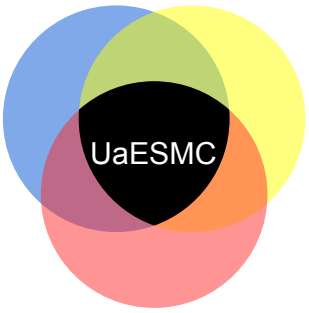


Project N°: **FP7-284731**

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 D2.2.2
Advances in Secure Multiparty Protocols

Due date of deliverable: 31st January 2014

Actual submission date: 31st January 2014

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 D2.2.2 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`.

The report contains an overview of the results of the second year of UaESMC, pertaining to secure multiparty computation techniques. The studies of these techniques have been directed by the example problems selected during the first year, as well as by the desire to have a comprehensive framework of privacy-preserving computation techniques by the end of the project.

In this deliverable, we report of the following findings and advances:

- We provide improved privacy preserving algorithms for giving an overview of the data. We also give privacy preserving versions of algorithms for several most common statistical tests. As result, we were able to conduct a full-scale experimental statistical study so that confidential data were always processed using SMC. The strengths of our solution are generality, precision and practicality. We show that secure multi-party computation is flexible enough for implementing complex applications.

- We have found that the class of techniques currently used for problem transformation based solving of linear programming tasks cannot be privacy-preserving. This leaves the implementations of standard LP-solving algorithms on top of generic protocol sets for privacy-preserving arithmetic as the only general method for privacy-preserving LP, unless some radically new ideas for transforming LP problems are proposed.

- We provide efficient algorithms for privacy-preserving finite automata execution, that achieve online efficiency through offline precomputations.

- We provide a protocol set for actively-secure two-party computation that also acheives efficiency through offline precomputations.

- We provide a protocol transformation that turns any passively secure multiparty computation protocol with honest majority to a protocol where any misbehaviour is detected after the execution.

We expect many of these advances to play a significant role in the UaESMC framework.

List of Authors

Dan Bogdanov (CYB) Liina Kamm (CYB) Peeter Laud (CYB) Alisa Pankova (CYB)
Pille Pullonen (CYB) Riivo Talviste (CYB) Jan Willemson (CYB)

Contents

Chapter 1

Introduction

This report gives a review of the advances in secure multiparty computation techniques made during the second year of the UaESMC project. We have investigated several different tasks and problems, making progress in solving particular classes of computational problems, as well as in improving the security guarantees of broad classes of protocols. Our investigations have been motivated by the example problems selected during the first year [7]. Even more, they have been motivated by the desire to have a comprehensive set of SMC techniques available for the UaESMC framework, due to be formulated during the final year of the project.

We have continued our work on statistical analysis of structured data, the results of which are reported in Chapter 2. The processing of structured data requires efficient database operations, which in turn depend on fast sorting methods. We have thus performed a thorough effeciency comparison of privacy-preserving sorting methods, described in Chapter 7. In Chapter 3, we describe our surprising results on linear programming, obtained during this year of UaESMC. Both statistical analysis and linear programming are among the selected example problems of UaESMC.

There is a different kind of problem that we have also investigated during the second year of UaESMC project. It pertains to the privacy-preserving execution of finite automata. This problem has applications in network management (also a selected example problem of UaESMC). It is also interesting because the access patterns for the algorithms solving it significantly depend on the data that we would like to remain private. Efficient privacy-preserving solutions for this problem would thus need new kinds of techniques. Our results are reported in Chapter 4.

We have investigated efficient methods to make the parties of a privacy-preserving computation or, more generally, any cryptographic protocol faithfully perform the instructions of the protocol. In Chapter 6 we describe a protocol set for actively secure multiparty computation among two parties. The efficiency of the protocols is achieved through offline precomputation. In Chapter 5, we show how to turn any protocol secure against semi-honest adversaries into a protocol secure against covert adversaries (such adversaries may deviate from the protocol, but only if they are not caught afterwards), under the condition that a majority of protocol participants are honest. Both techniques allow secure computation application to achieve stronger security properties.

To the end of this deliverable, we have annexed a number of papers and technical reports we have published during the second year of the project. These papers are referenced from the main body of the deliverable.

Chapter 2

Privacy-Preserving Statistical Analysis

2.1 Simple Statistics

This year, we continued our research into privacy-preserving statistical analysis. We finished and improved work begun last year and we added more statistical tests to the statistics suite in order to provide a wider choice for data analysts. In the following, let $[\![x]\!]$ denote a private value x, let $[\![\vec{a}]\!]$ denote a private value vector $\vec{a}$, and let binary operations between vectors be point-wise operations.

2.1.1 Quantiles and Outlier Detection

The first improvement to deliverable D2.2.1 [9] is the new quantile calculation method. As no one method for computing quantiles has been widely agreed upon in the statistics community, we use algorithm $\mathbf{Q_7}$ from [32], because it is the default choice in our reference statistical analysis package GNU R. Let p be the percentile we want to find and let $[\![\vec{a}]\!]$ be a vector of values sorted in ascending order. Then the quantile is computed using the following function:

$$\mathbf{Q_7}(p, [\![\vec{a}]\!]) = (1-\gamma) \cdot [\![\vec{a}]\!][j] + \gamma \cdot [\![\vec{a}]\!][j+1] \ ,$$

where $j = \lfloor (n-1)p \rfloor + 1$, n is the size of vector $[\![\vec{a}]\!]$, and $\gamma = np - \lfloor (n-1)p \rfloor - p$. Once we have the index of the quantile value, we can use oblivious versions of vector lookup or sorting to learn the quantile value from the input vector.

While data-independent oblivious sorting can easily be implemented using sorting networks, oblivious Hoare's selection is more complex, because the partitioning sub-procedure publishes random comparison results the same way as cutting does. We solve the problem in the same way, by running a shuffling procedure before the selection. As the elements of the resulting vector are in random order, even the declassification of all comparison results leaks no information about the input vector. Hence, it is straightforward to simulate the outcome of the entire selection algorithm. As Hoare's selection algorithm has linear asymptotic complexity whereas common sorting networks consist of $\Theta(n \log^2 n)$ comparison gates, selection is potentially faster[1] if we are dealing with large datasets. Although we implemented both approaches, we did not observe this in practice. In fact, using sorting networks turned out to be faster, so we chose this as our default solution.

We also implemented a very simple outlier elimination method using quantiles. We do not need to publish the quantile to use it for outlier filtering. Let q_0 and q_1 be the 5% and 95% quantiles of an attribute $[\![\vec{a}]\!]$. It is common to mark all values smaller than q_0 and larger than q_1 as outliers. The corresponding mask vector is computed by comparing all elements of $[\![\vec{a}]\!]$ to $\mathbf{Q_7}(0.05, [\![\vec{a}]\!])$ and $\mathbf{Q_7}(0.95, [\![\vec{a}]\!])$, and then multiplying the resulting index vectors.

[1] As the asymptotic complexity of shuffle is $\Theta(n \log n)$, which is the complexity of the optimal AKS sorting network, both approaches are theoretically equivalent.

Algorithm 1: Function **cut** for cutting the dataset according to a given filter.

Data: Data vector $[\![\vec{a}]\!]$ of size N and corresponding mask vector $[\![\vec{m}]\!]$.
Result: Data vector $[\![\vec{x}]\!]$ of size n that contains only elements of $[\![\vec{a}]\!]$ corresponding to the mask $[\![\vec{m}]\!]$

1 Obliviously shuffle the value pairs in vectors $([\![\vec{a}]\!], [\![\vec{m}]\!])$ into $([\![\vec{a'}]\!], [\![\vec{m'}]\!])$
2 $\vec{s} \leftarrow \textbf{publish}([\![\vec{m'}]\!])$
3 $[\![\vec{x}]\!] \leftarrow ([\![\vec{a'}]\!][i] \mid \vec{s}[i] = 1, i \in \{1, \ldots, N\})$
4 **return** $[\![\vec{x}]\!]$

Algorithm 2: Algorithm for finding the five-number summary of a vector.

Data: Input data vector $[\![\vec{a}]\!]$ and corresponding mask vector $[\![\vec{m}]\!]$.
Result: Minimum $[\![min]\!]$, lower quartile $[\![lq]\!]$, median $[\![me]\!]$, upper quartile $[\![uq]\!]$, and maximum $[\![max]\!]$ of $[\![\vec{a}]\!]$ based on the mask vector $[\![\vec{m}]\!]$

1 $[\![\vec{x}]\!] \leftarrow \textbf{cut}([\![\vec{a}]\!], [\![\vec{m}]\!])$
2 $[\![\vec{b}]\!] \leftarrow \textbf{sort}([\![\vec{x}]\!])$
3 $[\![min]\!] \leftarrow [\![\vec{b}]\!][1]$
4 $[\![max]\!] \leftarrow [\![\vec{b}]\!][n]$
5 $[\![lq]\!] \leftarrow \mathbf{Q_7}(0.25, [\![\vec{b}]\!])$
6 $[\![me]\!] \leftarrow \mathbf{Q_7}(0.5, [\![\vec{b}]\!])$
7 $[\![uq]\!] \leftarrow \mathbf{Q_7}(0.75, [\![\vec{b}]\!])$
8 **return** $([\![min]\!], [\![lq]\!], [\![me]\!], [\![uq]\!], [\![max]\!])$

2.1.2 Five-Number Summary and Frequency Tables

It is important for a data analyst to get an overview of the data. As it is not possible to see the data in SMC format, we give an overview using the five number summary and the histogram.

First, for reference, we give Algorithm 1 for obliviously cutting the dataset based on a given filter. First the value and mask vector pairs are obliviously shuffled, retaining the correspondence of the elements. Next, the mask vector is declassified and values for which the mask vector contains 0 are removed from the value vector. The obtained cut vector is then returned to the user. This process leaks the number of values that correspond to the filters that the mask vector represents. This makes cutting trivially safe to use, when the number of records in the filter would be published anyway. Oblivious shuffling ensures that no other information about the private input vector and mask vector is leaked [37]. Therefore, all algorithms that use oblivious cut provide source privacy.

Algorithm 2 describes the computation of the five-number summary of a value vector $[\![\vec{a}]\!]$ with the corresponding mask vector $[\![\vec{m}]\!]$. Function **cut** leaks the count of elements n that correspond to the filter signified by the mask vector $[\![\vec{m}]\!]$. However, the filter size is often one of the descriptive statistics that analysts want to learn. If we want to keep n secret, we can use Algorithm 3. This hides n, but runs slower than Algorithm 2.

More information about the data can be obtained by looking at the distribution of a data attribute. For categorical attributes, this can be done by computing the frequency of the occurrences of different values. For numerical attributes, we must split the range into bins specified by breaks and compute the corresponding frequencies. The resulting frequency table can be visualised as a histogram. The algorithm publishes the number of bins and the number of values in each bin. We also implemented the histogram calculation algorithm.

Algorithm 4 computes a frequency table for a vector of values similarly to a public frequency calculation algorithm.

Algorithm 3: Oblivious algorithm for finding the five-number summary of a vector.

Data: Input data vector $[\![\vec{a}]\!]$ and corresponding mask vector $[\![\vec{m}]\!]$.
Result: Minimum $[\![min]\!]$, lower quartile $[\![lq]\!]$, median $[\![me]\!]$, upper quartile $[\![uq]\!]$, and maximum $[\![max]\!]$ of $[\![\vec{a}]\!]$ based on the mask vector $[\![\vec{m}]\!]$

1 $([\![\vec{b}]\!], [\![\vec{m'}]\!]) \leftarrow \mathbf{sort}([\![\vec{a}]\!], [\![\vec{m}]\!])$
2 $[\![n]\!] \leftarrow \mathbf{sum}([\![\vec{m}]\!])$
3 $[\![os]\!] \leftarrow [\![N - n]\!]$
4 $[\![min]\!] \leftarrow [\![\vec{b}]\!][1 + [\![os]\!]]$
5 $[\![max]\!] \leftarrow [\![\vec{b}]\!][N]$
6 $[\![lq]\!] \leftarrow \mathbf{Q_7}(0.25, [\![\vec{a}]\!], [\![os]\!])$
7 $[\![me]\!] \leftarrow \mathbf{Q_7}(0.5, [\![\vec{a}]\!], [\![os]\!])$
8 $[\![uq]\!] \leftarrow \mathbf{Q_7}(0.75, [\![\vec{a}]\!], [\![os]\!])$
9 **return** $([\![min]\!], [\![lq]\!], [\![me]\!], [\![uq]\!], [\![max]\!])$

Algorithm 4: Algorithm for finding the frequency table of a data vector.

Data: Input data vector $[\![\vec{a}]\!]$ and corresponding mask vector $[\![\vec{m}]\!]$.
Result: Vector $[\![\vec{b}]\!]$ containing breaks against which frequency is computed, and vector $[\![\vec{c}]\!]$ containing counts of elements

1 $[\![\vec{x}]\!] \leftarrow \mathbf{cut}([\![\vec{a}]\!], [\![\vec{m}]\!])$
2 $n \leftarrow \mathbf{count}([\![\vec{x}]\!])$
3 $k \leftarrow \lceil \log_2(n) + 1 \rceil$
4 $[\![min]\!] \leftarrow \mathbf{min}([\![\vec{x}]\!])$, $[\![max]\!] \leftarrow \mathbf{max}([\![\vec{x}]\!])$
5 Compute breaks according to $[\![min]\!]$, $[\![max]\!]$ and k, assign result to $[\![\vec{b}]\!]$
6 $[\![\vec{c}]\!][1] = (\mathbf{count}([\![\vec{x}]\!][i]) \mid [\![\vec{b}]\!][1] \leq [\![\vec{x}]\!][i] \leq [\![\vec{b}]\!][2], i = 1, \ldots, n)$
7 $[\![\vec{c}]\!][j] = (\mathbf{count}([\![\vec{x}]\!][i]) \mid [\![\vec{b}]\!][j] < [\![\vec{x}]\!][i] \leq [\![\vec{b}]\!][j+1], i = 1, \ldots, n), j = 2, \ldots, k$
8 **return** $([\![\vec{b}]\!], [\![\vec{c}]\!])$

2.2 Statistical Tests

2.2.1 Wilcoxon Rank Sum Test and Signed Rank Test

In addition to the t-test and the paired t-test, we created privacy preserving algorithms for the Wilcoxon rank sum test and signed rank test. As t-tests are formally applicable only if the distribution of attribute values in case and control groups follows the normal distribution. If this assumption does not hold, it is appropriate to use non-parametric Wilcoxon tests. The Wilcoxon rank sum test [30] works on the assumption that the distribution of data in one group significantly differs from that in the other.

A privacy-preserving version of the rank sum test follows the standard algorithm, but we need to use several tricks to achieve output privacy. Algorithm 5 gives an overview of how we compute the test statistic $[\![w]\!]$ using the Wilcoxon rank sum test.

For this algorithm to work, we need to cut the database similarly to what was done for the five-number summary. But we need the dataset to retain elements from both groups—cases and controls. On line 1, we combine the two input mask vectors into one making sure that the values that appear in both masks as 1 are removed from the analysis. The function **cut** on line 2 differs from its previous usage in that several vectors are cut at once based on the combined filter $[\![\vec{m}]\!]$.

Similarly to Student's paired t-test, the Wilcoxon signed-rank test [49] is a paired difference test. Our version, given in Algorithm 6, takes into account Pratt's correction [30] for when the values are equal and their difference is 0. As with the rank sum test, we do not take into account the ranking of equal values, but this only gives us a more pessimistic test statistic.

Algorithm 5: Wilcoxon rank sum test

Data: Value vector $[\![\vec{a}]\!]$ and corresponding mask vectors $[\![\vec{m_1}]\!]$ and $[\![\vec{m_2}]\!]$
Result: Test statistic $[\![w]\!]$
1 $[\![\vec{m}]\!] \leftarrow [\![\vec{m_1}]\!] + [\![\vec{m_2}]\!] - ([\![\vec{m_1}]\!] \cdot [\![\vec{m_2}]\!])$
2 $([\![\vec{x}]\!], [\![\vec{m'_1}]\!], [\![\vec{m'_2}]\!]) \leftarrow \mathbf{cut}(([\![\vec{a}]\!], [\![\vec{m_1}]\!], [\![\vec{m_2}]\!]), [\![\vec{m}]\!])$
3 $([\![\vec{\hat{x}}]\!], [\![\vec{\hat{m}_1}]\!][\![\vec{\hat{m}_2}]\!]) \leftarrow \mathbf{sort}(([\![\vec{x}]\!], [\![\vec{m'_1}]\!], [\![\vec{m'_2}]\!]))$
4 $[\![\vec{r}]\!] \leftarrow \mathbf{rank}([\![\vec{\hat{x}}]\!])$
5 $[\![\vec{r_1}]\!] \leftarrow [\![r \cdot \vec{\hat{m}_1}]\!]$ and $[\![\vec{r_2}]\!] \leftarrow [\![r \cdot \vec{\hat{m}_2}]\!]$
6 $[\![R_1]\!] \leftarrow \mathbf{sum}([\![\vec{r_1}]\!])$ and $[\![R_2]\!] \leftarrow \mathbf{sum}([\![\vec{r_2}]\!])$
7 $[\![n_1]\!] \leftarrow \mathbf{sum}([\![\vec{m_1}]\!])$ and $[\![n_2]\!] \leftarrow \mathbf{sum}([\![\vec{m_2}]\!])$
8 $[\![u_1]\!] \leftarrow [\![R_1]\!] - [\![\frac{n_1 \cdot (n_1+1)}{2}]\!]$ and $[\![u_2]\!] \leftarrow [\![n_1 \cdot n_2]\!] - [\![u_1]\!]$
9 **return** $[\![w]\!] \leftarrow \mathbf{min}([\![u_1]\!], [\![u_2]\!])$

Algorithm 6: Wilcoxon signed-rank test

Data: Paired value vectors $[\![\vec{a_1}]\!]$ and $[\![\vec{a_2}]\!]$ for n subjects, mask vector $[\![\vec{m}]\!]$
Result: Test statistic $[\![w]\!]$
1 $([\![\vec{x_1}]\!], [\![\vec{x_2}]\!]) \leftarrow \mathbf{cut}(([\![\vec{a_1}]\!], [\![\vec{a_2}]\!]), [\![\vec{m}]\!])$
2 $[\![\vec{d}]\!] \leftarrow [\![\vec{x_1}]\!] - [\![\vec{x_2}]\!]$
3 Let $[\![\vec{d'}]\!]$ be the absolute values and $[\![\vec{s}]\!]$ be the signs of elements of $[\![\vec{d}]\!]$
4 $[\![\vec{\hat{s}}]\!] \leftarrow \mathbf{sort}(([\![\vec{d'}]\!], [\![\vec{s}]\!]))$
5 $[\![\vec{r}]\!] \leftarrow \mathbf{rank_0}([\![\vec{\hat{s}}]\!])$
6 **return** $[\![w]\!] \leftarrow \mathbf{sum}([\![\vec{\hat{s}} \cdot \vec{r}]\!])$

First, on line 3, both data vectors are cut based on the mask vector similarly to what was done in Algorithm 5. The signs are then sorted based on the absolute values $[\![\vec{d'}]\!]$ (line 4) and the ranking function $\mathbf{rank_0}$ is called. This ranking function differs from the function **rank** because we need to exclude the differences that have the value 0. Let the number of 0 values in vector $[\![\vec{d}]\!]$ be $[\![k]\!]$. As $[\![\vec{d}]\!]$ has been sorted based on absolute values, the 0 values are at the beginning of the vector so it is possible to use $[\![k]\!]$ as the offset for our ranks. Function $\mathbf{rank_0}$ assigns $[\![\vec{r}]\!][i] \leftarrow 0$ while $[\![\vec{\hat{s}}[i] = 0]\!]$, and works similarly to **rank** on the rest of the vector $[\![\vec{\hat{s}}]\!]$, with the difference that $i \in \{1, \ldots, [\![n-k]\!]\}$.

2.2.2 The χ^2-Tests for Consistency.

If the attribute values are discrete such as income categories then it is impossible to apply t-tests or their non-parametric counterparts and we have to analyse frequencies of certain values in the dataset. The corresponding statistical test is known as χ^2-test.

The standard χ^2-test statistic is computed as

$$\chi^2 = \sum_{i=1}^{k} \sum_{j=1}^{2} \frac{(f_{ji} - e_{ji})^2}{e_{ji}} \ ,$$

where f_{ji} is the observed frequency and e_{ji} is the expected frequency of the i-th option and j-th group. For simplification, we denote $c_i = f_{1i}$ and $d_i = f_{2i}$, then the frequencies can be presented as the contingency table 2.1. Let p_i be the sum of column i, r_j be the sum of row j and n be the number of all observations. The estimated frequency e_{ji} is computed as

$$e_{ji} = \frac{p_i \cdot r_j}{n} \ .$$

	Option 1	**Option 2**	...	**Total**
Cases	c_1	c_2	...	r_1
Controls	d_1	d_2	...	r_2
Total	p_1	p_2	...	n

Table 2.1: Contingency table for the standard χ^2 test

Algorithm 7: χ^2 test

Data: Value vector $[\![\vec{a}]\!]$, corresponding mask vectors $[\![\vec{m_1}]\!]$ and $[\![\vec{m_2}]\!]$ for cases and controls respectively and a contingency table $[\![\mathbf{C}]\!]$ of size $2 \times k$

Result: The test statistic χ^2

1 Let $[\![n]\!]$ be the total count of elements

2 Let $[\![r_1]\!]$ and $[\![r_2]\!]$ be the row subtotals and $[\![p_1]\!], \ldots, [\![p_k]\!]$ be the column subtotals

3 Let $[\![\mathbf{E}]\!]$ be a table of expected frequencies such that $[\![\mathbf{E}]\!][i][j] = \frac{[\![\vec{r_i}]\!] \cdot [\![\vec{p_j}]\!]}{n}$

4 $[\![\chi^2]\!] = \sum_{j=1}^{k} \frac{([\![\mathbf{C}]\!][1][j] - [\![\mathbf{E}]\!][1][j])^2}{[\![\mathbf{E}]\!][1][j]} + \frac{([\![\mathbf{C}]\!][2][j] - [\![\mathbf{E}]\!][2][j])^2}{[\![\mathbf{E}]\!][2][j]}$

5 **return** $[\![\chi^2]\!]$

Algorithm 7 shows how to calculate the χ^2 test statistic based on a contingency table. If the null hypothesis is supported if the computed χ^2 value does not exceed the critical value from the χ^2 table with $k - 1$ degrees of freedom.

Often, the number of options in the contingency table is two—subjects who have a certain property and those who do not. Therefore, we look at an optimised version of this algorithm that works where the number of options in our test be 2. Then the test statistic can be simplified and written as

$$[\![t]\!] = [\![\frac{(c_1 + d_1 + c_2 + d_2)(d_1 c_2 - c_1 d_2)^2}{(c_1 + d_1)(c_1 + c_2)(d_1 + d_2)(c_2 + d_2)}]\!] \ .$$

The privacy-preserving version of the χ^2-test is implemented simply by evaluating the algorithm using SMC operations.

2.3 Conclusion

As a result, we were able to conduct a full-scale experimental statistical study so that confidential data were always processed using SMC. The strengths of our solution are generality, precision and practicality. We show that secure multi-party computation is flexible enough for implementing complex applications.

The work described in this chapter is discussed in more detail in the paper [10] that is available in Appendix A of this deliverable.

Chapter 3

Transformation-based Linear Programming

This chapter introduces some problems of the transformation-based linear programming that have been present in the previous works and demonstrates its insecurity. It presents concrete attacks against published methods following this approach. It has been proven that there are issues that cannot be resolved at all using the particular known class of efficient transformations that has been used before.

3.1 Privacy-preserving linear programming

We consider linear programming tasks in the canonical form

$$\text{minimize } \mathbf{c}^{\mathrm{T}} \cdot \mathbf{x}, \text{ subject to } A\mathbf{x} = \mathbf{b}, \mathbf{x} \geq \mathbf{0} \ . \tag{3.1}$$

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 canonical form (3.1) of LP is equivalent to its *standard form*

$$\text{maximize } \mathbf{c}^{\mathrm{T}} \cdot \mathbf{x}, \text{ subject to } A\mathbf{x} = \mathbf{b}, \mathbf{x} \geq \mathbf{0} \ . \tag{3.2}$$

Indeed, the inequalities of the canonical form may be replaced with equalities by introducing slack variables. Each equality may be substituted with two inequalities of opposite directions.

3.2 Attacks against Transformation-based Linear Programming

In [33] we demonstrate a number of attacks against proposed protocols for privacy-preserving linear programming based on publishing and solving a transformed version of the problem instance. Our attacks exploit the geometric structure of the problem, which has mostly been overlooked in the previous analyses and is largely preserved by the proposed transformations.

3.2.1 Transformations Used in the Previous Works

Let a linear programming task be given in its standard form (3.2). The main basic transformations from the related works are the following.

Multiplying from the left. Multiplying A and $\mathbf{b}$ by a random invertible matrix P from the left does not change the feasible region of the linear program at all, and it remains revealed. All the solutions to the system, including the optimal solution, remain the same.

Multiplying from the right. Multiplying A and $\mathbf{c}$ by a positive monomial matrix Q from the right results in scaling and permuting the variables.

Shifting In some works, the initial variable vector $\mathbf{x}$ is not only scaled, but also shifted. This is done by introducing special slack variables for each shifted variable.

3.2.2 The Problems of Slack Variables

We introduce an attack that allows to remove the scaling and the permutation of variables. This is possible in the setting where all the constraints are represented by inequalities, one of the parties knows at least two inequality constraints, and the locations of the slack variables can be traced down. Our attack is polynomial-time. Let us state this problem in general.

Suppose that the initial linear program is given in its canonical form (3.1). In the standard approach proposed by the previous works, the inequalities are transformed to equalities by bringing the linear program to its standard form (3.2) introducing slack variables $\mathbf{x_s}$.

$$\text{maximize } \begin{pmatrix}\mathbf{c}\\ \mathbf{0}\end{pmatrix}^{\mathrm{T}} \cdot \begin{pmatrix}\mathbf{x}\\ \mathbf{x_s}\end{pmatrix}, \text{ subject to } \begin{pmatrix}A & I\end{pmatrix}\begin{pmatrix}\mathbf{x}\\ \mathbf{x_s}\end{pmatrix} = \mathbf{b}, \begin{pmatrix}\mathbf{x}\\ \mathbf{x_s}\end{pmatrix} \geq \mathbf{0} \ .$$

The columns of the matrix are first being scaled and permuted multiplying it by a monomial matrix Q from the right. Then it is multiplied by a random invertible matrix P from the left.

If the vector $\mathbf{c}$ is treated separately from the constraints, as it was done in the previous works, then the locations of the slack variables are clearly visible even after the permutation since their values in the cost vector are 0, and scaling does not affect 0 in any way. This issue allows to use row operations to reduce the constraints back to the standard form $\begin{pmatrix}A' & I\end{pmatrix}\begin{pmatrix}\mathbf{x}\\ \mathbf{x_s}\end{pmatrix} = \mathbf{b}'$. Here A' and $\mathbf{b}'$ may be different from A and $\mathbf{b}$. However, since we know that the feasible region is just scaled, the inequalities $A'\mathbf{x} \leq \mathbf{b}'$ define the scaled polyhedron of the initial program. Then we may use the knowledge about the initial inequalities to cancel out the scaling and permutation. This attack is described more precisely in [33].

The locations of slack variables can be potentially hidden by adding the equation $\mathbf{c}^{\mathrm{T}} \cdot \mathbf{x} - w = w_0$ to the constraints and minimizing over the variable w only. However, we present attacks that detect the slack variables due to their special behaviour.

1. If the entries of P and Q are sampled from a uniform or a (folded) normal distribution, the columns that correspond to the slack variables have in general greater variance. This problem can be resolved by bringing the matrix A to its reduced row echelon form instead of multiplying it by P. However, since this form has to be computed in a privacy-preserving way, this transformation is more expensive.

2. The other attack is based on the geometric properties of the feasible region. It turns out that, optimizing the LP task in random directions, slack variables tend to take the value 0 much more often than the initial variables. The attack is efficient in practice, and for numerous LP tasks the slack variables have been located perfectly all at once. The particular results can be seen in [33].

The slack variables also allow to undo shifting. Namely, although the values of the initial variables $\mathbf{x}$ are hidden by a randomness vector $\mathbf{r}$, the scaled values of $\mathbf{x}$ are leaked into the special slack variables $\mathbf{s}$ that have been introduced by shifting. Distinguishing slack and non-slack variables in this case is especially easy due to the special relations between these variables. After removing non-slack variables by gaussian elimination, we obtain an LP with variables $\mathbf{s}$ whose feasible region is just a scaled version of the original one, without shifting. The attack is described in more details in [33].

3.3 Impossiblity of Secure Transformation-based Linear Programming

As a conclusion of the previous section, the attacks are efficient in practice and cast serious doubt to the viability of transformation-based approaches in general. In our next paper [34], we study the security definitions and methods for transformation-based outsourcing of linear programming in general. The recent attacks have shown the deficiencies of existing security definitions; thus we propose a stronger, indistinguishability-based definition of security of problem transformations that is very similar to IND-CPA security of encryption systems. We study the realizability of this definition for linear programming and find that barring radically new ideas, there cannot exist transformations that are secure information-theoretically or even computationally.

The privacy requirements for our task are the following. The sizes m and n, and the bounding box of the feasible region are public. Matrix A and the vector $\mathbf{b}$ have to remain secret. The solution $\mathbf{x}_{\text{opt}}$ 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 $A'\mathbf{y} = \mathbf{b}'$, $\mathbf{y} \geq \mathbf{0}$", such that from the solution $\mathbf{y}_{\text{opt}}$ and secret data generated during the transformation, we could efficiently recover the solution $\mathbf{x}_{\text{opt}}$ to the original task.

Quite clearly, the transformations described in the previous subsection do not satisfy this privacy requirement. Since these transformations are all some instances of affine transformations (linear transformation + shifting), we have studied if a more general class of affine transformations can satisfy our security requirement.

- **Information-theoretic Security** First of all, we studied if information-theoretic security is possible. Since computational security would most probably require to introduce some new assumptions (since cryptography over real numbers has not received too much attention so far), achieving information-theoretic or at least statistical security would be really preferable. Theoretically, it could be achieved by introducing more dimensions to the feasible region and encoding a scaled/shifted initial feasible region as some projection. However, even if we are able to encode all possible shapes as some non-trivial projection, we will still have problems with scaling. The idea behind the proof is that an affine transformation preserves distances between the hyperplanes, and although the distances in the transformed feasible region may also depend on the randomness, they still depend also on the initial distances, what makes the difference traceable in average. The particular construction used as a counterexample can be seen in [34].

  Since in our counterexample we had to use the assumption of perfect secrecy, we did not give up and started studying computational security.

- **Computational Security** We could still hope to hide the initial feasible region by using an affine transformation in such a way that it would be computationally difficult to recover it. We studied the methods used in the previous works and proposed our own affine transformations, but they still were obviously vulnerable. We tried to understand what is the reason why all the proposed methods fail. The problem is that all the hiding except scaling and permutation requires introducing additional variables (even multiplication by an invertible matrix from the left in general cannot be done without introducing slack variables first). And both in the previous works and in our solutions, these new variables have been distinguishable by their behaviour, so that permutation has not helped to hide them. After being revealed, these variables can be removed by gaussian elimination, thus undoing all the hiding that they provide.

  For that reason, we empirically state the necessary requirement: for a security parameter t, any set of t variables should look the same to the adversary. However, if we want this claim to hold for a reasonable t, it should hold at least for $t = 2$. We have tried to define the properties of a feasible region that would satisfy this requirement. It has turned out that the only suitable geometric shape is a simplex: a polyhedron defined by an equation $\{(x_1, \ldots, x_n) \mid x_1 + \ldots + x_n \leq c\}$ and inequalities $x_1 > 0, \ldots, x_n \geq 0$ for some $c \geq 0$. Such a linear program has only one dimension and hence cannot be used to encode anything reasonable. The full proof can be found in [34].

3.4 Conclusions

We have shown that the current approaches towards transformation-based privacy-preserving outsourcing or multiparty linear programming are unlikely to be successful. Success in this direction requires some radically new ideas in transforming polyhedra and/or in cryptographic foundations violating the rather generous assumptions we have made. We conclude that for solving linear programming problems in privacy-preserving manner, cryptographic methods for securely implementing Simplex or some other linear programming solving algorithm are the only viable approach.

Chapter 4

Privacy-Preserving Execution of Finite Automata

A *deterministic finite automaton (DFA)* over an alphabet Σ is a tuple $A = (Q, q_0, \delta, F)$, where Q is the set of *states*, $q_0 \in Q$ is the *initial state* of the automaton, $F \subseteq Q$ is the set of *accepting states* and $\delta : Q \times \Sigma \to Q$ is the *transition function*. The transition function can be extended to have the type $Q \times \Sigma^* \to Q$, by defining $\delta(q, \varepsilon) = q$ and $\delta(q, sa) = \delta(\delta(q, s), a)$ for all $q \in Q$, $s \in \Sigma^*$ and $a \in \Sigma$. The automaton *accepts* a string s if $\delta(q_0, s) \in F$.

A *non-deterministic finite automaton (NFA)* is also a tuple $A = (Q, q_0, \delta, F)$, but now δ is a function with the type $Q \times \Sigma \to 2^Q$. It can again be extended to the arguments of type $Q \times \Sigma^*$ by defining $\delta(q, \varepsilon) = \{q\}$ and $\delta(q, sa) = \bigcup_{q' \in \delta(q,s)} \delta(q', a)$. The automaton *accepts* a string s if $\delta(q_0, s) \cap F \neq \emptyset$.

4.1 Problem description

We have a private string over the alphabet Σ (which is public). As it is difficult to hide the size of inputs without applying a lot of padding, we assume that the length of the string is public. The individual characters, however, are sensitive data.

We also have a private finite automaton. Again, we are not trying to hide the size of our inputs, hence we assume that the number of states $|Q|$ is public. The transfer function δ and the set of accepting states F are, however, private.

We want to compute whether the automaton accepts the string. The result must remain private, too.

"Privacy" may mean many different things, depending on the number of parties in the system, the coalitions which know or may know different data items, and the potential collusions between parties. We are considering the most general setup described also in D5.2.1 [7, Chap. 1]. We are working in the *Arithmetic Black Box (ABB)* model. In this model, the protocol set for SMC, run by the parties, is such that some kind of *private storage* is implemented. Values held in this private storage can become public only if several of the computing parties cooperate. The input string and the description of the automaton are kept in this private storage. We want to execute the automaton in this string so, that the result of this execution is added to the private storage while nothing about the inputs is made public. Private computation tasks implemented in such manner are universally composable, meaning that the security of some composition of these tasks follows from the security of each task separately.

4.2 Private selection

If privacy were not an issue, then one checks whether a DFA A accepts a string $s = a_1 \cdots a_\ell$ by computing $q_1 = \delta(q_0, a_1), q_2 = \delta(q_1, a_2), \ldots, q_\ell = \delta(q_{\ell-1}, a_\ell)$ and checks whether $q_\ell \in F$. The function δ is typically given in *tabular form*. I.e. to find $\delta(q, a)$, one has to locate the cell (q, a) of δ and read its contents. Such operation is typically much more expensive if the index of the cell is private. Indeed, in this case, the

selection algorithm must "touch" all[1] cells of δ, otherwise one would reveal which cells definitely do not correspond to (q, a). At each cell, the selection algorithm typically has to perform some *non-free* operations with private values[2]. A typical example of private selection performs the scalar product of the table with the characteristic vector of the index. The computation of the characteristic vector requires $\Omega(|\delta|)$ work.

We have shown how almost all expensive operations of a private selection can be moved *offline*, i.e. performed before the automaton and/or the input string are available [36]. In the online phase, we have to perform only a couple of multiplications of private values, irrespective of the sizes of Q and Σ. We will now describe our protocols for that.

We use the usual notation $[\![v]\!]$ for the value v stored in the ABB. The notation $[\![v_1]\!]$ *op* $[\![v_2]\!]$ denotes the computation of v_1 *op* v_2 by the ABB (translated to a protocol in the implementation of the ABB). Let a private table v with m elements be given, let the indices of the cells be $i_1, \ldots, i_m$ (these indices are public and we require them to be non-zero). We require that both $i_1, \ldots, i_m$ and the elements of the table $v_{i_1}, \ldots, v_{i_m}$ are elements of a finite field $\mathbb{F}$ with at least $m+1$ elements. There exist protocols for generating a uniformly random element of $\mathbb{F}$ inside the ABB (denote: $[\![r]\!] \xleftarrow{\$} \mathbb{F}$), and for generating a uniformly random non-zero element of $\mathbb{F}$ together with its inverse (denote: $([\![r]\!], [\![r^{-1}]\!]) \xleftarrow{\$} \mathbb{F}^*$). These protocols require a small constant number of multiplications on average for any ABB [17]. For any $\mathbf{I} = \{i_1, \ldots, i_m\}$ there also exist Lagrange interpolation coefficients $\lambda^{\mathbf{I}}_{j,k}$ depending only on the set $\mathbf{I}$, such that for any polynomial V over $\mathbb{F}$ with degree at most $m-1$ we have $V(x) = \sum_{j=0}^{m-1} c_j x^j$, where $c_j = \sum_{k=1}^{m} \lambda^{\mathbf{I}}_{j,k} V(i_k)$. These coefficients are public and can be computed in the offline phase, too.

Our private selection algorithm Alg. 8 receives as inputs the private values $[\![v_{i_1}]\!], \ldots, [\![v_{i_m}]\!]$ and the private index $[\![j]\!]$, where $j \in \{i_1, \ldots, i_m\}$. It responds with the private value $[\![v_j]\!]$. The work of this algorithm is divided into three phases. During the vector-only phase the table $([\![v_{i_1}]\!], \ldots, [\![v_{i_m}]\!])$ is available, while $[\![j]\!]$ can be used only in the online phase. This corresponds to common use cases (e.g. spam filtering), where the DFA is known before the actual input string.

Algorithm 8: Private selection protocol

Data: Vector of indices $i_1, \ldots, i_m \in \mathbb{F} \backslash \{0\}$
Data: Vector of values $([\![v_{i_1}]\!], \ldots, [\![v_{i_m}]\!])$ with $v_{i_1}, \ldots, v_{i_m} \in \mathbb{F}$.
Data: Index $[\![j]\!]$ to be looked up, with $j \in \{i_1, \ldots, i_m\}$.
Result: The looked up value $[\![w]\!] = [\![v_j]\!]$.

Offline phase
$([\![r]\!], [\![r^{-1}]\!]) \xleftarrow{\$} \mathbb{F}^*$
for $k = 2$ **to** $m-1$ **do** $[\![r^j]\!] \leftarrow [\![r]\!] \cdot [\![r^{j-1}]\!]$ Compute the coefficients $\lambda^{\mathbf{I}}_{j,k}$ from $i_1, \ldots, i_m$.
Vector-only phase
foreach $k \in \{0, \ldots, m-1\}$ **do** $[\![c_k]\!] \leftarrow \sum_{l=1}^{m} \lambda^{\mathbf{I}}_{k,l} [\![v_l]\!]$ **foreach** $k \in \{0, \ldots, m-1\}$ **do** $[\![y_k]\!] \leftarrow [\![c_k]\!] \cdot [\![r^k]\!]$
Online phase
$z \leftarrow \mathsf{retrieve}([\![j]\!] \cdot [\![r^{-1}]\!])$
$[\![w]\!] = \sum_{k=0}^{m-1} z^k [\![y_k]\!]$

Algorithm Alg. 8 represents the vector $(v_{i_1}, \ldots, v_{i_m})$ as a polynomial $V(x) = \sum_{k=0}^{m-1} c_k x_k$, such that $V(i_k) = v_{i_k}$. The value $V(j)$ is computed as $\sum_{k=0}^{m-1} (jr^{-1})^k (c_k r^k)$, where r is a random non-zero element of $\mathbb{F}$. In this way, jr^{-1} is also a random non-zero element of $\mathbb{F}$ and may be made public. The private random element r together with its inverse r^{-1} and its powers can be computed in the offline phase while the coefficients c_i of V and the products $c_k r^k$ can be computed in the vector-only phase. In the online phase, we perform a single multiplication (to find jr^{-1}) with private values, and a single declassification.

[1] Unless the table has been somehow scrambled before, as in implementations of *Oblivious RAM* [21]

[2] All operations except the computation of linear combinations of private values with public coefficients; which does not require any expensive computations or communication in any existing implementations of ABBs

Algorithm Alg. 8 can be used with any ABB implementation that operates on elements of a finite field of sufficient size, and it inherits the security guarantees of the ABB.

