum) of an axis of values is computed by simply adding all the values in the axis together.

Five-number summary

The five-number summary is a descriptive statistic that includes the minimum, lower quartile, median, upper quartile and maximum of a vector. This statistic can be quite revealing and can leak information as we are, in some cases, outputting a single value instead of an aggregated value. Hence, these statistics must be used with care.

Algorithm 4 describes how to find the five-number summary of an axis a in table $T' = (T, E, m)$, where T is the original table, E is empty and m is the mask vector. We look at this problem as if there is already a mask filter present and we assume that the 'is available' axis previously calculated by E is already incorporated into the mask vector. If it is known that all values in the axes exist and there are no mask vectors applied, we simply assume that m is the identity vector.

We begin by extracting the values that belong to the set defined by the mask vector m. Note, that we are only trying to find the quartiles of one axis a of table T. We begin by shuffling the pair (a, m) using the oblivious shuffle protocol from [17]. Next we declassify the mask vector m and make a new vector a' by

selecting only the values that are indicated by m: $a' = (a[i] \mid m(i) = 1, i \in range(T))$. During this, a and, hence, a' remain secret shared. Now we have a vector of values for the current selection.

We go on by obliviously sorting the obtained vector ($b \leftarrow \mathsf{Sort}(a')$). We assume that we have a sorting interface that handles the public sorting network generation for us when we provide it with the vector we want to sort. We count the elements in a' and compute the initial indices for the lower quartile L_{lq}, median L_{me} and upper quartile L_{uq} by multiplying the element count with 0.25, 0.5, and 0.75, respectively.

The quartiles $\alpha \in \{lq, me, uq\}$ are chosen from the sorted vector b based on the initial indices L_α:

$$\alpha \leftarrow \begin{cases} \frac{b[L_\alpha]+b[L_\alpha+1]}{2}, & \text{if } L_\alpha \in \mathbb{N} \\ b[\lceil L_\alpha \rfloor], & \text{otherwise} \end{cases} .$$

If the index L_α is a natural number, the value lies between L_α and $L_\alpha + 1$. If not, then the index L_α is rounded to the nearest integer ($\lceil L_\alpha \rfloor$).

The five-number summary can be graphically represented as a box plot that gives a good overview of the data. If the data is separated into different classes (defined by mask vectors), the box plots based on the summaries calculated from these classes can be used for gaining a visual idea of what the data distribution looks like in each class. This can be used, for example, for gaining an overview before carrying out Student's t-test.

Mean, variance and standard deviation

Let n be the number of subjects in the cohort. The most common measures for data are the arithmetic mean, variance and standard deviation. These measures require the system to be able to handle private addition, multiplication, division and square root. If the size n of a chosen cohort is public, we can use division with a public divisor instead. This operation is faster and less complex than division with a private divisor. Depending on whether these values can be published, we can also use public square root instead of its private counterpart. As with all private computations, taking the square root of a private value is considerably slower than the public version of this protocol. If protocols for these operations exist (they do in the Sharemind framework), the implementation of these statistics is quite straightforward.

2.6 Comparison of two populations

Firstly we want to compare the means of two populations. For this we chose to to study the feasibility of the private t-test. The t-test is a statistical hypothesis test based on the t-distribution.

2.6.1 Filtering the data

To perform the t-test, we first need to distribute the data into two groups based on some condition. There are two slightly different ways of doing this: on one hand, we can choose the subjects in one group and assume all the rest are in group two (e.g., people with high blood pressure and everyone else); on the other hand, we can choose subjects into both groups (e.g., men who are older than 65 and have high blood pressure and men who are older than 65 who do not have high blood pressure). There is a clear difference between these selection categories and they yield either one or two mask vectors. In the former case, we calculate the second mask vector by flipping all the bits in the existing mask vector.

To calculate the means, variances and standard deviations, we first multiply point-wise the axis a with the mask vector m so that the values that do not belong to the population do not interfere with the calculations. The count of elements can be found by calculating the sum of values in the mask vector.

2.6.2 Computing the test statistic

In the following, let $s_i^2 = \mathcal{VAR}(R_i, a)$ and $\overline{x_i} = \mathcal{M}_a(R_i, a)$, where a is the axis we want to test, $R_i = (T, E, m_i)$, T is the original table, E is empty and m_i is the mask vector for population $i \in \{1, 2\}$. Let $n_i = \mathsf{Sum}(m_i)$ be the count of subjects in population $i \in \{1, 2\}$.

We look at two cases. Firstly, we assume that the variances of both populations are equal, and secondly we assume that the variances are different. If the variances of both populations are equal, we calculate an estimator of the common variance of the two populations as

$$s^2 = \frac{(n_1 - 1)s_1^2 + (n_2 - 1)s_2^2}{n_1 + n_2 - 2} \ .$$

The t-test statistic is

$$t = \frac{\overline{x_1} - \overline{x_2}}{s} \sqrt{\frac{n_1 n_2}{n_1 + n_2}} \ .$$

Secondly, If the variances of the populations are not equal, we calculate the estimate of the standard deviation as:

$$s_{\overline{x_1} - \overline{x_2}} = \sqrt{\frac{s_1^2}{n_1} + \frac{s_2^2}{n_2}} \ ,$$

where s_i^2 is the unbiased variance of the population $i \in \{1, 2\}$. The t-test statistic is

$$t = \frac{\overline{x_1} - \overline{x_2}}{s_{\overline{x_1} - \overline{x_2}}} \ .$$

If the null hypothesis is supported, the t-test statistic follows Student's t-distribution with $(n_1 + n_2 - 2)$ degrees of freedom.

Similarly to the simple statistic measures, this kind of hypothesis testing requires the system to be able to handle private addition, multiplication, division and square root. The same argumentation of public versus private computations applies in this case as well.

2.6.3 Evaluating the significance of the statistic

In addition, we may make a second branching in our solution depending on whether the sizes of the two cohorts are public or not. The t-test requires us to compare the calculated t-test statistic to the t-distribution values based on the degrees of freedom. If this value is public and the test statistic can be made public as well, the analyst can check the t-distribution table to find out whether the calculated result was significant or not.

However, if these values cannot be published for some reason, we have to consider another approach. Namely, we can pre-encode the t-distribution table and use oblivious lookup algorithms to check whether our calculated t-statistic is significant or not.

2.7 Outlier detection

We elaborate on how to do outlier detection in oblivious manner. That means that once we have identified the outliers we cannot show the data analyst which data points are classified as outliers nor can we show how many outliers there are. Because of that we cannot actually throw out the outliers from the dataset. Instead, the outlier detection algorithm should produce a (boolean) mask vector indicating which data points are outliers and which are not. Before using the data vector in a computation, we can multiply it with the mask vector rendering outliers to zeroes (or other suitable values) that are not taken into account in further computations.

There are many different outlier detection methods and which one to use is highly dependent on the properties of the dataset. In this Section, we will only consider univariate outlier detection mechanisms. For example, one of the most simple outlier detection methods is to just remove the minimal and maximal $n\%$ of the data values.

2.7.1 Median absolute deviation

Another robust measure of detecting outliers that does not depend too much on the specific data scheme is to pick a property that captures the shape of the concrete dataset. Traditionally, sample mean and sample variance are used for this, but these properties are highly influenced by the existence of outliers. So using them to detect outliers may give suboptimal results. Instead, Hampel [12, 13] suggests to use median and median absolute deviation (MAD) as the properties to describe a given dataset and detect possible outliers. For a dataset X, its element n is considered an outlier if

$$median - n > \lambda \cdot \mathsf{MAD}, \tag{2.2}$$

where

$$\mathsf{MAD} = median_i(|X_i - median_j(X_j)|)$$

and λ is a constant. The exact value of λ depends on the dataset, but anything from 3 to 5 could be used as a generic starting value.