The complexity of private selection is thus shifted to the offline and vector-only phases. In the vector-only phase we still perform m multiplications with private values (while computing $[\![y_k]\!]$). For certain ABB implementations, it is possible to avoid that cost. Namely, for ABBs based on Shamir's secret sharing [46] and using the Gennaro-Rabin-Rabin multiplication protocol [27], the computation of the scalar product of two private vectors is no more expensive than a single multiplication of private values. We hence rewrite the vector-only and online phases of the private selection protocol as depicted in Alg. 9.

Algorithm 9: Improved vector-only and online phases of the private lookup protocol

Data: Lagrange interpolation coefficients $\lambda^{\mathbf{I}}_{j,k}$
Data: Random non-zero $[\![r]\!]$ and its powers $[\![r^{-1}]\!], [\![r^2]\!], \ldots, [\![r^{m-1}]\!]$.
Data: Vector of values $([\![v_{i_1}]\!], \ldots, [\![v_{i_m}]\!])$ with $v_{i_1}, \ldots, v_{i_m} \in \mathbb{F}$.
Data: Index $[\![j]\!]$ to be looked up, with $j \in \{i_1, \ldots, i_m\}$.
Result: The looked up value $[\![w]\!] = [\![v_j]\!]$.
1 Vector-only phase
2 **foreach** $k \in \{0, \ldots, m-1\}$ **do** $[\![c_k]\!] \leftarrow \sum_{l=1}^{m} \lambda^{\mathbf{I}}_{k,l} [\![v_l]\!]$ Online phase
3 $z \leftarrow \mathsf{retrieve}([\![j]\!] \cdot [\![r^{-1}]\!])$
4 **foreach** $j \in \{0, \ldots, m-1\}$ **do** $[\![\zeta_j]\!] \leftarrow z^j [\![r^j]\!]$ $[\![w]\!] = ([\![c_0]\!], \ldots, [\![c_{m-1}]\!]) \cdot ([\![\zeta_0]\!], \ldots, [\![\zeta_{m-1}]\!])$

Compared to Alg. 8, we have moved the entire computation of the products $z^j [\![c_j]\!] [\![r^j]\!]$ to the online phase, thereby reducing the vector-only phase to the computation of certain linear combinations. The complexity of the online phase has increased by the computation of the scalar product in the last line. The total cost of the online phase is thus two multiplications and one declassification.

4.3 DFA execution

Private selection is all that we need to execute a private DFA on a private string. For a string of length ℓ, we sequentially perform ℓ private selections to find $[\![q_1]\!], \ldots, [\![q_\ell]\!]$. Finally, we will perform one more private selection on the characteristic vector of the set of accepting states F, using q_ℓ as the index. In this way, we have emulated the sequential DFA execution algorithm.

One can also execute a DFA $A = (Q, q_0, \delta, F)$ on a string $s = a_1, \ldots, a_\ell$ in a parallel manner. For any $t \in \Sigma^*$, let $\delta(\cdot, t) : Q \to Q$ be the mapping we get from δ (extended to $Q \times \Sigma^*$) by fixing its second argument as t. The function $\delta(\cdot, \varepsilon)$ is the identity function on Q and the equality $\delta(\cdot, t_1 t_2) = \delta(\cdot, t_2) \circ \delta(\cdot, t_1)$ holds for all $t_1, t_2 \in \Sigma^*$. If all functions are represented in tabular form, then the computation of the composition requires $|Q|$ lookups. We can now compute $\delta(\cdot, s)$ in a divide-and-conquer fashion, and then check whether $\delta(q_0, s) \in F$. The computation requires $\log \ell$ "steps". The total amount of work is $|Q|$ times the work performed by the sequential execution.

We have not implemented the parallel version of the DFA execution algorithm in privacy-preserving manner, using our private selection algorithm as a subroutine. For certain parameters (small —Q— and large ℓ), this algorithm may in practice perform much better than the sequential algorithm, due to its smaller round complexity.

4.4 NFA execution

Due to their non-deterministic nature, NFAs are more complicated to handle in a secure manner. We see that even though the NFA execution starts from a single state, after the intermediate steps it can generally be in a subset of states. In order to account for this, we will use characteristic vectors of the intermediate

sets $\mathcal{Q}_i = \boldsymbol{\delta}_A(a_1 \cdots a_i)$ to represent them. Let $\mathbf{q}^i = (q_0^i, q_1^i, \ldots, q_{m-1}^i)$ be a binary vector, where $q_j^i = 1$ iff the state $q_j \in \mathcal{Q}_i$.

As $\mathcal{Q}_0 = \{q_0\}$, we have $\mathbf{q}^0 = (1, 0, \ldots, 0)$. Subsequent $\mathbf{q}^i$-s will depend both on the given automaton A and the string s. Namely, in order to determine $\mathbf{q}^i$ from $\mathbf{q}^{i-1}$, δ and a_i, we can compute

$$q_j^i = \bigvee_{q \in \mathcal{Q}_{i-1}} [q_j \in \delta(q, a_i)] = \bigvee_{k=0}^{m-1} q_k^{i-1} \& [q_j \in \delta(q_k, a_i)] \tag{4.1}$$

for all the components q_j^i of the characteristic vector $\mathbf{q}^i$.

In order to determine efficiently whether $q_j \in \delta(q_k, a_i)$, we need an efficient representation of δ as well. We will represent it as a look-up table $\overline{\delta} : Q \times Q \to \mathcal{P}(\Sigma)$, where $a_i \in \overline{\delta}(q_k, q_j)$ iff $q_j \in \delta(q_k, a_i)$. To encode subsets of Σ, we will once again use characteristic vectors; let $\mathcal{S} \subseteq \Sigma$ be encoded by vector $\mathbf{s} = (s_1, \ldots, s_n)$ where $s_i = 1$ iff the corresponding $\sigma_i \in \mathcal{S}$. Similarly, we also represent the characters of the string using binary characteristic vectors $\mathbf{a}_1, \ldots, \mathbf{a}_\ell$, where $\mathbf{a}_i = (a_i^1, \ldots, a_i^n)$ and $a_i^j = 1$ iff $a_i = \sigma_j$. As a result, the value of the predicate $q_j \in \delta(q_k, a_i)$ can be computed as a dot product $\overline{\delta}(q_k, q_j) \cdot \mathbf{a}_i$.

The complexity of NFA execution is significantly higher than DFA execution. Some of its steps can also be performed offline. Namely, the disjunction in (4.1) can be computed by adding up the elements and comparing the result to 0 (and flipping the outcome) [37]. Working in a suitable field, comparison to 0 may be implemented using just one round of online multiplications using the protocol by Lipmaa and Toft [40] (though some precomputation is necessary).

4.5 Applications of our results

We have shown that arithmetic black boxes support fast lookups from private tables according to a private index. We have used this operation to obtain very efficient DFA execution algorithms. Our results show that for private lookups in an ABB, complex techniques based on Oblivious RAMs [21] are not necessary. We expect our techniques to have wide applicability in privately processing graph-like data structures.

Chapter 5

Public Verifiability for Parties in SMC

In this chapter we propose a method that allows to detect the parties that have violated the protocol rules after the computation has ended, thus making the protocol secure against covert attacks. This approach can be useful in the settings where for any party it is fatal to be accused in violating protocol rules. In this way, up to the verification, all the computation can be performed in semi-honest model, which makes it very efficient in practice. The verification is statistical zero-knowledge, and it it based on linear probabilistically checkable proofs (PCP) for verifiable computation. Hence each malicious party is detected with probability $1 - \varepsilon$ for a negligible ε that is defined by the failure of the corresponding linear PCP. The initial protocol has to be executed only once, and the verification requires in total 3 additional rounds. The verification also ensures that all the parties have sampled all the randomness from an appropriate distribution. Its efficiency does not depend on whether the inputs of the parties have been shared, or each party uses its own private input.

5.1 Introduction

The semi-honest and the malicious model are the two main models in which cryptographic protocols are studied. In the semi-honest model, the adversary is curious about the values it gets, and it tries to extract information out of them, but it follows the protocol rules honestly. In the malicious model, the adversary is allowed to do whatever it wants. In addition to these traditional models, a notion of covert security was proposed in [2]. In this model, the adversary is malicious, but it will not cheat if it will be caught with a non-negligible probability, which can be defined more precisely as a security parameter. This notion is very realistic in many computational models, where the participants care about their reputation and will not cheat even if this probability is not close to 1.

Some works have been dedicated to covert security [38, 19], where [38] treats the security for two-party computation based on garbled circuits, both the covert and the malicious cases, and [19] deals with honest majority protocols for an arbitrary number of parties. A more precise definition of *covert security with public verifiability* has been proposed in [1]. This allows the cheater to be blamed publicly.

5.2 Our Contribution

In this work we propose a scheme that is based on succinct computation verification. Our work is closely related to [19] that is dealing with honest majority protocols for an arbitrary number of parties. The solution proposed in [19] is based on running the initial protocol on two inputs, the real shares and the dummy shares. In this case, the real shares should be indistinguishable from random, and hence in the beginning the protocol is being rewritten to a shared form. Differently from [19], our solution does not require rewriting the original protocol. The original protocol has to be run only once, and each malicious party is detected with probability $1 - \varepsilon$ for a negligible ε. Our approach is statistical zero-knowledge, and it it based on linear probabilistically checkable proofs for verifiable computation. The particular PCP that we

are using is the one proposed in [4]. The quantity ε is defined by the failure of the corresponding linear PCP behind the protocol. The verification requires in total 3 additional rounds. Additionally, it ensures that all the parties have sampled all the randomness from an appropriate distribution.

The major drawback of our scheme is that the number of values sent per one round is exponential in the number of parties. In [19], efficiency is achieved by reducing the probability of being detected from 1/2 to 1/4. We cannot use the same approach in our case since the probability of being detected would immediately become negligible. Nevertheless, the settings make the verification very efficient for a small number of parties.

Similarly to [19], we prove the security of our scheme in UC model [14].

5.3 Protocol Description

This section gives a general overview of the protocol. We state the assumptions on which our protocol is based, describe which precomputation has to be performed before running the original protocol, and which messages should be sent in addition to the initial ones during the execution. We briefly explain what happens in the final verification, without going into details. Similarly to [19], all the inputs and the communication values are committed, but in a special shared way, using signatures. Due to the sharing, the signatures do not have to hide the messages at all, and should be rather perfectly binding. The entire verification is still zero-knowledge.

5.3.1 Notation

Throughout this chapter, we use the following notation:

- the upper case letters A denote matrices;
- the bold lower case letters $\mathbf{b}$ denote vectors;
- $\langle \mathbf{a}, \mathbf{b} \rangle$ denotes the scalar product of $\mathbf{a}$ and $\mathbf{b}$;
- $(\mathbf{a}||\mathbf{b})$ is a concatenation of vectors $\mathbf{a}$ and $\mathbf{b}$.

5.3.2 Assumptions

Our verification protocol is based on security of some other schemes. Here is the list of used assumptions.

- Secure point-to-point channels between each pair of parties.
- Broadcast channels between subsets of parties.
- Honest Verifier Statistical Zero-Knowledge Linear Probabilistically Checkable Proofs for verifiable computation [39, 26, 4, 6]. In particular, all the complexity estimations in this chapter are based on the solution proposed in [4].
- Functionality that allows to prove to third parties which messages one received during the protocol, and to further transfer such revealed messages. This allows to protect the initial protocol from halting problem (when the computation cannot proceed due to some malicious party that is either just doing nothing, or causes some other party to wait by sending wrong messages). We use the solution proposed in [19].

5.3.3 The Protocol Outline

We describe briefly the initial settings, and how the new verifiable protocol differs from the original one.

- In the initial settings, we have a set of arithmetic circuits C_i^j over some finite field $\mathbb{F}$, where C_i^j is the circuit computed by the party M_i on the j-th round of computation. Some outputs of C_i^j may be used as inputs for some C_k^{j+1}, so there is some communication between the parties. Each circuit may use some randomness that comes from random uniform distribution in $\mathbb{F}$ (this is sufficient to model any other distribution). The circuits could be boolean as well, since there also exist linear probabilistically checkable proofs based on boolean circuits [39], so our verification is not restricted to computation over some certain field.

- The computation is performed by n parties. Let them be denoted M_i for $i \in \{1, \ldots, n\}$. A necessary condition is that at least $t = \lfloor n/2 \rfloor + 1$ are honest.

- Before the execution of original protocol starts, the inputs of the parties are committed in a special way. Let the input of the party M_i be represented by a vector $\mathbf{x}_i$ over $\mathbb{F}$. M_i represents $\mathbf{x}_i$ as $\binom{n-1}{t}$ distinct sums of the form $\mathbf{x}_i = \sum_{k \in T_j} \mathbf{x}_{ikT_j}$ for $j \in \left\{1, \ldots, \binom{n-1}{t}\right\}$, where each T_j represents a distinct subset of t other parties, and k corresponds to one particular party in that subset. The idea is that each party has to prove its honestness to any subset of t other parties. All the shares are signed and distributed amongst the corresponding parties. Although the number $\binom{n-1}{t}$ is exponential, computing all $t \cdot \binom{n-1}{t}$ signatures is not less efficient than computing just one, for example using hash Merkle tree.

- The randomness used in the protocols should also be committed in the same way. Moreover, we want to ensure that it indeed comes from random uniform distribution, without revealing to anyone its value.

  - Let an arbitrary set of t parties be responsible for generating the randomness. Let these parties be called "generators". By honest majority assumption, at least one of them is honest. For each M_i, they generate the randomness $\mathbf{r}_i$ as follows. Each generator M_j generates $\mathbf{r}_{ji}$ of the same length that $\mathbf{r}_i$ should be. The idea is to take $\mathbf{r}_i = \mathbf{r}_{i1} + \ldots + \mathbf{r}_{it}$. Since at least one party is honest, the vector $\mathbf{r}_i$ comes from a random uniform distribution.
  - Each generator M_j represents its $\mathbf{r}_{ij}$ as $\binom{n-1}{t}$ distinct sums of the form $\mathbf{r}_{ij} = \sum_{k \in T_\ell} \mathbf{r}_{ijkT_\ell}$ for $\ell \in \left\{1, \ldots, \binom{n-1}{t}\right\}$. All the shares are signed and sent to M_i. After M_i receives $\mathbf{r}_{ij}$ from all generators M_j, it may compute the sum of all $\mathbf{r}_{ij}$ and use it as $\mathbf{r}_i$ (M_i has to verify if the shares for different sets T_ℓ indeed all represent the same value). Then M_i signs all the received shares also by itself, and distributes the shares and the signatures (both signed by M_i and the corresponding generator M_j) amongst appropriate subsets of t parties, similarly to $\mathbf{x}_i$.

- The original protocol is computed in the same way as before. Additionally, each communicated vector $\mathbf{c}_{ij}^\ell$ sent by M_i to M_j on the round ℓ is presented as $\binom{n}{t}$ distinct sums $\mathbf{c}_{ij}^\ell = \sum_{k \in T_{j'}} \mathbf{c}_{ijkT_{j'}}^\ell$ (here we have $\binom{n}{t}$ instead of $\binom{n-1}{t}$ since both communicating parties should later verify the consistency of this value from each other). Along with each $\mathbf{c}_{ij}^\ell$, M_j receives the signature of $\mathbf{c}_{ij}^\ell$ and the signatures of all the shares $\mathbf{c}_{ij\ell kT_{j'}}^\ell$. M_j checks if the signatures are all indeed valid, and in turn signs them. M_j distributes the corresponding signatures (both signed by M_i and M_j) amongst each T_j. Here M_j is unable to check whether the shares under the signatures are valid and indeed sum up to $\mathbf{c}_{ij}^\ell$. All the shares will be distributed after the protocol execution, and then M_i may present the signature of $\mathbf{c}_{ij}^\ell$ to complain.

- After the protocol computation ends, all the communication shares are finally distributed. Each party M_j is verified for honestness. A party is honest iff it can prove that it acted according to the protocol, given the signed input, randomness, and communication that it had with the other parties. It has to

perform a 3-round interactive proof with each subset of t parties in parallel. Since each subset of t parties holds all the shares of all the committed values, they are able to reconstruct the committed values and check if the proof indeed corresponds to them.

In general, in a linear PCP the prover has to prove the knowledge of a vector $\pi = (\mathbf{p}||\mathbf{d})$ such that certain combinations of $\langle \pi, \mathbf{q}_i \rangle$ for special challenges $\mathbf{q}_1, \ldots, \mathbf{q}_5$ should be equal to 0, and $\mathbf{d}$ corresponds to the committed values. The problem is that the prover cannot see any of the $\mathbf{q}_i$ before committing the proof, but at the same time π should remain private.

In particular, for any subset of t verifiers, the following has to be done (ordered by rounds).

1. The verifiers agree on a random $\tau \in \mathbb{F}$ that is sufficient to generate all $\mathbf{q}_1, \ldots, \mathbf{q}_5$ (in one round). The prover generates shares $\pi = \pi_1 + \ldots + \pi_t$ (where the $\mathbf{d}$ part is shared in the same way as it was committed to the given set of t verifiers) and distributes them amongst the parties. Each verifier checks if the part that corresponds to $\mathbf{d}$ is consistent with the signatures of shares sent during the computation.
2. Each verifier V_i computes and publishes $\langle \pi_i, \mathbf{q}_j \rangle$ for $j \in \{1, \ldots, 5\}$. The τ is published. Everyone may compute $\langle \pi_1, \mathbf{q}_j \rangle + \ldots + \langle \pi_t, \mathbf{q}_j \rangle = \langle \pi, \mathbf{q}_j \rangle$ for $j \in \{1, \ldots, 5\}$ and locally verify the necessary combinations. The prover checks if all the scalar products are computed correctly, and complains if necessary.

A party is claimed honest iff it succeeds in all the $\binom{n-1}{t}$ proofs against t other parties. This means that even if it was in collaboration with $t-2$ other malicious parties, there exists a subset of t all-honest parties that will definitely accept only the correct proof. We also need to ensure that the presence of malicious parties will not make the proof fail for an honest prover, and this can be done by revealing the signatures that correspond to the shares of incorrect scalar products. An honest party is safe to open them since they are known by the adversary anyway. The details of accusations and the security proofs can be seen in [35].

5.3.4 Properties

In our settings, we have n parties M_i. Compared to the original protocol, for each M_i the proposed solution has the following computational overheads.

- Let $p = \binom{n-2}{t-1}$. This is the number of t-sets in which one party participates as a verifier. If everyone is honest, then in order to verify M_j's honestness, M_i has to send the following messages:
  - In the initial protocol, send two signatures and $tp+p$ vectors of length $O(|C|)$ to each of the $n-1$ parties (tp for the randomness, and p for the inputs).
  - During the protocol execution, in addition to the original protocol communication, send $r(1+n)$ signatures to each of the $n-1$ receiver parties, where r is the number of rounds (one signature for the entire message and n for its shares). Each receiver M_j produces rn more signatures of the same values (that correspond to the shares). All these signatures are distributed by each receiver M_j amongst corresponding $n-1$ remaining parties (including M_i).
  - After the protocol execution, compute locally the auxiliary values for the proof in $O(|C| \log |C|)$ steps, as shown in [4]. Send to each of the other $n-1$ parties in parallel $1+(n-1)$ signatures and $p+p(n-1)$ vectors of length $O(|C|)$: p for intermediate variables (for each proof separately, signed with one signature), and $p(n-1)$ for communication.

  As a verifier, in the verification process each M_i has to do the following:
  - Locally generate, sign and broadcast a random element of $\mathbb{F}$.
  - Locally generate p state informations $\mathbf{u}$ and $5p$ challenge vectors $\mathbf{q}_k$ of length $O(|C|)$ (according to the arithmetic circuit). This can be done in $O(|C|)$ steps, as shown in [4].

  - Locally sum up p times t^2 vectors of length $O(|C|)$.
  - Locally concatenate 4 vectors of total result length $O(|C|)$: the shares of the input, randomness, communication, and the intermediate values. This is done p times, for each verification set.
  - Locally compute $5p$ scalar products of vectors of length $O(|C|)$ and broadcast them (5 for each proof).
  - In the end, compute a constant number of local operations based on these scalar products: 2 multiplications, 3 additions, 1 scalar product of length $O(|\mathbf{v}|)$ for the part of the input $\mathbf{v}$ whose value is public (which is in general just the constant 1), all operations in $\mathbb{F}$. Everything is done p times, for each proof.

- If something goes wrong with the proof of M_j's honestness, then in the worst case each sent message has to be sent in such a way that it is possible to prove afterwards what has been sent to whom. The $\mathcal{F}_{transmit}$ functionality from [19] requires each message to be broadcast to all $n-1$ parties, and then this message should be delivered by each of the $n-2$ remaining parties to the receiver. No additional signatures are needed since we have already considered all of them in the case where everyone acts honestly.

According to the Linear PCP description from [4], a dishonest prover may cheat with probability $\frac{2m}{|\mathbb{F}|}$ where m is the number of multiplication gates in the circuit. This means that either the field should be large enough, or the verification should be repeated k times, so that $\left(\frac{2m}{|\mathbb{F}|}\right)^k$ is negligible. All the k verifications can be done in parallel, by generating k sets of challenges instead of one, thus do not increasing the number of rounds at all, and increasing the communication in total by $p(n-1)k$ field elements and $p(n-1)k$ proof vector shares.

5.4 Using the Proposed Protocol in Secure Multiparty Computation Platforms

In this section we discuss how the proposed verification could be used in Secure Multiparty Computation Platforms. More precisely, here we should consider the case where in addition to *computing* parties (that participate in the protocol) we may have *input* parties (that provide the inputs, sharing them in some way amongst the computing parties) and the *result* parties (that receive the final output). In our protocol, the computing parties do commit the inputs before the computation starts, but we must ensure that these are indeed the same inputs that have been provided by the input parties.

5.4.1 Treating Inputs/Outputs as Communication

As a simpler solution, we may just handle the input and the output similarly to communication. Hence the following enhancements are made.

1. Let the number of input parties be N. In the beginning, each input party P_i generates the shares $\mathbf{x}_{i1}, \ldots, \mathbf{x}_{in}$ (according to an arbitrary sharing scheme) from all the computing parties $M_1, \ldots, M_n$, as it would do without the verification. Each M_j should now use the input vector $\mathbf{x}_j = (\mathbf{x}_{1j}|| \ldots ||\mathbf{x}_{Nj})$, where each $\mathbf{x}_{ij}$ is provided by an input party P_i. Now, for each $\mathbf{x}_{ij}$, P_i generates by itself all the $\binom{n-1}{t}$ shares $\mathbf{x}_{ijkT_\ell}$ such that $\sum_{k \in T_j} \mathbf{x}_{ijkT_\ell} = \mathbf{x}_{ij}$, signs all these shares, and sends them to M_j. As before, M_j should also sign all of these shares before redistributing them amongst all the verifier t-sets. The verifiers should now check both signatures, similarly to how it was done to communication.

2. In the end, each receiver party R_i gets the shares $\mathbf{y}_{i1}, \ldots, \mathbf{y}_{in}$ from all the computing parties $M_1, \ldots, M_n$. Now each M_j has to generate $\binom{n-1}{t}$ sums $\sum_{k \in T_j} \mathbf{y}_{ijkT_\ell} = \mathbf{y}_{ij}$, exactly in the same way as it would do with an ordinary communication value. M_j sends the shares and their signatures to P_i, and P_i redistributes them amongst the t-sets. In the verification process, they check both signatures, similarly to how it was done to communication.

Since the parties P_i and R_i do not participate in the computation, they do not have to participate in the verification. However, they will still be punished if they provide multiple signatures for the same value.

5.4.2 Possible Issues

The main drawback of the previous proposition is the numerous amount of signatures that the computing parties may have to check. While in the initial scheme each party M_j has to provide just one share $\mathbf{x_{jkT}}_\ell$ for each party M_k in each T_ℓ, now it has to provide N shares, where N is the number of input parties, and all their signatures have to be checked (for M_j it is still sufficient to use just one signature, but it does not help much). Depending on the settings, N can be very large. In the worst case, each input party provides only one bit, and hence $N \in O(|C|)$. However, each computing party would have to verify the source of all the inputs anyway. For P_i, sending $t \cdot \binom{n-1}{t}$ shares instead of one is not worse since all the values used by the same P_i may have the same signature. The problem comes when M_j wants to redistribute the shares and the signatures to all T-sets, since each receiver will again have to check all N of them. Fortunately, this happens only in the beginning and in the end of the protocol.

Additionally, depending on the performed computation, the covert security may just not work with the input parties, especially in some anonymous statistical projects. Any participant may cheat without reason and complain afterwards. In our scheme, the verification of input share signatures is done already in the beginning, an hence the computing parties will not spend their time on clearly malicious parties whose shares do not correspond to their signatures. The problem still remains with the output, since the malicious output party R_i may sign wrong values just for fun, without fear of being detected. However, since such cheating would require just one additional broadcast (revealing the signatures to everyone), this is not too much different from the case if R_i has not complained. In any case, even if no one complains, it may still be some kind of attack where the input party is completely honest, but it just performs the computation without needing.

5.4.3 Deviations from the Initial Settings

In real Secure Multiparty Computation Platforms, it may happen that the number of input parties is initially unknown. For example, in the case of some statistical computation, the input parties may come and submit their inputs during the execution, and hence the shape of the computational circuit may be even unknown in the beginning, since the input length is undefined. Nevertheless, the proposed techniques still work. The coming input parties may commit the inputs as they come. In the end of the computation, the structure of the circuit will be known anyway.

5.5 Conclusions and Future Work

In this work we have proposed a scheme that allows to verify the computation of each party in a passively secure protocol, thus converting passive security to covert security. Each malicious party will be detected with probability close to 1, depending on the parameters of selected field.

While our verification is being done only after the entire computation has ended, it might be interesting to do something more similar to the active security model. Namely, we could require each party to prove the correctness after each round. If implemented straightforwardly, repeating our verification algorithm on each round, it multiplies the verification complexity by the number of rounds (actually, a bit less since in the beginning the vectors will be of smaller length). Doing it more cleverly, we could make use of the proofs of the previous rounds, making the next proof steps reliable on the proofs of the previous steps. The ideas can be taken for example from [15].

Chapter 6

Actively Secure Two-Party Computation with Precomputing

This chapter introduces two initialisations of actively secure two-party computation and new ideas for precoputation using additively homomorphic encryption. The main focus of secure computation in the precomputation model is usually on improving the efficiency of the online phase. However, the efficiency of the precomputation phase is also of importance. This work proposes ideas based on packing several values to one ciphertext to improve precomputation based on additive secret sharing. In addition, a symmetric and asymmetric setting for secret sharing are introduced where the symmetric gains most from the improved precomputation.

The aim of this work is to adapt the SPDZ general actively secure computation framework [24] for the two-party case and focus on optimising the precomputation phase. An important distinction between our work and SPDZ is that we use an additively homomorphic cryptosystem instead of the somewhat-homomorphic cryptosystem for the precomputation phase. We prefer an additively homomorphic cryptosystem because it is more conventional, easier to implement and currently more thoroughly studied. The resulting protocol set is implemented in Sharemind version 3 [47].

6.1 Related work

Secure computation is currently an active research field and has reached the state where is is efficient enough for practical applications. The work is mainly divided to three branches, one focusing on the development of garbled circuits, the second on secure multi-party computing on secret shared elements and the third on fully homomorphic encryption. This work is about the latter and considers secure computations on secret shared data. Sharemind is one of the more mature secure multi-party computation frameworks that currently offers *passive* security guarantees [12, 13]. This work uses the principles also combined in the SPDZ framework [24] to add an *actively* secure protocol set to the Sharemind framework.

Our protocols are divided between the *online* and *offline* world also known as the *precomputation* model. The precomputation model originates from Beaver [3] and has found wider usage in SMC after [22] as it has been used by [18, 23, 5, 41, 24]. Firstly, the precomputation phase is independent of the secret information and produces some random shares or sets of shares in a specific relation. Secondly, the online phase uses the secrets and the precomputation results to efficiently evaluate necessary functions.

The SPDZ framework utilises three important tools: oblivious message authentication codes (MAC) [45], Beaver triples [3], and vectorized homomorphic encryption [28, 48]. The first is used to ensure security against an active adversary and the second as a precomputation mechanism for multiplication. These two have been previously used together for SMC in BDOZ [5]. However, SPDZ adds an important idea that MAC is used to authenticate the shared secret as a whole and not for authenticating independent shares.

Vectorised somewhat-homomorphic encryption is used to generate Beaver triples in a communication-efficient way and is a SPDZ-specific property. Currently, SPDZ precomputes Beaver triples and single

random shares. The covertly secure extension of SPDZ [20] also precomputes squaring pairs analogously to Beaver triples and shared bits for comparison, bit-decomposition, fixed point and floating point operations. It is an open question if other operations can be efficiently precomputed.

Our work also uses oblivious MAC and Beaver triples, but differently from SPDZ we use additively homomorpic Paillier cryptosystem [42] instead of fully homomorphic encryption.

6.2 Secure two-party computation

We require three things for secure computation that is secure against an active adversary: definition of the share, protection mechanisms to ensure the correctness of the computation results, and computation protocols. We use additive secret sharing and homomorphic message authentication as our main protection mechanism. In addition, the secret sharing method and homomorphic MAC should have the same operations that can be computed locally.

6.2.1 Possible setups

Frameworks analogous to SPDZ can be used with two computing parties and could have three considerably different initialisations. The main difference between them is the setup and means of using the MAC algorithm. Two parties are denoted by $\mathcal{CP}_1$ and $\mathcal{CP}_2$.

Asymmetric setup

Asymmetric setup differentiates the computing parties so that one gets the role of a master node ($\mathcal{CP}_1$) who defines the MAC key and the client ($\mathcal{CP}_2$) is using the keys from the master. Using the MAC to either authenticate the secret value or the share of the other party enables $\mathcal{CP}_1$ to easily verify the correctness of the declassification result. However, $\mathcal{CP}_2$ is unable to verify the MAC as it must not know the MAC secret key. It is up to the master to also define something that $\mathcal{CP}_2$ can check. For example, we can use commitments.

The MAC tag for the whole value or the share of $\mathcal{CP}_2$ can not be kept by the master node. MAC algorithms are not designed to protect the privacy of the message. Thus, seeing the whole tag might leak the secret to the master node, who also knows the MAC secret key. In addition, storing the tag on the side of $\mathcal{CP}_2$ might also leak some information about the secret or the key. Hence, we need to store the tag t in a secret-shared manner and both parties must be able to update their parts of the tags during the computation.

For our initialisation of the asymmetric protocol set, we use a MAC algorithm defined as $t = k \cdot x \bmod N$ for tag t, key k, Paillier modulus N and secret value x. In addition, we use additively homomorphic perfectly binding commitments based on the Paillier' cryptosystem that the party $\mathcal{CP}_2$ can verify. This results in the fact that all our computations will be with respect to the Paillier modulus.

Each secret value x is represented by a tuple

$$[\![x]\!]_N = \langle \Delta, x_1, x_2, r, (\![x_1]\!)_{pk}, z_1, z_2 \rangle$$

such that $x = x_1 + x_2 + \Delta \bmod N$ and $z_1 + z_2 = k \cdot (x_1 + x_2) \bmod N$. The values Δ and $(\![x_1]\!)_{pk} = \mathsf{Enc}_{pk}(x_1, r)$ are public whereas $\mathcal{CP}_i$ has private values z_i and x_i. The public modifier Δ is always 0 for random values and is used to enable fast addition of a share and public constant. Value r is kept by $\mathcal{CP}_1$ to open the commitment to $(\![x_1]\!)_{pk}$ of share $[\![x]\!]_N$. This randomness also enables us to write protocols so that actually only $\mathcal{CP}_2$ computes $(\![x_1]\!)_{pk}$ and $\mathcal{CP}_1$ recomputes the encryption if needed. This is a reasonable step because, in reality, $\mathcal{CP}_1$ rarely needs this value.

Symmetric setup

A symmetric setup means that both computing parties define similar parameters. A direct continuation of the previous asymmetric setting would be that both parties $\mathcal{CP}_1$ and $\mathcal{CP}_2$ in the symmetric setting define

their own MAC keys k_i. This would mean that on top of the secret sharing method we have two MAC tags $z^{(1)}$, $z^{(2)}$ where both parties can verify one of them during the declassification phase. As in the asymmetric, case we need a to keep the tags in shares.

The main benefit of this setup over the asymmetric one is that the protocol descriptions would also become symmetric. This simplifies the notation and also means that the parties can do exactly the same workload in parallel. In some sense, this enables us to gain more efficient time usage. More precisely, it is unlikely that in such protocols one party has to wait between sending and receiving network message without having any computations to perform. Furthermore, we can only use the cheap MAC algorithm and do not need more expensive homomorphic commitments that we used in the asymmetric case.

For our initialisation, we use the same MAC as for the asymmetric case for both of the participants, but we use a different prime modulus p. We propose a share representation as

$$[\![x]\!]_p = \langle \Delta, x_1, x_2, z_1^{(1)}, z_2^{(1)}, z_1^{(2)}, z_2^{(2)} \rangle \ ,$$

where $x = x_1 + x_2 + \Delta \bmod p$ and Δ is the public modifier. The remaining values belong to the MAC tags as $z_1^{(i)} + z_2^{(i)} = k_i \cdot (x_1 + x_2) \bmod p$. Both parties know Δ and, in addition, $\mathcal{CP}_i$ has values x_i, $z_i^{(1)}$ and $z_i^{(2)}$.

Shared setup

The shared key model is a further extension changing the symmetric setup so that instead of both parties defining a key they share one key k between them. This defines a threshold MAC algorithm where all parties must participate in the verification of the tag. It can give additional efficiency gains as the parties only have to update a single tag t during the computations. However, the sharing of the key k is special as it has to define some additional information, allowing parties to verify the correctness of the restored key and checked tags. The shared key setup is the approach currently used by the SPDZ framework, however, we do not define our version of the shared setup.

There are well-known difficulties with shared approach as the knowledge of the secret key is usually needed to verify the MAC tags. One possible solution is to not verify any opened results before all computations are done. Afterwards, it is possible to restore the MAC key and verify all the results at once. However, in such case parties can only notice cheating very late and they must agree on a new key before next computations. In addition, changing the key means that after verifying the correctness of opened values, the shares of the outputs or intermediate results from the checked computations can not be reused.

6.2.2 Protocols

The description of some SPDZ protocols is independent from the secret sharing method as long as the scheme defines protocols for publishing shares privately to each computing or result party, generating a random share, and generating random Beaver triples. There are three main protocols: classifying the secret input, publishing the secret shared value, and multiplying two shared values. Classifying and multiplication protocols depend on the precomputation protocols. Publishing is actually dependant on the share representation, but the overall idea remains the same - open the value and verify that the protection mechanisms hold. We use these general protocols, but in addition have to define specific protocols for our share representation. In addition, we use a common protocol for verifying the correctness of Beaver triples from [23].

Our protocols include the addition protocol for the online phase and random share and triple generation protocols for the precomputation phase. The addition protocol is in a sense trivial, because we define our share representation so that the addition can be done locally. For precomputation we need protocols that generate either single random shares or random multiplicative triples. The main idea of the latter is that we at first generate two random shares and then use some multiplication functionality to obtain the third triple element. The main drawback is that we can not use the previously mentioned multiplication protocol because it requires triples as input. Therefore we need special protocols for *slow multiplication* that are discussed afterwards. All our protocols are fully specified in [43].

For practical computations, we would also need additional protocols. For example to compute division or exponentiation. However, addition and multiplication, together with the supporting protocols to work with secret sharing and precomputation, are sufficient for testing the feasibility of the proposed framework.

6.3 Beaver triple generation

In Beaver triple generation, we focus on the following question. Given two random elements a and b, we want to find their product. The main algorithm that we can use is a basic additive share multiplication using the Paillier cryptosystem. However, there is one problem with this algorithm, namely that it gives correct results for the Paillier modulus, but we might want to use different moduli for our triples. The other limitation is that if we are interested in very short a and b values compared to the Paillier modulus, then the encryption plaintext will have a lot of unused bits and hence, the computations are not very efficient. We try to overcome the first problem by introducing error correction and the second by packing several elements into one ciphertext.

6.3.1 Packing

B-ary packing

Packing as B-ary numbers means that each element modulo $M < B$ represents a digit and we pack them as numbers of base B. A three-digit base-B number could be written out as

$$x = B^2 \cdot x_3 + B \cdot x_2 + x_1 \ ,$$

where $x_i < B$ are digits. If we assume, that y is written out in a similar manner, then the corresponding multiplication becomes a degree 4 polynomial of B where we can learn x_3y_3 and x_1y_1, assuming that these do not overflow a B-ary digit.

Packing as straightforward B-ary numbers is, therefore, not very beneficial, as we did not receive a triple x_2, y_2, x_2y_2. However, we could consider another example, with x as before, but y is modified, giving $y = B^6y_3 + B^3 \cdot y_2 + y_1$. After multiplying it out, xy contains all the triples x_i, y_i and x_iy_i, but also some elements x_iy_j, $i \neq j$ that we do not need. Thus, in this packing we only get the same number of triples as the square-root of the number digits in the base-B representation of xy where the maximal size is limited by the plaintext size and, therefore, it is not very space-efficient.

Another problem with using this approach in a straightforward manner is that we have to assume that $y_ix_i < B$, which essentially means that the result xy contains integer results of all triples as digits. However, the common version of the Paillier multiplication would destroy this structure in the outcome. This packing can be used with a redefinition of the randomiser in the Paillier multiplication, but it will also result in decreased security. A version of partial B-ary packing was also introduced in [44], that also suffers from the reduced security level.

Packing using the Chinese Remainder Theorem

The Chinese remainder theorem (CRT) can also be used for packing several elements into one ciphertext. However, it can only be used, if we are using elements with pairwise coprime moduli p_i. By definition, CRT can be used to combine all those single random values x_i modulo p_i, for a modulus $M = p_1 \cdot \ldots \cdot p_k$ and execute the triple generation protocol to obtain the corresponding third triple element xy modulo M. We are interested in learning the shares for x_iy_i. The CRT allows us to reduce the final result xy respectively for all moduli p_i to learn the third triple element for all initial random value pairs. Therefore, we can learn x_iy_i from $xy \bmod M$ as $x_iy_i = xy \bmod p_i$. Packing with CRT enables us to get exactly $|M|$-bit triples from one execution of the triple generation protocol where the maximal size of M is bounded by the used Paillier modulus.

However, in most use-cases, we would like to always use the same modulus p, not different p_i. Actually, we could use a modulus $p < p_i$ and convert the final results for different moduli p_i to one modulus p using the ideas from the following error correction section.

6.3.2 Error correction

Error correction is the most important part of the share conversion step. Here, share conversion means that we have a secret value x shared using a modulus M_1 and we need to obtain a sharing of x for a modulus M_2. In addition, we are only interested in cases where $M_2 \leq M_1$ and M_1 is odd, because these are what we need for using Paillier multiplication or packing with the Chinese remainder theorem. Especially, we have some $x = a \cdot b \bmod M_1$ and we need to obtain $x = a \cdot b \bmod M_2$. By picking a and b, we can actually ensure that $a \cdot b \leq M_1$, which means that the value x can be reduced as usual.

The second problem is that we are working with secret sharing either modulo M_1 or M_1. Initially, we have shared value $x = x_1 + x_2 \bmod M_1$ with shares x_1 and x_2 and we actually need to learn shares x_1^* and x_2^* such that $x = x_1^* + x_2^* \bmod M_1$.

There are two possibilities, $x_1 + x_2 < M_1$ or $2M_1 > x_1 + x_2 \geq M_1$. If we are in the first case, then $x_i^* = x_i \bmod M_2$. However, for the second case $x_1^* = x_1 \bmod M_2$, but $x_2^* = x_2 - M_1 \bmod M_2$. These cases can be distinguished easily using only the least significant bits of x, x_1 and x_2, because if $x \bmod 2 = x_1 + x_2 \bmod 2$ then we are in the first case and otherwise in the second. Moreover, this condition can be securely checked using oblivious transfer, if we know what the last bit of x should be. We can use this, for example, to instead check the value $2x$, where we know that the least significant bit is 0.

The main drawback of using $2x$ as a check value is that in such a case, we can not use modulus 2 as any of our target moduli. However, it is usable for most use-cases, especially for transforming the Paillier modulus to a prime modulus needed for the symmetric and shared versions of secure computation.

Finally, it would also be possible that instead of doing error correction we use values where the probability of an error is small enough and use the triple verification procedure to check that the triple was indeed valid.

6.4 Conclusion

Current results show that actively secure multi-party computation is significantly slower than passively secure versions. However, our results indicate that fully implemented symmetric protocol set could be close to the performance of the SPDZ framework that is the current leader in actively secure multi-party computation frameworks. In addition, achieving security against malicious adversaries can be very important for data mining tasks that have important economical or societal outcomes. Therefore, in many cases the extra time consumption is a reasonable trade-off for the additional layer of security.

In conclusion, the symmetric and possibly shared setups are the most reasonable setups for secure two-party computation. Follow up work should focus more on specifying the missing details of the symmetric setup and on achieving the setup phase of the shared setup.

Chapter 7

Comparison of oblivious sorting algorithms

7.1 Introduction

Sorting is an important operation in privacy-preserving data analysis and data mining. In addition to its obvious use in ordering data, sorting is used for finding ranked elements (top-k, quantiles), performing group-level aggregations and implementing statistical tests.

In UaESMC deliverable D2.2.1 [8] we showed how sorting networks can be evaluated in SMC. In this chapter, we concentrate on constructions based on oblivious shuffling and introduce further optimizations for sorting networks.

7.2 Oblivious sorting techniques

7.2.1 Constructions based on comparisons

Comparison-based sorting algorithms use the comparison operation to determine the correct sequence of elements of a given array. Such algorithms are inherently data-dependent, as the execution flow depends on the outcomes of the comparison operations. Hence, changes in the input data will affect the running time of the algorithm. A simple solution would be to evaluate all the branches of the sorting algorithm and obliviously select the correct output in the end, but this dramatically reduces efficiency.

Hamada *et al.* [29] propose a generic solution to *obliviously shuffle* [37] the inputs before performing a comparison-based sort. Then, as we are comparing values in a randomly shuffled vector, any declassified (published) comparison results are also random. However, the pattern of comparisons in the algorithm can still leak information, such as the number of equal elements in the input vector.

Because many SMC implementations have highly efficient vector operations, vectorized naive protocols may sometimes be more efficient than protocols with a lower computational complexity and a lower degree of vectorization. In this paper, we propose a naive sorting protocol based on shuffle and vectorized comparisons called NaiveCompSort (Algorithm 10). In this algorithm, we first shuffle the input array and then compare every element with every other element in the array in one big vector operation. Finally, we rearrange the elements according to the declassified comparison results. This algorithm always works in the worst case time of $\mathcal{O}(n^2)$ and its runtime is, therefore, data-independent.

7.2.2 Constructions specific for bitwise secret-sharing schemes

If data is secret-shared using a bitwise secret-sharing scheme, access to individual bits is cheap. This allows us to design a very efficient count/radix sorting algorithm. Counting sort [16, 25] is a sorting algorithm that can sort an array of integers in a small range by first constructing a frequency table and then rearranging items in the array according to this table. Algorithm 11 describes a counting sort algorithm for binary data.

Algorithm 10: NaiveCompSort

Data: Input array $[\![\mathcal{D}]\!] \in \mathbb{Z}_{2^k}^n$
Result: Sorted array $[\![\mathcal{D}']\!]$
Let $[\![T]\!] = \mathsf{Shuffle}([\![T]\!])$
`// All comparisons here are done in parallel.`
for $i < j \in \{1, 2, \ldots, n\}$ **do**
  Let $[\![g_{i,j}]\!] = [\![\mathcal{D}_i]\!] \leq [\![\mathcal{D}_j]\!]$
end
Declassify the values $[\![g_{i,j}]\!]$ and sort $[\![\mathcal{D}]\!]$ according to them, obtaining $[\![\mathcal{D}']\!]$
return $[\![\mathcal{D}']\!]$

Algorithm 11: Counting sort algorithm for binary arrays.

Data: Binary input array $\mathcal{D} \in \mathbb{Z}_2^n$.
Result: Array $\mathcal{D}' \in \mathbb{Z}_2^n$ with elements of $\mathcal{D}$ in increasing order.
$n_0 \leftarrow n - sum(\mathcal{D})$; `// Count number of zeros.`
$c_0 \leftarrow 0$; $c_1 \leftarrow 0$; `// Keep counters for processed zeros and ones.`
`// Put each element in right position:`
foreach $i \in 1 \ldots n$ **do**
  if $\mathcal{D}_i == 0$ **then**
    $c_0 = c_0 + 1$
    $\mathcal{D}'_{c_0} = \mathcal{D}_i$
  else
    $c_1 = c_1 + 1$
    $\mathcal{D}'_{n_0+c_1} = \mathcal{D}_i$
  end
end
return $\mathcal{D}'$

Radix sort [31] sorts an array of integers by rearranging them based on counting sort results on digits in the same positions. Radix sort sorts data one digit position at a time, starting with the least significant digit. This works as the underlying counting sort is a stable sorting algorithm. Algorithm 12 shows the full protocol of oblivious radix sort that uses binary counting sort as a subroutine. The underlying counting sort is made data-independent by obliviously updating counters $[\![c_0]\!]$, $[\![c_1]\!]$ and the order vector $[\![ord]\!]$. Such a data-independent counting sort is sufficient to make our radix sort data-independent as well.

As our radix sort algorithm does not use comparison operations, it is not bound by the computational complexity lower bound of $\Omega(n \log n)$ for comparison-based sorting algorithms. Counting sort has a complexity of $\mathcal{O}(n)$ and radix sort on k-digit elements that uses counting sort as a subroutine, has a computational complexity of $\mathcal{O}(kn)$. However, the data-independent counting sort protocol also uses addition and multiplication operations which are expensive protocols on bitwise shared data. Therefore, after creating a vector with bits on a given position, we convert it to additively shared data and work in this domain. The output of the algorithm is still in a bitwise form.

7.3 Optimization methods and matrix sorting

7.3.1 Vectorization

In their work, Hamada *et al.* parallelize the invocations of comparisons on secret-shared data as network communication is the main bottleneck for SMC protocols. Such SIMD (single instruction, multiple data) operations are very efficient as they allow to put messages of many parallel operations into a single network

Algorithm 12: Data-independent radix sort.

Data: Input array $[\![\mathcal{D}]\!] \in \mathbb{Z}_{2^k}^n$.
Result: Sorted array $[\![\mathcal{D}]\!] \in \mathbb{Z}_{2^k}^n$.

```
// Iterate over all digits starting with the least significant digit:
```
foreach $m \in 1 \dots k$ **do**
  ```
  // Construct a binary vector consisting of m-th digits.
  // Convert it to additively shared data.
  ```
  $[\![d]\!] \leftarrow \mathsf{ShareConv}(([\![\mathcal{D}_1]\!]_m, [\![\mathcal{D}_2]\!]_m, \dots, [\![\mathcal{D}_n]\!]_m))$
  $[\![n_0]\!] \leftarrow n - sum([\![d]\!])$; `// Count number of zeros.`
  $[\![c_0]\!] \leftarrow 0$; $[\![c_1]\!] \leftarrow 0$; `// Keep counters for processed zeros and ones.`
  $[\![ord]\!]$; `// Keep` n`-element shared order vector.`
  ```
  // Put each element in the right position:
  ```
  foreach $i \in 1 \dots n$ **do**
    $[\![c_0]\!] = [\![c_0]\!] + 1 - [\![d_i]\!]$
    $[\![c_1]\!] = [\![c_1]\!] + [\![d_i]\!]$
    ```
    // Obliviously update order vector:
    ```
    $[\![ord_i]\!] = (1 - [\![d_i]\!]) * [\![c_0]\!] + [\![d_i]\!] * ([\![n_0]\!] + [\![c_1]\!])$
  end
  $([\![\mathcal{D}]\!], [\![ord]\!]) \leftarrow \mathsf{Shuffle}([\![\mathcal{D}]\!], [\![ord]\!])$; `// Shuffle two column database.`
  $ord \leftarrow \mathsf{Declassify}([\![ord]\!])$
  Rearrange elements in $[\![\mathcal{D}]\!]$ according to ord.
end
return $[\![\mathcal{D}]\!]$

message, saving on networking overhead. They did not only parallelize comparison invocations in a single partitioning subroutine, but rather all the comparisons at each depth of the quicksort algorithm. For this work we implemented the oblivious quicksort algorithm proposed in [29] as a reference, using the same idea for parallelization.

Similarly, we vectorize all secure operations in our counting sort algorithm design. As sorting by a given digit position is dependent on the previous position outcome, radix sort cannot be vectorized further. We could apply counting sort on chunks of 2 or more bits and reduce the number of rounds for radix sort. However, this requires substituting the cheap oblivious choice subprotocol for a more expensive comparison protocol.

7.3.2 Changing the share representation

Both comparison-based sorting algorithms and sorting networks rely on the comparison operation. Comparison is a bit-level operation and works faster on bitwise shared data. Therefore, we can convert additively shared inputs into bitwise shared form and run the intended algorithm on the converted shares. The results can be converted back to additively shared form at the end of the algorithm.

Converting additive shares to bitwise shares requires a bit extraction protocol. However, for algorithms that perform many comparisons after one another, the benefits of many fast comparisons outweigh one costly conversion.

7.3.3 Optimizations specific to sorting networks

In software implementations, the generation of sorting networks can take a significant amount of time. As the sorting network structure is data-independent, we can store the sorting network after generation to re-use it later.

If we shuffle the inputs before sorting, we can optimize the CompEx function implementations by declassifying comparison results and performing the exchanges non-obliviously. The running time of the resulting algorithm is data-independent because of the constant structure of the sorting network.

7.3.4 Sorting matrices

Sorting secret shared matrices using sorting networks is covered in UaESMC deliverable D2.2.1 [8].

Similarly, shuffle-based algorithms can be easily modified to support matrix sorting. Assume that our input data is in the form of a matrix $\mathcal{D}_{i,j}$ where $i = 1 \ldots n$ and $j = 1 \ldots m$. Let us also fix a column k by which we want to sort the rows.

First, we obliviously shuffle the rows in the whole matrix. Note that shuffling is already a part of comparison-based sorting protocols like quicksort and NaiveCompSort. However, this extra step has to be added for radix sort. Next, we extract the k-th column from the matrix and pass it to the sorting algorithm of our choice together with an n-element index vector $(1, 2, \ldots, n)$.

The sorting protocol now swaps elements in the data vector and the index vector together. After sorting these two vectors, we declassify the output index vector and use it as a permutation to rearrange rows in the matrix. Declassifying the index vector leaks information on how the elements were rearranged. However, as the input matrix was obliviously shuffled, this leaks no information on the original placement of rows in the initial matrix.

7.4 Conclusion

We describe three designs for oblivious versions of known sorting algorithms—naive comparison-based sort, quicksort and radix sort. As opposed to sorting networks, all of these perform some declassifications to improve efficiency. Our novel oblivious radix sorting algorithm leaks less information than constructions based on shuffling and declassified comparison results.

In addition, we recommend to use precomputing or caching of network structure for oblivious sorting networks. In that case, the algorithm provides perfect privacy with a reasonable performance.

Bibliography

[1] Gilad Asharov and Claudio Orlandi. Calling out cheaters: Covert security with public verifiability. In Xiaoyun Wang and Kazue Sako, editors, *ASIACRYPT*, volume 7658 of *Lecture Notes in Computer Science*, pages 681–698. Springer, 2012.

[2] Yonatan Aumann and Yehuda Lindell. Security against covert adversaries: Efficient protocols for realistic adversaries. *J. Cryptology*, 23(2):281–343, 2010.

[3] Donald Beaver. Efficient multiparty protocols using circuit randomization. In Joan Feigenbaum, editor, *CRYPTO*, volume 576 of *Lecture Notes in Computer Science*, pages 420–432. Springer, 1991.

[4] Eli Ben-Sasson, Alessandro Chiesa, Daniel Genkin, Eran Tromer, and Madars Virza. Snarks for c: Verifying program executions succinctly and in zero knowledge. In *CRYPTO (2)*, pages 90–108, 2013.

[5] Rikke Bendlin, Ivan Damgård, Claudio Orlandi, and Sarah Zakarias. Semi-homomorphic encryption and multiparty computation. In Kenneth G. Paterson, editor, *EUROCRYPT*, volume 6632 of *Lecture Notes in Computer Science*, pages 169–188. Springer, 2011.

[6] Nir Bitansky, Alessandro Chiesa, Yuval Ishai, Rafail Ostrovsky, and Omer Paneth. Succinct non-interactive arguments via linear interactive proofs. In *TCC*, pages 315–333, 2013.

[7] 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.

[8] Dan Bogdanov, Roberto Guanciale, Liina Kamm, Peeter Laud, Riivo Talviste, and Jan Willemson. Advances in SMC techniques, January 2013. UaESMC Deliverable 2.2.1.

[9] Dan Bogdanov, Liina Kamm, Peeter Laud, Alisa Pankova, Pille Pullonen, Riivo Talviste, and Jan Willemson. Advances in SMC techniques, January 2014. UaESMC Deliverable 2.2.2.

[10] Dan Bogdanov, Liina Kamm, Sven Laur, and Pille Pruulmann-Vengerfeldt. Secure multi-party data analysis: end user validation and practical experiments. Cryptology ePrint Archive, Report 2013/826, 2013. `http://eprint.iacr.org/`.

[11] Dan Bogdanov, Sven Laur, and Riivo Talviste. A practical analysis of oblivious sorting algorithms for secure multi-party computation. Submitted to ASIACCS 2014, 2014.

[12] 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.

[13] 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.

[14] Ran Canetti. Universally composable security: A new paradigm for cryptographic protocols. In *FOCS*, pages 136–145. IEEE Computer Society, 2001.

[15] Alessandro Chiesa and Eran Tromer. Proof-carrying data and hearsay arguments from signature cards. In Andrew Chi-Chih Yao, editor, *ICS*, pages 310–331. Tsinghua University Press, 2010.

[16] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to Algorithms*, chapter 8.2 Counting Sort, pages 168–170. MIT Press and McGraw-Hill, 2nd edition, 2001.

[17] Ivan Damgård, Matthias Fitzi, Eike Kiltz, Jesper Buus Nielsen, and Tomas Toft. Unconditionally secure constant-rounds multi-party computation for equality, comparison, bits and exponentiation. In Shai Halevi and Tal Rabin, editors, *TCC*, volume 3876 of *Lecture Notes in Computer Science*, pages 285–304. Springer, 2006.

[18] Ivan Damgård, Martin Geisler, Mikkel Krøigaard, and Jesper Buus Nielsen. Asynchronous Multiparty Computation: Theory and Implementation. In Stanislaw Jarecki and Gene Tsudik, editors, *Public Key Cryptography*, volume 5443 of *Lecture Notes in Computer Science*, pages 160–179. Springer, 2009.

[19] Ivan Damgård, Martin Geisler, and Jesper Buus Nielsen. From passive to covert security at low cost. In Daniele Micciancio, editor, *TCC*, volume 5978 of *Lecture Notes in Computer Science*, pages 128–145. Springer, 2010.

[20] Ivan Damgård, Marcel Keller, Enrique Larraia, Valerio Pastro, Peter Scholl, and Nigel P. Smart. Practical covertly secure mpc for dishonest majority - or: Breaking the spdz limits. In Jason Crampton, Sushil Jajodia, and Keith Mayes, editors, *ESORICS*, volume 8134 of *Lecture Notes in Computer Science*, pages 1–18. Springer, 2013.

[21] Ivan Damgård, Sigurd Meldgaard, and Jesper Buus Nielsen. Perfectly secure oblivious ram without random oracles. In Yuval Ishai, editor, *TCC*, volume 6597 of *Lecture Notes in Computer Science*, pages 144–163. Springer, 2011.

[22] Ivan Damgård and Jesper Buus Nielsen. Scalable and unconditionally secure multiparty computation. In Alfred Menezes, editor, *CRYPTO*, volume 4622 of *Lecture Notes in Computer Science*, pages 572–590. Springer, 2007.

[23] Ivan Damgård and Claudio Orlandi. Multiparty computation for dishonest majority: From passive to active security at low cost. In Tal Rabin, editor, *CRYPTO*, volume 6223 of *Lecture Notes in Computer Science*, pages 558–576. Springer, 2010.

[24] Ivan Damgård, Valerio Pastro, Nigel P. Smart, and Sarah Zakarias. Multiparty computation from somewhat homomorphic encryption. In Reihaneh Safavi-Naini and Ran Canetti, editors, *CRYPTO*, volume 7417 of *Lecture Notes in Computer Science*, pages 643–662. Springer, 2012.

[25] Jeff Edmonds. *How to Think about Algorithms*, chapter 5.2 Counting Sort (a Stable Sort), pages 72–75. Cambridge University Press, 2008.

[26] Rosario Gennaro, Craig Gentry, Bryan Parno, and Mariana Raykova. Quadratic span programs and succinct nizks without pcps. In Thomas Johansson and Phong Q. Nguyen, editors, *EUROCRYPT*, volume 7881 of *Lecture Notes in Computer Science*, pages 626–645. Springer, 2013.

[27] Rosario Gennaro, Michael O. Rabin, and Tal Rabin. Simplified vss and fact-track multiparty computations with applications to threshold cryptography. In *PODC*, pages 101–111, 1998.

[28] Craig Gentry. Fully homomorphic encryption using ideal lattices. In Michael Mitzenmacher, editor, *STOC*, pages 169–178. ACM, 2009.

[29] Koki Hamada, Ryo Kikuchi, Dai Ikarashi, Koji Chida, and Katsumi Takahashi. Practically Efficient Multi-party Sorting Protocols from Comparison Sort Algorithms. In *Proc. of ICISC'12*, volume 7839 of *LNCS*, pages 202–216. Springer, 2013.

[30] Myles Hollander and Douglas A Wolfe. *Nonparametric statistical methods*. John Wiley New York, 2nd ed. edition, 1999.

[31] Herman Hollerith. US395781 (A) - ART OF COMPILING STATISTICS. European Patent Office, 1889. `http://worldwide.espacenet.com/publicationDetails/biblio?CC=US&NR=395781`.

[32] Rob J Hyndman and Yanan Fan. Sample quantiles in statistical packages. *The American Statistician*, 50(4):361–365, 1996.

[33] Peeter Laud and Alisa Pankova. New Attacks against Transformation-Based Privacy-Preserving Linear Programming. In Rafael Accorsi and Silvio Ranise, editors, *Security and Trust Management (STM) 2013, 9th International Workshop*, volume 8203 of *Lecture Notes in Computer Science*, pages 17–32. Springer, 2013.

[34] Peeter Laud and Alisa Pankova. On the (Im)possibility of Privately Outsourcing Linear Programming. In Ari Juels and Bryan Parno, editors, *Proceedings of the 2013 ACM Workshop on Cloud computing security, CCSW 2013*, pages 55–64. ACM, 2013.

[35] Peeter Laud and Alisa Pankova. Verifiable computation in multiparty protocols with honest majority. Cryptology ePrint Archive, Report 2014/060, 2014. `http://eprint.iacr.org/`.

[36] Peeter Laud and Jan Willemson. Universally composable privacy preserving finite automata execution with low online and offline complexity. Cryptology ePrint Archive, Report 2013/678, 2013. `http://eprint.iacr.org/`.

[37] 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.

[38] Yehuda Lindell. Fast cut-and-choose based protocols for malicious and covert adversaries. In Ran Canetti and Juan A. Garay, editors, *CRYPTO (2)*, volume 8043 of *Lecture Notes in Computer Science*, pages 1–17. Springer, 2013.

[39] Helger Lipmaa. Succinct non-interactive zero knowledge arguments from span programs and linear error-correcting codes. *IACR Cryptology ePrint Archive*, 2013:121, 2013.

[40] Helger Lipmaa and Tomas Toft. Secure equality and greater-than tests with sublinear online complexity. In Fedor V. Fomin, Rusins Freivalds, Marta Z. Kwiatkowska, and David Peleg, editors, *ICALP (2)*, volume 7966 of *Lecture Notes in Computer Science*, pages 645–656. Springer, 2013.

[41] Jesper Buus Nielsen, Peter Sebastian Nordholt, Claudio Orlandi, and Sai Sheshank Burra. A new approach to practical active-secure two-party computation. *CoRR*, abs/1202.3052, 2012.

[42] Pascal Paillier. Public-key cryptosystems based on composite degree residuosity classes. In *EUROCRYPT*, pages 223–238, 1999.

[43] Pille Pullonen. Actively secure two-party computation: Efficient Beaver triple generation. Master's thesis, University of Tartu, Aalto University, 2013.

[44] Pille Pullonen, Dan Bogdanov, and Thomas Schneider. The design and implementation of a two-party protocol suite for Sharemind 3. Technical report, Cybernetica AS Infoturbeinstituut, 2012. `http://research.cyber.ee`.

[45] Tal Rabin and Michael Ben-Or. Verifiable secret sharing and multiparty protocols with honest majority (extended abstract). In David S. Johnson, editor, *STOC*, pages 73–85. ACM, 1989.

[46] Adi Shamir. How to share a secret. *Commun. ACM*, 22(11):612–613, 1979.

[47] Sharemind. `http://sharemind.cyber.ee`. Last accessed 2013-10-10.

[48] Nigel P. Smart and Frederik Vercauteren. Fully homomorphic SIMD operations. *IACR Cryptology ePrint Archive*, 2011:133, 2011.

[49] Frank Wilcoxon. Individual Comparisons by Ranking Methods. *Biometrics Bulletin*, 1(6):80–83, 1945.

Appendix A

Secure multi-party data analysis: end user validation and practical experiments

The paper "Secure multi-party data analysis: end user validation and practical experiments" [10] follows.

Secure multi-party data analysis: end user validation and practical experiments

Dan Bogdanov[1], Liina Kamm[1,2], Sven Laur[2], Pille Pruulmann-Vengerfeldt[3]

[1] Cybernetica, Mäealuse 2/1, 12618 Tallinn, Estonia
{dan,liina}@cyber.ee
[2] University of Tartu, Institute of Computer Science, Liivi 2, 50409 Tartu, Estonia
swen@ut.ee
[3] University of Tartu, Institute of Journalism, Communication and Information Studies, Lossi 36, 51003 Tartu, Estonia
pille.vengerfeldt@ut.ee

Abstract. Research papers on new secure multi-party computation protocols rarely confirm the need for the developed protocol with its end users. One challenge in the way of such validation is that it is hard to explain the benefits of secure multi-party computation to non-experts. We present a method that we used to explain the application models of secure multi-party computation to a diverse group of end users in several professional areas. In these interviews, we learned that the potential users were curious about the possibility of using secure multi-party computation to share and statistically analyse private data. However, they also had concerns on how the new technology will change the data analysis processes. Inspired by this, we implemented a secure multi-party computation prototype that calculates statistical functions in the same way as popular data analysis packages like R, SAS, SPSS and Stata. Finally, we validated the practical feasibility of this application by conducting an experimental study that combined tax records with education records.

1 Introduction

Secure multi-party computation (SMC) has been researched and developed for several decades. For years, SMC was rightfully considered too inefficient for practical use. However, in recent years, several fast implementations have been developed [2, 8]. Still, this powerful secure data manipulation tool has not become as popular in practice as one would hope. People have managed without such a technology for a long time and have replaced it with social solutions like non-disclosure agreements and hoped that their shared data is kept safe by their partners. Alternatively, they have legally been forbidden to do shared analysis.

Furthermore, SMC is not effective in every setting and knowledge about its capabilities is still relatively uncommon. The goal of our research is to develop usable and efficient SMC applications that meet the needs of the potential end users and, through communicating those solutions, raise general awareness of SMC in order to support sharing data without the fear of abuse.

At first, we directed our attention to the potential end users of SMC. We interviewed several stakeholders from a variety of fields to find out whether data holders see a need for this technology. As previous research has indicated, a serious obstacle in user-driven innovation and involving users in the early stages of development work is the problem of explaining such a complex technology to the end-user who is rarely an expert [15, 23].

In order to overcome the communication challenge, we decided to describe SMC visually. Our aim was to make it understandable and accurate, without focusing on the mathematics behind SMC. Hence, in the models we designed to assist the interview process, SMC is essentially a black box and different stakeholders are shown to communicate with this box. Each of these stakeholders has a set of roles that determines what that party is doing in the model.

We prepared 12 visual deployment models of SMC applications and used them to interview 25 people from across different fields. We asked them whether they can see a need for this technology in their field and what kind of social or cultural obstacles they see in implementing such technology. We also asked them to propose other fields that, in their opinion, could benefit from SMC.

The two most mentioned usage areas were statistical analysis and optimisation of supply and demand. We reviewed existing literature and saw that research on cryptographically secure statistical analysis has largely been focused on protocols for a particular function. We decided to find our whether general SMC can be efficient enough to support large-scale statistical data analysis.

Related work. To our knowledge, this is the first time that a study of this size has been conducted to determine the real-world need for SMC. However, there have been several efforts for implementing statistical functions.

Cryptographic primitives for evaluating statistical functions like mean, variance, frequency analysis and regression were proposed in [7, 9]. Early implementations of filtered sums and scalar products are described in [26]. Solutions based on secret sharing include a protocol for mean value proposed in [20, 19].

In 2004, Feigenbaum *et al.* proposed to use SMC for analysing faculty incomes in the annual Taulbee Survey [11]. The protocols designed for this study can be found in [1]. In 2011, Bogdanov *et al.* deployed SMC for financial data analysis for the Estonian Association of Information Technology and Telecommunications [3]. Kamm *et al.* have shown how to conduct secure genome-wide association studies using secure multi-party computation [17].

Our contribution. We present a novel way for introducing SMC to non-cryptographers and report the results of interviews we conducted with potential end users of the technology. We analyse the responses of our interviewees and identify their main expectations toward SMC.

As several interviewees reported a need for secure data analysis, we focus our efforts on implementing statistical functions using SMC. We show how to perform standard statistical procedures with SMC while preserving privacy and without simplifying the algorithms.

We describe the secure computation of statistical measures (mean, variance, standard deviation), frequency tables and quantiles. We show how to clean the data and apply custom filters. We give algorithms for privacy-preserving hypothesis testing using standard and paired t-tests, χ^2-tests and Wilcoxon tests.

We implement all algorithms on the Sharemind SMC platform. We use these implementations to conduct a complete privacy-preserving study featuring statistical measure computation, filtering, database transformation, linking and statistical tests. Performance results are provided for all implemented operations.

2 End user validation methodology

2.1 Modelling SMC deployments

We define three fundamental roles in an SMC system—the input party $\mathcal{I}$, the computation party $\mathcal{C}$ and the result party $\mathcal{R}$. Input parties collect and send data to the SMC system. The SMC system itself is hosted by computation parties who carry out the SMC protocols on the inputs and send results to result parties in response of queries.

We use the following notation for modelling SMC applications. Let $\mathcal{I}^k = (\mathcal{I}_1, \ldots, \mathcal{I}_k)$ be the list of input parties, $\mathcal{C}^m = (\mathcal{C}_1, \ldots, \mathcal{C}_m)$ be the list of computing parties and $\mathcal{R}^n = (\mathcal{R}_1, \ldots, \mathcal{R}_n)$ be the list of result parties. Let Π be an SMC protocol for performing a specific task.

In the following, $\mathcal{ICR}$ refers to a party that fills all three roles, similarly, $\mathcal{IC}$ refers to a party with roles $\mathcal{I}$ and $\mathcal{C}$. We use superscripts $(k, m, n \geq 1)$ to denote that there are several parties with the same role combination in the system.

Real world parties can have more than one of these roles assigned to them. The set $\{\mathcal{I}, \mathcal{C}, \mathcal{R}\}$ has 7 non-empty subsets and there are 2^7 possibilities to combine them. However, we want to look only at cases where all three roles are present. This leaves us with $128 - 16 = 112$ possible combinations. Not all of these make sense in a real-world setting, but we claim that all deployments of SMC can be expressed using these 112 combinations.

2.2 Visualisation of SMC deployment models

As our aim was to find out what stakeholders expect from SMC, we discussed SMC with people from different areas and asked them if they had had problems with sharing data in their field. We assumed that the interviewees did not have a background in computer science so approaching them with the usual SMC descriptions was out of the question.

We planned to visualise typical SMC applications to make the idea understandable. Fortunately, our role-based model translates easily into illustrative diagrams. See Table 1 for examples of deployment models inspired by published research on SMC applications.

We prepared for the interviews by designing 12 deployment models, some of which were based on existing SMC applications and some were imaginary. We

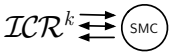
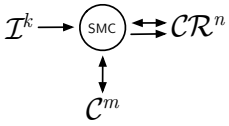
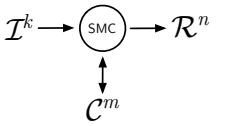
Basic deployment model	Example applications
	**The classic millionaires' problem [28]** *Parties:* Two—Alice and Bob (both $\mathcal{ICR}$) *Overview:* Millionaires Alice and Bob use SMC to determine who is richer. **Joint genome studies [17]** *Parties:* Any number of biobanks (all $\mathcal{ICR}$) *Overview:* The biobanks use SMC to create a joint genome database and study a larger population.
	**Studies on linked databases (this paper)** *Parties:* Ministry of Education, Tax Board, Population Register (all $\mathcal{IC}$) and Statistics Bureau ($\mathcal{R}$). *Overview:* Databases from several government agencies are linked to perform statistical analyses and tests.
	**Outsourcing computation to the cloud [12]** *Parties:* Cloud customer ($\mathcal{IR}$) and cloud service providers (all $\mathcal{C}$). *Overview:* The customer deploys SMC on one or more cloud servers to process her/his data.
	**Collaborative network anomaly detection [6]** *Parties:* Network administrators (all $\mathcal{IR}$) a subset of whom is running computing servers (all $\mathcal{ICR}$). *Overview:* A group of network administrators uses SMC to find anomalies in their traffic.
	**The sugar beet auction [4]** *Parties:* Sugar beet growers (all $\mathcal{I}$), Danisco and DKS (both $\mathcal{CR}$) and the SIMAP project ($\mathcal{C}$). *Overview:* The association of sugar beet growers and their main customer use SMC to agree on a price for buying contracts.
	**The Taulbee survey [11]** *Parties:* Universities in CRA (all $\mathcal{I}$), universities with computing servers (all $\mathcal{IC}$) and the CRA ($\mathcal{R}$). *Overview:* The CRA uses SMC to compute a report of faculty salaries among CRA members. **Financial reporting in a consortium [3]** *Parties:* Members of the ITL (all $\mathcal{I}$), Cybernetica, Microlink and Zone Media (all $\mathcal{IC}$) and the ITL board ($\mathcal{R}$). *Overview:* The ITL consortium uses SMC to compute a financial health report of its members.

Table 1: SMC deployment models and example applications

designed large colourful and easily readable figures to help us describe SMC to stakeholders during the interviews. On these figures we did not use the $\mathcal{ICR}$ syntax, but rather real-world roles that the interviewee could relate to. The description of each model included the security and trust guarantees that SMC provides for the parties. We could not include the figures here due to size constraints, but they can be found in [25].

2.3 Interview process and results

Our sample of 25 people was designed with the aim to get as much diversity as possible. The interviewees were always given a possibility to propose additional fields outside of their own where this kind of technology could be beneficial. Not all of our interviewees could be considered potential users, some could rather be described as stakeholders with knowledge of a potential social barrier. For instance, among others, we interviewed a lawyer and an ethics specialist in order to understand the larger societal implications. The interviewees originated from six different countries, they came from academia, from both public and private sector organizations, from small and medium sized enterprises to large multinational corporations, from local government to state level. The people we interviewed included representatives from the financial sector, agriculture, retail, security, mobile technologies, statistics companies and IT in general.

We sent the materials to the interviewees beforehand to let them prepare for the interview. We also used the figures during the interview process to trigger conversation and to assist in understanding the principles of the technology. During the interviews, we asked whether our interviewees recognised situations in their field of expertise where they need to share protected data with others.

Of all the possible cases brought out in the deployment models, the cases concerning the use of databases from different data sources for performing statistical analysis were most discussed. It seems that the benefits of merging different databases for statistical analysis were easily comprehensible for the interviewees with different professional backgrounds. On the one hand, the interviewees had many concerns, such as SMC conflicting with the traditional ways of doing things and problems related to the existing legal and regulatory framework. At times, the interviewees could not distinguish between anonymisation and SMC, or understand the operational challenges of using this kind of solution in practice. On the other hand, the interviewees also saw many potential benefits of the possible applications of SMC. They brought out examples how SMC could be advantageous in their professional field: for example, an expert working in the dairy industry said that there is a need to find a way to efficiently collect and analyse sensitive data concerning the activities of dairy companies as the studies form government research units do not fulfil the needs of the industry. Another example comes from biomedicine:

"For example, if I as a researcher get the data about the number of abortions but I also want to know how much all kind of associated complications cost, I need to get data from the national Health Insurance Fund. But I only get data from the Health Insurance Fund if I have the data from the abortion registry

with names and national identification numbers and then I ask the medical cost records of those people. What I think is actually a really big security risk. If it would be possible to link them differently, so I would receive impersonalised data, that would be really good." (I11, Academic sector, Biomedicine)

Several interviewees also pointed out how SMC could be used on a more general level. The idea of using different state databases for statistical analysis was seen as highly beneficial. For instance, an official working in a state institution that coordinates the work of the national information system stated that making more data and information available for public use is a relevant problem.

"After the presentation I thought that the state data should be made available for people this way: for researches, statisticians, universities. Publishing this data has always been a topic in the state, all the data has to be public, we should put it on the cloud or somewhere else. But do it in a secure way, I haven't thought about it before, but it seemed to me that there were no good solutions." (I8, Public sector, IT security)

Interestingly, interviewees whose work involves data processing remained somewhat critical, mostly because of the practical issues. Although an interviewee working in biomedicine saw the benefits of using different databases in scientific research, he also foresaw possible issues that could hinder their work. The main concern could be expressed as the necessity to "see" the data.

"But in the context of genetics, the researcher who does the calculations, he has to see the data. He has to understand tho data, because there the future work will be combined. You never take just means, but when you are already calculating genotypes and their frequencies, then you have to take into account some other factors all the time. Adjust them according to age, height, weight. And you need to see these data. Without understanding the data, you cannot analyse them." (I11, Academic sector, Biomedicine)

This obviously raises the question as to what is actually meant by "seeing" and "understanding" the data. The visibility of the data seems to be crucial, but it does not necessarily mean that no alternative solutions or procedures are possible. The interviewees remarked that it would be possible to do scientific analysis without "seeing" the data but that it would make their work more complicated and therefore would be met with hesitation. Hence, it may be possible that the barrier here is the practiced and accepted way of doing things. Even now statistics offices often respond to data requests by disclosing sample databases that resemble the data so that researchers can script their queries.

However, the interviews also revealed that the visibility of data is necessary to guarantee their quality. This aspect was for instance stressed by an expert working in the Statistics Office. Similarly, the interviewee doing scientific research thought it possible that the quality of their work and data suffers if they do not have the full overview.

"We cannot combine different statistical works if we don't have the identifiers. To do statistics, to have good quality information, we need to have it /full overview of data/." (I13, Public sector, Statistics)

This quote illustrates nicely the way new technologies are understood first and foremost in the context of existing practices and boundaries. Similarly, people considering the importance of statistical analysis with SMC can imagine the activities they do in their current framework. Hence, statistical analysis comes down to finding means, comparing samples in valid ways, finding correlations and relationships within the data. And all this preferably with a user environment that is recognisable. While, for instance, the Statistics Office employees can write their own scripts for queries, for wider usability, future SMC systems will need to be similar to existing tools.

2.4 Goal for practical validation

Based on the insights from the interviews, we decided to evaluate the feasibility of a statistical analysis tool based on SMC. We designed (and later implemented) SMC protocols that compute various statistical analysis functions. We set efficiency and reusability as our two main goals as both are critical for providing a user experience similar to that of popular statistics tools.

We decided to use an example scenario to help us select the statistical data analysis functions to implement in our experiments. This scenario is inspired by a problem faced by governments that have enacted data protection laws—how to evaluate the effect of state investments without breaching the privacy of individual citizens? More specifically, we consider a government that wants to learn the efficiency of its investments in the education system.

One way for assessing the quality of educational institutions is to analyse the incomes of their graduates. For a fair analysis, the Statistics Bureau has to combine data from the Tax Office, the Ministry of Education and the Population Register. However, in some countries, laws prohibit the aggregation of citizen databases into a single database. Hence, the Statistics Bureau needs to maintain privacy throughout the analysis. First, data owners need a secure way for providing data. Second, the data analyst has to assess the distributions and quality of the data without seeing individual records. Third, the analyst must combine the data from three sources to an analysis database. Finally, he or she performs statistical tests to find the educational factors that have a significant impact on future income.

3 A security model for the analysis of private data

3.1 Privacy expectations and definitions

When describing SMC to potential end users, we focused on its outstanding privacy-preserving properties. Therefore, the main security goal in the proposed applications was that the private inputs of the input parties remain hidden from the computing parties and the result parties.

While it is tempting to define privacy so that the computing parties and result parties learn nothing about the values of the input parties, such a definition

would be rather impractical. First, we would need to hide the sizes of all inputs from the computing parties. There are several techniques for hiding the input size (e.g., [13, 24]), but no generic solution exists and practical protocols often leak the upper bound of the size.

Second, we would need to hide all branching decisions based on the private inputs. While this can be done by always executing both branches and obliviously choosing the right result, we can significantly save resources when we perform some branching decisions based on published values. However, such behaviour can partially or fully leak the inputs to the computing parties (and also to the result parties, should they measure the running time of Π).

This directs us to a relaxed privacy definition, that allows the computing parties to learn the sizes of inputs and make limited branching decisions based on published values that do not directly leak private inputs. Finally, to support practical statistical analysis tasks, we also allow the result parties to learn certain aggregate values based on the inputs (e.g., percentiles). In a real-world setting, we prevent the abuse of such queries using query auditing techniques, that reject queries or query combinations that are extracting many private inputs.

Definition 1 (Relaxed privacy of a multi-party computation procedure). *A multi-party computation procedure Π evaluated by parties $\mathcal{I}^k$, $\mathcal{C}^m$, $\mathcal{R}^n$ preserves the privacy of the input parties if the following conditions hold:*

Source privacy *During the evaluation of Π, computing parties cannot associate a particular computation result with the input of a certain input party.*

Cryptographic privacy *During the evaluation of Π, computing parties learn nothing about the intermediate values used to compute results, including the individual values in the inputs of input parties, unless any of these values are among the allowed output values of Π. As an additional exception, if a computing party is also an input party, it may learn the individual values in the input of only that one input party.*

Restricted outputs *During the evaluation of Π, the result parties learn nothing about the intermediate values used to compute results, including the individual values in the inputs of input parties, unless any of these values are among the allowed outputs of Π. Additionally, if a result party is also an input party, it may learn the input of only that one input party.*

Output privacy *The outputs of Π do not leak significant parts of the private inputs.*

3.2 Implementing private data analysis procedures with SMC

We now describe general guidelines for designing privacy-preserving algorithms that satisfy Definition 1.

For source privacy, we require that computing parties cannot associate an intermediate value with an individual input party that contributed to this value. For instance, we may learn the smallest value among the private inputs, but we will not know which input party provided it. This can be achieved by starting the protocol by *obliviously shuffling* the data [22].

Cryptographic privacy is achieved by using SMC protocols that collect and store inputs in a protected (e.g., encrypted, secret-shared) form. This prevents the computing parties from recovering private inputs on their own. Furthermore, the protection mechanism must be maintained for private values throughout the algorithm execution. The computing parties must not remove the protection mechanism to perform computations. Examples of suitable techniques include homomorphic secret sharing, homomorphic encryption and garbled circuits.

Restricting outputs is quite straightforward. First, the computing parties must publish to other parties only the result values that Π allows to publish. Everything else must remain protected. Trivially, it follows that the computing parties must run only the procedures to which the computational parties have agreed. Furthermore, the computing parties must reject all queries from the result parties that the computing parties have not agreed to among themselves.

Output privacy is the most complex privacy goal, requiring a more creative approach. The most complex part in algorithm design is to control the leakage of input value bits through published outputs. There are many measures for this leakage, including input entropy estimation and differential privacy [10]. Regardless of the approach, the algorithm designer must analyse the potential impact of publishing the results of certain computations. In some cases, such an analysis is straightforward. For example, publishing the results of aggregations like sum and mean is a negligible leak unless there are only a few values.

Typically, directly publishing a value from the private inputs should not be allowed. However, there are exceptions to this rule. For example, descriptive values, such as the minimal value in a private input, are used by statisticians to evaluate data quality. The main concern of data analysts in our interviews was that if we take away their access to individual data values, we need to give them a way to get an overview of the data in return. That is the reason why our privacy model allows the publishing of descriptive statistics.

4 Privacy-preserving algorithms for statistical analysis

4.1 Data import and filtering

We present a suite of privacy-preserving algorithms for statistical data analysis that are private according to Definition 1. The algorithms described are not dependent on any particular protection method. However, we assume that the protection method provides privacy-preserving primitive operations required by the algorithm. We describe one example implementation in Section 5.