As MPC protocols are often network-bound, we would like to have algorithms that are easily parallelizable so we can use SIMD style operations. This algorithm is a good candidate for this as we do not have to compute everything one element at a time. For a given dataset we can compute the median and MAD once and use them as constants in equation 2.2 so classifying a given element becomes an evaluation of an inequality with a constant. Hence, it is possible to classify all elements of a dataset at once.

2.7.2 Local outlier factor

Breunig *et al.* [8] have found that in many scenarios assigning an object a degree of being an outlier works better than just binary classification. This degree is called local outlier factor (LOF) of an object. As the name implies, this degree is local in the sense that it expresses how isolated the given object is. To this end, LOF shares some principles with density-based clustering algorithms.

Here we will summarize the main notions of LOF, for an in depth overview refer to the original paper [8].

Definition 2.7.1 (k-distance of an object p)**.** *For any positive integer k, the k-*distance *of object p, denoted as k-*distance*(p), is defined as the distance $d(p, o)$ between p and an object $o \in D$ such that*

1. *for at least k objects $o' \in D \setminus \{p\}$ it holds that $d(p, o') \leq d(p, o)$; and*
2. *for at most $k - 1$ objects $o' \in D \setminus \{p\}$ it holds that $d(p, o') < d(p, o)$.*

Definition 2.7.2 (k-distance neighborhood of an object p)**.** *Given the k-*distance *of object p, the k-distance neighborhood of p contains every object whose distance from p is not greater than the k-*distance*: $N_k(p) = \{q \in D \setminus \{p\} | d(p, q) \leq k\text{-distance}(p)\}$.*

Definition 2.7.3 (reachability distance of an object p w.r.t. object o)**.** *Let k be a natural number. The reachability distance of object p with respect to object o is defined as*

$$\text{reach-dist}_k(p, o) = max\{k\text{-distance}(o), d(p, o)\}.$$

Definition 2.7.4 (local reachability density of an object p)**.** *Local reachability density of an object p is defined as:*

$$\text{lrd}_k(p) = 1/\left(\frac{\sum_{o \in N_k(p)} \text{reach-dist}_k(p, o)}{|N_k(p)|}\right).$$

Definition 2.7.5 (local outlier factor of an object p)**.** *The local outlier factor of p is defined as*

$$\text{LOF}_k(p) = \frac{\sum_{o \in N_k(p)} \frac{\text{lrd}_k(o)}{\text{lrd}_k(p)}}{|N_k(p)|}.$$

As noted before, LOF does not give a binary output value but rather describes the degree of being an outlier for a given point. A LOF value approximately equal to 1 means that the given object is comparable to its neighbors and thus not an outlier. $\mathrm{LOF}_k(p) < 1$ would imply that p is an inlier and LOF value larger than 1 means that p resides outside of a cluster. The greater the LOF value, the farther away from the cluster a given point is. The exact threshold of LOF from which an object is considered an outlier, depends on a given dataset.

Implementation considerations

Here we consider the implementation-specific details for the local outlier factor (LOF) method. We will concentrate on a linear additive secret sharing scheme used by the Sharemind framework.

Let us have a dataset D with n objects. We do not hide the value of n. Assume that the (integer) distances of objects are given as a $n \times n$ matrix Dst, such that $\forall o \in D\, \forall p \in D : Dst_{o,p} = d(o,p)$. Now we can compute k-distance of any object p by sorting the corresponding row in the matrix Dst and then picking the k-th element from that row. However, constructing $N_k(p)$ is more complicated as it may contain more than k objects if some objects are equally distant from p as the k-th object in the sorted row. Checking values one-by-one introduces branching that potentially leaks some information and also breaks parallelization. Therefore, constructing $N_k(p)$ obliviously and efficiently is an open problem that needs further research. As we must not leak the size of $N_k(p)$, we should implement it as an n-element binary mask vector. Then finding its size is a local sum operation.

Combining the k-distance of each object and the distance matrix Dst we can construct a new $n \times n$ matrix that contains $\text{reach-dist}_k(p,o)$ for all pairs (p,o). Constructing this matrix is efficient as oblivious max can be easily parallelized.

Now if we write out the equation to compute $\mathrm{LOF}_k(p)$, we can see that it is one large fraction with mainly sums and some multiplications in both numerator and denominator. Remind that addition is a local operation and multiplications can be vectorized separately for the numerator and the denominator. If in a given application the LOF value may be made public, the numerator and denominator should be published separately so that the division operation could be done on public values to get an exact value of LOF. However, if the LOF value must stay private, the numerator should be scaled up (for example 10 or 100 times) so integer division could be used. Of course, in this case the LOF threshold should be scaled as well. One must take care that the (possibly) integers appearing in the numerator and denominator do not overflow the data type boundary.

2.8 Data classification

The protocols for secure classification have already been composed before. For example, the Kernel Perceptron algorithm has been securely implemented in [16], where different cryptographic techniques are used for different steps of the algorithm. Now this algorithm has been implemented using the available functionality of Sharemind.

Without loss of generality, assume that our goal is to split the data into two classes. If we want to have more classes, then we may just classify the already classified data again. The two classes be denoted for example 1 and -1.

Let $\mathbf{x_1}, \ldots, \mathbf{x_n}$ be the data parameter vectors (that in fact represent row vectors of a data matrix $\mathbf{X}$), and $y_1, \ldots, y_n$ the corresponding classes of $\mathbf{x_1}, \ldots, \mathbf{x_n}$. The idea of classification is the following: given a training set $S = \{(\mathbf{x_1}, y_1), \ldots, (\mathbf{x_n}, y_n)\}$, find a function g such that, $g(\mathbf{x_i}) = y_i$ for as many i as possible.

One of the easiest methods is to separate two classes with a straight line. If the data is linearly separable, then we may take $g(\mathbf{x}) = \langle \mathbf{w}, \mathbf{x} \rangle$ for a suitable vector $\mathbf{w}$. The question is how to find a suitable $\mathbf{w}$, and this is based on finding the best solution for the equation $\mathbf{X} \cdot \mathbf{w} = \mathbf{y}$ (that matches as many y coordinates as possible). The algorithm is described in more details in [20, Chapter 2].

Since not every data is linearly separable (for example, a circle on a 2D-plane), it has to be transformed in such a way that it would become linearly separable. We may add more coordinates (for example, transform-

ing the 2D plane to a cone makes it possible to separate circles with a $z = 0$ plane), but higher-dimensional data is more difficult to handle.

Assume that we have a transformation φ that maps given data vectors to some higher-dimensional linearly separable data vectors. The idea is to train the vector based on the values of some function $k(\mathbf{x_i}, \mathbf{x_j})$ that is computed for all possible data vectors. The question is what this k could be. One example of k would be $k(\mathbf{x_i}, \mathbf{x_j}) = \langle \mathbf{x_i}, \mathbf{x_j} \rangle^2$, which is equal to $\langle \varphi(\mathbf{x_i}), \varphi(\mathbf{x_j}) \rangle$ where φ maps $\mathbf{x_i}$ to a higher-dimensional vector that contains all the possible squares of the initial coordinates of $\mathbf{x_i}$. In this case we can directly compute $k(\mathbf{x_i}, \mathbf{x_j})$ without applying φ to the data vectors. There also exist SVM training algorithms based on scalar product, so this example of k is indeed useful. This k is called a *kernel function*, and there are more examples of suitable kernel functions in [20].

The kernel values can be for simplicity represented by a matrix $K = (k_{ij}) = (k(\mathbf{x_i}, \mathbf{x_j}))$.