When collecting data from several input parties, a common data model has to be agreed upon and key values for linking data from different parties have to be identified. For efficiency, it is often useful to preprocess and clean data at the input parties before sending it to computing parties. This will not compromise data privacy as the data will be processed by the input party itself. We now look at how to filter and clean data once it has been sent to the computing parties.

In the following, let $[\![x]\!]$ denote a private value x, let $[\![\boldsymbol{a}]\!]$ denote a private value vector $\boldsymbol{a}$, and let binary operations between vectors be point-wise operations.

Encoding missing values. Sometimes, single values are missing from the imported dataset. There are two options for dealing with this situation: we can use a special value in the data domain for missing values; or add an extra attribute for each attribute to store this information. Only one shared bit of extra data needs to be held per entry. Let the availability mask $[\![\mathsf{available}(\boldsymbol{a})]\!]$ of vector $[\![\boldsymbol{a}]\!]$ contain 0 if the corresponding value in the attribute $[\![\boldsymbol{a}]\!]$ is missing and 1 otherwise. The overall count of records in storage is public. If missing elements exist, that value does not reflect the number of available elements and it is not possible to make sure which elements are available by looking at the data. However, the count of available elements can be computed by summing the values in the availability mask.

Evaluating filters and isolating filtered data. To filter data based on a condition, we compare each element in the the corresponding private attribute vector $[\![\boldsymbol{a}]\!]$ to the filter value in a privacy-preserving manner and obtain a private vector of comparison results. This mask vector $[\![\boldsymbol{m}]\!]$ contains 1 if the condition holds and 0 otherwise. If there are several conditions in a filter, the resulting mask vectors are multiplied to combine the filters. Such filters do not leak which records correspond to the conditions. To learn the number of filtered records we find the sum of elements in the mask vector.

Most of our algorithms are designed so that filter information is taken into account during computations. However, in some cases, it is necessary to build a subset vector containing only the filtered data.

For obliviously cutting the dataset based on a given filter, first the value and mask vector pairs are obliviously shuffled, retaining the correspondence of the elements. Next, the mask vector is declassified and values for which the mask vector contains 0 are removed from the value vector. The obtained cut vector is then returned to the user. This process leaks the number of values that correspond to the filters that the mask vector represents. This makes cutting trivially safe to use, when the number of records in the filter would be published anyway. Oblivious shuffling ensures that no other information about the private input vector and mask vector is leaked [22]. Therefore, all algorithms that use oblivious cut provide source privacy.

4.2 Data quality assurance and visibility

Quantiles and outlier detection. Datasets often contain errors or extreme values that should be excluded from the analysis. Although there are many elaborate outlier detection algorithms like [5], outliers are often detected using quantiles. As no one method for computing quantiles has been widely agreed upon in the statistics community, we use algorithm $\mathbf{Q_7}$ from [16], because it is the default choice in our reference statistical analysis package GNU R. Let p be the percentile we want to find and let $[\![\boldsymbol{a}]\!]$ be a vector of values sorted in ascending order. Then the quantile is computed using the following function:

$$\mathbf{Q}_7(p, [\![\boldsymbol{a}]\!]) = (1-\gamma) \cdot [\![\boldsymbol{a}]\!][j] + \gamma \cdot [\![\boldsymbol{a}]\!][j+1] \ ,$$

where $j = \lfloor (n-1)p \rfloor + 1$, n is the size of vector $[\![\boldsymbol{a}]\!]$, and $\gamma = np - \lfloor (n-1)p \rfloor - p$. Once we have the index of the quantile value, we can use oblivious versions of vector lookup or sorting to learn the quantile value from the input vector.

We do not need to publish the quantile to use it for outlier filtering. Let q_0 and q_1 be the 5% and 95% quantiles of an attribute $[\![\boldsymbol{a}]\!]$. It is common to mark all values smaller than q_0 and larger than q_1 as outliers. The corresponding mask vector is computed by comparing all elements of $[\![\boldsymbol{a}]\!]$ to $\mathbf{Q_7}(0.05, [\![\boldsymbol{a}]\!])$ and $\mathbf{Q_7}(0.95, [\![\boldsymbol{a}]\!])$, and then multiplying the resulting index vectors. This way, data can be filtered to exclude the outlier data from further analysis. It is possible to combine the mask vector with the availability mask $[\![\mathsf{available}(a)]\!]$ and cache it as an updated availability mask to reduce the filtering load. Later, this mask can be used with the data attributes as they are passed to the statistical functions.

Descriptive statistics. As discussed in Section 2.3, one of the data analysts' main concerns was that they will lose the ability to see individual values before analysing them. However, such access is not always needed and it is sufficient to have a range of descriptive statistics about the data attributes that help discover anomalies.

We claim, that given access to these aggregate values and the possibility to filter out outliers, we can ensure data quality without compromising the privacy of individual data owners. Indeed, the aggregated values of individual attributes leak information about inputs. However, the leakage is small and strictly limited to previously agreed aggregate values.

The most common aggregate for individual attributes is the five-number summary—a descriptive statistic that includes the minimum, lower quartile, median, upper quartile and maximum of an attribute. We compute the five-number summary of a data vector using the previously discussed quantile formula. Based on the five-number summary and quantiles, box-plots can be drawn that give a visual overview of the data and effectively draw attention to outliers.

It is also important to see the distribution of a data attribute. For categorical attributes, this can be done by computing the frequency of the occurrences of different values. For numerical attributes, we must split the range into bins specified by breaks and compute the corresponding frequencies. The resulting frequency table can be visualised as a histogram. The algorithm publishes the number of bins and the number of values in each bin.

4.3 Linking multiple tables

After collecting input values and compiling filters for the outliers, we can link the input databases to form the final analysis database. There are various ways for linking databases in a privacy-preserving manner. As a minimum, we desire linking algorithms that do not publish private input values and only disclose the sizes of the input and output databases. Such algorithms are known to exist [21].

4.4 Statistical testing

The principles of statistical testing. Many statistical analysis tasks conclude with the comparison of different populations. For instance, we might want to know whether the average income of graduates of a particular university is significantly higher than that of other universities. In such cases, we first extract two groups—the case and control populations. In our example, the case population corresponds to graduates of the particular university in question and the control group is formed of persons from other universities. Note that a simple comparison of corresponding means is sufficient as the variability of income in the subpopulations might be much higher than the difference between means.

Statistical tests are specific algorithms, which formally quantify the significance of the difference between means. These test algorithms return the test statistic value that has to be combined with the sizes of the compared populations to determine the significance of the difference. While we could also implement a privacy-preserving lookup to determine this significand and prevent the publication of the statistic value, statisticians are used to including the statistic values and group sizes in their reports.

The construction of case and control populations. We first need to privately form case and control groups before starting the tests. One option is to select the subjects into one group and assume all the rest are in group two, e.g., students who go to city schools and everyone else. Alternatively, we can choose subjects into both groups, e.g., men who are older than 35 and went to a city school and men who are older than 35 who did not go to a city school. These selection categories yield either one or two mask vectors. In the former case, we compute the second mask vector by flipping all the bits in the existing mask vector. Hence, we can always consider the version where case and control groups are determined by two mask vectors.

In the following, let $[\![\boldsymbol{a}]\!]$ be the value vector we are testing and let $[\![\boldsymbol{m_1}]\!]$ and $[\![\boldsymbol{m_2}]\!]$ be mask vectors for case and control groups, respectively. Then $[\![n_i]\!] = \mathbf{sum}([\![\boldsymbol{m_i}]\!])$ is the count of subjects in the corresponding population.

The tests need to compute the mean, standard deviation or variance of a a population. We do this by evaluating the standard formulae using SMC. For improved precision, these metrics should be computed using real numbers.

Student's t-tests. The two-sample Student's t-test is the simplest statistical tests that allows us to determine whether the difference of group means is significant or not compared to variability in groups. There are two common flavours of this test [18] depending on whether the variability of the populations is equal.

In some cases, there is a direct one-to-one dependence between case and control group elements. For example, the data consists of measurements from the same subject (e.g., income before and after graduation), or from two different subjects that have been heuristically paired together (e.g., a parent and a child). In that case, a paired t-test [18] is more appropriate to detect whether a significant change has taken place.

The algorithm for computing both t-tests is a straightforward evaluation of the respective formulae using SMC, preferably with privacy-preserving real number operations. Both algorithms only publish the statistic value and the population sizes.

Wilcoxon rank sum test and signed rank test. T-tests are formally applicable only if the distribution of attribute values in case and control groups follows the normal distribution. If this assumption does not hold, it is appropriate to use non-parametric Wilcoxon tests. The Wilcoxon rank sum test [14] works on the assumption that the distribution of data in one group significantly differs from that in the other.

A privacy-preserving version of the rank sum test follows the standard algorithm, but we need to use several tricks to achieve output privacy. First, we need a more complex version of the cutting procedure to filter the database, the cases and controls using the same filter. Second, to rank the values, we sort the filtered values together with their associated masks by the value column.

Similarly to Student's paired t-test, the Wilcoxon signed-rank test [27] is a paired difference test. Often, Pratt's correction [14] is used for when the values are equal and their difference is 0. In a privacy-preserving version of this algorithm, we again need to cut several columns at once. We also need to obliviously separate absolute values and signs from the signed inputs values and later sort these two vectors by the sign vector.

The computation of both tests is simplified by the fact that most operations are done on signed integers and secure real number operations are not required before computing the final z-score statistic. Both algorithms only publish the statistic value and the population sizes.

The χ^2-tests for consistency. If the attribute values are discrete such as income categories then it is impossible to apply t-tests or their non-parametric counterparts and we have to analyse frequencies of certain values in the dataset. The corresponding statistical test is known as χ^2-test.

The privacy-preserving version of the χ^2-test is implemented simply by evaluating the algorithm using SMC operations. The algorithm can be optimised, if the number of classes is small, e.g., two. The algorithm publishes only the statistic value and the population sizes.

5 Practical results

5.1 An experimental statistical study using SMC

We demonstrate our privacy-preserving statistics capability by designing, implementing and conducting an experimental study. In the scenario, we use a table of subjects and their demographic information from the Population Register, a table specifying whether a subject attended a city school from the Ministry of

Education, and a table of taxed income payments for the same subjects from the Tax Office. We used artificially generated data in our experiments.

For our implementation, we chose the Sharemind SMC platform, because it supports operations needed in our implementation, including integer, boolean and floating point arithmetic, table join and sorting. We implemented the statistical algorithms using the SecreC programming language. We uploaded data using a data importer application developed using the Sharemind controller library. Details of our implementation are given in Appendix A.

Sharemind provides cryptographic security against an honest-but-curious adversary. This is enough for performing statistical analysis on private databases held by organizations united by a common cause (e.g., government agencies, hospitals, companies). While our implementation is built on and optimised for Sharemind, our algorithms can be adapted to other secure computation systems with similar capabilities.

5.2 Performance measurements

We conducted the experiments on a Sharemind installation running on three computers with 3 GHz 6-core Intel CPUs with 8 GB RAM per core (a total of 48 GB RAM). While monitoring the experimental scenario, we did not notice memory usage above 500 MB per machine. The computers were connected using gigabit ethernet network interfaces.

Table 2 contains the operations, input sizes and running times for our experimental scenario. We see that most operations in our experimental study take under a minute to complete. The most notable exceptions is the group median computation, as median computation has to be applied to the payments of 2000 subjects. This time can be reduced by vectorising the median invocations or conduct this aggregation before the data is converted into secret-shared form.

To check scalability, we performed some tests on ten times larger data vectors. We found that increasing input data size 10 times increases running time about 5 times. Only histogram computation is actually slower, because it uses a more detailed frequency table for larger databases.

The improved efficiency per input data element is explained by the use of vectorised operations of the Sharemind framework. The operations in the Sharemind framework are more efficient when many are performed in parallel using the SIMD (single instruction, multiple data) model.

6 Conclusion

In this paper we presented an easy-to-visualise model for explaining the capabilities and deployment of SMC to end users. These models helped us conduct a series of interviews with potential stakeholders of SMC to learn how SMC could be valuable to them.

Based on the end user needs gathered from the interviews we identified the need for an SMC-based statistical analysis toolkit. We designed and implemented

Step 1: Data import

Operation	*Record count*	*Time*
Data import from offsite computer	2 000	3 s
	53 977	24 s

Step 2: Descriptive statistics

Operation	*Record count*	*Time*
5-number summary (publish filter size)	2000	21 s
	20000	97 s
5-number summary (hide filter size)	2000	27 s
	20000	107 s
Frequency table	2000	16 s
	20000	222 s

Step 3: Grouping and linking

Operation	*Record count*	*Time*
Median of incomes by subject	53 977	3 h 46 min
Linking two tables by a key column	2000×5 and 2000×3	28 s
Linking two tables by a key column	2000×7 and 2000×2	29 s

Step 4: Statistical tests

Operation	*Record count*	*Time*
Student's t-test, equal variance	2000	167 s
	20000	765 s
Student's t-test, different variance	2000	157 s
paired t-test, known mean	2000 and 2000	98 s
paired t-test, unknown mean	2000 and 2000	102 s
χ^2-test, 2 classes	2000	9 s
	20000	10 s
χ^2-test, n-class version, 2 classes	2000	20 s
χ^2-test, n-class version, 5 classes	2000	23 s
Wilcoxon rank sum	2000	34 s
Wilcoxon signed-rank	2000 and 2000	38 s

Table 2: Running times of privacy-preserving statistics (in seconds)

privacy-preserving versions of several statistical functions. As a result, were able to conduct a full-scale experimental statistical study so that confidential data were always processed using SMC.

The strengths of our solution are generality, precision and practicality. First, we show that secure multi-party computation is flexible enough for implementing complex applications. Second, our use of secure floating point operations makes our implementation more precise. Third, we use the same algorithms as popular statistical toolkits like GNU R without simplifying the underlying mathematics.

Acknowledgements

This work was supported by the European Regional Development Fund through the Estonian Center of Excellence in Computer Science, EXCS. It has also

received funding from the European Union Seventh Framework Programme (FP7/2007-2013) under grant agreement no. 284731.

The authors wish to thank the interviewees for their time and cooperation and the Estonian Center for Applied Research for their help in generating the artificial data used in the experiments of this paper.

References

1. Gagan Aggarwal, Nina Mishra, and Benny Pinkas. Secure computation of the median (and other elements of specified ranks). *Journal of Cryptology*, 23(3):373–401, 2010.
2. Dan Bogdanov, Margus Niitsoo, Tomas Toft, and Jan Willemson. High-performance secure multi-party computation for data mining applications. *International Journal of Information Security*, 11(6):403–418, 2012.
3. Dan Bogdanov, Riivo Talviste, and Jan Willemson. Deploying secure multi-party computation for financial data analysis (short paper). In *Proceedings of FC 2012*, pages 57–64, 2012.
4. Peter Bogetoft, Dan Lund Christensen, Ivan Damgård, Martin Geisler, Thomas P. Jakobsen, Mikkel Krøigaard, Janus Dam Nielsen, Jesper Buus Nielsen, Kurt Nielsen, Jakob Pagter, Michael I. Schwartzbach, and Tomas Toft. Secure Multiparty Computation Goes Live. In *Proceedings of FC 2009*, pages 325–343, 2009.
5. Markus M. Breunig, Hans-Peter Kriegel, Raymond T. Ng, and Jörg Sander. Lof: Identifying density-based local outliers. In *Proceedings of CM SIGMOD 2000*, pages 93–104, 2000.
6. Martin Burkhart, Mario Strasser, Dilip Many, and Xenofontas A. Dimitropoulos. SEPIA: Privacy-Preserving Aggregation of Multi-Domain Network Events and Statistics. In *Proceedings of USENIX 2010*, pages 223–240, 2010.
7. Ran Canetti, Yuval Ishai, Ravi Kumar, Michael K. Reiter, Ronitt Rubinfeld, and Rebecca N. Wright. Selective private function evaluation with applications to private statistics. In *Proceedings of PODC 2001*, pages 293–304. ACM, 2001.
8. Ivan Damgård, Valerio Pastro, Nigel P. Smart, and Sarah Zakarias. Multiparty computation from somewhat homomorphic encryption. In *Proceedings of CRYPTO 2012*, volume 7417 of *LNCS*, pages 643–662. Springer, 2012.
9. Wenliang Du, Shigang Chen, and Yunghsiang S. Han. Privacy-preserving multivariate statistical analysis: Linear regression and classification. In *Proceedings of SDM 2004*, pages 222–233, 2004.
10. Cynthia Dwork. Differential privacy. In *Proceedings of ICALP'06*, volume 4052 of *LNCS*, pages 1–12. Springer, 2006.
11. Joan Feigenbaum, Benny Pinkas, Raphael Ryger, and Felipe Saint-Jean. Secure computation of surveys. In *EU Workshop on Secure Multiparty Protocols*, 2004.
12. Craig Gentry. Fully homomorphic encryption using ideal lattices. In *Proceedings of STOC 2009*, pages 169–178. ACM, 2009.
13. Oded Goldreich and Rafail Ostrovsky. Software Protection and Simulation on Oblivious RAMs. *Journal of the ACM*, 43(3):431–473, 1996.
14. Myles Hollander and Douglas A Wolfe. *Nonparametric statistical methods*. John Wiley New York, 2nd ed. edition, 1999.
15. H.C.M. Hoonhout. Setting the stage for developing innovative product concepts: people and climate. *CoDesign*, 3(S1):19–34, 2007.

16. Rob J Hyndman and Yanan Fan. Sample quantiles in statistical packages. *The American Statistician*, 50(4):361–365, 1996.
17. Liina Kamm, Dan Bogdanov, Sven Laur, and Jaak Vilo. A new way to protect privacy in large-scale genome-wide association studies. *Bioinformatics*, 29(7):886–893, 2013.
18. Gopal K Kanji. *100 statistical tests*. Sage, 2006.
19. Florian Kerschbaum. Practical privacy-preserving benchmarking. In *Proceedings of IFIP TC-11 SEC 2008*, volume 278, pages 17–31. Springer US, 2008.
20. Eike Kiltz, Gregor Leander, and John Malone-Lee. Secure computation of the mean and related statistics. In *Procedings of TCC 2005*, volume 3378 of *LNCS*, pages 283–302. Springer, 2005.
21. Sven Laur, Riivo Talviste, and Jan Willemson. From Oblivious AES to Efficient and Secure Database Join in the Multiparty Setting. In *Proceedings of ACNS'13*, volume 7954 of *LNCS*, pages 84–101. Springer, 2013.
22. Sven Laur, Jan Willemson, and Bingsheng Zhang. Round-Efficient Oblivious Database Manipulation. In *Proceedings of ISC 2011*, pages 262–277, 2011.
23. Christopher Lettl. User involvement competence for radical innovation. *Journal of engineering and technology management*, 24(1):53–75, 2007.
24. Yehuda Lindell, Kobbi Nissim, and Claudio Orlandi. Hiding the input-size in secure two-party computation. Cryptology ePrint Archive, Report 2012/679, 2012. `http://eprint.iacr.org/`.
25. Pille Pruulmann-Vengerfeldt, Liina Kamm, Riivo Talviste, Peeter Laud, and Dan Bogdanov. Deliverable D1.1—Capability model. `http://usable-security.eu/files/D1.1.pdf.pdf`, 2012.
26. Hiranmayee Subramaniam, Rebecca N. Wright, and Zhiqiang Yang. Experimental analysis of privacy-preserving statistics computation. In *Proceedings of SDM 2004*, volume 3178 of *LNCS*, pages 55–66. Springer, 2004.
27. Frank Wilcoxon. Individual Comparisons by Ranking Methods. *Biometrics Bulletin*, 1(6):80–83, 1945.
28. Andrew Chi-Chih Yao. Protocols for Secure Computations (Extended Abstract). In *Proceedings of FOCS'82*, pages 160–164. IEEE, 1982.

A Implementation details

A.1 Data model and data flow

The data model and transformations are shown in Figure 1. All attributes that are marked as having a mask, actually represent two attributes—one with the value and another with the availability flag. After linking is complete, we use the CompleteSubject table to test statistical hypotheses.

A.2 Overview of implemented operations

Figure 2 shows an overview of the SMC functionality that was used to run our experiments. It also shows how the algorithms implemented using SMC depend on each other. Our statistical functionality is built on the arithmetical, comparison and oblivious vector operations provided by Sharemind (two top functionality groups in Figure 2). However, our protocols can be ported to any SMC framework that provides the same set of features.

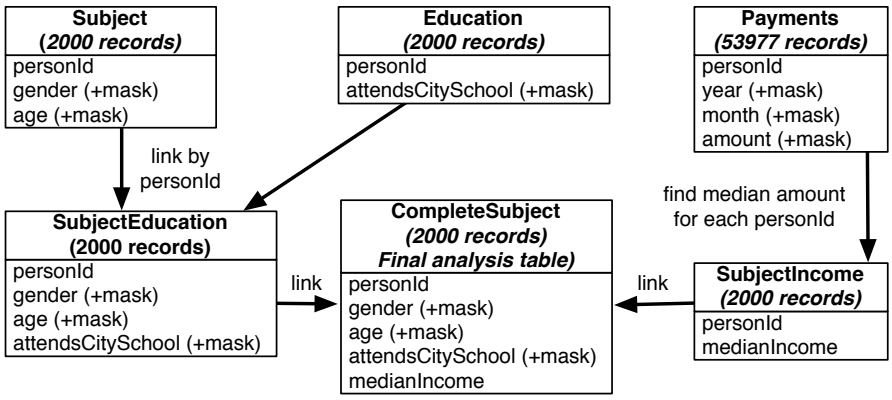


Fig. 1: The data model and table transformations in our experiment

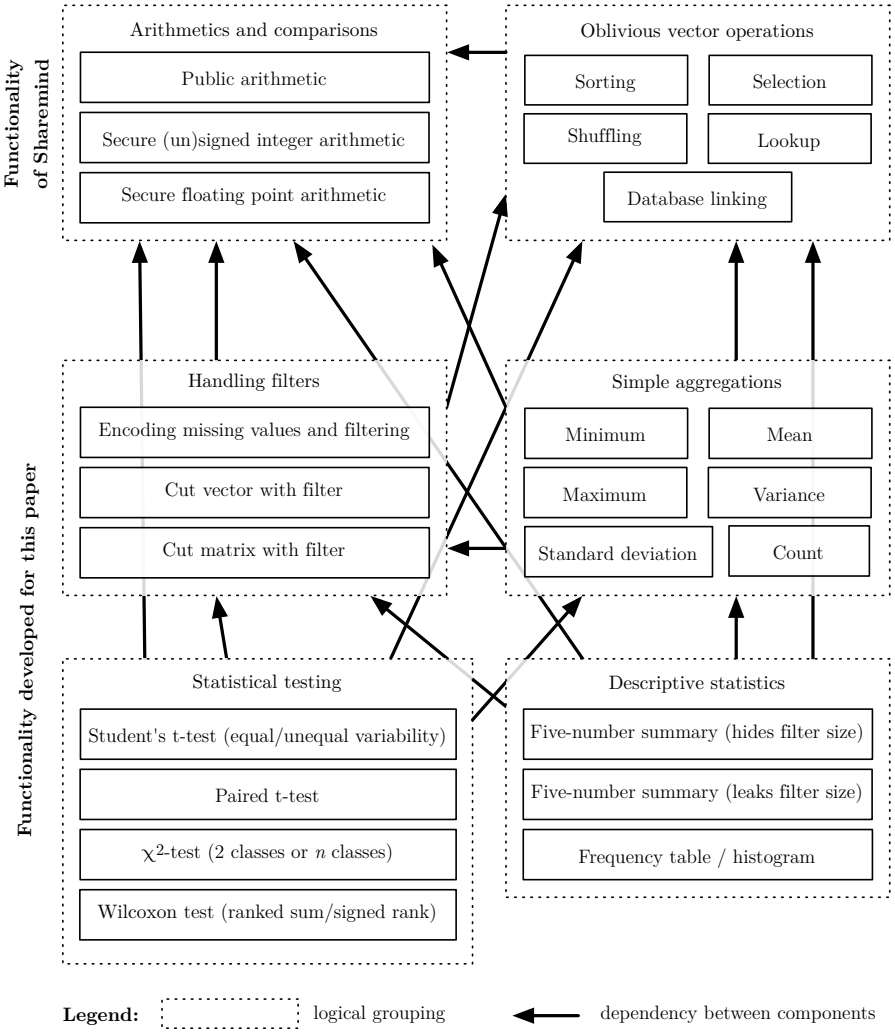


Fig. 2: Overview of operations implemented for our experiments

Appendix B

New Attacks against Transformation-Based Privacy-Preserving Linear Programming

The paper "New Attacks against Transformation-Based Privacy-Preserving Linear Programming" [33] follows.

New Attacks against Transformation-Based Privacy-Preserving Linear Programming

Peeter Laud[1] and Alisa Pankova[1,2,3]

[1] Cybernetica AS
[2] Software Technology and Applications Competence Centre (STACC)
[3] University of Tartu, Institute of Computer Science

Abstract. In this paper we demonstrate a number of attacks against proposed protocols for privacy-preserving linear programming, based on publishing and solving a transformed version of the problem instance. Our attacks exploit the geometric structure of the problem, which has mostly been overlooked in the previous analyses and is largely preserved by the proposed transformations. The attacks are efficient in practice and cast serious doubt to the viability of transformation-based approaches in general.
Keywords: Cryptanalysis, Secure multiparty computation, Linear programming

1 Introduction

Linear programming (LP) is one of the most versatile polynomial-time solvable optimization problems. It is usually straightforward to express various production planning and transportation problems as linear programs. There exist LP solving algorithms that are efficient both in theory and in practice. If the instances of these problems are built from data belonging to several mutually distrustful parties, the solving procedure must preserve the privacy of the parties. Thus it would be very useful to have an efficient privacy-preserving protocol that the data owners (and possibly also some other parties that help with computation) could execute for computing the optimal solution to a linear program that is obtained by combining the data of different owners. It is likely that such protocol would directly give us efficient privacy-preserving protocols for many other optimization tasks.

Several such protocols have indeed been proposed, following one of two main approaches. In the *secure multiparty computation (SMC) approach*, composable protocols for privacy-preserving arithmetic and relational operations are used to build a privacy-preserving implementation of some LP solving algorithm, typically the simplex algorithm. In the *transformation-based approach*, the algebraic structure of systems of linear inequalities and equations is used to apply a linear transformation to the description of the original problem, thus disguising it and allowing it to be solved publicly.

The security properties of the protocols of SMC approach can be derived from the properties of the protocols for primitive arithmetic and relational operations

through composability. The privacy guarantees these protocols offer are thus pretty well understood. The transformation-based methods have so far lacked the understanding of their privacy properties at a comparable level. The current paper demonstrates that such unavailability of security definitions is dangerous.

2 Privacy-Preserving Linear Programming

Throughout this paper, the upright upper case letters A denote matrices, and the bold lower case letters $\mathbf{b}$ denote column vectors. Writing two matrices/vectors together without an operator A$\mathbf{b}$ denotes multiplication, while separating them with a whitespace and putting into parentheses (A $\mathbf{b}$) denotes augmentation. By augmentation we mean attaching a column $\mathbf{b}$ to the matrix A from the right. This can be generalized to matrices: (A B) denotes a matrix that contains all the columns of A followed by all the columns of B. Row augmentation is defined analogously.

The *canonical form* for a linear programming task is the following:

$$\text{minimize } \mathbf{c}^{\mathrm{T}} \cdot \mathbf{x}, \text{ subject to } \mathrm{A}\mathbf{x} \leq \mathbf{b}, \mathbf{x} \geq \mathbf{0} \ . \tag{1}$$

Here A is an $m \times n$ matrix, $\mathbf{b}$ is a vector of length m and $\mathbf{c}$ is a vector of length n. There are n variables in the vector $\mathbf{x}$. The inequality of vectors is defined pointwise.

The LP solving algorithms, as well as protocols for privacy-preserving solution commonly expect the task to be in the *standard form*:

$$\text{minimize } \mathbf{c}^{\mathrm{T}} \cdot \mathbf{x}, \text{ subject to } \mathrm{A}\mathbf{x} = \mathbf{b}, \mathbf{x} \geq \mathbf{0} \ . \tag{2}$$

The inequality constraints of the canonical form can be transformed to equality constraints by introducing *slack variables*. The system of constraints $\mathrm{A}\mathbf{x} \leq \mathbf{b}$, $\mathbf{x} \geq \mathbf{0}$ is equivalent to the system $\mathrm{A}\mathbf{x} + \mathrm{I}\mathbf{x}_{\mathrm{s}} = \mathbf{b}$, $\mathbf{x}, \mathbf{x}_{\mathrm{s}} \geq 0$, where I is $m \times m$ identity matrix and $\mathbf{x}_{\mathrm{s}}$ is a vector of m new variables.

A *feasible solution* of a linear program is any vector $x_0 \in \mathbb{R}^n$ that satisfies its constraints. An *optimal solution* of a linear program is any feasible solution that maximizes the value of its cost function. The *feasible region* of a linear program is the set of all its feasible solutions. It is a polyhedron — the intersection of a finite number of hyperplanes and half-spaces. A feasible solution is *basic* if it is located in one of the vertices of that polyhedron.

In the privacy-preserving setting, the elements of the matrix A and the vectors $\mathbf{b}, \mathbf{c}$ are somehow contributed by several different parties. The cost vector $\mathbf{c}$ may be either held entirely by some party, or its entries may belong to different parties. Two standard ways of partitioning the constraints $\mathrm{A}\mathbf{x} \leq \mathbf{b}$ are the horizontal partitioning (each party contributes some of the constraints) and the vertical partitioning (each party knows certain columns of the matrix A). More general ways of data partitioning are possible, but these are not considered by the transformation methods that we are attacking.

In general, there are two main approaches to privacy-preserving linear programming. One approach is the straightforward cryptographic implementation

of a privacy-preserving version of some LP solving algorithm [14, 9]. Its main problem is efficiency since the entire optimization process must be performed in a manner that protects all intermediate values and comparison results. Another approach is transforming the program such a way that it could be given to a solver for offline computation. The optimal solution to the initial program has to be recoverable from the optimal solution to the transformed program.

In this work we present new attacks against some of the existing transformation methods. Without lessening the generality, we assume the number of parties to be 2, called Alice and Bob.

2.1 Transformation methods

Transformation-based methods have been proposed in [4, 3, 15, 11, 12, 16, 8, 2, 10, 7]. A set of "standard" transformations, applicable to the initial program, have been proposed over the years. Depending on the partitioning of constraints and the objective function, the application of a transformation may require cryptographic protocols of varying complexity. Each of the methods proposed in the literature typically uses several of these standard transformations.

Multiplying from the left. The idea of multiplying A and $\mathbf{b}$ in (2) by a random $m \times m$ invertible matrix P from the left was first introduced by Du [4]. This transformation conceals the outer appearance of A and $\mathbf{b}$, but the feasible region remains unchanged.

Multiplying from the right. The idea of multiplying A and $\mathbf{b}$ in (2) by a random invertible matrix Q from the right was also proposed by Du [4]. This hides also the cost vector $\mathbf{c}$. Unfortunately, it changes the optimal solution if some external constraints (e.g. the non-negativity constraints) of the form $\mathrm{B}\mathbf{x} \geq \mathbf{b}'$ are present, as it has been shown in [2]. In this case, the vector $\mathbf{b}'$ should also be modified according to the transformation, but that in fact reveals all the information about Q.

Scaling and Permutation. Bednarz et al. [2] have shown that, in order to preserve the inequality $\mathbf{x} \geq \mathbf{0}$, the most general type of Q is a positive generalized permutation matrix (a square matrix where each row and each column contains exactly one non-zero element). This results in scaling and permuting the columns of A. This transformation may also be applied to a problem in the canonical form (1).

Shifting. The shifting of variables has first been proposed in [3], and it has been also used in [16]. This transformation is achieved by replacing the constraints $\mathrm{A}\mathbf{x} \leq \mathbf{b}$ with $\mathrm{A}\mathbf{y} \leq \mathbf{b} + \mathrm{A}\mathbf{r}$, where $\mathbf{r}$ is a random non-negative vector of length n and $\mathbf{y}$ are new variables, related to the variables $\mathbf{x}$ through the equality $\mathbf{y} = \mathbf{x}+\mathbf{r}$. To preserve the set of feasible solutions, the inequalities $\mathbf{y} \geq \mathbf{r}$ have to be added to the system. A different transformation must then be used to hide $\mathbf{r}$.

2.2 Security Definition

There are no formal security definitions used in the transformation-based approach. The definition that has been used in the previous works is the *acceptable security*. This notion was first used in [5].

Definition 1. *A protocol achieves acceptable security if the only thing that the adversary can do is to reduce all the possible values of the secret data to some domain with the following properties:*

1. *The number of values in this domain is infinite, or the number of values in this domain is so large that a brute-force attack is computationally infeasible.*
2. *The range of the domain (the difference between the upper and lower bounds) is acceptable for the application.*

More detailed analysis [1, 3] estimates the probability that the adversary guesses some secret value. The leakage quantification analysis [3] is a compositional method for estimating the adversary's ability to make the correct guess when assisted by certain public information.

Although acceptable security could make the analysis simpler, it is not very well applicable in practice. Attacks on schemes that are secure by this definition have been found [2, 1]. The security of different transformation methods is very dependent on the initial settings of the problem — the partitioning of initial data, as well as on the type of used constraints (inequalities or equations).

2.3 Classification of Initial Settings

For each of the proposed transformation methods, the applicability and security strongly depend on the initial settings of the problem. For that reason, Bednarz [1] has introduced a classification of initial settings, provided with corresponding notation. She proposes to consider the following parameters:

Objective Function Partitioning How is the vector $\mathbf{c}$ initially shared? Is it known to Alice, to Bob, or to both of them? Are some entries known to Alice and others to Bob? Or does $\mathbf{c} = \mathbf{c}_{\text{Alice}} + \mathbf{c}_{\text{Bob}}$ hold, where $\mathbf{c}_{\text{Alice}}$ is "completely" unknown to Bob and vice versa?

Constraint Partitioning How is the matrix A initially shared? Is it public, known to one party, partitioned horizontally or vertically, or additively shared?

RHS Vector Partitioning How is the vector $\mathbf{b}$ initially shared?

Allowable Constraint Types Does the method admit only equality constraints, only inequalities, or both of them? Note that admitting only equality constraints means that the "natural" representation of the optimization problem is in terms of equalities. The use of slack variables to turn inequalities to equalities is not allowed.

Allowable Variable Types May the variables be assumed non-negative? Or may they be assumed free? Or can both types be handled?

Additionally, the classification considers which party or parties learn the optimal solution. This aspect does not play a role for our attacks.

The attacks described in this paper mostly target the transformation methods for LP tasks where the constraints are in the form of inequalities (1), and the set of constraints has been horizontally partitioned between Alice and Bob. The optimization direction $\mathbf{c}$ and its sharing does not play a big role in the main attacks, although some proposed transformation methods leave into it information that makes the attacks simpler. In our treatment, we assume all variables to be non-negative.

2.4 Overview of proposed methods

For exactly the setting described in the previous paragraph, Bednarz [1, Chap. 6] has proposed the following transformation. The set of constraints in (1) is transformed to

$$\hat{\mathrm{A}}\mathbf{y} = \hat{\mathbf{b}}, \mathbf{y} \geq \mathbf{0}, \tag{3}$$

where $\hat{\mathrm{A}} = \mathrm{P}\left(\mathrm{A}\ \mathrm{I}\right)\mathrm{Q}$, $\hat{\mathbf{b}} = \mathrm{P}\mathbf{b}$, I is the $m \times m$ identity matrix, P is a random invertible $m \times m$ matrix and Q is a random positive $(m+n) \times (m+n)$ generalized permutation matrix. New variables $\mathbf{y}$ are related to the original variables $\mathbf{x}$ and the slack variables $\mathbf{x}_\mathrm{s}$ by the equation $\binom{\mathbf{x}}{\mathbf{x}_\mathrm{s}} = \mathrm{Q}\mathbf{y}$. The objective function is disguised as $\hat{\mathbf{c}}^\mathrm{T} = (\mathbf{c}^\mathrm{T}\ \mathbf{0}^\mathrm{T})Q$, where $\mathbf{0}$ is a vector of m zeroes.

Other proposed transformations for horizontally partitioned constraints can be easily compared with Bednarz's. Du [4] applied the multiplication with both P and Q (where Q was more general) directly to the system of inequalities (1). Unfortunately, this transformation did not preserve the feasible region (and possibly the optimal solution) as shown by Bednarz et al. [2]. Vaidya [15] uses only the matrix Q, with similar correctness problems. Mangasarian [12] uses only the multiplication with P for a system with only equality constraints (2). Hong et al. [8] propose a complex set of protocols for a certain kind of distributed linear programming problems. Regarding the security, they prove that these protocols leak no more than what is made public by Bednarz's transformation. Li et al. [10] propose a transformation very similar to Bednarz's, only the matrix Q is selected from a more restricted set. This transformation is analyzed by Hong and Vaidya [7] and shown to provide no security (their attack has slight similarities with the one we present in Sec. 3.2). They propose a number of methods to make the transformation more secure and to also hide the number of inequalities in (1), including the addition of superfluous constraints and the use of more than one slack variable per inequality to turn them to equalities. We will further discuss the use of more slack variables in Sec. 3.1. The transformation by Dreier and Kerschbaum [3], when applied to (1), basically shifts the variables (Sec. 2.1), followed by Bednarz's transformation. We discuss the details and attacks specific to this transformation in Sec. 3.3.

3 Attacks

The system of constraints (1) consists of m inequalities of the form $\sum_{i=1}^{n} a_{ji}x_i \leq b_j$ for $j \in \{1, \ldots, m\}$, in addition to the non-negativity constraints. We assume that Alice knows the first r of these inequalities.

When Alice attempts to recover (1) from the result of Bednarz's transformation (3), she will first try to locate the slack variables, as described in Sec. 3.1. When she has located the slack variables, she can remove these, turning the equalities back to inequalities of the form $\mathrm{A}'\mathbf{x}' \leq \mathbf{b}'$. These constraints are related to (1) by $\mathrm{A}' = \mathrm{P}'\mathrm{A}\mathrm{Q}'$, $\mathbf{b}' = \mathrm{P}'\mathbf{b}$, where both P' and Q' are generalized permutation matrices (of size $m \times m$ and $n \times n$, respectively; Q' is also positive). Multiplication with P' from the left does not actually change the constraints, so the goal of Alice is to find Q'. The correspondence of the variables in $\mathbf{x}$ and $\mathbf{x}'$ can be found by looking at scale-invariant quantities related to constraints. Once the correspondence is found, the scaling factors can be easily recovered. All this is described in Sec. 3.2.

3.1 Identifying the Slack variables

Looking at the objective function When we add the slack variables to the system of inequalities in order to turn them to equations, then the coefficients of these slack variables in the cost vector $\mathbf{c}$ will be 0. In the existing transformation methods, the cost vector $\mathbf{c}$ is hidden by also multiplying it with a monomial matrix Q (product of a positive diagonal matrix and a permutation matrix) from the right. In this way, the zero entries in $\mathbf{c}$ are not changed. If all original variables had non-zero coefficients in the objective function, then the location of zeroes in the transformed vector $\mathbf{c}$ tells us the location of slack variables.

This issue can be solved by applying the transformation to the *augmented form* of linear program that includes the cost vector into the constraint matrix, and the cost value is expressed by a single variable:

$$\text{minimize } w, \text{ subject to } \begin{pmatrix} 1 & -\mathbf{c}^{\mathrm{T}} & 0 \\ 0 & \mathrm{A} & \mathrm{I} \end{pmatrix} \begin{pmatrix} w \\ \mathbf{x} \\ \mathbf{x_s} \end{pmatrix} = \begin{pmatrix} \mathbf{0} \\ \mathbf{b} \end{pmatrix}, \begin{pmatrix} w \\ \mathbf{x} \\ \mathbf{x_s} \end{pmatrix} \geq \mathbf{0} \ . \qquad (4)$$

The slack variables may be now hidden amongst the real variables by permutation. The location of the variable w should be known to the solver, although he may also solve all the n instances of linear programming tasks: for each variable in the task, try to minimize it.

There may be possibly other means of hiding $\mathbf{c}$. Hence we introduce more attacks that are not related to $\mathbf{c}$.

Looking at sizes of entries If the positions of slack variables have been hidden in the cost vector, they may be located by exploiting the structure of A. Namely, after the slack variables are introduced, they form an identity matrix

that is attached to A from the right. Thus each slack column contains exactly one non-zero entry. The columns of A are very unlikely to contain just one non-zero entry. We have found that the columns of P $\begin{pmatrix} \mathrm{A} & \mathrm{I} \end{pmatrix}$ can be distinguished by performing statistical analysis on the sizes of their entries. Even if using both positive and negative entries in A makes the mean more or less the same, the variance is smaller for the slack variables. The following scaling of the columns with the entries of Q does not provide any more protection.