Now the question is how this kernel will be used in classification. One of the particular kernel-based classifications algorithms is Kernel Perceptron (Algorithm 7.52 in [20]). Here we assume that the class vector y consists of values 1 and -1. We may think of looking at the sign of $\langle \mathbf{w}, \varphi(\mathbf{x}) \rangle$ in order to predict the class of $\mathbf{x}$.

Algorithm 5: Kernel Perceptron

Data: A kernel matrix K and class labels $\mathbf{y} \in \{-1, 1\}^n$.
Result: A weight vector $\alpha \in \mathbb{Z}^n$.

```
α = 0;
repeat
    for i=1 to n do
        if y_i · Σ_{j=1}^{n} k_ij · a_j ≤ 0 then
            α_i ← α_i + y_i;
until α is no more updated;
```

Since the goal is to implement the privacy-preserving version of this algorithm, it would be good for efficiency to parallelize as many computations as possible.

- **Kernel computation:** easy to parallelize. Since the entries of K are squared inner products of two vectors $\langle \mathbf{x_i}, \mathbf{x_j} \rangle^2$, each entry can be computed independently. The computation of each inner product can in turn be parallelized: all the product terms of the sum

$$\langle a, b \rangle = a_1 \cdot b_1 + a_2 \cdot b_2 + \ldots + a_n \cdot b_n$$

  can be computed independently of each other.

  If the entries of the kernel matrix are $\langle \mathbf{x_i}, \mathbf{x_j} \rangle^p$ for some $p > 2$, the exponentiation can in turn be parallelized for example using exponentiation by squaring.

- **Updating the classifier vector:** the iterations of the for-cycle are expensive. It would be good to compute the entire cycle in parallel, but the problem is that each step is actually dependent on the previous steps. Since computing them one by one is too slow, they are computed by parts in parallel. For example, a training vector of length 50 can be computed in 10 sequential blocks of length 5. Increasing the number of blocks (reducung the parallelization) gives better results, but is slower.

- **Evaluation:** the obtained vector α can be applied to the test set in order to predict the classes by computing $\mathsf{sign}(\sum_{j=1}^{n} k(\mathbf{x}, \mathbf{x_j}) \cdot a_j)$ for each $\mathbf{x}$ in the test set (here the $\mathbf{x_1}, \ldots, \mathbf{x_n}$ are the *training* set vectors). This can be also efficiently parallelized: first of all, compute the corresponding kernel matrix in the same way as for the training set, and then evaluate the classes of all the test set vectors independently in parallel.

Chapter 3

Privacy-preserving optimization

3.1 Privacy-preserving linear programming

We consider linear programming tasks in the form

$$\text{minimize } \mathbf{c}^{\mathrm{T}} \cdot \mathbf{x}, \text{ subject to } A\mathbf{x} = \mathbf{b}, 0 \leq \mathbf{x} \ .$$

Here A is an $m \times n$ matrix ($m \leq n$), $\mathbf{b}$ is a vector of length m and $\mathbf{c}$ is a vector of length n. There are n variables in the vector $\mathbf{x}$. Without lessening of generality we may assume that the quantity to be minimized is just the variable x_n, i.e. the vector $\mathbf{c}$ is of the form $(0, \ldots, 0, 1)^{\mathrm{T}}$. Any linear programming task can be brought to such a form by introducing a new variable w and adding the equation $\mathbf{c}^{\mathrm{T}} \cdot \mathbf{x} - w = w_0$ to the constraints, where $w_0 \in \mathbb{R}$ is determined (from out-of-band information about A, $\mathbf{b}$ and $\mathbf{c}$) so, that the minimal possible value of $\mathbf{c}^{\mathrm{T}} \cdot \mathbf{x} - w_0$ will certainly be positive.

The privacy requirements for our task are the following. The sizes m and n are public. Matrix A and the vector $\mathbf{b}$ have to remain secret. The solution $\mathbf{x}_{\mathrm{opt}}$ may become public. We wish to transform the linear programming task at hand to a task "minimize $\mathbf{c}'^{\mathrm{T}} \cdot \mathbf{y}$, subject to $A'\mathbf{y} = \mathbf{b}'$, $0 \leq \mathbf{y}$", such that from the solution $\mathbf{y}_{\mathrm{opt}}$ and secret data generated during the transformation, we could efficiently recover the solution $\mathbf{x}_{\mathrm{opt}}$ to the original task.

Note that we do not attempt to hide the result of the linear programming task. If its secrecy is desired, then it can be achieved (in statistical sense) by a separate, known problem transformation [9, 22] consisting of shifting the values of all variables by a random amount.

We are interested in a problem transformation based solution because of its potential of being much more efficient than the implementation of the simplex algorithm (or some other algorithm for solving linear programming tasks) using the primitives for secure computation. Several such transformations have been proposed beforehand [9, 22, 18, 19]. Unfortunately, these transformations either lack statements of security at all (or the statements are informal), or difficult to combine with usual cryptographic security definitions in the construction of larger applications. For our construction, we can state a pretty standard, indistinguishability-based confidentiality definition. We reduce the security of our transformation to the assumed intractability of a problem related to recognizing certain linear combinations among a set of vectors. As the values in the statement of a linear programming task belong to the set of real numbers, our transformation is also working with and producing structures made up of real numbers. Hence the assumed intractable problem is also defined over real numbers, and is therefore new, because cryptography over reals has not received much attention so far. Nevertheless, the assumption is tightly related to the *Strong Secret Hiding Assumption* [1] defined over finite fields.

3.1.1 Notation

A $m \times n$ matrix A is in the *reduced row-echelon form (RREF)* if it is the result of Gauss-Jordan elimination. This means, that

- all its zero rows are below its non-zero rows;
- the leading entry in each row (the leftmost non-zero entry) equals 1 and is strictly to the right of the leading entry of the row above it;
- the leading entry in each row is the only non-zero entry in its column.

For any $m \times n$ matrix A, there exists a $m \times m$ invertible matrix S, such that SA is in the RREF. Moreover, the matrix SA is uniquely determined by A. We call SA the *reduced row-echelon normal form* of A and denote it by $\mathrm{norm}(A)$. Two $m \times n$ matrices A and B satisfy $\mathrm{norm}(A) = \mathrm{norm}(B)$ iff there exists an invertible $m \times m$ matrix S, such that $SA = B$.

If A is a $m \times n$ matrix and π is a permutation of n elements, then let $A\pi$ denote the matrix obtained from A by permuting its columns according to π. If additionally B is a $m \times n'$ matrix, then $A|B$ denotes the matrix of size $m \times (n + n')$, the rows of which are the concatenations of corresponding rows of A and B.

The *ℓ_2-norm* of a vector $(v_1, \ldots, v_n)$ is the quantity $\sqrt{v_1^2 + \cdots + v_n^2}$.

The *null space* of A is the set of all such vectors $\mathbf{v} \in \mathbb{R}^n$, that $A\mathbf{v} = 0$. Let $A^\perp$ denote the *kernel* of A — the transpose of the basis of the *null space* of A. The kernel of A has n columns and $n - \mathit{rank}(A)$ rows. To make $A^\perp$ uniquely defined, we assume that it is in the RREF.

We denote the *normal distribution* with mean μ and standard deviation σ by $\mathcal{N}(\mu, \sigma^2)$. The probability density function of $\mathcal{N}(\mu, \sigma^2)$ is $f_{\mathcal{N}(\mu,\sigma^2)}(x) = \frac{1}{\sigma\sqrt{2\pi}} \exp(-\frac{1}{2}(\frac{x-\mu}{\sigma})^2)$. If a random variable X is distributed according to $\mathcal{N}(\mu, \sigma^2)$, then the absolute value $|X|$ is distributed according to the *folded normal distribution* $\mathcal{N}_{\mathrm{f}}(\mu, \sigma^2)$.