We have discovered this problem occasionally, just because the columns appeared too different after applying the existing transformation methods. The previous works do not state precisely the distribution from which the entries of P (and Q) should be sampled. We have made experiments where we have sampled these entries independently of each other, according to the uniform distribution, or the normal distribution (the parameters of the distribution are currently unimportant, they only affect the scale of the resulting matrix, as well as the variance of its entries relative to each other). It turns out that selecting the entries of P randomly according to either one of these distributions keeps the variables distinguishable.

We performed a series of experiments, described below in detail. The instances of linear programming tasks were generated from a certain distribution that may differ from the distributions typical to some particular real-life problems, but nevertheless covers a large class of linear programs.

First, let us define the following probability distribution:

Definition 2. *If a random variable X is distributed according to the normal distribution $\mathcal{N}(\mu, \sigma^2)$, then the distribution of the absolute value $|X|$ is called the* folded normal distribution *and is denoted $\mathcal{N}_{\mathrm{f}}(\mu, \sigma^2)$.*

Our experiments were parametrized by the following quantities:

- the number of variables n and the number of inequality constraints m in (1);
- the fraction $p \in [0, 1]$ of zero entries in A;
- the fraction $a \in [0, 1]$ of constraints with non-negative coefficients;
- the fraction $q \in [0, 1]$ of zero entries outside the main diagonal of P.

We performed two sets of experiments. In one of them we sampled the entries of P, Q from a uniform distribution, and in the other one from a normal distribution.

An experiment proceeded as follows.

1. Generate a random point $v = (v_1, \ldots, v_n) \in \mathbb{R}^n$ where v_i is chosen uniformly from $(0, 100]$. This point will be contained in the polyhedron defined by the constraints in (1), thereby ensuring its non-emptiness.
2. Generate a random $m \times n$ matrix $\mathrm{A} = (a_{ij})_{i,j=1,1}^{m,n}$ whose entries are assigned in the following way:
   - The value 0 is taken with the probability p.
   - A random value is sampled uniformly from $[-100, 100] \subseteq \mathbb{R}$ (or from a normal distribution $\mathcal{N}(0, 100)$) with probability $1 - p$.

- After a row of A is generated, with probability a all entries in this row are replaced with their absolute values.

3. Generate the entries of the vector $\mathbf{b}$ of length m in such a way that the polyhedron defined by $\mathrm{A}\mathbf{x} \leq \mathbf{b}$ definitely contains the point v. That is, for each $i \in \{1, \ldots, m\}$, compute $b_i = a_{i1}v_1 + \ldots + a_{in}v_n + s$, where s is a random positive number. In our experiments, s was chosen uniformly from $[1000, 2000]$.
4. Let P be a $m \times m$ random matrix, the entries of which are assigned in the following way:
   - The value 0 is taken with the probability q (except the main diagonal, which stays non-zero in any case).
   - A random value is sampled uniformly from $[-100, 100]$ (or from a normal distribution $\mathcal{N}(0, 100)$) with probability $1 - q$.

   Note that P is invertible with probability 1.
5. Let Q be a $(m + n) \times (m + n)$ random positive generalized permutation matrix. The permutation defined by Q was picked uniformly from S_{m+n} and the non-zero entries of Q were uniformly sampled from $[1, 100]$ (or sampled from a folded normal distribution $\mathcal{N}_f(0, 100)$).
6. Construct $\hat{\mathrm{A}}$ and $\hat{\mathbf{b}}$ according to Bednarz's transformation.
7. For each column of $\hat{\mathrm{A}}$ compute the mean and the variance of its entries. Find the sets of m columns where (a) the means are the largest, (b) the means are the smallest, (c) the variances are the largest, or (d) the variances are the smallest.
8. The experiment was considered *successful* if one of the four sets of m columns found in the previous step exactly corresponded to the slack variables in $\mathbf{y}$ introduced by Bednarz's transformation.

When sampling the entries of P, Q from the uniform distribution, we ran 5 experiments for all possible values of the parameters, where $m+n \in \{100, 250, 500\}$, $m/(m + n) \in \{25\%, 50\%, 75\%\}$, $p, q \in \{0\%, 25\%, 50\%, 75\%, 90\%\}$, and $a \in \{0\%, 25\%, 50\%, 100\%\}$. For almost all settings, there was at least one experiment that was successful. The experiments were less successful only if m was small and p was large. When sampling the entries of P, Q from the normal distribution, we ran the same number of experiments with the same parameters.Again, for most settings, at least one of the experiments was successful. Again, we had less success if many entries in A were 0 (i.e. p was large) and there were less constraints than variables (i.e. $m/(m + n)$ was small). As we assumed, the best metrics was the variance, larger for the initial variables and smaller for the slack variables. For the largest parameters ($m+n = 500$), an attack took just a couple of seconds on a server with two Intel X5670 processors with 12 MB cache running at 2.93 GHz, and with 48 GB of main memory. The linear algebra operations were imported from *sage* [13]. Since sage does not round floating point numbers in the process of matrix multiplication, the transformation itself turned out to be too inefficient for choosing the initial parameters with high precision. For example, while the attack still takes several seconds for 64-bit initial numbers, the transformation takes half an hour. However, this issue affects significantly the

transformation, but not the attack timing. The attack timing grows less than linearly with the number of bits. We also did not notice that choosing more precise numbers would affect the outcome of the attack.

This problem can be potentially resolved by scaling the columns by a value that comes from a sufficiently large distribution to hide these differences. Although this makes the columns approximately the same size, it makes the values of the slack variables in the optimal solution to the transformed LP task much smaller than the values of the original variables, still keeping them distinguishable. Also, this modification does not affect the variances of the variables.

Another way is to add extra constraints whose entries that are large enough to provide noise for all the variables. The problem is that introducing more constraints requires introducing more slack variables for correctness. These slack variables cannot be protected by the same method. Once they have been revealed, they may be removed from the system by Gaussian elimination.

We would also like to note that the adversary may always bring the transformed matrix to its reduced row echelon form. This means that this transformation provides the best possible hiding, and the security analysis should be performed on this form. Unfortunately, it cannot be used for hiding instead of P since it is expensive to compute it while preserving the privacy.

Sampling the vertices of the polyhedron If the previous attack does not work well because the random values used during the transformation have been sampled in such a way that the entries of the resulting matrix have similar distributions, then there are still more ways of locating the slack variables. Consider (3), where each of the new variables $y_i \in \mathbf{x}$ is either a scaled copy of some original variable $x_{i'} \in \mathbf{x}$ or a (scaled) slack variable. The constraints (3) define an n-dimensional polyhedron in the space $\mathbb{R}^{m+n}$ (due to its construction, the matrix $\hat{\mathrm{A}}$ has full rank). In each vertex of this polyhedron, at least n of the variables in $\mathbf{y}$ are equal to zero. We have hypothesized that for at least a significant fraction of linear programs, it is possible to sample the vertices of this polyhedron in such manner, that slack variables will be 0 more often than the original variables.

To verify our hypothesis, we performed a series of experiments, described below in detail. Our experiments were parametrized by the quantities m, n, p, a described at the previous experiment. Additionally, the number $k \in \mathbb{N}$ determines the number of vertex samples done in an experiment, and the fraction $e \in [0, 1]$ affects the polyhedron that we use to look for variables that most often take the value 0 in vertices.

An experiment proceeded as follows.

1–6. Generate A, $\mathbf{b}$, $\hat{\mathrm{A}}$, $\hat{\mathbf{b}}$ as in the previous experiment, using the current values of m, n, p, a, and taking $q = 0$. The entries of all matrices are sampled from the uniform distribution.

7. Modify $\hat{\mathrm{A}}$ [resp. $\hat{\mathbf{b}}$] by removing their first $e \cdot m$ rows [resp. elements]. This corresponds to discarding a fraction of e equations from the system $\hat{\mathrm{A}}\mathbf{y} = \hat{\mathbf{b}}$.

We have found that such removal increases the success rate of the experiments for certain parameters.

8. Initialize the counters $z_1, \ldots, z_{m+n}$ to 0.
9. Repeat the following k times.
   (a) Generate the optimization direction $\mathbf{c} \in \mathbb{R}^{m+n}$ sampling each entry from the distribution $\mathcal{N}_{\mathrm{f}}(0,1)$.
   (b) Find an optimal *basic* solution (a solution located in a vertex of the polyhedron) to the linear program

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

   (c) If the optimal solution $\mathbf{y}_{\mathrm{opt}}$ exists, then increase by one each z_i where the i-th element of $\mathbf{y}_{\mathrm{opt}}$ equals 0.
10. The experiment was considered *successful* if the counters with n largest values exactly corresponded to the slack variables in $\mathbf{y}$ introduced by Bednarz's transformation.

We have performed our experiments with different settings. In all experiments, k was fixed to 100 (larger values did not seem to give any significant difference). For each set of values for the parameters (m, n, p, a, e), we performed 20 experiments. The results for all sets of experiments are reported in Table 1. For given (m, n, p, a), the symbol $*$ in the corresponding cell of the table indicates that none of 20 experiments performed for all values of e we considered were successful. If at least one experiment was successful for some value of e, given the parameters (m, n, p, a), then this value of e is given in the corresponding cell of the table.

Each attack took a couple of minutes. The largest matrices were obtained for $\frac{m}{m+n} = 0.75$; for $m + n = 250$ it took less than one minute, and for $m + n = 500$ about five minutes.

We also performed some initial experiments where the entries of the optimization direction $\mathbf{c}$ were sampled from $\mathcal{N}(0,1)$. This choice did not perform better (and sometimes performed much worse) than the sampling from $\mathcal{N}_{\mathrm{f}}(0,1)$.

We see that the worst case for our algorithm is when m is much smaller than n and the fraction of zero entries in A is large. The problem is that there are too few inequalities already in the beginning, and the zeroes make the initial matrix A even sparser and less constraining. The initial variables thus do not differ too much from the slack variables. However, if A is sparse, there may possibly exist other attacks based looking for certain affine relationships between the variables, similarly to the attacks from Sec. 3.3.

For $m > n$ it may happen that even the slack variables will not be allowed to take the value 0 at all because of too tight bounds. In this case, some equations have been just eliminated from the transformed program. This is not equivalent to removing bounds from the initial polyhedron, and it is not quite clear what exactly happens to it. However, there are definitely less constraints than before, and the slack variables again have higher probabilities of becoming 0.

The results also show something interesting about the effect of the structure of A on the outcome of the attack. It can be seen than the attack performs better

m	n	p	a			
			0.0	0.25	0.5	1.0
25	75	0	*	0	0	0
		0.25	*	*	0	0
		0.5	0	0	0	0
		0.75	0	0	0	0
		0.9	*	*	*	*
50	50	0	0	0	0	0
		0.25	0	0	0	0
		0.5	0	0	0	0
		0.75	*	0	0	0
		0.9	*	*	*	*
62	188	0	*	*	*	*
		0.25	*	*	*	*
		0.5	*	*	*	0
		0.75	*	*	*	0
		0.9	*	*	*	*
75	25	0	0.75	0.5	0.75	0.5
		0.25	0.75	0.5	0.75	0.5
		0.5	0.75	0.5	0.75	0.5
		0.75	0.75	0.5	0.75	0.5
		0.9	*	*	0.75	0.5
125	125	0	0	0	0	0
		0.25	0	*	0	0
		0.5	0	0	0	0
		0.75	*	0	0	0
		0.9	*	*	*	*

m	n	p	a			
			0.0	0.25	0.5	1.0
125	375	0	*	*	*	*
		0.25	*	*	*	*
		0.5	*	*	*	*
		0.75	*	*	*	*
		0.9	*	*	*	*
187	63	0	0.75	0.75	0.75	0.5
		0.25	0.75	0.5	0.75	0.5
		0.5	0.75	0.75	0.75	*
		0.75	*	0.75	0.75	0.5
		0.9	*	*	*	0.9
250	250	0	*	0	*	0
		0.25	*	0	*	0
		0.5	*	*	*	0
		0.75	*	*	*	0
		0.9	*	*	*	*
375	125	0	*	0.75	0.75	0.5
		0.25	0.75	0.75	0.75	0.75
		0.5	0.75	*	*	0.5
		0.75	*	0.75	*	0.5
		0.9	*	0.75	*	0.75
475	25	0	0.9	0.9	0.9	0.9
		0.25	0.9	0.9	0.9	0.9
		0.5	0.9	0.9	0.9	0.9
		0.75	0.9	0.9	0.9	0.9
		0.9	0.9	0.9	0.9	0.9

Table 1. Results of the vertex-sampling experiments

when all the entries of A are non-negative. The success rate is in general higher for smaller fraction of zero elements in A, especially for the smaller number of constraints.

Our experimental results show that for many linear programs in canonical form (1), it is possible to identify the slack variables after Bednarz's transformation. The validity of our hypothesis has been verified.

Several slack variables per inequality The authors of [7] proposed introducing multiple slack variables for the same inequality. We have tried experimentally that in this case there is even higher probability that the slack variables are those that most often take the value 0 in a vertex sampled as described previously; this can also be explained in theory. Also, in this case, the columns in $\hat{\mathrm{A}}$, corresponding to slack variables added to the same inequality, are multiples of each other. This makes them easily locatable.

Removing the slack variables Once we have located the slack variables, we will reorder the variables in the constraints $\hat{\mathrm{A}}\mathbf{y} = \hat{\mathbf{b}}$ so, that the non-slack variables are the first n variables and the slack variables are the last m variables in $\mathbf{y}$. This corresponds to the first n columns of $\hat{\mathrm{A}}$ containing the coefficients of non-slack variables in the system of equations, and the last m columns containing the coefficients of slack variables. We will now use row operations to bring the system to the form $\left(\mathrm{A}'\ \mathrm{I}\right)\mathbf{y} = \mathbf{b}'$, where I is $m \times m$ identity matrix. This system, together with the non-negativity constraints, is equivalent to the system of inequalities $\mathrm{A}'\mathbf{x}' \leq \mathbf{b}'$, where $\mathbf{x}'$ are the first n elements of $\mathbf{y}$.

3.2 Finding the permutation of variables

We will now describe the attack that allows to remove the scaling and the permutation of variables. An attack based on exploiting the slack variables has been proposed in [3]. If the system contains only inequalities, then they completely reveal a scaled permutation of P that may be afterwards used to recover a scaled permutation of M whose scaling may be afterwards removed by searching for common factors. The factoring attack can be avoided by using real entries in Q. Our attack does not use factoring, but exploits the geometrical structure of the transformed program.

Recall that the initial linear program is partitioned horizontally, so each party holds some number of constraints. Suppose Alice knows r inequalities $\sum_{i=1}^{n} a_{ji}x_i \leq b_j$ (where $j \in \{1, \ldots, r\}$) of the original system of constraints, from a total of m. We assume that r is at least 2. Alice also knows all scaled and permuted constraints $\sum_{i=1}^{n} a'_{ji}x'_i \leq b'_j$ (where $j \in \{1, \ldots, m\}$). If we could undo the scaling and permuting, then this set of m inequalities would contain all original r inequalities known by Alice. Next we show how Alice can recover the permutation of the variables. Once this has been recovered, the scaling is trivial to undo.

Alice picks two of the original inequalities she knows (e.g. k-th and l-th, where $1 \leq k, l \leq r$) and two inequalities from the scaled and permuted system (e.g. k'-th and l'-th, where $1 \leq k', l' \leq m$). She makes the guess that k-th [resp. l-th] original inequality is the k'-th [resp. l'-th] scaled and permuted inequality. This guess can be verified as follows. If the guess turns out to be correct, then the verification procedure also reveals the permutation (or at least parts of it).

For the inequality $\sum_{i=1}^{n} a_{ji}x_i \leq b_j$ in the original system let H_j be the corresponding hyperplane where "$\leq$" has been replaced by "=". Similarly, let H'_j be the hyperplane corresponding to the j-th inequality in the scaled and permuted system. The hyperplane H_j intersects with the i-th coordinate axis in the point $(0, \ldots, 0, z_{ji}, 0, \ldots, 0)$, where $z_{ji} = b_j/a_{ji}$ (here z_{ji} is the i-th component in the tuple). Also, let $(0, \ldots, 0, z'_{ji}, 0, \ldots, 0)$ be the point where H'_j and the i-th coordinate axis intersect.

Note that scaling the (initial) polyhedron s times along the i-th axis would increase z_{ji} by s times, too, for all j. Scaling it along other axes would not change z_{ji}. Hence the quantities z_{ki}/z_{li} (for $i \in \{1, \ldots, n\}$) are scale-invariant.

To verify her guess, Alice computes the (multi)sets $\{z_{ki}/z_{li} \mid 1 \leq i \leq n\}$ and $\{z'_{k'i}/z'_{l'i} \mid 1 \leq i \leq n\}$. If her guess was correct, then these multisets are equal. Also, if they are equal, then the i-th coordinate in the original system can only correspond to the i'-th coordinate in the scaled and permuted system if $z_{ki}/z_{li} = z'_{k'i'}/z'_{l'i'}$. This allows her to recover the permutation. If there are repeating values in the multisets, or if division by 0 occurs somewhere, then she cannot recover the complete permutation. In this case she repeats with other k, l, k', l'. But note that the presence of zeroes in the coefficients also gives information about the permutation.

This attack does not allow to discover precise permutations if the known inequalities are symmetric with respect to some variables, and the scaling cannot be derived for the variables whose coefficients in all the known inequalities are 0. It is also impossible if the right sides of all the known inequalities are 0. However, it would reduce the number of secure linear programming tasks significantly. Also, if two variables in the system look the same for Alice (they participate in the same way in all inequalities she knows) then it should not matter to her how they end up in the recovered permutation.

We have followed up our experiments reported in the previous section, and verified that the attack works in practice.

3.3 Attacks specific to [3]

Dreier and Kerschbaum [3] propose a transformation that is applicable to LP tasks containing both equality and inequality constraints. In this paper, we only consider its application to tasks with inequality constraints only (although the operations presented in this section are also applicable to equations). In their transformation, the variables are first shifted by a positive vector (as described in Sec. 2.1), and then Bednarz's transformation is applied to the resulting system. In [3], the construction is described somewhat differently and the resulting positive generalized permutation matrix Q used to scale and permute the columns of the constraint system is not the most general matrix possible. The attacks described below work for any possible Q.

Shifting back The shifting of variables that has been used in [3] (and also in the transformation presented by Wang et al. [16], which only applies to LP tasks with equality constraints, and is thus outside the scope of this paper) reduces to scaling. The inequalities $\mathbf{y} \geq \mathbf{r}$ for the variables $\mathbf{y}$ are transformed to equalities by the introduction of new slack variables $\mathbf{s}$. For the variable $y_i \in \mathbf{y}$, related to the original variable x_i through the equality $y_i = x_i + r_i$, we have the equality $y_i - s_i = r_i$, where s_i is a new slack variable. After applying Bednarz's transformation, the variables are scaled and this equality becomes $q_i\hat{y}_i - q'_i\hat{s}_i = r_i$. The new variables $\hat{y}_i$ and $\hat{s}_i$ are related to the previous ones by $y_i = q_i\hat{y}_i$ and $s_i = q'_i\hat{s}_i$, where q_i and q'_i are certain non-zero entries in the matrix Q. Thus $\hat{s}_i = (q_i\hat{y}_i - r_i)/q'_i = (y_i - r_i)/q'_i = x_i/q'_i$. I.e. the slack variable $\hat{s}_i$ is a scaled copy of the original variable x_i.

We could now eliminate the variables $\mathbf{y}$ (the shifted versions of the original variables $\mathbf{x}$) from the system of constraints and the objective function. We will then be left with the system that involves only the slack variables $\mathbf{s}$ from the inequalities $\mathbf{y} \geq \mathbf{r}$ and the slack variables $\mathbf{x}_\mathrm{s}$ from the inequalities in the original system. The resulting LP task could have been obtained from the original task through Bednarz's transformation and the attacks described above can be applied to it.

To eliminate the variables $\mathbf{y}$, we need to know their location. Dreier's and Kerschbaum's transformation [3] does not actually hide these variables, due to their choice of Q. But even if the permutation encoded in Q were more general, we could still recover the locations of the variables $\mathbf{y}$ as described below. The procedure described below also recovers the pairs $(\hat{y}_i, \hat{s}_i)$ of variables and corresponding slack variables, the difficulty of which is postulated in the cryptanalysis performed in [3].

Affine relationships in small sets of variables Each variable from $\mathbf{y} = \mathbf{x}+\mathbf{r}$ is associated with exactly one slack variable from $\mathbf{s}$. To find the pairs $(\hat{y}_i, \hat{s}_i)$, the adversary can just pick pairs of variables and then verify that they correspond to each other. The correspondence that the adversary can verify is the affine relationship $q_i\hat{y}_i - q'_i\hat{s}_i = r_i$ between these variables.

This problem can be stated more generally. Suppose that we have a linear equation system $\mathrm{A}\mathbf{x} = \mathbf{b}$. Consider the solution space of this system. If the space contains small sets of t variables that are in affine relationship $\alpha_1 x_{i_1} + \ldots + \alpha_t x_{i_t} = \beta$ for some $\alpha_i, \beta \in \mathbb{R}$ (that may be not obvious from the outer appearance A), then these equations may be recovered by looking through all the sets of variables of size t. To expose the affine relationship between $x_{i_1}, \ldots, x_{i_t}$, we will just use Gaussian elimination to get rid of all other variables. The procedure can be described as follows:

1. Repeat the following, until only variables $x_{i_1}, \ldots, x_{i_t}$ remain in the system.
   (a) Pick any other variable x_j that has not been removed yet.
   (b) Take an equation where x_j has non-zero coefficient. Through this equation, express the variable x_j in terms of the other variables. Substitute it into all the other equations. Remove the equation and the variable x_j. If there are no equations where x_j has non-zero coefficient, then remove only x_j, without touching any remaining equations.
2. The previous operations do not change the solution set of the system (for the remaining variables). Therefore, if there are any equations left, then there exist $\alpha_i, \beta \in \mathbb{R}$ (not all $\alpha_i = 0$) such that $\alpha_1 x_{i_1} + \ldots + \alpha_t x_{i_t} = \beta$.

In this manner, the adversary is able to find all unordered pairs $\{\hat{y}_i, \hat{s}_i\}$ related to each other through $q_i\hat{y}_i + q'_i\hat{s}_i = r_i$. The signs of q_i, q'_i, r_i in this relationship determine, which one is the original variable ($q_i r_i > 0$), and which one the slack variable ($q'_i r_i < 0$).

4 Conclusions

We have presented attacks against transformation-based methods for solving LP tasks in privacy-preserving manner. The attacks are not merely theoretical constructions, but work with reasonable likelihood on problems of practical size. The aim of this paper was to show that the attacks work in practice. It was not intended to estimate their theoretical complexity.

We have presented our attacks against methods that handle LP tasks where the constraints are specified as inequalities. May the methods for differently-represented LP tasks, e.g. as systems of equations [12, 16], still be considered secure? Our attacks are not directly applicable against this setting because the set of equations representing the subspace of feasible solutions is not unique and the hyperplanes in the original and transformed systems of constraints cannot be directly matched against each other like in Sec. 3.2. In our opinion, one still has to be careful because there is no sharp line delimiting systems of constraints represented as equations, and systems of constraints represented as inequalities. The canonical form (1) and the standard form (2) can be transformed to each other and the actual nature of the constraints may be hidden in the specified LP task.

The lack of precise definitions of confidentiality for transformation-based methods makes it harder to argue about the (in)security of a particular method. Further advances in this field would benefit from an indistinguishability-based definition of security, similar to [6]. In such a definition, the adversary would be allowed to pick two LP tasks, one of which would then be transformed by the environment. The adversary's goal is to find out, which of the two tasks was transformed. In this definition, it would also be possible to precisely state which parts of the task the transformation will not attempt to protect: the environment would check that these parts are equal for the two tasks selected by the adversary.

Acknowledgements. This work has been supported by the European Regional Development Fund through the Estonian Center of Excellence in Computer Science, EXCS, and the Software Technologies and Applications Competence Centre, STACC. This research was also supported by the European Union Seventh Framework Programme (FP7/2007-2013) under grant agreement no. 284731 "Usable and Efficient Secure Multiparty Computation (UaESMC)".

References

1. Alice Bednarz. *Methods for two-party privacy-preserving linear programming.* PhD thesis, University of Adelaide, 2012.
2. Alice Bednarz, Nigel Bean, and Matthew Roughan. Hiccups on the road to privacy-preserving linear programming. In *Proceedings of the 8th ACM workshop on Privacy in the electronic society*, WPES '09, pages 117–120, New York, NY, USA, 2009. ACM.
3. Jannik Dreier and Florian Kerschbaum. Practical privacy-preserving multiparty linear programming based on problem transformation. In *SocialCom/PASSAT*, pages 916–924. IEEE, 2011.

4. Wenliang Du. *A Study Of Several Specific Secure Two-Party Computation Problems*. PhD thesis, Purdue University, 2001.
5. Wenliang Du and Zhijun Zhan. A practical approach to solve secure multi-party computation problems. In *New Security Paradigms Workshop*, pages 127–135. ACM Press, 2002.
6. Shafi Goldwasser and Silvio Micali. Probabilistic encryption. *J. Comput. Syst. Sci.*, 28(2):270–299, 1984.
7. Yuan Hong and Jaideep Vaidya. An inference-proof approach to privacy-preserving horizontally partitioned linear programs. *Optimization Letters*, 2013. To appear. Published online 05 October 2012.
8. Yuan Hong, Jaideep Vaidya, and Haibing Lu. Secure and efficient distributed linear programming. *Journal of Computer Security*, 20(5):583–634, 2012.
9. Jiangtao Li and Mikhail J. Atallah. Secure and private collaborative linear programming. In *International Conference on Collaborative Computing*, pages 1–8, 2006.
10. Wei Li, Haohao Li, and Chongyang Deng. Privacy-preserving horizontally partitioned linear programs with inequality constraints. *Optimization Letters*, 7(1):137–144, 2013.
11. Olvi L. Mangasarian. Privacy-preserving linear programming. *Optimization Letters*, 5(1):165–172, 2011.
12. Olvi L. Mangasarian. Privacy-preserving horizontally partitioned linear programs. *Optimization Letters*, 6(3):431–436, 2012.
13. W. A. Stein et al. *Sage Mathematics Software (Version 5.10)*. The Sage Development Team, 2013. `http://www.sagemath.org`.
14. Tomas Toft. Solving linear programs using multiparty computation. In Roger Dingledine and Philippe Golle, editors, *Financial Cryptography and Data Security*, pages 90–107, Berlin, Heidelberg, 2009. Springer-Verlag.
15. Jaideep Vaidya. Privacy-preserving linear programming. In Sung Y. Shin and Sascha Ossowski, editors, *SAC*, pages 2002–2007. ACM, 2009.
16. Cong Wang, Kui Ren, and Jia Wang. Secure and practical outsourcing of linear programming in cloud computing. In *INFOCOM, 2011 Proceedings IEEE*, pages 820–828, 2011.

Appendix C

On the (Im)possibility of Privately Outsourcing Linear Programming

The paper "On the (Im)possibility of Privately Outsourcing Linear Programming" [34] follows.

On the (Im)possibility of Privately Outsourcing Linear Programming

Peeter Laud
Cybernetica AS
peeter.laud@cyber.ee

Alisa Pankova
Cybernetica AS
Software Technologies and Applications Competence Centre
University of Tartu, Institute of Computer Science
alisa.pankova@cyber.ee

ABSTRACT

In this paper we study the security definitions and methods for transformation-based outsourcing of linear programming. The recent attacks have shown the deficiencies of existing security definitions; thus we propose a stronger, indistinguishability-based definition of security of problem transformations that is very similar to IND-CPA security of encryption systems. We will study the realizability of this definition for linear programming and find that barring radically new ideas, there cannot exist transformations that are secure information-theoretically or even computationally. We conclude that for solving linear programming problems in privacy-preserving manner, cryptographic methods for securely implementing Simplex or some other linear programming solving algorithm are the only viable approach.

Categories and Subject Descriptors

E.3 [**Data**]: Data Encryption; C.2.4 [**Computer-Communication Networks**]: Distributed Systems—*Client/server*

Keywords

Cryptanalysis; Linear programming; Secure outsourcing

1. INTRODUCTION

In linear programming (LP), one seeks the optimal value of a linear function of several arguments, subject to linear constraints on these arguments. In the canonical form, a LP task is

$$\text{maximize } \vec{c}^{\mathrm{T}}\vec{x}, \text{ subject to } A\vec{x} \leq \vec{b}, \vec{x} \geq \vec{0}, \qquad (1)$$

where $A \in \mathbb{R}^{m\times n}$, $\vec{b} \in \mathbb{R}^m$, $\vec{c} \in \mathbb{R}^n$ (all vectors are column vectors), and the inequalities hold componentwise. The solution $\vec{x}_{\mathrm{opt}}$ is a vector of length n over real numbers. There exist algorithms for solving LP tasks that are efficient in theory and/or in practice. A large number of practical optimization problems can be cast as LP tasks either exactly or approximately. As the specifications of these problems may also have privacy constraints, there has been ample interest towards privacy-preserving methods for solving LP tasks.

Secure outsourcing is conceptually the simplest case of privacy-aware computation. In this case, there are two parties, the *client* and the *server*. The client has a task T which it would like to solve, but has insufficient computational resources for doing so. The server is powerful enough to solve the task T. The client wants to keep T private from the server. Thus the client and server engage in a protocol that results in the client learning the solution to T, but is computationally less demanding for the client than solving T himself. Such outsourcing protocols have been proposed for different tasks, e.g. sequence matching [2], database operations [11], cryptographic operations [12, 6], and some linear algebra tasks [1] including linear programming [8, 22]. Quite often, the outsourcing protocol consists of the client transforming the task T in some manner that hides its original description, the server solving the transformed task, and the client transforming the solution of the transformed task back to the solution of the original task. To reap the benefits of outsourcing, a good, problem-specific transformation is probably necessary [7].

In this paper, we consider the feasibility of outsourcing linear programming in privacy-preserving manner. This problem is tightly related to multiparty privacy-preserving LP, where the specification $(A, \vec{b}, \vec{c})$ of a LP task is distributed among several mutually distrusting parties. There is a decent number of proposed protocols for this task that are actually based on outsourcing as well — the clients execute a protocol that results in a transformed task, this is solved publicly, and another protocol transforms this solution back to the solution of the original LP task [9, 21, 18, 19, 4, 17, 13, 14].

This paper is motivated by some recent attacks against privacy-preserving multiparty and/or outsourcing LP solving methods [3, 15]. They show that, for linear programs given in canonical form and split horizontally between parties (each party provides some of the constraints of the system), the existing transformations can in general be undone by some of the parties, using the partial knowledge they have about the initial task. There exist other forms of LP tasks (see Sec. 2.1) and proposed transformations for them, and the existing attacks are not directly applicable to these transformations, but as all forms are equivalent, it may be possible to adapt the attacks.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than the author(s) must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.
CCSW'13, November 8, 2013, Berlin, Germany.
Copyright is held by the owner/author(s). Publication rights licensed to ACM.
ACM 978-1-4503-2490-8/13/11 ...$15.00.
http://dx.doi.org/10.1145/2517488.2517490.

The outline of this paper is the following. In Sec. 2 we review LP-related notions and existing transformation techniques from related works. In Sec. 3 we study how to build LP transformations on top of solid cryptographic foundations. To this end, we propose a natural indistinguishability-based definition stating that any two LP tasks chosen by the adversary cannot be distinguishied by it after the transformation. In Sec. 4 we show that it is impossible to information-theoretically achieve this level of security at least with transformations from a large class that includes all transformations built on currently proposed techniques. We note that this does not yet rule out computational security, and in Sec. 5 we study a number of candidate secure transformations. It turns out that none of our proposals are secure. We generalize the reason why all these transformations fail, and then we *empirically* state another necessary property for secure transformations that excludes this way of failing. In Sec. 6 we see that this property is very strong and constrains the transformed tasks too much.We thus conclude that, barring any radically new ideas related to transforming the LP task to some different task, transformation-based techniques for privacy-preserving LP outsourcing and multi-party LP solving are unusable, and the only currently known feasible method is to use privacy-preserving building blocks in implementing existing LP solving algorithms [20, 16].

2. PRELIMINARIES

Throughout this paper, the upright upper case letters A denote matrices, and the lower case letters with vector signs $\vec{b}$ denote column vectors. Writing two matrices/vectors together without an operator $A\vec{b}$ denotes multiplication, while separating them with a whitespace and putting into parentheses $(A\ \vec{b})$ denotes column augmentation. By augmentation we mean attaching a column $\vec{b}$ to the matrix A from the right. This can be generalized to matrices: $(A\ B)$ denotes a matrix that contains all the columns of A followed by all the columns of B. Row augmentation is defined analogously. Since multiplication and augmentation may be used in the same expression at once, for clarity the augmentation is sometimes also denoted $(A|\vec{b})$, whereas the multiplication operation has higher priority.

2.1 Linear Programming

The canonical form (1) of LP is equivalent to its *standard form*

$$\text{maximize } \vec{c}^{\mathrm{T}}\vec{x}, \text{ subject to } A\vec{x} = \vec{b}, \vec{x} \geq \vec{0} \qquad (2)$$

and to its augmented form ($i \in \{1, \ldots, |\vec{x}|\}$):

$$\text{maximize } x_i, \text{ subject to } A\vec{x} = \vec{b}, \vec{x} \geq \vec{0}\ . \qquad (3)$$

Indeed, the inequalities of the canonical form may be replaced with equalities by introducing slack variables. Each equality may be substituted with two inequalities of opposite directions. Given a linear program in its standard form, the objective function vector $\vec{c}$ may be included into the matrix, and an additional variable represent the corresponding linear combination. The augmented form is an instance of standard form where the i-th entry of $\vec{c}^{\mathrm{T}}$ is 1, and the rest are 0.

A *feasible solution* of a linear program is any vector $x_0 \in \mathbb{R}^n$ that satisfies its constraints. An *optimal solution* of a linear program is any feasible solution that maximizes the value of its objective function. The *feasible region* of a linear program is the set of all its feasible solutions. It is a polyhedron — the intersection of a finite number of hyperplanes and half-spaces. A feasible solution is *basic* if it is located in one of the vertices of that polyhedron.

2.2 Existing Types of Transformations

Let a linear programming task be given in its standard form (2). The main basic transformations from the related works are the following.

Multiplying from the left. The idea of multiplying A and $\vec{b}$ by a random invertible matrix P from the left was first introduced in [9]. Since P is invertible, all the solutions to the system, including the optimal solution, remain the same. This means that the transformation is not harmful for the correctness. However, the feasible region remains revealed.

Multiplying from the right. The idea of multiplying A and $\vec{b}$ by a random invertible matrix Q from the right was also proposed in [9]. This operation hides also the objective function vector $\vec{c}$. Unfortunately it changes the optimal solution if some external constraints of the form $B\vec{x} \geq \vec{b'}$ are present. In this case, the vector $\vec{b'}$ should also be modified according to the transformation, but that in fact reveals all the information about Q. Since in practice linear programs do require such constraints (in general of the form $\vec{x} \geq \vec{0}$), this solution is not sufficient.

Scaling and Permutation. There have been attempts to implement multiplication from the right without affecting the correctness [21], and finally it was noticed in [4] that in order to preserve the inequality $\vec{x} \geq \vec{0}$, the most general type of matrix by which we may multiply from the right is a positive monomial matrix (product of a positive diagonal matrix and a permutation matrix). This results in scaling and permuting the variables.

Shifting. In [8], the initial variable vector $\vec{x}$ is not only scaled, but also shifted. This is done by introducing special slack variables for each shifted variable.

Several of the transformations presented above are special cases of the following transformation for a LP task in the augmented form (3). Generate a random $n \times n$ positive diagonal matrix D and $n \times n$ permutation matrix Q (corresponding to the permutation $\sigma \in S_n$), where $n = |\vec{x}|$. Output $i' = \pi(i)$ and $(A'|\vec{b'}) = \mathsf{RREF}(ADQ|\vec{b})$, where $\mathsf{RREF}(M)$ is the *reduced row-echelon form (RREF)* of the matrix M. We call this transformation the *basic transformation* of a LP task.

Recall that the RREF is an invariant of matrices: we have $\mathsf{RREF}(M_1) = \mathsf{RREF}(M_2)$ for $m \times n$ matrices M_1 and M_2 iff there exists an invertible $m \times m$ matrix P, such that $M_1 = PM_2$. Also, for each $m \times n$ matrix M there exists an invertible $m \times m$ matrix P, such that $\mathsf{RREF}(M) = PM$. Thus the computation of RREF generalizes the multiplication of the system of equations with an invertible matrix, occurring in almost every transformation proposed so far.

On input $i', A', \vec{b'}$, the server finds an optimal solution $\vec{y}_{\mathrm{opt}}$ for the LP task "maximize $y_{i'}$, subject to $A'\vec{y} = \vec{b'}$, $\vec{y} \geq \vec{0}$". Upon learning $\vec{y}_{\mathrm{opt}}$, the client can recover an optimal solution to (3) by $\vec{x}_{\mathrm{opt}} = \vec{y}_{\mathrm{opt}}Q^{-1}D^{-1}$.

Note that the basic transformation does not employ shifting. We have made this choice because the shifting transformation, as used in [8], is easy to undo [15].

3. DESIRED SECURITY DEFINITION

The security definition that has been used in the previous works related to the transformation-based approach is the acceptable security. This notion was first used in [10]. A protocol achieves acceptable security if the only thing that the adversary can do is to reduce all the possible values of the secret data to some domain with the following properties:

1. The number of values in this domain is infinite, or the number of values in this domain is so large that a brute-force attack is computationally infeasible.

2. The range of the domain (the difference between the upper and lower bounds) is acceptable for the application.

Some works provide more detailed analysis [3, 8] that estimates the probability that the adversary guesses some secret value. The leakage quantification analysis [8] is a compositional method for estimating the adversary's ability to make the correct guess when assisted by certain public information. However, even this analysis is still not formal enough and is related to the same acceptable security definition. The informal definitions allow the existence of some attacks that may be yet unknown, but may turn out to be efficient. For example, some vulnerabilities against settings that were assumed to be secure have been found [15]. Additionally, to argue about the security of complex protocols that use privacy-preserving LP transformations as a subprotocol, a more standard security definition for the LP transformation is necessary.

We now give the necessary notions to formally define a problem transformation and its security. Let $\mathfrak{T} \subseteq \{0,1\}^*$ be the set of all possible tasks and $\mathfrak{S} \subseteq \{0,1\}^*$ the set of all possible solutions. For $T \in \mathfrak{T}$ and $S \in \mathfrak{S}$ let $T \vDash S$ denote that S is a solution for T. A *problem transformation* is a pair of functions $\mathcal{F} : \mathfrak{T} \times \{0,1\}^* \to \mathfrak{T} \times \{0,1\}^*$ and $\mathcal{G} : \mathfrak{S} \times \{0,1\}^* \to \mathfrak{S}$. Both $\mathcal{F}$ and $\mathcal{G}$ must work in time polynomial to the length of their first argument. The pair $(\mathcal{F}, \mathcal{G})$ is a *correct problem transformation* if

$$\forall T, r, T', S', s : \\ \big((T', s) = \mathcal{F}(T; r) \ \wedge \ T' \vDash S'\big) \ \Rightarrow \ T \vDash \mathcal{G}(S', s) \ .$$

This implication shows the intended use of $\mathcal{F}$ and $\mathcal{G}$. To transform a task T, the mapping $\mathcal{F}$ uses randomness r, producing a transformed task T' and some state s for the mapping $\mathcal{G}$ that transforms a solution S' to T' back into a solution of the original task T. We write $\mathcal{F}(T)$ for a randomized function that first samples r and then runs $\mathcal{F}(T; r)$.

Note that we have not defined the transformation in the most general manner possible. Namely, we require that the result of transforming a task from the set $\mathfrak{T}$ is again a task from $\mathfrak{T}$. This corresponds to our goal that a transformed linear program is a linear program again.

The transformation $\mathcal{F}$ is intended to hide the important details of a task T. The meaning of hiding has been recently investigated by Bellare et al. [5] in the context of garbling circuits [23]. It is possible that it is unimportant and/or too expensive to hide certain details of T. It is also possible that certain tasks are inherently unsuitable for hiding. In the context of linear programming, we most probably do not want to hide the size of the task, because we would like to avoid padding all tasks to some maximum size. Also, some tasks may be ill-specified and thus unsuitable for transformation. For example, we may require the constraint matrix to have full rank.