3.1.2 Transformation

Recall that the linear programming task is to minimize x_n, subject to $A\mathbf{x} = \mathbf{b}$ and $\mathbf{x} \geq 0$.

Given A of size $m \times n$ and $\mathbf{b}$ of size m, perform the following operations.

1. Let k and l be the security parameters. We suggest to take k about equal to n, and l at least $2n$, but so, that 2^{-l} is negligibly small.
2. Let D be a random $k \times (n + 1)$ matrix with non-negative entries, such that
   - each set of k columns of D is linearly independent;
   - each column of the matrix $\left(\frac{A|-\mathbf{b}}{D}\right)$ has the same ℓ_2-norm τ.
3. Let C be the following matrix:
$$C = \left(\begin{array}{c|c|c} A & -\mathbf{b} & 0 \\ \hline D & & -\tau I \end{array}\right),$$
   where I is the unit matrix of the correct size $(k \times k)$.
4. Let V be a random $(n + k + 1) \times l$ matrix generated as follows:
   - the entries in the n-th row are negative;
   - the entries in the $(n + 1)$-st row are 0;
   - the entries in all other rows are positive;
   - the absolute values of each non-zero entry are mutually independent variables distributed according to $\mathcal{N}_{\mathrm{f}}(0, 1)$.

   Note that any set of $(m + k)$ rows (not containing the $(n + 1)$-st row) is linearly independent with probability 1.

5. Let T be a $(n+k+1)\times(n+k+1)$ positive diagonal matrix. I.e. the entries of T are strictly positive on its main diagonal, and 0 elsewhere. Each entry t_{ii} on the main diagonal is obtained by independently sampling $u_1,\ldots,u_{n+k}$ from $\mathcal{N}(0,1)$ and putting $t_{ii}=\sqrt{u_1^2+\cdots+u_{n+k}^2}$.
6. Let π be a random permutation of $n+k+l+1$ elements.
7. Let $A'=\mathrm{norm}(C(T|V)\pi)$. Output A', $\pi(n)$ and $\pi(n+1)$.

In public domain, solve the linear programming task

$$\text{minimize } y_{\pi(n)},\ \text{subject to } A'\mathbf{y}=0,\mathbf{y}\geq 0,y_{\pi(n+1)}=1\ . \tag{3.1}$$

In private domain, select the optimal solution $\mathbf{y}_{\mathrm{opt}}$ for (3.1). To recover the optimal solution $\mathbf{x}_{\mathrm{opt}}$ of the original problem in the private domain, compute $\mathbf{y}_{\mathrm{opt}}\pi^{-1}$, take the first n elements, and multiply them by $t\cdot T^{-1}$, where t is the entry in the $(n+1)$-st row and column of T (the multiplication by t is needed since $t\cdot t^{-1}=1$, and the variable that is associated with the $\mathbf{b}$ vector is not being scaled).

3.1.3 Correctness

Consider the following three linear programming problems:

$$\text{minimize } y_n \text{ subject to } C(T|V)\mathbf{y}=0,y_{n+1}=1,\mathbf{y}\geq 0 \tag{3.2}$$
$$\text{minimize } z_n \text{ subject to } C\mathbf{z}=0,z_{n+1}=1,\mathbf{z}\geq 0 \tag{3.3}$$
$$\text{minimize } x_n \text{ subject to } A\mathbf{x}-\mathbf{b}x_{n+1}=0,x_{n+1}=1,\mathbf{x}\geq 0 \tag{3.4}$$

The RREF of A' obviously does not change the feasibility of any solution $\mathbf{y}$. When discussing the correctness of the scheme, we can also discard the permutation π. The correctness of the scheme thus means that the first n elements of an optimal solution to (3.2), multiplied by $t\cdot T^{-1}$, are an optimal solution the original linear programming task, which is obviously equivalent to (3.4). Consider the null space of $C(T|V)$. It is generated by the rows of the following matrix

$$\left(\begin{array}{c|c} T^{-1}\cdot C^{\perp} & 0 \\ \hline -V^{\mathrm{T}} & I \end{array}\right)\ . \tag{3.5}$$

A linear combination of the rows of (3.5) is a feasible solution to (3.2) iff its $(n+1)$-st entry is 1 (corresponding to the requirement $y_{n+1}=1$) and all other entries are non-negative. This implies that l lower rows (below the line in (3.5)) must have non-negative coefficients in such a feasible linear combination. On the other hand, decreasing the coefficients of l lower rows will improve the value of the objective function y_n — the n-th column of $-V^{\mathrm{T}}$ contains only positive numbers. Also, decreasing the coefficients of l lower rows does not make the linear combination of the rows infeasible: it increases the values in first n entries and the entries $(n+2)$ to $(n+k+1)$, because all entries of $-V^{\mathrm{T}}$ are negative. The decreasing of the coefficients of l lower rows also does not change the $(n+1)$-st entry of the linear combination because $-V^{\mathrm{T}}$ has zeroes in that column. Hence the lower l rows have their coefficients equal to 0 in any optimal solution to (3.2).

Thus the optimal solutions to (3.2) are obtained from the optimal solutions of (3.3) by multiplying them with $(1/t)\cdot T$ and appending them with l zeroes. The null space of C is generated by the rows of $(A|-\mathbf{b})^{\perp}|\frac{1}{\tau}(A|-\mathbf{b})^{\perp}D^{\mathrm{T}}$. Again, a feasible solution to the linear programming problem (3.3) is obtained as a linear combination of the rows of this matrix, where the $(n+1)$-st entry equals 1 and all other entries are non-negative. In our case, the condition "all other entries are non-negative" can be replaced with the condition "the first $n+1$ entries are non-negative", because the $(n+2)$-nd and following entries are non-negative linear combinations of first $n+1$ entries.

The feasible solutions for the original problem (3.4) are obtained as linear combinations of the rows of the matrix $(A|-\mathbf{b})^{\perp}$, where the $(n+1)$-st entry equals 1 and all other n entries are non-negative. This is the same condition as for the feasible solutions of (3.3). As the objective functions are also the same, we conclude that the optimal solutions to (3.4) and (3.3) coincide in their first n components.

3.1.4 Security

3.1.4.1 Definition

Our security definition for the transformation looks standard. The adversary communicates with the environment and attempts to guess a bit b chosen by it. First, the adversary chooses the dimensions m and n, and two instances of linear programming tasks $A_0, \mathbf{b}_0$ and $A_1, \mathbf{b}_1$ that share a common optimal solution $\mathbf{x}_{\text{opt}}$ (recall that the objective function to minimize is x_n). The adversary sends $m, n, A_0, A_1, \mathbf{b}_0, \mathbf{b}_1, \mathbf{x}_{\text{opt}}$ to the environment. The environment checks that $\mathbf{x}_{\text{opt}}$ is an optimal solution to both linear programming tasks, performs the transformation on the task $A_b, \mathbf{b}_b$ and sends the transformed problem back to the adversary. The adversary attempts to guess b. Its *advantage* is equal to its success probability minus 1/2. The transformation is *secure* if no polynomial-time adversary has non-negligible advantage.

3.1.4.2 Hardness assumption

Atallah and Frikken [1] have defined the following *Strong Secret Hiding Assumption (SSHA)*:

Assumption 3.1.1. *Let V be a $m \times n$ Vandermonde matrix over a finite field $\mathbb{F}_q$ (here $m < n$). Let S be a random $m \times m$ invertible matrix, R a random $m \times n'$ matrix and R' a random $m \times (n + n')$ matrix over $\mathbb{F}_q$. Let π be a random permutation of $n + n'$ elements. Then $(SV|R)\pi \approx R'$, where the probability distributions are defined through the random choices in selecting S, R and R'.*

Atallah and Frikken postulate that to distinguish $(SV|R)\pi$ from the completely random matrix R', one has to locate a set of $(m + 1)$ columns in $(SV|R)\pi$ that correspond to columns in V (these sets of columns are called *special*). They note that any set of at most m columns of $(SV|R)$ is distributed identically to a random $m \times m$ matrix. At the same time, special sets of $(m+1)$ columns are an exponentially small fraction of all sets of $(m + 1)$ columns.

How does the distinguisher recognize that a set of $(m+1)$ columns is special? Any set of $(m+1)$ columns is linearly dependent, but for a special set, there is a linear combination with special coefficients (following from V) of these columns that is equal to 0.

We also note that as V is a matrix with full rank, there exists a $n \times n'$ matrix W, such that $SVW = R$. Thus $(SV|R)\pi$ can be rewritten as $S(V|VW)\pi$. Finally we note that multiplication by a random invertible matrix cannot be more "hiding" than bringing the matrix to RREF. Noticing these things, and generalizing from finite $\mathbb{F}_q$ to real numbers, motivates us to state the following hardness assumption.

Assumption 3.1.2 (SSHA for $\mathbb{R}$)**.** *Let C be a $m \times n$ matrix over $\mathbb{R}$ (here $m \leq n$), such that any k columns of C are linearly independent. Let R be a random $n \times n'$ matrix and R' a random $m \times n$ matrix over $\mathbb{R}$. Let π be a random permutation of $n + n'$ elements. Then* $\text{norm}((C|CR)\pi) \approx \text{norm}((R'|R'R)\pi)$.

In order to distinguish $M = \text{norm}((C|CR)\pi)$ (with known C) from $M' = \text{norm}((R'|R'R)\pi)$, one could again look for a *special* set of columns in the resulting matrix. As any k columns of C are linearly independent, any k columns of M will look like columns of a random matrix in RREF. To identify a special set of columns, it is necessary to find $k + 1$ columns from among $n + n'$ that all came from C. In other words, it is necessary to find a $(k + 1)$-element subset of $\pi^{-1}(\{1, \ldots, n\})$. There are $\binom{n}{k+1}$ such special subsets. In total, there are $\binom{n+n'}{k+1}$ sets of $k + 1$ columns, special or not. The probability of a randomly chosen subset to be special is

$$\frac{\binom{n}{k+1}}{\binom{n+n'}{k+1}} = \frac{n!(n+n'-k-1)!}{(n-k-1)!(n+n')!} = \frac{(n-k)(n-k+1)\cdots(n+n'-k-1)}{(n+1)(n+2)\cdots(n+n')} \leq \left(1 - \frac{k+1}{n+n'}\right)^{n'},$$

which approaches 0 exponentially fast, if $n \to \infty$ and $k, n' \in \Theta(n)$.

The assumption 3.1.2 does not yet seem to be sufficient to show the security of our problem transformation. Indeed, we have required the entries of the matrix R to have specific signs. This will not invalidate

the analysis above on the difficulty of finding a special set of columns, but opens up possible new attack vectors that are not present in finite fields.

A further problem with assumption 3.1.2 is, that it does not consider the relative sizes of the entries in C, R' and R. This may allow to distinguish the matrices in ways that rely on specific inequalities between the entries of the matrix that is made public. These ways to distinguish do not have any counterparts when considering finite fields.

Assumption 3.1.3. *Let τ be a positive real number. Let $\mathcal{D}$ be the distribution $\mathcal{N}_{\mathrm{f}}(0,1)$. Let C be a $m \times n$ matrix over $\mathbb{R}$ (here $m \leq n$), such that any k columns of C are linearly independent and all columns of C have the same ℓ_2-norm τ. Let $Q \subseteq \{1,\ldots,n\}$, such that $|Q|$ is constant (i.e. does not depend on m or n). Let R be a random $n \times n'$ matrix, where an entry in the i-th row is 0 if $i \in Q$, and distributed according to $\mathcal{D}$ otherwise. Let R' be a random $m \times n$ matrix, such that all columns of R have the same ℓ_2-norm τ. Let T be a random $n \times n$ positive diagonal matrix over $\mathbb{R}$, where each entry on the main diagonal is generated according to the distribution*

$$\sqrt{\underbrace{\mathcal{D}^2 + \cdots + \mathcal{D}^2}_{n-|Q|}} \ . \tag{3.6}$$

Let π be a random permutation of $n + n'$ elements. Then $\mathrm{norm}(C(T|R)\pi) \approx \mathrm{norm}(R'(T|R)\pi)$.

In order to distinguish $M = \mathrm{norm}(C(T|R)\pi)$ from $M' = \mathrm{norm}(R'(T|R)\pi)$, one could also consider polytopes in n-dimensional space, specified by constraints that include the rows of C. E.g. one could consider the vectors $\mathbf{x} \in \mathbb{R}^n$ satisfying $C\mathbf{x} = 0$ and $\mathbf{x} \geq 0$. One could then consider the vertices of this polytope and try to find similar-looking vertices of the polytope that is defined through $M\mathbf{y} = 0$ and some inequality constraints on $\mathbf{y}$. Of course, the equations and inequalities could be defined differently. E.g. one could set $x_i = c$ for a particular $i \in \{1,\ldots,n\}$ and a particular constant $c \in \mathbb{R}$. The distinguisher does not know $\pi(i)$ in order to also set $y_{\pi(i)} = c$, but it could try guessing it. The guess is correct with non-negligible probability. A distinguisher with polynomial time-success ratio could make a constant number of such guesses and thus could treat a constant number of columns of C somehow differently from others when constructing the polytope that depends on C.

We believe that due to the scaling of the columns of C using the matrix T, the distinguisher is not able to recognize the coordinates of a vertex of the polytope depending on C among the coordinates of the corresponding vertex of the polytope depending on M. This assumption, however, is in need of further study.

Why have the sizes and scaling factors in assumption 3.1.3 been chosen like that? Taking the RREF of a matrix is at least as hiding as multiplying that matrix from the left with a random $m \times m$ (invertible) matrix S. If all entries of S are mutually independently sampled according to the *standard normal distribution* $\mathcal{N}(0,1)$ then each entry of SC is distributed according to $\mathcal{N}(0,\tau^2)$. Each column of $SC(T|R)$ is a linear combination of the columns of SC. To make the columns of $SC(T|R)$ indistinguishable by their sizes only, we choose the entries of $(T|R)$ (the coefficients of the linear combinations) so, that the variance of each entry is roughly $\mathbf{E}[\mathcal{D}] \cdot (n - |Q|)\tau^2$. As T and R have very different shapes, their entries have to be selected from different distributions.

As the problem transformation also makes some points of π public, we need to strengthen our assumptions a little bit more.

Assumption 3.1.4. *Let τ, $\mathcal{D}$, m, n, C, k, Q, n', R, R', T, π be the same quantities as in assumption 3.1.3. Let $P \subseteq \{1,\ldots,n\}$, such that $|P|$ is constant. Then* $\langle \mathrm{norm}(C(T|R)\pi), \pi|_P \rangle \approx \langle \mathrm{norm}(R'(T|R)\pi), \pi|_P \rangle$.

Here $\pi|_P$ denotes the list of pairs $[(i_1, \pi(i_1)), \ldots, (i_{|P|}, \pi(i_{|P|}))]$, where $P = \{i_1, \ldots, i_{|P|}\}$. The opening of some points of π does not seem to make distinguisher's work much easier, as it could have guessed these values of π itself (with non-negligible success probability). Still, as there does not seem to be a simple method for the distinguisher to verify its guess, we do not know how to reduce this assumption to the previous one.

In stating these assumptions, the distinguisher itself is allowed to first generate the matrix C (and pick the set of positions P). The assumptions have been stated in the *real-or-random* sense [3]. An equivalent statement in the *left-or-right* sense would be the following.