Both the public details and suitability for hiding are captured by the notion of *side information function* $\Phi : \mathfrak{T} \to \{0,1\}^*$ that, again, must be polynomial-time computable. When transforming a task T, we do not try to hide the information in $\Phi(T)$. If T should not be transformed at all, then we set $\Phi(T) = T$. We can now state a rather standard, indistinguishability-based definition of privacy. Recall that a function $\alpha : \mathbb{N} \to \mathbb{R}$ is *negligible* if $\forall c \exists m \forall n \geq m : \alpha(n) < 1/n^c$.

Definition 1. *A transformation $(\mathcal{F}, \mathcal{G})$ for $\mathfrak{T}$ is* Φ-private *if the advantage of any probabilistic polynomial-time adversary $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$ is a negligible function of η in the following experiment $\mathbf{Exp}_{\mathcal{A}}^{\mathfrak{T},\Phi}$:*

$$\left[\begin{array}{l} (T_0, T_1, s) \leftarrow \mathcal{A}_1(\eta) \\ \textbf{\textit{if}}\ |T_0| \neq \eta \vee |T_1| \neq \eta \vee \Phi(T_0) \neq \Phi(T_1) \\ \quad \textbf{\textit{return}}\ \bot \\ b \xleftarrow{\$} \{0,1\} \\ (T', \_) \leftarrow \mathcal{F}(T_b) \\ b' \leftarrow \mathcal{A}_2(T', s) \\ \textbf{\textit{return}}\ (b \stackrel{?}{=} b') \end{array}\right.$$

where the advantage of the adversary is $1/2$ less the probability of the experiment returning true.

If $\mathfrak{T}$ is the set of all linear programming tasks "maximize $\vec{c}^{\mathrm{T}}\vec{x}$, subject to $A\vec{x} = \vec{b}$, $\vec{x} \geq \vec{0}$", determined by $A \in \mathbb{R}^{m \times n}$, $\vec{b} \in \mathbb{R}^m$ and $\vec{c} \in \mathbb{R}^n$, then we will in the following take $\Phi(A, \vec{b}, \vec{c}) = (m, n, \vec{x}_{\mathrm{opt}}, \mathsf{bb}(A, \vec{b}))$, where $\vec{x}_{\mathrm{opt}}$ is the unique optimal solution for the linear programming task and $\mathsf{bb}(A, \vec{b})$ is the bounding box for the polyhedron $A\vec{x} = \vec{b}$, $\vec{x} \geq \vec{0}$. If the task is infeasible, unbounded, or has several different optimal solutions, then we consider the task unsuitable for transformation and take $\Phi(A, \vec{b}, \vec{c}) = (A, \vec{b}, \vec{c})$. This choice of Φ is at least as permissive as the privacy goals for previously proposed transformations (note that these goals have often been implicit in the papers where these transformations were presented).

Quite clearly, the transformations described in Sec. 2.2, in particular the basic transformation, do not satisfy this definition. If $\mathcal{F}$ is the basic transformation then the adversary picks two LP tasks T_0 and T_1 with n variables, m equality constraints and the same optimal solution, such that their feasible regions P_0 and P_1 both have the unit hypercube with opposite corners at $(0, \ldots, 0)$ and $(1, \ldots, 1)$ as the bounding box, but a different number of vertices with coordinates $(0, \ldots, 0, 1, 0, \ldots, 0)$. Given the transformed task T', the adversary will find its bounding box (by solving a number of LP tasks with the constraints of T' and either maximizing or minimizing a single variable), scale it to the unit hypercube, and count the vertices with coordinates $(0, \ldots, 0, 1, 0, \ldots, 0)$. This count is not changed by the basic transformation.

4. NO PERFECT SECRECY

A transformation $(\mathcal{F}, \mathcal{G})$ provides *perfect secrecy* (or *information-theoretic secrecy*) if the advantage of any adversary in the experiment described in Def. 1 is zero. It is definitely possible to define a transformation that provides perfect secrecy with respect to this definition. Such a transformation

$\mathcal{F}$ would solve the LP task and then output a randomly selected (or even a constant) LP task that has the same optimal solution. But obviously, we are not looking for such transformations because the goal of the entire outsourcing approach is to make the client not pay the price of solving the original problem.

In this section we prove that any perfectly secure transformation with the properties listed below cannot be computationally much simpler than solving the LP task. W.l.o.g. we assume that both the inputs and the outputs of $\mathcal{F}$ are linear programs in the canonical form (1). The properties are the following:

1. The optimal solution $\vec{y}_{\text{opt}}$ to the transformed linear program $\mathcal{F}(A, \vec{b}, \vec{c}; r)$ only depends on r and $\Phi(A, \vec{b}, \vec{c})$.
2. The mapping $\mathcal{F}(\cdot; r)$ is continuous with respect to the optimal solutions of the initial and transformed problems. This means that for each $\varepsilon > 0$ there exists $\delta > 0$, such that if $(A^\circ, \vec{b^\circ}, \vec{c^\circ})$ and $(A^\bullet, \vec{b^\bullet}, \vec{c^\bullet})$ are two LP tasks with n variables, m constraints and the optimal solutions satisfying $\|\vec{x^\circ}_{\text{opt}} - \vec{x^\bullet}_{\text{opt}}\| \leq \delta$, then the optimal solutions of the transformed tasks $\mathcal{F}(A^\circ, \vec{b^\circ}, \vec{c^\circ}; r)$ and $\mathcal{F}(A^\bullet, \vec{b^\bullet}, \vec{c^\bullet}; r)$ satisfy $\|\vec{y^\circ}_{\text{opt}} - \vec{y^\bullet}_{\text{opt}}\| \leq \varepsilon$.

We find the properties natural; they are satisfied by all transformations proposed so far in the literature and seem to be natural consequences of the proposed transformation techniques. For example, the result of scaling and shifting the polyhedron or permuting its variables depends only on the random matrices by which it is multiplied (the range for the random values may depend on the bounding box). For a transformation satisfying these properties, we show how to turn it into a LP solving algorithm with only little extra computational effort.

According to property 2, there exists a function $\Delta : \mathbb{R}_{>0} \to \mathbb{R}_{>0}$ (where $\mathbb{R}_{>0}$ denotes the set of positive real numbers), mapping ε to δ. W.l.o.g. we may assume that the function Δ is monotone and continuous.

For the simplicity of the exposition, we need a further technical property of $\mathcal{F}$. Let Q_n be the n-dimensional unit hypercube. For a polyhedron P bounded by a number of hyperplanes, let its *roundness* $\text{rd}(P)$ be the minimum distance between a vertex of P and a hyperplane of P that does not contain this vertex. For $\vec{v} \in Q_n$, let $\text{rd}_n^m(\vec{v}; r)$ be the roundness of the polyhedron determined by the constraints of the LP task $\mathcal{F}(A, \vec{b}, \vec{c}; r)$, where $\Phi(A, \vec{b}, \vec{c}) = (m, n, \vec{v}, Q_n)$. Note that due to property 1, the quantity $\text{rd}_n^m(\vec{v}; r)$ is well-defined. The required property of $\mathcal{F}$ is the following:

3. The function $\text{rd}_n^m(\vec{x}; r)$ is continuous (as a function of $\vec{x}$).

We show that the existence of a perfectly secure transformation with listed properties allows us to perform offline precomputations for a fixed dimension n and number of bounding hyperplanes m that afterwards allow us to solve an arbitrarily large proportion of interesting LP tasks of dimension $n-1$ with $m-2$ facets with an effort that is only marginally larger than applying $\mathcal{F}$ and $\mathcal{G}$. Let our interest in these tasks be described by a probability distribution $\mathcal{D}$; the tasks we want to solve are sampled from this distribution. We assume that the probability density function $\mathbf{p}$ of the optimal solutions of tasks sampled according to $\mathcal{D}$ is continuous. We also assume that as side information, we know the bounding boxes of the polyhedra defined by the constraints of these tasks; this assumption holds for large classes of problems occurring in practice. We will actually use an assumption equivalent to the last one due to the ease of scaling and shifting: the bounding boxes of all tasks sampled from $\mathcal{D}$ are Q_{n-1}.

For the precomputation, we first fix the randomness r. We construct a LP task T^{pre} with n variables and m bounding hyperplanes and the objective function selected in such a way that the optimal solution of T^{pre} is $\vec{x}^{\text{pre}}_{\text{opt}} = (1, \ldots, 1)^{\text{T}}$. We perform the transformation – $U^{\text{pre}} = \mathcal{F}(T^{\text{pre}}; r)$ – and solve the resulting task U^{pre}. Let the solution to U^{pre} be $\vec{y}^{\text{pre}}_{\text{opt}}$.

Let $q \in (0, 1)$ be the desired success probability of the LP solving algorithm. The online phase of LP depends on a constant $\varepsilon > 0$ that is selected according to q (but is independent of T^{pre}). Let us first explain the algorithm, and then show that there is a choice of ε that guarantees the success probability at least q. Denote $\delta = \Delta(\varepsilon)$.

Let $T \leftarrow \mathcal{D}$ be a LP task with $n-1$ variables and $m-2$ constraints. Let P be the polyhedron defined by these constraints; let the bounding box of P be Q_{n-1}. To solve T, we subject it to the following transformations.

1. Scale the polyhedron P down, with the scalar multiplier being δ. This corresponds to substituting each variable x_i in each constraint and in the objective function vector by $(1/\delta)x_i$. Let T_0 be the resulting task and P_0 the polyhedron defined by the scaled constraints. The bounding box of P_0 is the hypercube in $n-1$ dimensions with the side length δ.
2. Add the n-th dimension and build an oblique hyperprism from P_0, with the bases at hyperplanes $x_n = 0$ and $x_n = 1$, and the bounding box of the hyperprism being equal to Q_n. This modifies the system of constraints as follows:
   - Two new constraints, $x_n \geq 0$ and $x_n \leq 1$, are added corresponding to the bases of the hyperprism.
   - Each existing constraint $\sum_{i=1}^{n-1} a_i x_i \leq b$ is replaced with $\sum_{i=1}^{n-1} a_i(x_i + (1-\delta)x_n) \leq b$, corresponding to the sides of the hyperprism.

The result of these two transformations is shown in Fig. 1.

Let T' be the LP task where the resulting hyperprism P' is the set of feasible solutions, and where the objective function of T' is $\sum_{i=1}^{n-1} c_i x_i + C x_n$, where $\vec{c} = (c_1, \ldots, c_{n-1})$ is the objective function vector of T and C is a large constant. Hence the optimal solution $\vec{x'}_{\text{opt}}$ of T' is located on the base P_1 of the hyperprism, otherwise being "at the same vertex" as the optimal solution $\vec{x}_{\text{opt}}$ to T. In particular, $\vec{x'}_{\text{opt}}$ can easily be transformed back to $\vec{x}_{\text{opt}}$. Also note that $\|\vec{x'}_{\text{opt}} - \vec{x}^{\text{pre}}_{\text{opt}}\| \leq \delta$.

Let $(U', s) = \mathcal{F}(T'; r)$. If we could find the optimal solution $\vec{y'}_{\text{opt}}$ to U', then we could compute $\vec{x'}_{\text{opt}} = \mathcal{G}(\vec{y'}_{\text{opt}}, s)$ and find $\vec{x}_{\text{opt}}$ from it. Note that $\|\vec{y'}_{\text{opt}} - \vec{y}^{\text{pre}}_{\text{opt}}\| \leq \varepsilon$. Let $\bar{P}$ be the polyhedron defined by the constraints of U'. The point $\vec{y'}_{\text{opt}}$ is determined as the unique intersection point of a number of hyperplanes bounding $\bar{P}$. All these hyperplanes are at distance of at most ε from the point $\vec{y}^{\text{pre}}_{\text{opt}}$.

We now have to explain the choice of ε. We have selected it so small, that $\Pr[\text{rd}(\bar{P}) > 2\varepsilon] \geq q$, where the probability

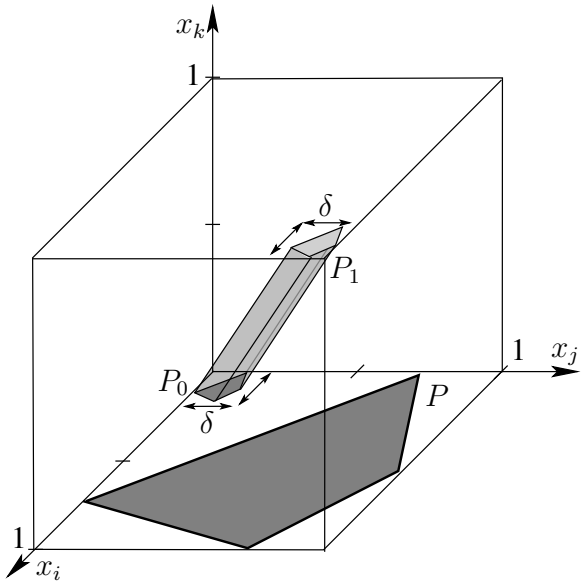


Figure 1: Results of preparatory transformations to task T

is taken over the choice of T (sampled from $\mathcal{D}$). Thanks to this choice, the following claim holds with probability at least q:

- If a hyperplane bounding $\bar{P}$ is at distance of at most ε from the point $\vec{y}^{\,\mathrm{pre}}_{\mathrm{opt}}$, then this hyperplane contains $\vec{y'}_{\mathrm{opt}}$.

Hence, to find $\vec{y'}_{\mathrm{opt}}$, we measure how far each hyperplane bounding $\bar{P}$ is from the point $\vec{y}^{\,\mathrm{pre}}_{\mathrm{opt}}$, and find the intersection point of these hyperplanes where the distance is at most ε. This amounts to solving a system of linear equations, which is much simpler than solving LP.

Existence of suitable ε.

Let $\mathsf{base}_\varepsilon \subseteq \mathbb{R}^n$ be the set of points $\vec{x}$, where $x_n = 1$ and $1 - \Delta(\varepsilon) \leq x_i \leq 1$ for all $i \in \{1, \ldots, n-1\}$. Let $\mathbf{p}'_\varepsilon$ be the probability density function of $\vec{x'}_{\mathrm{opt}} \in \mathsf{base}_\varepsilon$, where $\vec{x'}_{\mathrm{opt}}$ is the optimal solution to the LP task T', defined as before. The function $\mathbf{p}'_\varepsilon$ is continuous because it has been obtained via a continuous transformation from the continuous function $\mathbf{p}$. For a Boolean value b, let $[b]$ be 1 if b is true, and 0 if b is false. The probability of $\mathrm{rd}(\bar{P})$ being larger than 2ε is

$$PR(\varepsilon) = \int_{\mathsf{base}_\varepsilon} \mathbf{p}'_\varepsilon(\vec{x}) \cdot [\mathrm{rd}_n^m(\vec{x}; r) > 2\varepsilon]\, d\vec{x} \ .$$

We see that PR is a continuous function, because it has been constructed from continuous components in a manner that preserves continuity. As $PR(0) = 1$, there must exist some $\varepsilon > 0$, such that $PR(\varepsilon) \geq q$.

5. SOME INSECURE TRANSFORMATIONS

We have seen that perfect security is impractical. This does not yet rule out the existence of transformations with weaker security (computational), because we had to access the private randomness in order to obtain the optimal solution for the LP task.

In this section, we propose certain transformations, observe why these do not satisfy our privacy , and *empirically* derive a further necessary condition for the security of the transformation. In our opinion, this condition is a very natural one. The transformations satisfying this condition are then further explored in Sec. 6.

When aiming for computational security, the privacy has to follow from a plausible computational hardness assumption. In LP, we work with real numbers, hence the assumptions about discrete structures (e.g. the RSA assumption, various Diffie-Hellman assumptions) etc. are not directly applicable. Any assumption we base the security of the transformation on, will be a novel one. To reduce the novelty, we may take some known assumption on processes similar to transforming LP tasks producing indistinguishable distributions, and change it by stating that instead of elements of finite fields or groups, there are real numbers. We must be careful, though, because real numbers have extra structure (order) that finite fields lack.

The *Strong Secret Hiding Assumption (SSHA)* [1] may be suitable for such change. We refer to the original paper for its precise statement on the indistinguishability of certain distributions over matrices over finite fields. We only remark that if SSHA could be extended to matrices over $\mathbb{R}$, then it would allow us to hide a LP task by adding many extra columns (and rows) to the matrix A before applying the basic transformation. We must be careful, though, to not change the optimal solution of the task.

Let us now study different ways of performing this augmentation of A. Starting from the task (3), we will change it to "maximize w_j, subject to $A'\vec{w} = \vec{b'}$, $\vec{w} \geq 0$", to which we apply the basic transformation.

New variables with constraints.

We add a new variable z to the system, and fix its value with respect to existing variables: we also add a constraint $z = \vec{d}^{\mathrm{T}}\vec{x} + r$ to the system, where $\vec{d} \in \mathbb{R}^n$ and $r \in \mathbb{R}$. In this manner, we can add any number of variables $z_1, \ldots, z_k$ with constraints $z_j = \vec{d_j}^{\mathrm{T}}\vec{x} + r_j$. This corresponds to changing the task to "maximize x_i, subject to $\left(\begin{smallmatrix} A & 0 \\ D & -I \end{smallmatrix}\right) = \left(\begin{smallmatrix} \vec{b} \\ -\vec{r} \end{smallmatrix}\right)$, $\vec{x}, \vec{z} \geq \vec{0}$", where $D = (\vec{d_1} \ \cdots \ \vec{d_k})^{\mathrm{T}}$ and $\vec{r} = (r_1 \ \cdots \ r_k)^{\mathrm{T}}$.

The inequalities $\vec{z} \geq \vec{0}$ set new constraints also for the original variables $\vec{x}$. These must not change the optimal solution of the LP task. Also, we want to define the transformation in a manner that does not require it to start solving the original LP task. Hence we want that the newly introduced inequalities $z_j = \vec{d_j}^{\mathrm{T}}\vec{x} + r_j \geq 0$ are implied by the original inequalities $\vec{x} \geq \vec{0}$. This is possible only if all entries of $\vec{d_j}$ are non-negative and $r_j \geq 0$.

Why do we add the inequalities $\vec{z} \geq \vec{0}$ to the transformed LP task, instead of allowing these variables to be free? If we did so, we also would have to leave free the variables in $\vec{y}$ (after performing the basic transformation) corresponding to variables in $\vec{z}$. This means that in the transformed task, we can tell which of the variables $\vec{y}$ correspond to variables in $\vec{x}$ and which correspond to variables in $\vec{z}$. If we know which variables stem from $\vec{z}$, we can use Gaussian elimination to get rid of them. This leaves us with just the basic transformation being applied to the original task, which, as we saw before, is not sufficiently secure.

If we insist that all entries of D and $\vec{r}$ are non-negative, the variables from $\vec{x}$ and the variables from $\vec{z}$ can still be distinguished after the basic transformation. A variable z_j with the constraint $z_j = \vec{d_j}^{\mathrm{T}}\vec{x} + r_j$ can only be 0 if r_j is 0 and $x_k = 0$ for all $k \in \{1, \ldots, n\}$ where $\vec{d}_{jk} \neq 0$. If the actual transformation is such, that the values r_j are likely to be non-zero, then the adversary of Def. 1 has to pick the to-be-distinguished LP tasks so, that for each variable, there is a feasible solution where this variable equals 0. To distinguish variables from $\vec{x}$ and the variables from $\vec{z}$, the adversary minimizes each variable one by one (this amounts to solving a LP) and sees which ones have the minimal value 0. If the addends r_j are 0, but the matrix D is sufficiently dense then the variables from $\vec{z}$ have "smaller probability" of being 0 than the variables from $\vec{x}$. In this case, the adversary samples random vertices of the polyhedron $A'\vec{y} = \vec{b'}$ (by solving LP tasks with randomly chosen optimization directions, i.e randomly chosen objective functions) and records which variables are 0 with larger or smaller frequency. If the matrix D is sparse then the adversary picks the to-be-distinguished LP tasks so, that there are no affine relationships among small sets of initial variables $\vec{x}$. The transformation, however, introduces affine relationships among a variable from $\vec{z}$ and small sets of variables from $\vec{x}$. Such relationships are straightforwardly detected [15] and based on them, variables from $\vec{x}$ and variables from $\vec{z}$ can be distinguished.

Hence this way of augmenting the LP task is not sufficient to achieve privacy, at least when applied alone.

New variables without constraints.

We add k new variables $\vec{z}$ to the system (3) with m constraints and n variables, without adding new constraints. In this case, the original system of equations $A\vec{x} = \vec{b}$ must be modified to include $\vec{z}$, otherwise the variables corresponding to the ones in $\vec{z}$ can be recognized after the basic transformation and easily removed from the system.

We replace the original system of equations with $A\vec{x} + C\vec{z} = \vec{b}$, where $C = AV$, where $V \in \mathbb{R}^{n \times k}$ is a random matrix with non-negative entries. We also add the constraints $\vec{z} \geq \vec{0}$. Given an optimal solution $(\vec{x}_0, \vec{z}_0)$ of the modified task, we recover the optimal solution $\vec{x}_{\mathrm{opt}}$ of the original task as $\vec{x}_{\mathrm{opt}} = \vec{x}_0 + V\vec{z}_0$.

It is easy to see that $\vec{x'} = \vec{x}_0 + V\vec{z}_0$ is indeed the optimal solution to the original task. First, it satisfies the system of equations $A\vec{x} = \vec{b}$. Second, the value of the objective function x_i for $\vec{x'}$ is at least as good as its value in $\vec{x}_{\mathrm{opt}}$, because $(\vec{x}_{\mathrm{opt}}, \vec{0})$ is one of the solutions of the modified task and the i-th component of the vector $V\vec{z}_0$ is non-negative. Finally, all components of $\vec{x'}$ are non-negative because both $\vec{x}_0$ and $V\vec{z}_0$ are non-negative.

If we could distinguish the variables in $\vec{x}$ from the variables in $\vec{z}$ after the basic transformation, then we could remove the variables in $\vec{z}$ by setting them equal to 0, and obtain the original LP task with basic transformation. Unfortunately, we can indeed distinguish them. There are no upper bounds for variables in $\vec{z}$. If the adversary picks the to-be-distinguished LP tasks so, that their feasible solution sets are bounded, then it has to maximize each variable one by one to distinguish variables in $\vec{x}$ from those in $\vec{z}$.

The two augmentations could both be applied to the original LP task before the basic transformation. It is easy to see that the order of application does not matter here. Unfortunately, the application of both of them does not increase the security of the transformation — the variables introduced by both augmentations can still be located and removed.

Splitting the variables.

In previous transformations, each variable of the original LP task gave us one variable in the transformed task, while the transformation introduced new variables. We can also consider transformations where each variable x_j in the original task gives several variables $z_{j1}, \ldots, z_{jk}$ of the transformed task. In this case, the transformation picks non-negative $r_{j1}, \ldots, r_{jk}$ and replaces x_j with $r_{j1}z_{j1} + \cdots + r_{jk}z_{jk}$ in all equations of the original task (3). This replacement can be made for some, or all variables in the original system. After this replacement, we again apply the basic transformation to the system.

Unfortunately, the splitting of the variables can be undone, at least partially, even after the basic transformation, even if other transformations described above have also been applied. The adversary will pick the original tasks T_0, T_1 so that the set of feasible solutions is bounded. Let x_j be one of the original variables, with the upper bound $x_j^{\max}$. After the replacement, the variable z_{jt} is upper-bounded by $z_{jt}^{\max} = x_j^{\max}/r_{jt}$ (recall that the upper bounds can easily be found). A pair of variables z_{jt}, z_{ju} satisfies the inequality $r_{jt}z_{jt} + r_{ju}z_{ju} \leq x_j^{\max}$ or $z_{ju}^{\max} \cdot z_{jt} + z_{jt}^{\max} \cdot z_{ju} \leq z_{jt}^{\max} z_{ju}^{\max}$.

The basic transformation permutes the variables. To undo the splitting of variables, we have to detect whether two variables y_{j1} and y_{j2} could have resulted from the splitting of the same variable x_j. We just saw that a necessary condition for this is, that all feasible solutions of the transformed task satisfy the condition $y_{j2}^{\max} \cdot y_{j1} + y_{j1}^{\max} \cdot y_{j2} \leq y_{j1}^{\max} y_{j2}^{\max}$. In other words, this inequality must be redundant with respect to the constraints of the transformed task. The redundancy of an inequality constraint is easy to check (it corresponds to a LP task).

While this analysis may not be sufficient to completely identify which variables in $\vec{y}$ correspond to the same variable in $\vec{x}$ (we may get false positives due to certain pairs of original variables x_{j_1} and x_{j_2} also satisfying the inequality $x_{j_2}^{\max} \cdot x_{j_1} + x_{j_1}^{\max} \cdot x_{j_2} \leq x_{j_1}^{\max} x_{j_2}^{\max}$), it is sufficient to distinguish two LP tasks selected by the adversary.

A further condition.

We see that all our attempts so far have failed because in the transformed task, different variables had different roles. These roles could be determined and the variables eliminated. Thus we set an extra requirement that in the transformed task (in the polyhedron corresponding to this task), all variables "look the same". Besides the variables, we want the same condition to hold for (small) sets of variables, as the failure of our last attempt in augmenting A showed.

We need the following notions to formally define our requirement. Let $\vec{e}_i^{\,k} = (0, \ldots, 0, 1, 0, \ldots, 0)$ be the i-th *unit vector* in the k-dimensional space $\mathbb{R}^k$ (the length of $\vec{e}_i^{\,k}$ is k and the only 1 is on i-th position). For $I \subseteq \mathbb{N}$ and $i \in I$ let $\mathsf{idx}_I\, i$ be the *index* of i in the ordered set I, meaning that I has exactly $\mathsf{idx}_I\, i$ elements less than or equal to i. For $I \subseteq \{1, \ldots, k\}$, $|I| = n$ let $\pi_I^k : \mathbb{R}^k \to \mathbb{R}^n$ be the *projection* to dimensions in I. It is a linear mapping defined by $\pi_I^k(\vec{e}_i^{\,k}) = \vec{e}^{\,n}_{\mathsf{idx}_I\, i}$ if $i \in I$, and $\pi_I^k(\vec{e}_i^{\,k}) = 0$ otherwise. For a permutation $\sigma \in S_n$ let $\hat{\sigma} : \mathbb{R}^n \to \mathbb{R}^n$ be the permutation of

dimensions given by σ, i.e. $\hat{\sigma}$ is the linear mapping defined by $\hat{\sigma}(\vec{e}_i^n) = \vec{e}_{\sigma(i)}^n$.

DEFINITION 2. *Let $t \in \mathbb{N}$. A set of points $X \subseteq \mathbb{R}^k$ is* t-symmetric, *if for any $I, I' \subseteq \{1, \ldots, k\}$, $|I| = |I'| = t$, and $\sigma \in S_t$ we have $\pi_I^k(X) = \hat{\sigma}(\pi_{I'}^k(X))$.*

There is also a computational analogue to this definition.

DEFINITION 3. *Let $t, k : \mathbb{N} \to \mathbb{N}$. A family of probability distributions over sets of points $\{\mathcal{X}_n\}_{n\in\mathbb{N}}$, where each element $X_n \in \mathrm{supp}(\mathcal{X}_n) \subseteq \mathbb{R}^{k(n)}$ has a polynomial-size description, is* computationally t-symmetric, *if for any probabilistic polynomial-time (in n) algorithm $\mathcal{A}$, the following probability is negligible:*

$$\Pr[x \in \pi_I^{k(n)}(X) \backslash \hat{\sigma}(\pi_{I'}^{k(n)}(X)) \mid X \leftarrow \mathcal{X}_n, (x, I, I', \sigma) \leftarrow \mathcal{A}(X)],$$

where $x \in \mathbb{R}^{t(n)}$, $I, I' \subseteq \{1, \ldots, k(n)\}$, $|I| = |I'| = t(n)$, $\sigma \in S_{t(n)}$.

The previous definition says that computationally, t-symmetry is broken if an efficient algorithm can find two projections that are different, as well as a certificate of the non-emptiness of their difference. We want the transformation of a LP task be such, that the possible set of results of the transformation is computationally t-symmetric for small values of t, where the asymmetry could be exploited for classifying the variables in the transformed LP task. In particular, we want the transformed LP task to be computationally 1-symmetric (hence the bounding box of the transformed task must be a hypercube) and 2-symmetric.

6. 2-SYMMETRIC TRANSFORMATIONS

Let us now consider transformations that are 2-symmetric and see what properties of the feasible regions of transformed tasks this implies. From now on, let the constraints of the transformed LP task be $A\vec{x} = \vec{b}$, $\vec{x} \geq \vec{0}$.

6.1 A computable property of polyhedra

Consider the polyhedron P determined by $A\vec{x} = \vec{b}$, $\vec{x} \geq \vec{0}$, with m equality constraints and n variables. Let $i, j \in \{1, \ldots, n\}$ be two coordinates, such that $x_i = x_j = 0$ does not contradict the constraints, and consider the projection of P to the (x_i, x_j)-plane. Assume the projection does not equal the whole first quadrant of the plane. Also assume that it is not a point or a line (segment). In this case, the projection is a convex, possibly unbounded polygon. Let O_1 be the vertex of the polygon at the coordinates $(0, 0)$. Let O_2 be the next vertex of the polygon, at the coordinates $(c, 0)$. It is possible that O_2 coincides with O_1; this happens if $x_j = 0$ implies $x_i = 0$. Let O_3 be the next vertex of the polygon after O_2. Let $\alpha_{ij}^P \in \mathbb{R}$ be such, that the side O_2O_3 lies on the line $x_i + \alpha_{ij}^P x_j = c$.

LEMMA 1. *There exists a polynomial-time algorithm that on input A, $\vec{b}$, i and j, satisfying the conditions above, computes α_{ij}^P (to an arbitrary level of precision).*

PROOF. The algorithm works as follows. It will first find the coordinate c by solving the LP task "maximize x_i, subject to $\vec{x} \in P$, $x_j = 0$".

Let $D(\vec{x}, \gamma)$ denote the direction $(\cos\gamma)\cdot x_i + (\sin\gamma)\cdot x_j$. Using binary search, the algorithm will then find $\gamma \in [-\frac{\pi}{2}, \frac{\pi}{2}]$, such that the optimal solution for the LP task "maximize $D(\vec{x}, \gamma - \varepsilon)$ subject to $\vec{x} \in P$" is in a point $\vec{x'}$ with $x'_i = c$ and $x'_j = 0$, while the optimal solution for the LP task "maximize $D(\vec{x}, \gamma + \varepsilon)$, subject to $\vec{x} \in P$" is in a point $\vec{x''}$ with $(x''_i, x''_j) \neq (c, 0)$. Here $\varepsilon > 0$ is an arbitrarily small (depending on the required precision) angle. Fig. 2 depicts all these quantities.

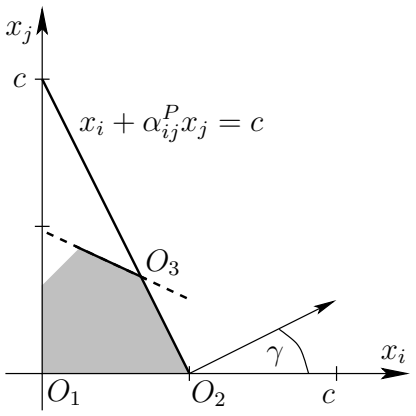


Figure 2: Finding the value of α_{ij}^P

On (x_i, x_j)-plane, the direction $D(\vec{x}, \gamma)$ is (almost) perpendicular to the line $x_i + \alpha_{ij}^P x_j = c$. Hence $\alpha_{ij}^P \approx \tan\gamma$. □

The existence of this algorithm shows that if a transformation producing the polyhedron $P \subseteq \mathbb{R}^n$ is computationally 2-symmetric, then the value α_{ij}^P must be the same for all coordinate pairs i, j. Let us denote it by α.

6.2 Scarcity

We show that any polyhedron whose α_{ij}^P is the same for all coordinate pairs i, j is a set of the form

$$\{(x_1, \ldots, x_n) \mid x_1 + \ldots + x_n \leq c\} \tag{4}$$

for some $c \geq 0$ ($c \in \mathbb{R}$) or $c = \infty$.

This result implies that any computationally 2-symmetric transformation $(\mathcal{F}, \mathcal{G})$ can be easily turned to a LP solving algorithm. Given a LP task, the solver will first apply $\mathcal{F}$ to it, resulting in the polyhedron (4) and an objective function. This polyhedron has at most $n+1$ vertices and the algorithm finds the optimal one by applying the objective function to each of them. The value c can also be straightforwardly found by adding the constraints $x_1 = \ldots = x_{n-1} = 0$ and then finding the maximum value of x_n (either by substituting 0 to $x_1, \ldots, x_{n-1}$, or by binary search).

Let the transformed polyhedron P be defined by $A\vec{x} = \vec{b}$, $\vec{x} \geq \vec{0}$, where A is a non-singular $m \times n$ matrix for $m \leq n-1$. If $m = n-1$, then P is one-dimensional, hence it has at most 2 vertices which can be easily found and $(\mathcal{F}, \mathcal{G})$ can again be turned into a LP solving algorithm. Thus we let $m \leq n-2$.

If $m \leq n-2$ and P contains at least one basic feasible solution, then at least two variables in this solution are 0. The reason is that basic solutions occur only on the intersections of constraining hyperplanes, and the system $A\vec{x} = \vec{b}$ provides at most m of them, so at least $n - m$ come from $x_j \geq 0$.

If there are two variables that can simultaneously have the value 0 in P, then *any* pair of two variables must have the

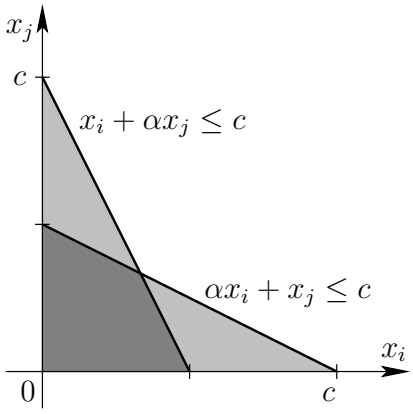


Figure 3: Symmetry property of a projection to (x_i, x_j)

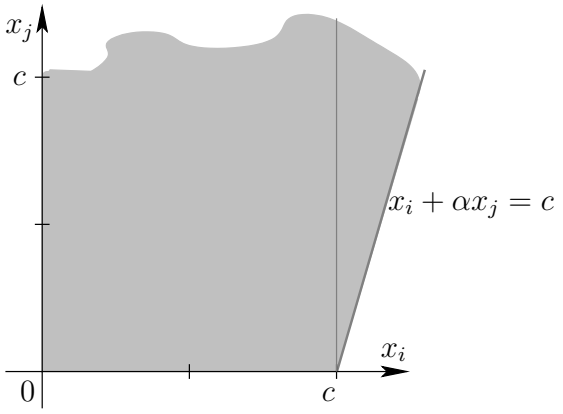


Figure 4: The case $\alpha \le 0$

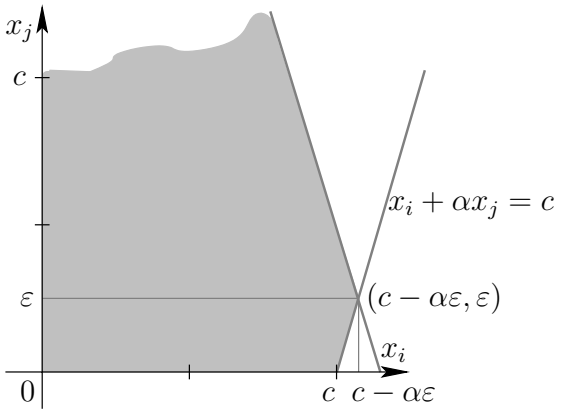


Figure 5: A point that exists on the line $x_i + \alpha x_j = c$ for $\alpha \le 0$

same property, otherwise the (computational) 2-symmetry would be broken. For arbitrary i, j, consider the projection of P to the dimensions x_i and x_j. Since P is a convex polyhedron, this projection is a convex polygon. As discussed before, it contains the point $(0, 0)$.

Let O_2, O_3, c, $\alpha_{ij}^P = \alpha$ and $\gamma = \arctan\alpha$ be defined as in Sec. 6.1. One of the sides of the polygon is located on the line $x_i + \alpha x_j = c$. By symmetry, there also exists a side of the polygon that lies on the line $x_j + \alpha x_i = c$. Due to the convexity of the polygon, $\alpha \le 1$, otherwise these lines pass through the interior of the convex hull of the points $(0, 0)$, $(c, 0)$ and $(0, c)$ that are contained in the polygon.

Since all projections onto any pair of variables (x_i, x_j) should have exactly the same angle γ and therefore the same $\tan\gamma = \alpha$, and exactly the same distance from the origin point c, a system of $2 \cdot \binom{n}{2}$ inequalities of the form $x_i + \alpha x_j \le c$ is implied by the constraints defining the polyhedron P. Here $\binom{n}{2}$ is the number of possible pair choices. This number is multiplied by 2 since the inequality $x_i + \alpha x_j \le c$ implies existence of the inequality $\alpha x_i + x_j \le c$ due to the symmetry requirement, as shown in Fig. 3. Due to convexity of the projection, any valuation (v_i, v_j) of x_i and x_j such that $v_i + v_j \le c$ must be possible.

Given n variables, any inequality $x_i + \alpha x_j \le c$ comes from some equation of the form $a_1 x_1 + a_2 x_2 + \ldots + x_i + \ldots + \alpha x_j + \ldots + a_n x_n = c$ implied by the constraints defining P, where $a_k \ge 0$ for any k, and $a_\ell > 0$ for some ℓ (the variable x_ℓ acts as a slack variable for the inequality). In total, we get $2 \cdot \binom{n}{2}$ equations that all follow from these constraints:

$$\begin{cases} \alpha x_1 + x_2 + a_{123}x_3 + \ldots + a_{12(n-1)}x_{n-1} + a_{12n}x_n = c \\ x_1 + \alpha x_2 + a_{213}x_3 + \ldots + a_{21(n-1)}x_{n-1} + a_{21n}x_n = c \\ a_{231}x_1 + \alpha x_2 + x_3 + \ldots + a_{23(n-1)}x_{n-1} + a_{23n}x_n = c \\ a_{321}x_1 + x_2 + \alpha x_3 + \ldots + a_{32(n-1)}x_{n-1} + a_{32n}x_n = c \\ \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots \\ a_{n(n-1)1}x_1 + a_{n(n-1)2}x_2 + \ldots + \alpha x_{n-1} + x_n = c \\ a_{(n-1)n1}x_1 + a_{(n-1)n2}x_2 + \ldots + x_{n-1} + \alpha x_n = c \end{cases} \tag{5}$$

where $a_{ijk} \ge 0$ for all $i, j, k \in \{1, \ldots, n\}$, and each equation contains at least one strictly positive a_{ijk}. We will show that these equations imply $x_1 + \ldots + x_n = c$. Consider the possible values of α.

1. **The case $\alpha \le 0$.** The side of the polygon represented by the line $x_i + \alpha x_j = c$ is either parallel to the x_j axis, or is tilted in the direction opposite from the x_j axis. This is illustrated by Fig. 4.

   Consider a point $v = (v_1, \ldots, v_n) \in P$ where $(v_i, v_j) = (c - \alpha\varepsilon, \varepsilon)$ for some $\varepsilon > 0$. Depending on ε, this can be any point that is located on the side on the line $x_i + \alpha x_j = c$. There definitely exist points on this side, otherwise that side would not be present on the projection at all. A possible location of such a point is shown in Fig. 5. Consider the equation $E \equiv a_{ij1}x_1 + \ldots + x_i + \ldots + \alpha x_j + \ldots + a_{ijn}x_n = c$ from the system (5). Since each equation represents an inequality, there exists some $a_k = a_{ijk} > 0$ that corresponds to some variable x_k, $k \ne i$, $k \ne j$ (x_k acts as a slack variable).

   (a) **The case $\alpha = 0$.** Due to the symmetry of the projections, there must exist a point $v' = (v'_1, \ldots, v'_n) \in P$ such that $(v'_i, v'_k) = (c - \alpha\varepsilon, \varepsilon) = (c, \varepsilon)$. But the point v' cannot possibly satisfy the equation E because already $v'_i + a_k v'_k = c + a_k\varepsilon > c$ and all other coefficients in E are non-negative.

   (b) **The case $\alpha < 0$.** We have $v_i + \alpha v_j = c - \alpha\varepsilon + \alpha\varepsilon = c$. The point v must satisfy the equation E. Hence $v_k = 0$, because $a_k > 0$. Now we have $(v_i, v_k) =$

$(c - \alpha\varepsilon, 0)$. The point v must also satisfy other equations of the system (5), including $a_{ik1}x_1 + \ldots + x_i + \ldots + \alpha x_k + \ldots + a_{ikn}x_n = c$. We have $v_i + \alpha v_k = c - \alpha\varepsilon + 0 > c$, hence this equation is violated.

Thus the only possible case is $\alpha > 0$.

2. **The case** $\alpha > 0$. If $c = 0$, then the only possible solution for the system (5) would be $x_1 = \ldots = x_n = 0$, since there are no negative entries at all. Consider the case $c > 0$.

   Let $v = (v_1, \ldots, v_n) \in P$ be a point where $v_i = c$. For any $j \neq i$, consider the equation $a_{ij1}x_1 + \ldots + x_i + \ldots + \alpha x_j + \ldots + a_{ijn}x_n = c$. As $v_i = c$, the left hand side of this equation, when applied to v, will be at least c. As it cannot be larger, we must have $\alpha v_j = 0$. This implies $v_j = 0$, because $\alpha > 0$. As j was arbitrary, we obtain $v = (v_1, \ldots, v_{i-1}, v_i, v_{i+1}, \ldots, v_n) = (0, \ldots, 0, c, 0, \ldots, 0)$.

   If the system (5) contains any equation where the coefficient of x_i is $a \neq 1$, then the point v cannot satisfy this equation, because $av_i \neq v_i = c$, and the other components of v cannot affect the left hand side of the equation since they are all 0. We get that the coefficient of x_i should be 1 in each equation.

   In the same way, we get that the coefficients of all the variables in all equations should be 1. This means that the only equation that remains is $x_1 + \ldots + x_n = c$.

Another thing that we would like to show is that if the equation system defined by P contains a constraint $x_1 + \ldots + x_n = c$ for some $c > 0$, then it is not allowed to have any other constraints. Suppose that the system contains the following two equations:

$$x_1 + \ldots + x_n = c \quad (6)$$

$$a_1x_1 + \ldots + a_nx_n = b \quad (7)$$

where $a_i, b \in \mathbb{R}$. We will show that the equation (7) can be at most a multiple of the equation (6), representing the same constraint.

Without loss of generality, let $a_1 = \min_i a_i$, $a_2 = \max_i a_i$. We may assume that $a_2 > a_1$ since if it was the case $a_2 = a_1$, then all the a_i would be equal, and the only possible way to avoid contradiction with (6) would be to assign $b = a_ic$, making the (7) a multiple of (6).

Multiplying (6) by a_1 and subtracting the result from (7), we get

$$(a_2 - a_1)x_2 + \ldots + (a_n - a_1)x_n = b - a_1c$$

Since $a_2 \neq a_1$, we may express the variable x_2 in terms of other variables:

$$x_2 = \frac{b - a_1c - \sum_{i=3}^{n}(a_i - a_1)x_i}{a_2 - a_1}$$

We know that the only allowed valuations of x_i are positive.

- Since $x_2 \geq 0$ and $a_2 > a_1$, the constraints defining the polyhedron P imply

$$b - a_1c \geq \sum_{i=3}^{n}(a_i - a_1)x_i$$

- From (6), $x_1 = c - \sum_{i=2}^{n} x_i$. We get

$$\begin{aligned} x_1 &= c - \frac{b - a_1c - \sum_{i=3}^{n}(a_i - a_1)x_i}{a_2 - a_1} - \sum_{i=3}^{n} x_i \\ &= \frac{a_2c - b - \sum_{i=3}^{n}(a_2 - a_i)x_i}{a_2 - a_1} \end{aligned}$$

- Since $x_1 \geq 0$ and $a_2 > a_1$, the constraints defining the polyhedron P imply

$$a_2c - b \geq \sum_{i=3}^{n}(a_2 - a_i)x_i$$

Recall that any variable must be allowed to take any value in the span $[0, c]$. Consider the valuation where $x_i = c$ for some i. From (6) and the requirement $\vec{x} \geq 0$, we get that $\forall j \neq i: \; x_j = 0$.

$$\begin{cases} b - a_1c \geq \sum_{i=3}^{n}(a_i - a_1)x_i \\ a_2c - b \geq \sum_{i=3}^{n}(a_2 - a_i)x_i \end{cases} \implies \begin{cases} b - a_1c \geq a_ic - a_1c \\ a_2c - b \geq a_2c - a_ic \end{cases}$$

$$\implies \begin{cases} b \geq a_ic \\ -b \geq -a_ic \end{cases} \implies a_i = \frac{b}{c} \; .$$

Similarly, we can show that for any $i \neq 1, 2$ we have $a_i = \frac{b}{c}$. What about a_1 and a_2?

From (6), we get that $x_3 + \ldots + x_n = c - x_1 - x_2$.

$$\begin{aligned} a_1x_1 + \ldots + a_nx_n &= b \\ a_1x_1 + a_2x_2 + \frac{b}{c}x_3 + \ldots + \frac{b}{c}x_n &= b \\ \left(a_1 - \frac{b}{c}\right)x_1 + \left(a_2 - \frac{b}{c}\right)x_2 &= 0 \end{aligned}$$

If either $\left(a_1 - \frac{b}{c}\right) \neq 0$ or $\left(a_2 - \frac{b}{c}\right) \neq 0$, then one variable may be expressed in terms of the other one, meaning that the projection to (x_1, x_2) can be only a line segment. This would mean that the entire polygon P is actually a line segment. We have explored this case before. But if this is not the case, then $a_1 = a_2 = \frac{b}{c}$.

We have obtained an equation $\frac{b}{c}x_1 + \ldots + \frac{b}{c}x_n = b$, and multiplying both sides by $\frac{c}{b}$ we get the same equation (6).

7. CONCLUSIONS

We have shown that the current approaches towards privacy-preserving outsourcing or multiparty linear programming are unlikely to be successful. Success in this direction requires some radically new ideas in transforming polyhedra and/or in cryptographic foundations violating the rather generous assumptions we have made in this paper. Alternatively, it may be fruitful to optimize privacy-preserving implementations of LP solving algorithms in order to have universal privacy-preserving optimization methods for large classes of tasks.

8. ACKNOWLEDGEMENTS

This work was supported by the European Regional Development Fund through the Estonian Center of Excellence in Computer Science, EXCS, and through the Software Technologies and Applications Competence Centre, STACC. It has also received funding from the European Union Seventh Framework Programme (FP7/2007-2013) under grant agreement no. 284731.

9. REFERENCES

[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] Mikhail J. Atallah and Jiangtao Li. Secure outsourcing of sequence comparisons. *Int. J. Inf. Sec.*, 4(4):277–287, 2005.

[3] Alice Bednarz. *Methods for two-party privacy-preserving linear programming*. PhD thesis, University of Adelaide, 2012.

[4] Alice Bednarz, Nigel Bean, and Matthew Roughan. Hiccups on the road to privacy-preserving linear programming. In *Proceedings of the 8th ACM workshop on Privacy in the electronic society*, WPES '09, pages 117–120, New York, NY, USA, 2009. ACM.

[5] Mihir Bellare, Viet Tung Hoang, and Phillip Rogaway. Foundations of garbled circuits. In Ting Yu, George Danezis, and Virgil D. Gligor, editors, *ACM Conference on Computer and Communications Security*, pages 784–796. ACM, 2012.

[6] Xiaofeng Chen, Jin Li, Jianfeng Ma, Qiang Tang, and Wenjing Lou. New algorithms for secure outsourcing of modular exponentiations. In Sara Foresti, Moti Yung, and Fabio Martinelli, editors, *ESORICS*, volume 7459 of *Lecture Notes in Computer Science*, pages 541–556. Springer, 2012.

[7] Yao Chen and Radu Sion. On securing untrusted clouds with cryptography. *IEEE Data Eng. Bull.*, 35(4):9–20, 2012.

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

[9] Wenliang Du. *A Study Of Several Specific Secure Two-Party Computation Problems*. PhD thesis, Purdue University, 2001.

[10] Wenliang Du and Zhijun Zhan. A practical approach to solve secure multi-party computation problems. In *New Security Paradigms Workshop*, pages 127–135. ACM Press, 2002.

[11] Sergei Evdokimov and Oliver Günther. Encryption techniques for secure database outsourcing. In Joachim Biskup and Javier Lopez, editors, *ESORICS*, volume 4734 of *Lecture Notes in Computer Science*, pages 327–342. Springer, 2007.

[12] Susan Hohenberger and Anna Lysyanskaya. How to securely outsource cryptographic computations. In Joe Kilian, editor, *TCC*, volume 3378 of *Lecture Notes in Computer Science*, pages 264–282. Springer, 2005.

[13] Yuan Hong and Jaideep Vaidya. An inference-proof approach to privacy-preserving horizontally partitioned linear programs. *Optimization Letters*, 2013. To appear. Published online 05 October 2012.

[14] Yuan Hong, Jaideep Vaidya, and Haibing Lu. Secure and efficient distributed linear programming. *Journal of Computer Security*, 20(5):583–634, 2012.

[15] Peeter Laud and Alisa Pankova. New Attacks against Transformation-Based Privacy-Preserving Linear Programming. In Rafael Accorsi and Silvio Ranise, editors, *Security and Trust Management (STM) 2013, 9th International Workshop*, volume 8203 of *Lecture Notes in Computer Science*. Springer, 2013.

[16] Jiangtao Li and Mikhail J. Atallah. Secure and private collaborative linear programming. In *International Conference on Collaborative Computing*, pages 1–8, 2006.

[17] Wei Li, Haohao Li, and Chongyang Deng. Privacy-preserving horizontally partitioned linear programs with inequality constraints. *Optimization Letters*, 7(1):137–144, 2013.

[18] Olvi 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] Tomas Toft. Solving linear programs using multiparty computation. In Roger Dingledine and Philippe Golle, editors, *Financial Cryptography and Data Security*, pages 90–107, Berlin, Heidelberg, 2009. Springer-Verlag.

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

[22] Cong Wang, Kui Ren, and Jia Wang. Secure and practical outsourcing of linear programming in cloud computing. In *INFOCOM, 2011 Proceedings IEEE*, pages 820–828, 2011.

[23] Andrew Chi-Chih Yao. Protocols for secure computations (extended abstract). In *FOCS*, pages 160–164. IEEE, 1982.

Appendix D

Universally composable privacy preserving finite automata execution with low online and offline complexity

The paper "Universally composable privacy preserving finite automata execution with low online and offline complexity" [36] follows.

Universally composable privacy preserving finite automata execution with low online and offline complexity*

Peeter Laud and Jan Willemson

Cybernetica AS

Abstract. In this paper, we propose efficient protocols to obliviously execute non-deterministic and deterministic finite automata (NFA and DFA) in the arithmetic black box (ABB) model. In contrast to previous approaches, our protocols do not use expensive public-key operations, relying instead only on computation with secret-shared values. Additionally, the complexity of our protocols is largely offline. In particular, if the DFA is available during the precomputation phase, then the online complexity of evaluating it on an input string requires a small constant number of operations per character. This makes our protocols highly suitable for certain outsourcing applications.

Keywords. Finite automata, secure multiparty computation, arithmetic black box

1 Introduction

Finite automata (FA) are among the most often used algorithmic tools for analyzing textual data. They are used in filtering spam, recognizing malware, genetic analysis, log mining, etc. Often, these applications make use of data with various owners, having certain expectations of privacy (e.g. genetic microdata may reveal the subject's medical condition, network log items may show security vulnerabilities, etc.). Hence, the problem of executing finite automata in a privacy-preserving manner is highly relevant.

Usually, the execution of the FA does not comprise the whole algorithm for a particular task. E.g. in spam filtering, the automata are used to recognize whether the e-mail message matches certain signatures. Afterwards, these matchings are suitably weighted and combined to decide whether the message was spam or not. Hence the result of privacy-preserving FA execution should be obtained in a manner that is easily usable by further secure computation algorithms — we need *composable* protocols for FA execution (as well as for other algorithmic tasks that are used in the application).

* The research leading to these results has received funding from the European Union Seventh Framework Programme (FP7/2007-2013) under grant agreement no. 284731 (Usable and Efficient Secure Multiparty Computation, UaESMC), and from the European Regional Development Fund through the Estonian Center of Excellence in Computer Science, EXCS.

In the setting of secure multiparty computation, the composability is well-captured by the notion of the *arithmetic black box (ABB)* [9, 22]. It is an ideal functionality that stores the data items that we wish to keep private and performs computations with them. It is realized with the help of suitable cryptographic protocols. Higher-level protocols use the protocols of ABB as their subprotocols. To prove the security of the higher-level protocol, it is sufficient to consider its execution together with the ideal ABB-functionality.

In this paper, we are considering algorithms for FA execution where the description of the FA, as well as the input string have been stored in the ABB. As the result of the algorithm, the information about the reached state(s) of the automaton is also stored in the ABB. During the computation, no information about the input string (except its length) nor the FA (except the number of its states) is leaked outside the ABB.

Without security considerations, the execution of a DFA is trivial — at each step, read the next state of the automaton from the transition table, using the current state and the next character of the input string. Inside the ABB, this is complicated due to the lack of an efficient privacy-preserving look-up operation. In this paper, we consider DFA execution algorithms making use of only such operations that are typically provided in an efficient manner by ABB implementations.

NFAs have the same expressive power as DFAs, but they can offer considerably smaller size of the automata. However, this efficiency comes with a price of a more complicated execution paradigm. Generally, the subsequent states are not uniquely determined and this non-uniqueness is classically implemented by using backtracking. However, backtracking and other constructions with complex control flow are inefficient to implement on ABB; our proposal is based on different ideas [21].

For certain tasks, it is possible to partition the necessary computations into the offline part — these that can be done without knowing the actual inputs —, and the online part — these that require the inputs. To increase the responsiveness of algorithm implementations, one tries to minimize the online computations by structuring the algorithm in such a way that more computations can be performed off-line. In case of FA execution, it makes sense to consider even three stages of input availability — offline, FA-only, and online. In the FA-only stage, the description of the automaton is already available for the computation, but the input string is still missing. This naturally corresponds to certain practical settings, e.g. spam filtering, where the filters are known before the e-mail messages to which they are applied.

There are applications where the input string is private, but the description of the FA is "public" — known to all parties implementing the ABB. The outsourcing of spam filtering into the cloud can be one of such applications. Here the service provider sets up the multiparty computation for realizing the ABB, deploying the servers implementing the ABB at different cloud providers and providing them all with the descriptions of filters. The customers upload their e-mail messages to the ABB and receive back their classification, with no sin-

gle server learning the contents of the messages. The DFA execution algorithms presented in this paper are extremely well suited for the use in these settings, rapidly returning answers while preserving the privacy of the customers against subsets of servers.

Our contribution For a DFA with m states over an alphabet with n characters, we propose an execution algorithm in the ABB model, processing the input string character-wise and performing $(1 + o(1))mn$ ABB multiplications in the offline stage, $(1 + o(1))mn$ ABB multiplications in the FA-only stage and only a single ABB multiplication in the online stage (all these costs are per character of the input string). If the DFA description is public, then the FA-only stage has zero computational cost. Also, for a particular ABB, the cost of offline stage can be reduced to $O(\sqrt{mn})$. For a different ABB, all computations of the FA-only stage may be moved to the on-line stage without increasing the cost of the latter. As usual, the additions and public linear combinations of private values are considered to have zero execution cost due to being local operations in all implementations. The online performance of this protocol exceeds all other protocols in the state of the art, all of which perform at least $O(m + n)$ online work (computation and/or communication; whichever is the bottleneck for the particular method) per character. Also, we stress that our protocol works in the ABB model, making DFA execution usable as a subprotocol in protocols for more complex tasks.

As a separate contribution, we also propose private execution algorithm for the NFAs. In case the automaton description is also captured in the ABB, its online complexity is $m(mn + m + 1)$ multiplications in 3 rounds per input character. If the description of the automaton is public, these complexities drop to $m(m + 1)$ and 2, respectively. In both cases, the amount of required precomputation is $O(m \log m)$ per input character.

Interestingly, the protocols in this paper are the first *information-theoretically* secure protocols for FA execution, if an information-theoretically secure ABB is used. All previous protocols have used cryptographic constructions (encryption) that rely on computational hardness assumptions for security.

To the best of our knowledge, this paper presents the first protocol for secure NFA execution.

Structure of the paper We review work related to privacy-preserving DFA execution in Sec. 2, describe the necessary preliminaries and notation in Sec. 3, and present our basic DFA protocol in Sec. 4. In Sec. 5 we show how to reduce the complexity of the offline phase. Sec. 7 compares the performance of our protocol with the protocols constructed from generic building blocks with good asymptotic complexity. As the last part of the contribution, Sec. 8 presents our private NFA execution protocol. Finally, we draw the conclusions in Sec. 9.

2 Related work

The possibility of *secure multiparty computation* SMC in general has been known for a long time [25, 14]. Unfortunately, the generic protocols resulting from these possibility results are too inefficient in practice for anything but the simplest functionalities.

The question of privacy-preserving DFA execution seems to have been first considered by Troncoso-Pastoriza et al. [23]. In their setting, there are two parties, one of them (Alice) knowing the DFA description, while the other one (Bob) knows the input string. I.e. they were not considering the ABB model. During the computation, the current state of the DFA is additively shared between these two parties (*modulo* m). Rotations of this table combined with homomorphic encryptions of the shares of the current state and an oblivious transfer allow parties to learn the shares of the next state. The protocol has been improved in several ways by Blanton and Aliasgari [1]. Beside reducing the complexity, they also adapt the protocol for the ABB model. Through a clever reshaping of the transition table, they are actually able to get the communication complexity down to $O(\sqrt{mn})$ per character of the input string. Their techniques resemble those of *private information retrieval* (PIR) protocols using homomorphic encryption [17]; we believe that through a more thorough application of these techniques, the complexity per character might even be lowered to $O(\log^2 mn)$. Nevertheless, we are not pursuing these avenues of research here, as the constant hidden inside the O-notation makes these protocols less efficient than our protocols (which do not use expensive public-key operations) for realistic problem sizes.

Troncoso-Pastoriza et al.'s protocol has been adapted to malicious setting by Gennaro et al. [11], using zero-knowledge protocols to force honest behaviour of parties. Malicious setting is also considered by Wei and Reiter [24]. They propose two-party (called "client" and "server") protocols in the ABB model that are secure against the malicious behaviour of the server party. Similarly to the protcols in this paper, they treat the DFA transition table as a polynomial over some field. All protocols described so far make heavy use of homomorphic encryption, some of them also requiring some extra properties [4].

Garbled circuits originally proposed by Yao [25] can be adapted for DFA execution [10, 19]. In these protocols, the party knowing the transition table δ constructs and garbles "δ-gates", connected sequentially. The party with the input string executes the circuit and learns the last state of the DFA or some property of it.

3 Preliminaries

In this work, a *deterministic finite automaton (DFA)* over the alphabet Σ is a tuple $A = (Q, \delta, q_0)$, where Q is the set of states, $q_0 \in Q$ the starting state of the automaton, and $\delta : Q \times \Sigma \to Q$ the transition table. Given a string $s = a_1 \cdots a_\ell \in \Sigma^*$, the DFA A maps s to the state $\boldsymbol{\delta}_A(s) = \delta(\cdots(\delta(\delta(q_0, a_1), a_2)\cdots), a_\ell)$. Compared to usual definition of DFA, we have left out the set of accepting states

from the structure of a DFA, and consider the function $\boldsymbol{\delta}_A : \Sigma^* \to Q$ as the behaviour of A; this is also computed by our protocol. This omission is justified by our work in the ABB model, as the computation can continue from the last state reached by the DFA in any manner deemed necessary by the designer of the whole system.

Similarly, a *nondeterministic finite automaton (NFA)* over the alphabet Σ is a tuple $A = (Q, \delta, q_0)$, with Q and q_0 meaning the same as before, and $\delta : Q \times \Sigma \to \mathcal{P}(Q)$ being the transition table (here $\mathcal{P}(Q)$ is the set of subsets of Q). We can extend δ to sets of states — $\delta(\mathcal{Q}, a) = \bigcup_{q \in \mathcal{Q}} \delta(q, a)$ for $\mathcal{Q} \subseteq Q$ and $a \in \Sigma$. Given a string $s = a_1 \cdots a_\ell \in \Sigma^*$, the NFA A maps s to the set of states $\boldsymbol{\delta}_A(s) = \delta(\cdots(\delta(\delta(\{q_0\}, a_1), a_2)\cdots), a_\ell)$.

The *arithmetic black box* is an ideal functionality $\mathcal{F}_{\mathsf{ABB}}$. It allows its users (a fixed number p of parties) to securely store and retrieve values, and to perform computations with them. When a party sends the command $\mathsf{store}(v)$ to $\mathcal{F}_{\mathsf{ABB}}$, where v is some value, the functionality assigns a new *handle* h (sequentially taken integers) to it by storing the pair (h, v) and sending h to all parties. If a sufficient number (depending on implementation details) of parties send the command $\mathsf{retrieve}(h)$ to $\mathcal{F}_{\mathsf{ABB}}$, it looks up (h, v) among the stored pairs and responds with v to all parties. When a sufficient number of parties send the command $\mathsf{compute}(op; h_1, \ldots, h_k; params)$ to $\mathcal{F}_{\mathsf{ABB}}$, it looks up the values $v_1, \ldots, v_k$ corresponding to the handles $h_1, \ldots, h_k$, performs the operation op (parametrized with $params$) on them, stores the result v together with a new handle h, and sends h to all parties. In this way, the parties can perform computations without revealing anything about the intermediate values or results, unless a sufficiently large coalition wants a value to be revealed. In this paper, we use the functionality $\mathcal{F}_{\mathsf{ABB}}$ to implement privacy-preserving FA execution.

The existing implementations of ABB are based on either secret sharing [7, 2, 5] or threshold homomorphic encryption [9, 15]. Fully homomorphic encryption [13] may also be used to implement ABB in a conceptually very simple way, but with prohibitively slow performance. Depending on the implementation, the ABB offers protection against a honest-but-curious, or a malicious party, or a number of parties (up to a certain limit). E.g. the implementation of the ABB by Sharemind [2] consists of three parties, providing protection against one honest-but-curious party.

Typically, the ABB performs computations with values v from some ring $\mathbb{R}$. The set of operations definitely includes addition/subtraction, multiplication of a stored value with a public value (this operation motivates the *params* in the compute-command), and multiplication. Even though all algorithms can be expressed using just these operations, most ABB implementations provide more operations (as primitive protocols) for greater efficiency of the implementations of algorithms on top of the ABB. In all ABB implementations, addition, and multiplication with a public value occur negligible costs; hence they're not counted when analyzing the complexity of protocols using the ABB. Other operations may require a variable amount of communication (in one or several rounds) between parties, and/or expensive computation. The ABB can execute several

operations in parallel; the *round complexity* of a protocol is the number of communication rounds all operations of the protocol require, when parallelized as much as possible.

It is common to use $[\![v]\!]$ to denote the value v stored in the ABB. The notation $[\![v_1]\!]$ *op* $[\![v_2]\!]$ denotes the computation of v_1 *op* v_2 by the ABB (translated to a protocol in the implementation of $\mathcal{F}_{\mathsf{ABB}}$).

4 Basic protocol for DFA execution

Our basic protocol combines the idea to consider the transition table δ of the DFA as a polynomial over a field $\mathbb{F}$ [24] with a method to move offline most of the computations for polynomial evaluation in the ABB [18]. Both ideas have been slightly improved and expanded here.

We have a DFA $A = (Q, \delta, q_0)$ with $|Q| = m$, working over the alphabet Σ with $|\Sigma| = n$. Let $\mathbb{F}$ be a finite field with at least $mn + 1$ elements; moreover, let $Q \subseteq \mathbb{F}$, $\Sigma \subseteq \mathbb{F}$ and let $\gamma \in \mathbb{F}$ be such, that $(q, a) \mapsto \gamma q + a$ is an injective mapping from $Q \times \Sigma$ to $\mathbb{F}\backslash\{0\}$.

There exists a polynomial $f : \mathbb{F} \to \mathbb{F}$, such that $f(\gamma q + a) = \delta(q, a)$ for all $q \in Q$ and $a \in \Sigma$; this polynomial has the degree of at most $mn - 1$. There exist Lagrange interpolation coefficients λ_{iqa} with $0 \leq i \leq mn - 1$, $q \in Q$, $a \in \Sigma$, depending only on m and n (i.e. these coefficients are public), such that $f(x) = \sum_{i=0}^{mn-1} c_i x^i$, where $c_i = \sum_{q \in Q} \sum_{a \in \Sigma} \lambda_{iqa} \delta(q, a)$.

Let our ABB work with values from the field $\mathbb{F}$. In this case, there exists a protocol for generating a uniformly random element of $\mathbb{F}$ inside the ABB (denote: $[\![r]\!] \xleftarrow{\$} \mathbb{F}$), and for generating a uniformly random non-zero element of $\mathbb{F}$ together with its inverse (denote: $([\![r]\!], [\![r^{-1}]\!]) \xleftarrow{\$} \mathbb{F}^*$). These protocols require a small constant number of multiplications on average [6]. Using these subprotocols, Algorithm 1 gives the protocol for executing the DFA A on an ℓ-character string. Note that all inputs to the algorithm (except for the sizes m, n and ℓ, which are public) are stored inside the ABB. Its result, the state of the DFA $[\![q_l]\!]$ after processing ℓ characters is similarly stored inside the ABB.

Correctness First we note that the polynomial $f(x) = \sum_{j=0}^{mn-1} c_j x^j$ satisfies the equality $f(\gamma q + a) = \delta(q, a)$ due to the construction of c_j in the DFA-only stage and the definition of the coefficients λ_{jqa}. We also note that the values r_i^j constructed in the offline stage are indeed the j-th powers of the values r_i.

We can now easily show that the value q_i computed by the on-line loop is equal to the state of the DFA A after processing the characters $a_1, \ldots, a_i$. For $i = 0$, this claim trivially holds. If it holds for $i - 1$, then it also holds for i:

$$q_i = \sum_{j=0}^{mn-1} z_i^j y_{ij} = \sum_{j=0}^{mn-1} (\gamma q_{i-1} + a_i)^j r_i^{-j} c_j r_i^j = f(\gamma q_{i-1} + a_i) = \delta(q_{i-1}, a) \ .$$

Algorithm 1: DFA execution protocol

Data: DFA components $[\![\delta(q,a)]\!]$ and $[\![q_0]\!]$, where $q \in Q$, $a \in \Sigma$.
Data: Characters of the input string $[\![a_1]\!], \ldots, [\![a_\ell]\!]$.
Result: Last state $[\![q_\ell]\!]$ in ABB.

1 offline processing
2 **foreach** $i \in \{1, \ldots, \ell\}$ **do**
3 $\quad ([\![r_i]\!], [\![r_i^{-1}]\!]) \xleftarrow{\$} \mathbb{F}^*$
4 $\quad$ **for** $j = 2$ **to** $mn - 1$ **do** $[\![r_i^j]\!] \leftarrow [\![r_i]\!] \cdot [\![r_i^{j-1}]\!]$
5 DFA-only processing
6 **foreach** $j \in \{0, \ldots, mn-1\}$ **do** $[\![c_j]\!] \leftarrow \sum_{q \in Q} \sum_{a \in \Sigma} \lambda_{jqa} [\![\delta(q,a)]\!]$
7 **foreach** $i \in \{1, \ldots, \ell\}$, $j \in \{0, \ldots, mn-1\}$ **do** $[\![y_{ij}]\!] \leftarrow [\![c_j]\!] \cdot [\![r_i^j]\!]$
8 online processing
9 **for** $i = 1$ **to** ℓ **do**
10 $\quad [\![z_i]\!] \leftarrow (\gamma [\![q_{i-1}]\!] + [\![a_i]\!]) \cdot [\![r_i^{-1}]\!]$
11 $\quad z_i \leftarrow \mathsf{retrieve}([\![z_i]\!])$
12 $\quad [\![q_i]\!] \leftarrow \sum_{j=0}^{mn-1} z_i^j [\![y_{ij}]\!]$

Privacy Except for computing z_i, al operations in Alg. 1 are performed either inside the ABB, or with public values. Hence all guarantees provided by the ABB against certain kinds of attacks involving certain coalitions of parties carry directly over to Alg. 1, if there weren't the computations involving the values z_i. Regarding the values z_i — they do not leak anything about the inputs to the algorithm, because each of them is a product of a non-zero secret value with a uniformly randomly distributed non-zero value. Hence z_i is also a uniformly randomly distributed element of $\mathbb{F}^*$. As independent values r_i^{-1} are used for computing different z_i, the different z_i-s are mutually independent as well. Regarding the correctness of the use of z_i-s in further computation — as these values become known to all parties, we can be sure that in the computation of q_i, correct z_i is used.

Complexity It is straightforward to count the number of operations Alg. 1 performs. In the offline stage, we perform $mn - 2$ multiplications per character of the input string. We also generate one random invertible element together with its inverse, this generation costs the same as a couple of multiplications [6] (in the ABB of Sharemind [3], the random number generation is free, while verifying that it is invertible and computing the inverse takes one multiplication, with the probability of the element being rejected being equal to $2/|\mathbb{F}|$). The round complexity of this computation is also $O(mn)$, which would be bad for online computations, but, in our opinion, does not matter for computations where latency is unimportant. We note that the offline phase could be performed in $O(1)$ rounds [6] at the cost of increasing the number of multiplications a couple of times. In the DFA-only stage, we perform a number of multiplications with constants λ_{jqa}; we count these operations as free. We also perform $mn - 1$ mul-

tiplications per character in order to compute $[\![y_{ij}]\!]$ (no multiplication is needed to obtain $[\![y_{i0}]\!]$). But if the DFA description had been public, then the values c_j would have been public, too, and the values $[\![y_{ij}]\!]$ would have been linear combinations of $[\![r_i^j]\!]$ with public coefficients. In this case, the DFA-only stage would have contained no costly operations at all. In the online stage, the only costly operation is the computation of $[\![z_i]\!]$, which takes a single multiplication of private values. Also, the retrieveoperation has the complexity similar to a multiplication in most implementations of the ABB.

5 Improving offline performance

We will now consider the ABB implementation of Sharemind [3] and show how it can be leveraged to speed up the offline stage of Alg. 1, the goal of which was to compute $[\![v^2]\!], \ldots, [\![v^k]\!]$ from $[\![v]\!]$ and $k \in \mathbb{N}$. Let us give a short overview of the relevant protocols in Sharemind.

The Sharemind ABB is realized by three parties, offering protection against passive attacks by one of the parties. The ABB stores elements of some ring $\mathbb{R}$; a value $v \in \mathbb{R}$ is stored in the ABB as $[\![v]\!] = ([\![v]\!]_1, [\![v]\!]_2, [\![v]\!]_3) \in \mathbb{R}^3$ satisfying $[\![v]\!]_1 + [\![v]\!]_2 + [\![v]\!]_3 = v$, where the *share* $[\![v]\!]_i$ is kept by the i-th party P_i. Messages depending on these shares are sent among the parties, hence it is important to rerandomize $[\![v]\!]$ before each use. The *resharing* protocol [3, Algorithm 1] (repeated here as Alg. 5 in Appendix A) is used for this rerandomization. We note that in this algorithm, the generation and distribution of random elements can take place offline. Even better, only random seeds can be distributed ahead of the computation and new elements of $\mathbb{R}$ generated from them as needed. Hence we consider the resharing protocol to involve only local operations and have the cost 0 in our complexity analysis.

Sharemind's multiplication protocol [3, Algorithm 2] (repeated as Alg. 6 in Appendix A) is based on the equality $([\![u]\!]_1 + [\![u]\!]_2 + [\![u]\!]_3)([\![v]\!]_1 + [\![v]\!]_2 + [\![v]\!]_3) = \sum_{i,j=1}^{3} [\![u]\!]_i [\![v]\!]_j$. After the party P_i has sent $[\![u]\!]_i$ and $[\![v]\!]_i$ to party P_{i+1} (here and subsequently, all party indices are *modulo* 3), each of these nine components of the sum can be computed by one of the parties. We see that in order to perform one multiplication in Sharemind, six elements of $\mathbb{R}$ have to be sent from one party to another. All these can be done in parallel. The multiplication protocol is secure against one honest-but-curious party [3, Theorem 2].

We see that sometimes the multiplication or a series of multiplications can be performed more efficiently. To compute $[\![u^2]\!]$ from $[\![u]\!]$, only $[\![u]\!]_i$ has to be sent from P_i to P_{i+1}. To compute $([\![uv_1]\!], \ldots, [\![uv_n]\!])$ from $[\![u]\!]$ and $([\![v_1]\!], \ldots, [\![v_n]\!])$, we start n copies of the multiplication protocol, but the shares $[\![u]\!]_i$ have to be sent only once. The security of the protocol is not affected by such optimizations.

Alg. 1 requires the ring $\mathbb{R}$ to be a field $\mathbb{F}$. In computing $([\![v^2]\!], \ldots, [\![v^k]\!])$ from $[\![v]\!]$, more substantive optimizations are possible if we take $\mathbb{F}$ to be of characteristic 2. In this case, the cardinality of $\mathbb{F}$ is a power of 2 and the equality $1 + 1 = 0$ holds. We note that squaring a value in the ABB now requires only local operations: $(x_1 + x_2 + x_3)^2 = x_1^2 + x_2^2 + x_3^2$ if the characteristic of $\mathbb{F}$ is 2.

Similarly, if parties P_i have sent the share $[\![v]\!]_i$ to parties P_{i+1} (as they do in lines 2&4 of Alg.6), then they have also sent the shares $[\![v^{2^n}]\!]_i$ for all $n \in \mathbb{N}$. The algorithm for computing the powers of $[\![v]\!]$ up to $[\![v^k]\!]$ is given as Alg. 2.

Algorithm 2: Computing $([\![v^2]\!], \ldots, [\![v^k]\!])$ from $[\![v]\!]$

Data: $k \in \mathbb{N}$ and the value $[\![v]\!]$, where $v \in \mathbb{F}$, char $\mathbb{F} = 2$
Result: Values $[\![u_0]\!], \ldots, [\![u_k]\!]$, where $u_j = v^j$
$q \leftarrow \lceil \log \sqrt{k+1} \rceil$
$[\![u_0]\!] \leftarrow (1, 0, 0)$
$[\![u_1]\!] \leftarrow \mathsf{Reshare}([\![v]\!])$
Party P_i sends $[\![u_1]\!]_i$ to party P_{i+1}
for $j = 2$ **to** $2^q - 1$ **do**
  if j *is even* **then**
    Party P_i computes $[\![u_j]\!]_i \leftarrow [\![u_{j/2}]\!]_i^2$ and $[\![u_j]\!]_{i-1} \leftarrow [\![u_{j/2}]\!]_{i-1}^2$
  else
    Party P_i computes
      $[\![t]\!]_i \leftarrow [\![u_1]\!]_i \cdot [\![u_{j-1}]\!]_i + [\![u_1]\!]_i \cdot [\![u_{j-1}]\!]_{i-1} + [\![u_1]\!]_{i-1} \cdot [\![u_{j-1}]\!]_i$
    $[\![u_j]\!] \leftarrow \mathsf{Reshare}([\![t]\!])$
    Party P_i sends $[\![u_j]\!]_i$ to party P_{i+1}
  `// At this point,` P_i `knows` $[\![u_0]\!]_i, \ldots, [\![u_j]\!]_i, [\![u_0]\!]_{i-1}, \ldots, [\![u_j]\!]_{i-1}$
foreach $j \in \{2^q, \ldots, k\}$ **do**
  Let $(r, s) \in \{0, \ldots, 2^q - 1\}$, such that $2^q r + s = j$
  Party P_i computes $[\![t]\!]_i \leftarrow [\![u_r]\!]_i^{2^q} \cdot [\![u_s]\!]_i + [\![u_r]\!]_i^{2^q} \cdot [\![u_s]\!]_{i-1} + [\![u_r]\!]_{i-1}^{2^q} \cdot [\![u_s]\!]_i$
  $[\![u_j]\!] \leftarrow \mathsf{Reshare}([\![t]\!])$

Correctness For $j < 2^q$, the values v^j in the ABB are computed as $v^j = (v^{j/2})^2$ (if j is even) or $v^j = v \cdot v^{j-1}$ (if j is odd). We note that all necessary shares for computing these values are available to the parties. If $j \geq 2^q$ then v^j is computed as $v^j = (v^r)^{2^q} \cdot v^s$, where $2^q r + s = j$. Because char $\mathbb{F} = 2$, the shares of $(v^r)^{2^q}$ are just the shares of v^r, squared q times. This squaring can be performed locally. Again, all shares are available to the parties that need them.

Privacy Similarly to the multiplication protocol, each party knows at most two out of the three shares of $[\![v^j]\!]$, for each j. The last share is a uniformly randomly distributed element of $\mathbb{F}$.

In the second loop of Alg. 2, all $[\![u_j]\!]$ are rerandomized. In the first loop, the values $[\![u_j]\!]$ are not rerandomized for even j. This rerandomization is unnecessary, because of the locality of the computation. We note that all values sent to other parties result from the Reshare protocol.

Complexity The second loop of Alg. 2 has only local computation (recall that Reshare is counted as requiring local computation only). In the first loop, the iterations with odd index j incur the communication of three elements of $\mathbb{F}$, while

the iterations with even j incur no communication. The first loop has at most $\lceil 2\sqrt{k+1} \rceil$ iterations, hence the communication is at most $3\lceil \sqrt{k+1} \rceil$ elements of $\mathbb{F}$.

If we use SHAREMIND's representation of values in ABB, Alg. 2 in place of the offline stage of Alg. 1, and if the DFA description is public, i.e. known to all parties implementing the ABB, then the total offline communication (per character) of executing a m-state DFA on a string over an alphabet of size n is at most $3\lceil \sqrt{mn} \rceil (\lceil \log(m+1) \rceil + \lceil \log(n+1) \rceil) = 3\sqrt{mn}\log(mn) + o(1)$ bits. Here we have assumed that the states of the DFA are encoded as bit-strings $1, \ldots, m$ of length $\lceil \log(m+1) \rceil$, while the characters of the alphabet are encoded as bit-strings $1, \ldots, n$ of length $\lceil \log(n+1) \rceil$. In this way, a suitable γ exists and the elements of $\mathbb{F}$ are encoded as bit-strings of length $\lceil \log(m+1) \rceil + \lceil \log(n+1) \rceil$.

With SHAREMIND's protocols, the online communication (per character of the input string) is 12 elements of $\mathbb{F}$, distributed equally between the multiplication and the retrieve-operation.

6 Improving FA-only / online performance

A different kind of optimization is possible if the ABB implementation is based on Shamir's secret sharing [20] and using the multiplication protocol of Gennaro et al. [12], which is the case for e.g. VIFF [7] or SEPIA [5]. Using this implementation, the computation of a scalar product $[\![\sum_{i=1}^{k} a_i b_i]\!]$ from the values $[\![a_1]\!], \ldots, [\![a_k]\!]$ and $[\![b_1]\!], \ldots [\![b_k]\!]$ stored inside the ABB has the same cost as performing a single multiplication of stored values.

Hence the following modification of the DFA execution algorithm, presented as Algorithm 3 will have the same offline and online complexity as the original algorithm, but perform no costly operations during the FA-only stage.

7 Performance comparison

Private Information Retrieval (PIR) protocols can be adapted to compute $\delta(q, a)$ with asymptotically better communication complexity, if the description of δ is public. A PIR protocol allows the *client* to query for a specific element in *server*'s database, without the server learning the index of that element. In our setting, the ABB would be the client, while the server's computations can be executed in the public. Each character of the input string would need one instance of the PIR protocol to be executed on the transition table δ.

Lipmaa's communication-efficient PIR protocol [17] internally uses the homomorphic cryptosystem by Damgård and Jurik [8]. Its encryption and decryption are usually not considered to be part of the set of operations offered by ABB, but they are often readily available (also in SHAREMIND) using the threshold version of this cryptosystem. The PIR protocol requires the communication of $O(k \log^2(mn))$ bits per query, where mn is the number of elements in the database and k is the size of the RSA-modulus N of the cryptosystem.