Lemma 3.1.5. *For any probabilistic polynomial-time adversary $\mathcal{A}$ and $c \in \mathbb{N}$, the probability of the following experiment to output* **true** *is no more than $1/2 + negl(n)$.*

1. *Pick a random bit $b \xleftarrow{\$} \{0,1\}$.*
2. *Obtain C_0, C_1, P, Q from $\mathcal{A}$, where $C_0, C_1 \in \mathbb{R}^{m \times n}$, $P, Q \subseteq \{1, \ldots, n\}$, $|P| \leq c$, $|Q| \leq c$ and all columns of C_0 and C_1 have the same ℓ_2-norm.*
3. *Let T be random $n \times n$ positive diagonal matrix, where each entry is sampled according to (3.6).*
4. *Let R be random $n \times n'$ where the i-th row is 0 iff $i \in Q$ and is sampled according to $\mathcal{N}_{\mathrm{f}}(0,1)$ otherwise.*
5. *Let π be a random permutation of $n + n'$ elements.*
6. *Give $\mathrm{norm}(C_b(T|R)\pi)$ and $\pi|_P$ to the adversary $\mathcal{A}$.*
7. *The adversary returns a bit b^*. Output whether $b = b^*$.*

3.1.4.3 Security reduction

Assume that the transformation is not secure, i.e. there exists an adversary $\mathcal{B}$ that breaks the security definition given in Sec 3.1.4.1, i.e. it can distinguish the transformed problem $A_0, \mathbf{b}_0$ from the transformation of $A_1, \mathbf{b}_1$. In this case, we can build an adversary $\mathcal{A}$ breaking the Assumption 3.1.4 as follows. The adversary $\mathcal{A}$ executes the following steps.

1. Invoke $\mathcal{B}$ and obtain the problem size $m \times n$, the linear programming tasks $A_0, \mathbf{b}_0$ and $A_1, \mathbf{b}_1$, and the common optimal solution $\mathbf{x}_{\mathrm{opt}}$ from it. Check that $\mathbf{x}_{\mathrm{opt}}$ is indeed an optimal solution to both linear programming tasks.
2. Let $k = n$. Let D be a random $k \times (n+1)$ matrix with non-negative entries, such that each set of k columns of D is linearly independent and each column of the matrix $\left(\frac{A|-\mathbf{b}}{D}\right)$ has the same ℓ_2-norm τ.
3. For $b \in \{0,1\}$, let C_b be the following matrix:
$$C = \left(\begin{array}{c|c|c} A & -\mathbf{b} & 0 \\ \hline D & & -\tau I \end{array}\right) .$$
4. Let $P = \{n, n+1\}$ and $Q = \{n+1\}$. Recall that the n-th column in C corresponds to the coefficients of the variable x_n, the value of which we try to minimize. The $(n+1)$-st column corresponds to the right-hand side $-\mathbf{b}$ of the system of constraints.
5. Let C'_b be obtained from C_b by negating the entries in the n-th column.
6. Submit C'_0, C'_1, P and Q to the environment described in Lemma 3.1.5. Get back a matrix M (in RREF) and the values $\pi(n)$ and $\pi(n+1)$.
7. Negate the entries in the $\pi(n)$-th column of M. Give the matrix thus obtained, as well as $\pi(n)$ and $\pi(n+1)$ to $\mathcal{B}$.
8. The adversary $\mathcal{B}$ returns a bit b^*. Output this bit.

It is straightforward to verify that the matrix M returned by the environment in step 6 is distributed identically to the matrix obtained as the result of the problem transformation in Sec. 3.1.2, except for the sign of the entries in the $\pi(n)$-th column. Hence the advantage of $\mathcal{A}$ in distinguishing instances of linear programming tasks is equal to the advantage of $\mathcal{B}$ in breaking Assumption 3.1.4.

3.1.5 Efficiency considerations

The operations performed in the private domain during the transformation described in Sec. 3.1.2 are generally quite efficient in a typical SMC platform, e.g. SHAREMIND [6, 7], involving the generation of random matrices and the multiplication of matrices. The transformation to RREF, however, is not so efficient, involving comparisons of numbers and branching on results. Hence we may want to replace that transformation with the multiplication with a random invertible matrix from the left. Our arguments in Sec. 3.1.4.2 hint that the entries of this matrix may be sampled from $\mathcal{N}(0,1)$.

3.2 Solving secure subset covering problem via a genetic algorithm

In this Section, we will consider the following problem setting. Let us have a target set $\mathcal{T} = \{T_1, T_2, \ldots, T_m\}$ and a covering set $\mathcal{X} = \{X_1, X_2, \ldots, X_n\}$. We are also given a binary relation *covers* between the elements of these sets. This relation can be presented using a binary matrix $A = (a_{ij})_{i,j=1}^{m,n}$ defined as

$$a_{ij} = \begin{cases} 1 & \text{if the element } X_j \textit{ covers} \text{ the elment } T_i, \text{ and} \\ 0 & \text{otherwise.} \end{cases}$$

In this way, each element X_j of $\mathcal{X}$ is identified with a subset of $\mathcal{T}$.

As input, we are given the characteristic vector $\mathbf{t} \in \{0,1\}^m$, and our aim is to generate an output characteristic cover vector $\mathbf{x} \in \{0,1\}^n$, so that the subset of $\mathcal{T}$ characterized by $\mathbf{t}$ would be covered by the subset of $\mathcal{X}$ characterized by $\mathbf{x}$. Alternatively, we can write this requirement as

$$A\mathbf{x} \geq \mathbf{t}\,. \tag{3.7}$$

Additionally, we are given a vector of (non-negative) *costs* $\mathbf{c} = (c_1, c_2, \ldots, c_n)^T$, where c_j can be interpreted as the price to pay for selecting the element X_j into the cover. Our optimization task is to minimize the overall cost of the cover, i.e.

$$\min\ \mathbf{c} \cdot \mathbf{x} = \min \sum_{j=1}^{n} c_j \cdot x_j\,. \tag{3.8}$$

The equations (3.7) and (3.8) together also specify a certain kind of linear programming problem known as 0-1 linear programming, or binary (integer) linear programming problem (BIP). We will consider secure BIP in the setting where the matrix A and the cost vector $\mathbf{c}$ are public, but both $\mathbf{t}$ and $\mathbf{x}$ must remain secret. As an example of such a setting we may consider the situation, where the elements T_j represent potential threats against an information system, and the elements X_i are the possible countermeasures, each of which may be effective against certain threats. We do not want to disclose the vulnerabilities the system may have, nor the selected countermeasures to the outside observers, but we nevertheless want to select the countermeasure set to be as cheap as possible.

Even though BIP is a special case of linear programming, generic linear programming methods (like simplex algorithm) generally fail to solve it well since they do not take the 0-1 restriction into account. BIP can be solved by branch-and-bound type of algorithms like Balas Additive Algorithm [2]. However, in order to efficiently prune the search tree, such methods need to make decisions on control bits, and their runs differ on different input data. This kind of behaviour is unwanted in a privacy-preserving algorithm, as the running time of the program could be used to infer details about the private inputs.

Hence, we decided not to choose a branch-and-bound algorithm and take a totally different approach. To solve the underlying BIP problem, we have implemented a genetic algorithm. This approach has several advantages. First, we do not have to leak any bits, since the control flow does not depend on the private inputs. Second, a genetic algorithm can be made to run for a predefined number of iterations or a predefined amount of time. On the other hand, genetic algorithms are inherently heuristic and are not guaranteed to produce the globally optimal result. Nevertheless, they have been proven to yield results good enough for practical use.

Algorithm 6: Basic genetic algorithm

Data: Vector $\mathbf{t} \in \{0,1\}^m$ of relevant threats; matrix $A \in \{0,1\}^{m \times n}$ of the correspondences between the threats and countermeasures; vector $\mathbf{c} \in (\mathbb{Z}_{2^{32}})^n$ expressing the costs of countermeasures

Result: A set of k candidate countermeasure suites

```
Generate a random generation (x_1, x_2, ..., x_k)
while there is time to compute do
    For each pair of individuals x_i and x_j produce their offspring by crossover
    For some offspring mutate some of their bits
    Sort the offspring pool by the fitness c · x
    Choose k fittest for the new generation (x_1, x_2, ..., x_k).
end
return (x_1, x_2, ..., x_k)
```

Genetic algorithms work on *generations* of individuals. In our case, the individuals are 0-1 vectors $\mathbf{x}_i$ corresponding to the candidate countermeasure suites. Each generation has k individuals where k is a system-wide configurable parameter. Computation proceeds in iterations, where both the input and output of each iteration is a k-element generation. General structure of the routine is presented in Algorithm 6.

There are several implementation details to fill in in the basic algorithm. We have to choose the size of the generation, crossover strategy and mutation strategy. Since these parameters depend on each other non-linearly, making the optimal choices is a highly non-trivial task.

For our demo application we ran tests with the size of the generation set to $k = 8, 12, 16, 23, 32$, and for the number of iterations $g = 5, 10, 20$. We applied uniform crossover and mutated the bits of individuals also randomly with the probability 2^{-s}. The last two choices were made because of the need to hide the control flow. Next to the uniform crossover, another frequently used strategy is one- or two-point crossover. However, selecting a few random cutting points has no straightforward implementation in the oblivious setting. At the same time, uniform selection between the parent genes is rather easy to achieve by generating random selection vectors and performing n oblivious choices for each candidate offspring. Similar reasoning applies to the mutation operation as well. In order to flip the bits of individuals with probability 2^{-s}, we can generate s random bit vectors and multiply them bitwise. In our experiments we set $s = 4$ giving 6.25% of probability for any bit to be flipped.

The pool of candidates for the next generation consists of k members of the previous generation plus $\binom{k}{2}$ of their offspring. Since technically, it is simpler to sort 2^t elements, some of the offspring are discarded to get the closest power of two for the pool size. E.g. when $k = 8$, we get the original pool size $8 + \binom{8}{2} = 36$ and we drop 4 of them to get down to 32. For $k = 12, 16, 23, 32$, we sort arrays of size $64, 128, 256, 512$, respectively.

In order to select the k fittest individuals, several steps need to be taken. First, for every candidate vector we need to verify the matrix inequality in Equation 3.7, and if it is not satisfied, we obliviously assign a very high cost to this vector. Next, we need to sort all the candidate vectors by the costs. Full sorting is a rather expensive operation, and it is not really needed for the purposes of genetic algorithms. Hence, we decided to implement Swiss tournament sorting. It is known that this sorting method works better in both ends on the sorted array, whereas the middle part is not guaranteed to be linearly ordered [10]. In our case, we obliviously evaluate as much of the the Swiss tournament sorting network as is needed for finding the top k elements. However, our experiments show that compared to full sorting, the degradation of the precision of the whole genetic algorithm is rather small, but the gain in computing time is significant.

3.3 Finding the shortest path in a graph

In this Section, we apply a genetic algorithm similar to the one described in previous Section to find a shortest path in a given directed graph. As a starting point, we use the following linear programming

formulation of the shortest path problem[1]:

Given a directed graph (V, A) with source node s, target node t, and cost w_{ij} for each arc (i, j) in A, consider the program with variables x_{ij}: minimize $\sum_{ij \in A} w_{ij} x_{ij}$ subject to $x \geq 0$ and for all i,

$$\sum_j x_{ij} - \sum_j x_{ji} = \begin{cases} 1, & \text{if } i = s; \\ -1, & \text{if } i = t; \\ 0, & \text{otherwise.} \end{cases} \tag{3.9}$$

This gives us a binary linear programming problem and a cost function to optimize. We will first tackle the single pair shortest path problem $SPSP(s, t)$ as described in Deliverable D4.1 [11]. In our privacy-preserving implementation, the source and target nodes s and t are secret. However, the total number of vertices n in the graph is public knowledge. Input graph G is split into two matrices: a public boolean adjacency matrix S showing the structure of the original graph and a private (secret shared) matrix W holding the weights (costs) of edges. Holding the graph structure separately as a boolean matrix allows us to ignore the graph structure while performing randomized operations, e.g. generating the initial generation or applying mutations. When computing the cost of an individual or determining if an individual is a valid path from s to t, that is if equation 3.9 holds, we can then point-wise multiply the individual with the public graph structure matrix to eliminate all edges that might have appeared outside the original structure of G. Since S is not secret shared, this multiplication is a local operation that requires no network communication if we use a linear secret sharing scheme. Note that the structure of the graph could also be given as secret input. However, removing edges outside the actual graph structure then requires multiplication of secret shared values, which in turn requires network communication. All the intermediate results in the algorithm are secret shared, including the final list of individuals.

The basic genetic algorithm used to solve this problem is the same as Algorithm 6 in Section 3.2. However, instead of countermeasure vectors the individuals in this setting are graphs represented as $n \times n$ adjacency matrices and the fitness function is based on the total cost (weight) of an individual matrix.

Computing the cost of an individual G is as simple as first point-wise multiplying the individual structure graph with the original structure graph S to remove any non-relevant edges and then point-wise multiplying again with the cost matrix W and summing all the values in the resulting graph:

$$cost(G) := \sum_{1 \leq i \leq n} \sum_{1 \leq j \leq n} G_{i,j} S_{i,j} W_{i,j}.$$

However, while determining which individuals are the fittest and should be carried over to the next generation, we would actually want to consider only these individuals that contain a valid path from s to t, that means the individuals where equation 3.9 holds. For all other individuals that do not match this criteria, we would like to set $cost(G) = \infty$ so they are less probable to be carried over to the next generation. Computing row and column sums and their difference can be easily done in equation 3.9, however, oblivious evaluation of the piece-wise function is a non-trivial task. In our implementation we obliviously build an n-element mask vector that is filled with zeros except at positions s and t where there are 1 and -1 respectively. This mask vector is then point-wise compared against the vector of row and column sum differences. If the two vectors are equal, we set a comparison bit b equal to one and zero otherwise. For a given individual G, we set its actual cost c obliviously as:

$$c = b \cdot cost(G) + (1 - b)\infty.$$

Another problem with the equation 3.9 is that other than the actual path from s to t, it only allows a graph G to contain closed cycles. This greatly affects the performance of the algorithm as we will throw away individuals that contain the correct path but also have a few non-relevant edges. However, such individuals may in fact be the best candidates as they are close to the optimal solution. To also consider such individuals, we must come up with a different privacy-preserving way to check the existence of a valid path.

[1]Shortest path problem – `http://en.wikipedia.org/wiki/Shortest_path_problem#Linear_programming_formulation`

Chapter 4

Privacy-preserving network management and operation

Applications described in Section 3.6 of Deliverable 4.1 [11] require identifying links among vertices in a network. These links can represent routing paths or intrusion detection events. This problem can be formally phrased as finding the transitive closure of a graph.

For simplicity we focus on a scenario involving only two parties. Each party P_k owns a graph $G_k = (V, E_k)$, consisting of a set of vertices V and directed edges $E_k \subseteq V \times V$. Notice that the set of vertices is a piece of information shared between the two parties. Let G be the union of the two graphs ($G = (V, E_1 \cup E_2)$), the transitive closure of G is a graph $G^* = (V, E^*)$ with $(i, j) \in E^*$ if there is a path from the vertex i to the vertex j in G.

Xiaoyun et all. [14] proposed a solution based on adjacency matrix multiplication. Let Adj be the adjacency matrix of the graph G ($Adj[i, j] = 1$ if $(i, j) \in E$ otherwise $Adj[i, j] = 0$) and n be the number vertices, then Adj^n computes the adjacency matrix of G^*. More precisely, $Adj^n[i, j] \neq 0$ iff there exists a path from the vertex i to the vertex j in graph G. The proposed solution is based on an iterative multiplication algorithm ($Adj^{2m} = Adj^m \cdot Adj^m$) that requires $\log_2 n$ iterations, each of them consisting of two secure multiplications of $n \times n$ matrices. At each round each party obtains a $n \times n$ matrix consisting of random shares of the global adjacency matrix. At the end of the computation, the parties can share their results to obtain the transitive closure.

Algorithm 7: Transitive closure algorithm

Data: Party P_k has input Adj_k

1 P_2 generates a random invertible matrix R
2 P_2 generates a random $n-$squared coefficient matrix C
3 P_2 computes $Adj'_2[i, j] = C[i, j] \cdot Adj_2[i, j]$ for all $i, j \in \{1, \ldots, n\}$
4 P_1 obtains the matrix $S = R(Adj'_2 + Adj_1)R^{-1}$
5 P_1 locally computes $T = S^n = R(Adj'_2 + Adj_1)^n R^{-1}$

We propose an asymmetric protocol for the two-party case (Algorithm 7). The algorithm requires secure multiplication and addition of matrices at the step 4. This algorithm step can be accomplished exploiting the existing techniques (e.g. using the matrix multiplication algorithm based on homomorphic encryption proposed by Xiaoyun et al.). However, the matrix exponentiations can be performed by the party P_1 without involving the other party. At the end of the algorithm three scenarios are possible:

- The transitive closure is delivered to a third party for independent analysis. If the third party is allowed to discover $(Adj'_2 + Adj_1)^n$, then the two participants send it the matrices T and R.
- If the party P_2 is allowed to discover $(Adj'_2 + Adj_1)^n$, then P_1 sends the matrix T to P_2. The party P_2 can compute $(Adj'_2 + Adj_1)^n$ as $R^{-1}TR$ and send back the normalized adjacency matrix ($A \mid A[i, j] = 1$

if $(Adj_2' + Adj_1)^n[i,j] \neq 0$ else 0) to the party P_1. Notice that P_1 may not obtain $(Adj_2' + Adj_1)^n$, otherwise it could compute the matrix R.

- The two parties perform a further secure matrix multiplication to obtain A_1 and A_2 such that $A_1 + A_2 = R^{-1}TR$. The obtained shares are used as input to a garbled circuit that computes the normalized adjacency matrix $A \mid A[i,j] = 1$ if $(A_1 + A_2)[i,j] \neq 0$ else 0.

We plan to implement the three proposals on the SHAREMIND system in order to compare their performances.

Bibliography

[1] Mikhail J. Atallah and Keith B. Frikken. Securely outsourcing linear algebra computations. In Dengguo Feng, David A. Basin, and Peng Liu, editors, *ASIACCS*, pages 48–59. ACM, 2010.

[2] Egon Balas. An additive algorithm for solving linear programs with zero-one variables. *Operations Research*, 13(4):517–546, July/August 1965.

[3] Mihir Bellare, Anand Desai, E. Jokipii, and Phillip Rogaway. A concrete security treatment of symmetric encryption. In *FOCS*, pages 394–403. IEEE Computer Society, 1997.

[4] Dan Bogdanov, Yiannis Giannakopoulos, Roberto Guanciale, Liina Kamm, Peeter Laud, Pille Pruulmann-Vengerfeldt, Riivo Talviste, Kadri Tõldsepp, and Jan Willemson. Scientific Progress Analysis and Recommendations, January 2013. UaESMC Deliverable 5.2.1.

[5] Dan Bogdanov, Roman Jagomägis, Liina Kamm, Peeter Laud, Alisa Pankova, Riivo Talviste, and Jan Willemson. Implementations of UaESMC techniques, January 2013. UaESMC Deliverable 5.1.1.

[6] Dan Bogdanov, Sven Laur, and Jan Willemson. Sharemind: A framework for fast privacy-preserving computations. In Sushil Jajodia and Javier López, editors, *ESORICS*, volume 5283 of *Lecture Notes in Computer Science*, pages 192–206. Springer, 2008.

[7] Dan Bogdanov, Margus Niitsoo, Tomas Toft, and Jan Willemson. High-performance secure multi-party computation for data mining applications. *Int. J. Inf. Sec.*, 11(6):403–418, 2012.

[8] Markus M. Breunig, Hans-Peter Kriegel, Raymond T. Ng, and Jörg Sander. Lof: Identifying density-based local outliers. In *Proceedings of the 2000 ACM SIGMOD International Conference on Management of Data, May 16-18, 2000, Dallas, Texas, USA*, pages 93–104, 2000.

[9] Jannik Dreier and Florian Kerschbaum. Practical privacy-preserving multiparty linear programming based on problem transformation. In *SocialCom/PASSAT*, pages 916–924. IEEE, 2011.

[10] Wilfried Elmenreich, Tobias Ibounig, and István Fehérvári. Robustness versus performance in sorting and tournament algorithms. *Acta Polytechnica Hungarica*, 6(5):7–18, 2009.

[11] Roberto Guanciale. Identification of application scenarios, January 2013. UaESMC Deliverable 4.1.

[12] Frank R. Hampel. A general qualitative definition of robustness. *The Annals of Mathematical Statistics*, 42(6):1887–1896, December 1971.

[13] Frank R. Hampel. The influence curve and its role in robust estimation. *Journal of the American Statistical Association*, 69(346):383–393, June 1974.

[14] Xiaoyun He, Jaideep Vaidya, Basit Shafiq, Nabil R. Adam, Evimaria Terzi, and Tyrone Grandison. Efficient privacy-preserving link discovery. In Thanaruk Theeramunkong, Boonserm Kijsirikul, Nick Cercone, and Tu Bao Ho, editors, *PAKDD*, volume 5476 of *Lecture Notes in Computer Science*, pages 16–27. Springer, 2009.

[15] Donald E. Knuth. *The art of computer programming, volume 3: (2nd ed.) sorting and searching*. Addison Wesley Longman Publishing Co., Inc., Redwood City, CA, USA, 1998.

[16] Sven Laur, Helger Lipmaa, and Taneli Mielikäinen. Cryptographically private support vector machines. In Tina Eliassi-Rad, Lyle H. Ungar, Mark Craven, and Dimitrios Gunopulos, editors, *KDD*, pages 618–624. ACM, 2006.

[17] Sven Laur, Jan Willemson, and Bingsheng Zhang. Round-Efficient Oblivious Database Manipulation. In *Proceedings of the 14th International Conference on Information Security. ISC'11*, pages 262–277, 2011.

[18] O. L. Mangasarian. Privacy-preserving linear programming. *Optimization Letters*, 5(1):165–172, 2011.

[19] Olvi L. Mangasarian. Privacy-preserving horizontally partitioned linear programs. *Optimization Letters*, 6(3):431–436, 2012.

[20] John Shawe-Taylor and Nello Cristianini. *Kernel Methods for Pattern Analysis*. Cambridge University Press, New York, NY, USA, 2004.

[21] Kadri Tõldsepp, Pille Pruulmann-Vengerfeldt, and Peeter Laud. Requirements specification based on the interviews, July 2012. UaESMC Deliverable 1.2.

[22] Jaideep Vaidya. Privacy-preserving linear programming. In Sung Y. Shin and Sascha Ossowski, editors, *SAC*, pages 2002–2007. ACM, 2009.