Algorithm 3: DFA execution protocol

Data: DFA components $[\![\delta(q,a)]\!]$ and $[\![q_0]\!]$, where $q \in Q$, $a \in \Sigma$.
Data: Characters of the input string $[\![a_1]\!], \ldots, [\![a_\ell]\!]$.
Result: Last state $[\![q_\ell]\!]$ in ABB.

offline processing
foreach $i \in \{1, \ldots, \ell\}$ **do**
  $([\![r_i]\!], [\![r_i^{-1}]\!]) \xleftarrow{\$} \mathbb{F}^*$
  for $j = 2$ **to** $mn - 1$ **do** $[\![r_i^j]\!] \leftarrow [\![r_i]\!] \cdot [\![r_i^{j-1}]\!]$
DFA-only processing
foreach $j \in \{0, \ldots, mn-1\}$ **do** $[\![c_j]\!] \leftarrow \sum_{q \in Q} \sum_{a \in \Sigma} \lambda_{jqa} [\![\delta(q,a)]\!]$
online processing
for $i = 1$ **to** ℓ **do**
  $[\![z_i]\!] \leftarrow (\gamma[\![q_{i-1}]\!] + [\![a_i]\!]) \cdot [\![r_i^{-1}]\!]$
  $z_i \leftarrow \mathsf{retrieve}([\![z_i]\!])$
  foreach $j \in \{0, \ldots, mn-1\}$ **do** $[\![(r_i z_i)^j]\!] = z_i^j [\![r_i^j]\!]$
  $[\![q_i]\!] \leftarrow \sum_{j=0}^{mn-1} [\![c_j]\!] \cdot [\![(r_i z_i)^j]\!]$

To have a valid comparison of the PIR-protocol based DFA execution and our protocols, we have to estimate the constant hidden in the O-notation for the PIR protocol's query complexity, particularly when implemented on top of Sharemind. Per character, the query belongs to the domain of δ, its size is $\alpha = \lceil \log(mn) \rceil$ bits. The single bits of the query must be available, hence we assume that the current pair (q, a) is stored as α separate bits in the ABB. In the PIR protocol, the client encrypts all bits, resulting in α ciphertexts of size $2k$, $3k$, $\ldots$, $(\alpha+1)k$ bits, respectively. In Sharemind's ABB, each of the three parties implementing it may encrypt its share of each bit; these ciphertexts can be combined using the homomorphic properties of the encryption scheme. To minimize the communication, we let two parties send their ciphertexts to the third party that will then perform the operations of the server in the PIR protocol. In this case, the total number of communicated bits for the client's message in the PIR protocol is $2(2k + 3k + \cdots + (\alpha+1)k) = \alpha(\alpha+3)k$.

The third party, performing the operations of the server in the PIR protocol, combines these ciphertexts to a multiply encrypted ciphertext of the query result. This ciphertext must be decrypted using the decryption protocol of the threshold cryptosystem; this causes significant extra communication. To simplify our analysis, let us not estimate the communication costs of these operations, but only compare the total cost of our proposed protocol (per character of the input string) — $\approx 3\sqrt{mn}\log(mn)$ — with the communication costs for producing just the client's message in the PIR protocol — $(\log^2(mn) + 3\log(mn))k$. For acceptable level of security, we have to take $k \geq 1024$. In this case, our protocol has less communication if $mn \leq 10^8$. We are unlikely to have DFA larger than that in real applications.

8 NFA execution

Due to their non-deterministic nature, NFAs are more complicated to handle in a secure manner. We see that even though the NFA execution starts from a single state, after the intermediate steps it can generally be in a subset of states. In order to account for this, we will use characteristic vectors of the intermediate sets $\mathcal{Q}_i = \boldsymbol{\delta}_A(a_1 \cdots a_i)$ to represent them (using the notation of Sec. 3). Let $\mathbf{q}^i = (q_0^i, q_1^i, \ldots, q_{m-1}^i)$ be a binary vector, where $q_j^i = 1$ iff the state $q_j \in \mathcal{Q}_i$.

As $\mathcal{Q}_0 = \{q_0\}$, we have $\mathbf{q}^0 = (1, 0, \ldots, 0)$. Subsequent $\mathbf{q}^i$-s will depend both on the given automaton A and the string s. Namely, in order to determine $\mathbf{q}^i$ from $\mathbf{q}^{i-1}$, δ and a_i, we can compute

$$q_j^i = \bigvee_{q \in \mathcal{Q}_{i-1}} [q_j \in \delta(q, a_i)] = \bigvee_{k=0}^{m-1} q_k^{i-1} \& [q_j \in \delta(q_k, a_i)] \tag{1}$$

for all the components q_j^i of the characteristic vector $\mathbf{q}^i$.

The exact complexity of computing (1) in the ABB depends on which components of the NFA execution problem need to be private. Even if the automaton itself is public and only the string s is private, the characteristic vectors $\mathbf{q}^i$ (for $i > 0$) still need to be protected. However, in order for the term $q_k^{i-1} \& [q_j \in \delta(q_k, a_i)]$ to evaluate to 1, there has to exist a transition from q_k to q_j (even if we do not know whether its label matches a_i or not). Hence, from equation (1) we can leave out all the terms for which there is no such transition. If the NFA has to be private, no such omission is possible.

In order to determine efficiently whether $q_j \in \delta(q_k, a_i)$, we need an efficient representation of δ as well. We will represent it as a look-up table $\overline{\delta} : Q \times Q \to \mathcal{P}(\Sigma)$, where $a_i \in \overline{\delta}(q_k, q_j)$ iff $q_j \in \delta(q_k, a_i)$. To encode subsets of Σ, we will once again use characteristic vectors; let $\mathcal{S} \subseteq \Sigma$ be encoded by vector $\mathbf{s} = (s_1, \ldots, s_n)$ where $s_i = 1$ iff the corresponding $\sigma_i \in \mathcal{S}$. The characteristic vectors in the look-up table $\overline{\delta}$ may be private or public depending on whether A needs to be protected or not. Note that if we have for some q_k and q_j that $\overline{\delta}(q_k, q_j) = \emptyset$, then in the case of public automaton the respective term may be omitted from equation (1).

In order to execute NFA on the (private) string $a_1 \cdots a_\ell$, we also represent the characters of the string using binary characteristic vectors $\mathbf{a}_1, \ldots, \mathbf{a}_\ell$, where $\mathbf{a}_i = (a_i^1, \ldots, a_i^n)$ and $a_i^j = 1$ iff $a_i = \sigma_j$. As a result, the value of the predicate $q_j \in \delta(q_k, a_i)$ can be computed as a dot product $\overline{\delta}(q_k, q_j) \cdot \mathbf{a}_i$. Assuming that addition is free in the underlying secure computation platform (as it is in the case of Sharemind), dot product requires n multiplications that can be performed in parallel.

Next, computing the whole term $q_k^{i-1} \& [q_j \in \delta(q_k, a_i)]$ on equation (1) requires an additional round of multiplication to add the conjunction with q_k^{i-1}. Finally, we need to compute the disjunction over all the states where the transitions to q_i might have come from. This can be accomplished by adding the respective terms, comparing the result to 0 and inverting the comparison result [16]. Working in a suitable field, comparison to 0 may be implemented

using just one round of online multiplications using the protocol by Lipmaa and Toft [18] (though some precomputation is necessary). These operations require that the underlying ring $\mathbb{R}$ of the ABB is a field $\mathbb{F}$ with char $\mathbb{F} \geq m+1$.

The overall procedure of NFA execution is presented as Algorithm 4. The algorithm is obviously private because only ABB operations are used and nothing is declassified. The correctness of the algorithm follows from the discussions above.

Algorithm 4: NFA execution protocol

Data: NFA components $[\![\overline{\delta}(q_k, q_j)]\!]$ for all $q_k, q_j \in Q$, and $[\![\mathbf{q}_0]\!]$.
Data: Characteristic vectors of the characters of the input string $[\![\mathbf{a}_1]\!], \ldots, [\![\mathbf{a}_\ell]\!]$.
Result: Characteristic vector of the last set of achievable states $[\![\mathbf{q}_\ell]\!]$ in ABB.

1 **for** $i = 1$ **to** ℓ **do**
2 **foreach** $j \in \{0, \ldots, m-1\}$ **do**
3 **foreach** $k \in \{0, \ldots, m-1\}$ **do** $[\![t_{ijk}]\!] = [\![q_k^{i-1}]\!] \cdot ([\![\overline{\delta}(q_k, q_j)]\!] \cdot [\![\mathbf{a}_i]\!])$
4 $[\![p_j^i]\!] = \sum_{k=0}^{m-1} [\![t_{ijk}]\!]$
5 $[\![q_j^i]\!] = 1 - [\![p_j^i \stackrel{?}{=} 0]\!]$

Complexity Computing the dot product $[\![\overline{\delta}(q_k, q_j)]\!] \cdot [\![\mathbf{a}_i]\!]$ in line 3 requires n parallel multiplications and the product with $[\![q_k^{i-1}]\!]$ adds an additional multiplication and another round. Altogether, this cycle requires $m(n+1)$ private online multiplications in two rounds.

Summation on line 4 can be performed by the parties without any communication, and the comparisons on 5 require one private online multiplication per comparison, which can all be performed in parallel in one more round. Hence, in order to process each of the input symbols, $m(m(n+1)+1)$ multiplications in three rounds are needed. The overall online complexity of Algorithm 4 is $\ell m(m(n+1)+1)$ multiplications in 3ℓ rounds.

In case of the public automaton, the characteristic vectors $\overline{\delta}(q_k, q_j)$ will be public. Hence the dot product on line 3 will become a local operation performed by the computing parties. As a result, the whole Algorithm 4 requires $\ell m(m+1)$ private multiplications in 2ℓ rounds.

In both cases, the offline complexity consists of the precomputation to facilitate the fast online comparisons. According to [18], the amount of precomputation required for one comparison is $O(\log m)$. Since we need to perform ℓm comparisons, the total work needed in the offline phase is $O(\ell m \log m)$.

9 Conclusions

We have given the first ever algorithm for privacy-preserving NFA execution, as well as fast algorithms for privacy-preserving DFA execution. All our algorithms

are composable and can be easily used as components in the design of larger systems. In case of public FA, our DFA execution algorithms are the fastest for reasonably-sized DFAs. In any case, our DFA execution algorithm has by far the best *online* performance.

References

1. Marina Blanton and Mehrdad Aliasgari. Secure Outsourcing of DNA Searching via Finite Automata. In Sara Foresti and Sushil Jajodia, editors, *DBSec*, volume 6166 of *Lecture Notes in Computer Science*, pages 49–64. Springer, 2010.
2. 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.
3. 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.
4. Dan Boneh, Eu-Jin Goh, and Kobbi Nissim. Evaluating 2-DNF Formulas on Ciphertexts. In Joe Kilian, editor, *TCC*, volume 3378 of *Lecture Notes in Computer Science*, pages 325–341. Springer, 2005.
5. Martin Burkhart, Mario Strasser, Dilip Many, and Xenofontas Dimitropoulos. SEPIA: Privacy-preserving aggregation of multi-domain network events and statistics. In *USENIX Security Symposium*, pages 223–239, Washington, DC, USA, 2010.
6. Ivan Damgård, Matthias Fitzi, Eike Kiltz, Jesper Buus Nielsen, and Tomas Toft. Unconditionally secure constant-rounds multi-party computation for equality, comparison, bits and exponentiation. In Shai Halevi and Tal Rabin, editors, *TCC*, volume 3876 of *Lecture Notes in Computer Science*, pages 285–304. Springer, 2006.
7. Ivan Damgård, Martin Geisler, Mikkel Krøigaard, and Jesper Buus Nielsen. Asynchronous Multiparty Computation: Theory and Implementation. In Stanislaw Jarecki and Gene Tsudik, editors, *Public Key Cryptography*, volume 5443 of *Lecture Notes in Computer Science*, pages 160–179. Springer, 2009.
8. Ivan Damgård and Mads Jurik. A Generalisation, a Simplification and Some Applications of Paillier's Probabilistic Public-Key System. In Kwangjo Kim, editor, *Public Key Cryptography*, volume 1992 of *Lecture Notes in Computer Science*, pages 119–136. Springer, 2001.
9. Ivan Damgård and Jesper Buus Nielsen. Universally composable efficient multiparty computation from threshold homomorphic encryption. In Dan Boneh, editor, *CRYPTO*, volume 2729 of *Lecture Notes in Computer Science*, pages 247–264. Springer, 2003.
10. Keith B. Frikken. Practical Private DNA String Searching and Matching through Efficient Oblivious Automata Evaluation. In Ehud Gudes and Jaideep Vaidya, editors, *DBSec*, volume 5645 of *Lecture Notes in Computer Science*, pages 81–94. Springer, 2009.
11. Rosario Gennaro, Carmit Hazay, and Jeffrey S. Sorensen. Text search protocols with simulation based security. In Phong Q. Nguyen and David Pointcheval, editors, *Public Key Cryptography*, volume 6056 of *Lecture Notes in Computer Science*, pages 332–350. Springer, 2010.
12. Rosario Gennaro, Michael O. Rabin, and Tal Rabin. Simplified vss and fact-track multiparty computations with applications to threshold cryptography. In *PODC*, pages 101–111, 1998.

13. Craig Gentry. Fully homomorphic encryption using ideal lattices. In Michael Mitzenmacher, editor, *STOC*, pages 169–178. ACM, 2009.
14. Oded Goldreich, Silvio Micali, and Avi Wigderson. How to Play any Mental Game or A Completeness Theorem for Protocols with Honest Majority. In *STOC*, pages 218–229. ACM, 1987.
15. Wilko Henecka, Stefan Kögl, Ahmad-Reza Sadeghi, Thomas Schneider, and Immo Wehrenberg. TASTY: tool for automating secure two-party computations. In *CCS '10: Proceedings of the 17th ACM conference on Computer and communications security*, pages 451–462, New York, NY, USA, 2010. ACM.
16. 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.
17. Helger Lipmaa. An oblivious transfer protocol with log-squared communication. In Jianying Zhou, Javier Lopez, Robert H. Deng, and Feng Bao, editors, *ISC*, volume 3650 of *Lecture Notes in Computer Science*, pages 314–328. Springer, 2005.
18. Helger Lipmaa and Tomas Toft. Secure equality and greater-than tests with sublinear online complexity. In Fedor V. Fomin, Rusins Freivalds, Marta Z. Kwiatkowska, and David Peleg, editors, *ICALP (2)*, volume 7966 of *Lecture Notes in Computer Science*, pages 645–656. Springer, 2013.
19. Payman Mohassel, Salman Niksefat, Seyed Saeed Sadeghian, and Babak Sadeghiyan. An Efficient Protocol for Oblivious DFA Evaluation and Applications. In Orr Dunkelman, editor, *CT-RSA*, volume 7178 of *Lecture Notes in Computer Science*, pages 398–415. Springer, 2012.
20. Adi Shamir. How to share a secret. *Commun. ACM*, 22(11):612–613, 1979.
21. Ken Thompson. Regular Expression Search Algorithm. *Commun. ACM*, 11(6):419–422, 1968.
22. Tomas Toft. *Primitives and Applications for Multi-party Computation*. PhD thesis, University of Aarhus, Denmark, BRICS, Department of Computer Science, 2007.
23. Juan Ramón Troncoso-Pastoriza, Stefan Katzenbeisser, and Mehmet Utku Celik. Privacy preserving error resilient DNA searching through oblivious automata. In Peng Ning, Sabrina De Capitani di Vimercati, and Paul F. Syverson, editors, *ACM Conference on Computer and Communications Security*, pages 519–528. ACM, 2007.
24. Lei Wei and Michael K. Reiter. Third-Party Private DFA Evaluation on Encrypted Files in the Cloud. In Sara Foresti, Moti Yung, and Fabio Martinelli, editors, *ESORICS*, volume 7459 of *Lecture Notes in Computer Science*, pages 523–540. Springer, 2012.
25. Andrew Chi-Chih Yao. Protocols for secure computations (extended abstract). In *FOCS*, pages 160–164. IEEE, 1982.

A Basic protocols of Sharemind

The rerandomization protocol of Sharemind is depicted as Alg. 5 and the multiplication protocol as Alg. 6. We have reordered some steps of the protocols in order to have a grouping more relevant to the other algorithms in this paper. All indices of the parties are *modulo* 3.

Algorithm 5: Resharing protocol $[\![w]\!] \leftarrow \mathsf{Reshare}([\![u]\!])$ in SHAREMIND [3]

Data: Value $[\![u]\!]$.
Result: Value $[\![w]\!]$ such that $w = u$ and the components of $[\![w]\!]$ are independent of everything else.

Party P_i generates $r_i \xleftarrow{\$} \mathbb{R}$, sends it to party P_{i+1}
Party P_i computes $[\![w]\!]_i \leftarrow [\![u]\!]_i + r_i - r_{i-1}$

Algorithm 6: Multiplication protocol in the ABB of SHAREMIND [3]

Data: Values $[\![u]\!]$ and $[\![v]\!]$
Result: Value $[\![w]\!]$, such that $w = uv$

1 $[\![u']\!] \leftarrow \mathsf{Reshare}([\![u]\!])$
2 Party P_i sends $[\![u']\!]_i$ to party P_{i+1}
3 $[\![v']\!] \leftarrow \mathsf{Reshare}([\![v]\!])$
4 Party P_i sends $[\![v']\!]_i$ to party P_{i+1}
5 Party P_i computes $[\![w']\!]_i \leftarrow [\![u']\!]_i \cdot [\![v']\!]_i + [\![u']\!]_i \cdot [\![v']\!]_{i-1} + [\![u']\!]_{i-1} \cdot [\![v']\!]_i$
6 $[\![w]\!] \leftarrow \mathsf{Reshare}([\![w']\!])$

Appendix E

Verifiable Computation in Multi-Party Protocols with Honest Majority

The paper "Verifiable Computation in Multi-Party Protocols with Honest Majority" [35] follows.

Verifiable Computation in Multiparty Protocols with Honest Majority

Peeter Laud Alisa Pankova

Tuesday 28th January, 2014

Abstract

A lot of cryptographic protocols have been proposed for semi-honest model. In general, they are much more efficient than those proposed for the malicious model. In this paper, we propose a method that allows to detect the parties that have violated the protocol rules after the computation has ended, thus making the protocol secure against covert attacks. This approach can be useful in the settings where for any party it is fatal to be accused in violating protocol rules. In this way, up to the verification, all the computation can be performed in semi-honest model, which makes it very efficient in practice. The verification is statistical zero-knowledge, and it is based on linear probabilistically checkable proofs (PCP) for verifiable computation. Each malicious party is detected with probability $1 - \varepsilon$ for a negligible ε that is defined by the failure of the corresponding linear PCP. The initial protocol has to be executed only once, and the verification requires in total 3 additional rounds (if some parties act dishonestly, in the worst case they may force the protocol to substitute each round with 4 rounds, due to the transmission functionality that prevents the protocol from stopping). The verification also ensures that all the parties have sampled all the randomness from an appropriate distribution. Its efficiency does not depend on whether the inputs of the parties have been shared, or each party uses its own private input.

The major drawback of the proposed scheme is that the number of values sent before and after the protocol is exponential in the number of parties. Nevertheless, the settings make the verification very efficient for a small number of parties.

1 Introduction

The semi-honest and the malicious model are the two main models in which cryptographic protocols are studied. In the semi-honest model, the adversary is curious about the values it gets, and it tries to extract information out of them, but it follows the protocol rules honestly. In the malicious model, the adversary is allowed to do whatever it wants. In addition to these traditional models, a notion of covert security was proposed in [AL10]. In this model, the adversary is malicious, but it will not cheat if it will be caught with a non-negligible probability, which can be defined more precisely as a security parameter. This notion is very realistic in many computational models, where the participants care about their reputation and will not cheat even if this probability is not close to 1.

Some works have been dedicated to covert security [Lin13, DGN10], where [Lin13] treats the security for two-party computation based on garbled circuits, both the covert and the malicious cases, and [DGN10] deals with honest majority protocols for an arbitrary number of parties. A more precise definition of *covert security with public verifiability* has been proposed in [AO12]. This allows the cheater to be blamed publicly.

2 Our Contribution

In this paper, we propose a scheme that is based on succinct computation verification. Our work is closely related to [DGN10] that is dealing with honest majority protocols for an arbitrary number of parties. The solution proposed in [DGN10] is based on running the initial protocol on two inputs, the real shares and the dummy shares. In this case, the real shares should be indistinguishable from random, and hence

in the beginning the protocol is being rewritten to a shared form. Differently from [DGN10], our solution does not require rewriting the original protocol. The original protocol has to be run only once, and each malicious party is detected with probability $1 - \varepsilon$ for a negligible ε. Our approach is statistical zero-knowledge, and it it based on linear probabilistically checkable proofs (PCP) for verifiable computation. The particular PCP that we are using is the one proposed in [BSCG$^+$13]. The quantity ε is defined by the failure of the corresponding linear PCP behind the protocol. Additionally, the verification ensures that all the parties have sampled all the randomness from an appropriate distribution. If everyone is indeed honest, the verification requires in total 3 additional rounds. If some parties are dishonest, then in the worst case they may force the protocol to substitute each round with 4 rounds, due to the transmission functionality.

One question is whether there are indeed no additional rounds if everyone is honest, since there should be some time offset when the parties wait for possible complaints, and if nothing happens, only then they proceed to the next round. Since there are no complaints if everyone is honest, there is no communication during that time offset, and hence such additional rounds are much cheaper.

Another solution is to assume that the complaints can be presented not immediately, but on the next round, when instead of ordinary messages some party may sent complaints it had on the previous round. In this way, if everyone acts honestly, we will have just 3 additional rounds compared to the initial protocol (2 rounds are needed for the verification, and one more is the final time offset in the end of the protocol, where the parties should wait for the possible lastmost complaints).

The major drawback of our scheme is that the number of values sent per one round is exponential in the number of parties. In [DGN10], efficiency is achieved by reducing the probability of being detected from 1/2 to 1/4. We cannot use the same approach in our case since the probability of being detected would immediately become negligible. Nevertheless, the settings make the verification very efficient for a small number of parties.

Similarly to [DGN10], we prove the security of our scheme in UC model [Can01].

2.1 Notation

Throughout this work, we use the following notation:

- the upper case letters A denote matrices;
- the bold lower case letters $\mathbf{b}$ denote vectors;
- $\langle \mathbf{a}, \mathbf{b} \rangle$ denotes the scalar product of $\mathbf{a}$ and $\mathbf{b}$;
- $(\mathbf{a}||\mathbf{b})$ is a concatenation of vectors $\mathbf{a}$ and $\mathbf{b}$.

2.2 Assumptions

Our verification protocol is based on security of some other schemes. Here is the list of used assumptions.

- Secure point-to-point channels between each pair of parties.
- Broadcast channels between subsets of parties.
- Honest Verifier Statistical Zero-Knowledge Linear Probabilistically Checkable Proofs for verifiable computation [Lip13, GGPR13, BSCG$^+$13, BCI$^+$13]. In particular, all the complexity estimations in this chapter are based on the solution proposed in [BSCG$^+$13].
- Functionality that allows to prove to third parties which messages one received during the protocol, and to further transfer such revealed messages. This allows to protect the initial protocol from stopping, when the computation cannot proceed due to some malicious party that is either just doing nothing, or causes some other party to wait by sending wrong messages. We use the solution proposed in [DGN10].

2.3 The Protocol Outline

We describe briefly the initial settings, and how the new verifiable protocol differs from the original one.

- In the initial settings, we have a set of arithmetic circuits C_i^j over some finite field $\mathbb{F}$, where C_i^j is the circuit computed by the party M_i on the j-th round of computation. Some outputs of C_i^j may be used as inputs for some C_k^{j+1}, so there is some communication between the parties. Each circuit may use some randomness that comes from random uniform distribution in $\mathbb{F}$ (this is sufficient to model any other distribution). The circuits could be boolean as well, since there also exist linear probabilistically checkable proofs based on boolean circuits [Lip13], so our verification is not restricted to computation over some certain field.
- The computation is performed by n parties. Let them be denoted M_i for $i \in \{1, \ldots, n\}$. A necessary condition is that at least $t = \lfloor n/2 \rfloor + 1$ are honest.
- Before the execution of original protocol starts, the inputs of the parties are committed in a special way. Let the input of the party M_i be represented by a vector $\mathbf{x}_i$ over $\mathbb{F}$. M_i represents $\mathbf{x}_i$ as $\binom{n-1}{t}$ distinct sums of the form $\mathbf{x}_i = \sum_{k \in T_j} \mathbf{x}_{ikT_j}$ for $j \in \{1, \ldots, \binom{n-1}{t}\}$, where each T_j represents a distinct subset of t other parties, and k corresponds to one particular party in that subset. The idea is that each party has to prove its honestness to any subset of t other parties. All the shares are signed and distributed amongst the corresponding parties. Although the number $\binom{n-1}{t}$ is exponential, computing all $t \cdot \binom{n-1}{t}$ signatures is not less efficient than computing just one, for example using hash Merkle tree.
- The randomness used in the protocols should also be committed in the same way. Moreover, we want to ensure that it indeed comes from random uniform distribution, without revealing to anyone its value.
  - Let an arbitrary set of t parties be responsible for generating the randomness. Let these parties be called "generators". By honest majority assumption, at least one of them is honest. For each M_i, they generate the randomness $\mathbf{r}_i$ as follows. Each generator M_j generates $\mathbf{r}_{ji}$ of the same length that $\mathbf{r}_i$ should be. The idea is to take $\mathbf{r}_i = \mathbf{r}_{i1} + \ldots + \mathbf{r}_{it}$. Since at least one party is honest, the vector $\mathbf{r}_i$ comes from a random uniform distribution.
  - Each generator M_j represents its $\mathbf{r}_{ij}$ as $\binom{n-1}{t}$ distinct sums of the form $\mathbf{r}_{ij} = \sum_{k \in T_\ell} \mathbf{r}_{ijkT_\ell}$ for $\ell \in \{1, \ldots, \binom{n-1}{t}\}$. All the shares are signed and sent to M_i. After M_i receives $\mathbf{r}_{ij}$ from all generators M_j, it may compute the sum of all $\mathbf{r}_{ij}$ and use it as $\mathbf{r}_i$ (M_i has to verify if the shares for different sets T_ℓ indeed all represent the same value). Then M_i signs all the received shares also by itself, and distributes the shares and the signatures (both signed by M_i and the corresponding generator M_j) amongst appropriate subsets of t parties, similarly to $\mathbf{x}_i$.
- The original protocol is computed in the same way as before. Additionally, each communicated vector $\mathbf{c}_{ij}^\ell$ sent by M_i to M_j on the round ℓ is presented as $\binom{n}{t}$ distinct sums $\mathbf{c}_{ij}^\ell = \sum_{k \in T_{j'}} \mathbf{c}_{ijkT_{j'}}^\ell$ (here we have $\binom{n}{t}$ instead of $\binom{n-1}{t}$ since both communicating parties should later verify the consistency of this value from each other). Along with each $\mathbf{c}_{ij}^\ell$, M_j receives the signature of $\mathbf{c}_{ij}^\ell$ and the signatures of all the shares $\mathbf{c}_{ij\ell kT_{j'}}^\ell$. M_j checks if the signatures are all indeed valid, and in turn signs them. M_j distributes the corresponding signatures (both signed by M_i and M_j) amongst each T_j. Here M_j is unable to check whether the shares under the signatures are valid and indeed sum up to $\mathbf{c}_{ij}^\ell$. All the shares will be distributed after the protocol execution, and then M_i may present the signature of $\mathbf{c}_{ij}^\ell$ to complain.
- After the protocol computation ends, all the communication shares are finally distributed. Each party M_j is verified for honestness. A party is honest iff it can prove that it acted according to the protocol, given the signed input, randomness, and communication that it had with the other parties. It has to perform a 3-round interactive proof with each subset of t parties in parallel. Since each subset of t parties holds all the shares of all the committed values, they are able to reconstruct the committed values and check if the proof indeed corresponds to them.

  In general, in a linear PCP the prover has to prove the knowledge of a vector $\pi = (\mathbf{p}||\mathbf{d})$ such that certain combinations of $\langle \pi, \mathbf{q}_i \rangle$ for special challenges $\mathbf{q}_1, \ldots, \mathbf{q}_5$ should be equal to 0, and $\mathbf{d}$ corresponds to the committed values. The problem is that the prover cannot see any of the $\mathbf{q}_i$ before committing the proof, but at the same time π should remain private.

  In particular, for any subset of t verifiers, the following has to be done (ordered by rounds).

  1. The verifiers agree on a random $\tau \in \mathbb{F}$ that is sufficient to generate all $\mathbf{q}_1, \ldots, \mathbf{q}_5$ (in one round). The prover generates shares $\pi = \pi_1 + \ldots + \pi_t$ (where the $\mathbf{d}$ part is shared in the same way as it

was committed to the given set of t verifiers) and distributes them amongst the parties. Each verifier checks if the part that corresponds to $\mathbf{d}$ is consistent with the signatures of shares sent during the computation.

2. Each verifier V_i computes and publishes $\langle \pi_i, \mathbf{q}_j \rangle$ for $j \in \{1, \ldots, 5\}$. The τ is published. Everyone may compute $\langle \pi_1, \mathbf{q}_j \rangle + \ldots + \langle \pi_t, \mathbf{q}_j \rangle = \langle \pi, \mathbf{q}_j \rangle$ for $j \in \{1, \ldots, 5\}$ and locally verify the necessary combinations. The prover checks if all the scalar products are computed correctly, and complains if necessary.

A party is claimed honest iff it succeeds in all the $\binom{n-1}{t}$ proofs against t other parties. This means that even if it was in collaboration with $t-2$ other malicious parties, there exists a subset of t all-honest parties that will definitely accept only the correct proof. We also need to ensure that the presence of malicious parties will not make the proof fail for an honest prover, and this can be done by revealing the signatures that correspond to the shares of incorrect scalar products. An honest party is safe to open them since they are known by the adversary anyway.

3 A Linear Probabilistically Checkable Proof for Verifiable Computation

In this section we describe in more details the PCP that we use in our verification. This will be necessary since we do not use it just as a black box, but commit some parts of the proof in a special way, so we need to ensure that everything still works. Additionally, writing it down allows to estimate the complexity more precisely.

We start from a statistical honest verifier zero knowledge (HVZK) linear PCP based on translating each arithmetic circuit to a quadratic arithmetic program [BSCG$^+$13].

Definition 1 (Linear PCP)**.** *[BCI$^+$13] Let $\mathcal{R}$ be a binary relation, $\mathbb{F}$ a finite field, P_{LPCP} a deterministic prover algorithm, and V_{LPCP} a probabilistic oracle verifier algorithm. We say that the pair (P_{LPCP}, V_{LPCP}) is an input-oblivious k-query linear PCP for $\mathcal{R}$ over $\mathbb{F}$ with knowledge error ε and query length m if it satisfies the following requirements.*

1. ***Syntax.*** *On any input $\mathbf{v}$ and oracle π, the verifier $V_{LPCP}(\mathbf{v})$ makes k input-oblivious queries to π and then decides whether to accept or reject. More precisely, V_{LPCP} consists of a probabilistic query algorithm Q_{LPCP} and a deterministic decision algorithm D_{LPCP} working as follows. Based on its internal randomness τ, and independently of $\mathbf{v}$, Q_{LPCP} generates k query vectors $\mathbf{q_1}, \ldots, \mathbf{q_k} \in \mathbb{F}^m$ to π and state information u. Then, given $\mathbf{v}$, u, and the k oracle answers $a_1 = \langle \pi, \mathbf{q_1} \rangle, \ldots, a_k = \langle \pi, \mathbf{q_k} \rangle$, D_{LPCP} accepts or rejects.*
2. ***Completeness.*** *For every $(\mathbf{v}, \mathbf{w}) \in \mathcal{R}$, the output of $P_{LPCP}(\mathbf{v}, \mathbf{w})$ is a description of a linear function $\pi : \mathbb{F}^m \to \mathbb{F}$ such that $V^{\pi}_{LPCP}(\mathbf{v})$ accepts with probability 1.*
3. ***Knowledge.*** *There exists a knowledge extractor E_{LPCP} such that for every linear function $\pi^* : \mathbb{F}^m \to \mathbb{F}$ if the probability that $V^{\pi^*}_{LPCP}(\mathbf{v})$ accepts is greater than ε then $E^{\pi^*}_{LPCP}(\mathbf{v})$ outputs $\mathbf{w}$ such that $(\mathbf{v}, \mathbf{w}) \in \mathcal{R}$.*

In relation to verifiable computation, this definition is used in the following context:

1. $\mathbf{v}$ is the vector of committed input/randomness/communication/public variables.
2. $\mathbf{w}$ is an extension of $\mathbf{w}$ that contains non-deterministic auxiliary input and the values of intermediate gates.
3. $(\mathbf{v}, \mathbf{w}) \in \mathcal{R}$ iff $\mathbf{w}$ is a valid vector of values that the prover party indeed would have got if it followed the protocol honestly, according to the committed input/randomness/communication/public variables $\mathbf{v}$.

A particular solution proposed in [BSCG$^+$13] is a statistical HVZK. Namely, it means that the answers to the queries $\langle \pi, \mathbf{q}_i \rangle$ do not reveal any information about the input, unless the randomness $\tau \in \mathbb{F}$ that has been used in query generation is chosen in a bad way, which happens with negligible probability. In this particular solution, it is sufficient to generate 5 challenges $\mathbf{q}_1, \ldots, \mathbf{q}_5$.

Let us describe the solution of [BSCG$^+$13] in a bit more details. First of all, it has been shown that any arithmetic circuit can be represented by a quadratic arithmetic program.

Definition 2 (Quadratic Arithmetic Program)**.** *Consider some integers m, n, k such that $n - 1 \geq k$. A* strong quadratic arithmetic program (QAP) *over a field $\mathbb{F}$, denoted $\mathsf{P}(A, B, C)$, consists of three $m \times n$ matrices A, B, C over a field $\mathbb{F}$. P accepts a vector $\mathbf{v} \in \mathbb{F}^k$ iff there exists a vector $\mathbf{w} = (1, w_1, \ldots, w_{n-1})$ such that $(w_1, \ldots, w_k) = \mathbf{v}$ and $A\mathbf{w} \circ B\mathbf{w} = C\mathbf{w}$.*

The relation $R_{\mathsf{P}(A,B,C)}$ is defined as

$$(\mathbf{v}, \mathbf{w}) \in R_{\mathsf{P}(A,B,C)} \iff \mathsf{P}(A, B, C) \text{ accepts on input } \mathbf{v} \ .$$

The problem of program verification is now reduced to the problem of proving the existence of $\mathbf{w}$ such that $(\mathbf{v}, \mathbf{w}) \in R_{\mathsf{P}(A,B,C)}$, where A,B,C are defined by the arithmetic circuits that the parties compute. The values m and n are both $O(|C|)$. More precisely, n is the number of wires in the circuit (all the input/randomness/communication/intermediate variables), and m is the number of multiplication gates.

Preprocessing: Denote $A = (a_{ij})$, $B = (b_{ij})$, $C = (c_{ij})$ for $i \in \{0, \ldots, m-1\}$, $j \in \{0, \ldots, n-1\}$. Let $S = \{\omega^0, \ldots, \omega^{m-1}\} \subseteq \mathbb{F}$ for a principal root of unity ω in $\mathbb{F}$.

Let A_j, B_j, C_j for $j \in \{0, \ldots, n-1\}$ be polynomials of degree $m-1$ defined in such a way that $A_j(\omega^i) = a_{ij}$, $B_j(\omega^i) = b_{ij}$, $C_j(\omega^i) = c_{ij}$. The coefficients of these polynomials can be computed by interpolation, for example using Fast Fourier Transform, and they are of degree $m-1$ since they are defined on m points. Let $\mathbf{A}(x) := (A_0(x), \ldots, A_{n-1}(x))$, $\mathbf{B}(x) := (B_0(x), \ldots, B_{n-1}(x))$, $\mathbf{C}(x) := (C_0(x), \ldots, C_{n-1}(x))$ denote the vectors of corresponding polynomials.

Let $Z_S(x) = \prod_{s \in S}(x - s)$ be an m-degree polynomial over $\mathbb{F}$. In this way, $Z_S(x)$ has exactly m roots which are the elements of S.

The set S and the coefficients of $\mathbf{A}(x)$, $\mathbf{B}(x)$, $\mathbf{C}(x)$, $Z_S(x)$ are published.

Linear PCP Prover $P(\mathbf{v}, \mathbf{w})$: Let $\mathbf{v} \in \mathbb{F}^k$, $\mathbf{w} \in \mathbb{F}^n$. The prover works as follows:

- Let $\delta_A, \delta_B, \delta_C \in \mathbb{F}$ be random field elements.
- Let $A(x)$, $B(x)$, $C(x)$ be polynomials of degree m such that:

$$A(x) := \langle \mathbf{w}, \mathbf{A}(x) \rangle + \delta_A Z_S(x) \ ;$$

$$B(x) := \langle \mathbf{w}, \mathbf{B}(x) \rangle + \delta_B Z_S(x) \ ;$$

$$C(x) := \langle \mathbf{w}, \mathbf{C}(x) \rangle + \delta_C Z_S(x) \ ;$$

  where the degree m comes from the fact that the degree of each polynomial in $\mathbf{A}(x)$, $\mathbf{B}(x)$, $\mathbf{C}(x)$ is $m-1$, and the degree of $Z_S(x)$ is m.
- Let $\mathbf{h} = (h_0, \ldots, h_m)$ be the coefficients of the polynomial

$$H(x) := \frac{A(x)B(x) - C(x)}{Z_S(x)} \ .$$

The algorithm returns the proof $\pi = ((\delta_A, \delta_B, \delta_C)||\mathbf{w}||\mathbf{h})$. It can be done locally by the prover in time $O(|C| \log |C|)$, and the details can be seen in [BSCG+13].

Linear PCP Verifier $V = (Q_{LPCP}, D_{LPCP}(\mathbf{v}, \mathbf{u}, \mathbf{a}))$: The work of the verifier is split into two parts: the query algorithm Q_{LPCP} and the decision algorithm D_{LPCP}.

- Q_{LPCP}: First of all, a random element $\tau \in \mathbb{F}$ is generated. Then the following queries $\mathbf{q}_i \in \mathbb{F}^{3+m+(n+1)=4+m+n}$ are computed:
  1. $\mathbf{q}_1 = ((Z_S(\tau), 0, 0)||\mathbf{A}(\tau)||(0, 0, \ldots, 0))$;
  2. $\mathbf{q}_2 = ((0, Z_S(\tau), 0)||\mathbf{B}(\tau)||(0, 0, \ldots, 0))$;
  3. $\mathbf{q}_3 = ((0, 0, Z_S(\tau))||\mathbf{C}(\tau)||(0, 0, \ldots, 0))$;
  4. $\mathbf{q}_4 = ((0, 0, 0)||(0, 0, \ldots, 0, 0, \ldots, 0)||(1, \tau, \ldots, \tau^m))$;
  5. $\mathbf{q}_5 = ((0, 0, 0)||(1, \tau, \ldots, \tau^k, 0, \ldots, 0)||(0, 0, \ldots, 0))$.

  The state information is $\mathbf{u} := (1, \tau, \tau^2, \ldots, \tau^k, Z_S(\tau))$. The query results are $a_i = \langle \pi, \mathbf{q}_i \rangle$ for $i \in \{1, \ldots, 5\}$. Everything can be computed in $O(|C|)$, the details can be seen in [BSCG+13].
- $D_{LPCP}(\mathbf{v}, \mathbf{u}, \mathbf{a})$: Let $\mathbf{u} = (u_1, \ldots, u_{k+2})$, $\mathbf{a} = (a_1, \ldots, a_5)$. The algorithm accepts iff:

1. $a_1 a_2 - a_3 - a_4 u_{k+2} = 0$,
2. $a_5 - u_1 - \langle \mathbf{v}, (u_2, \ldots, u_{k+1}) \rangle = 0$.

Now our goal is to perform this verification in the end of the computation. It is necessary to compute 5 scalar products $\langle \pi, \mathbf{q}_1 \rangle, \ldots, \langle \pi, \mathbf{q}_5 \rangle$, where π cannot depend on any of the $\mathbf{q}_k$, and at the same time π cannot be revealed to any other party. The values $\langle \pi, \mathbf{q}_k \rangle$ themselves may be revealed since it has been proven in [BSCG$^+$13] that they are already statistical zero-knowledge on the assumption that all the vectors $\mathbf{q}_k$ have been chosen according to the rules. In general, the proposed linear interactive proof can be converted to a zero-knowledge succinct non-interactive argument of knowledge, as is shown for example in [BCI$^+$13]. The problem is that it requires homomorphic encryption, and the number of encryptions has to be linear in the size of the circuit. Additionally, according to the previous description, the only way to check if $\mathbf{w}$ indeed contains $(\mathbf{x}||\mathbf{r}||\mathbf{c})$ that correspond to the committed shares is to put $(\mathbf{x}||\mathbf{r}||\mathbf{c})$ into $\mathbf{v}$, but we would not like to make these values public. We propose a solution that uses the honest majority assumption instead of homomorphic encryption, and that is more suitable to our settings.

4 The Proposed Protocol

In this section we describe our protocol in details. First, we list the complexity overheads, based on the particular PCP that has been presented in the previous section. Then we define the ideal functionality that we would like to get (which is very similar to the one proposed in [DGN10]), and describe the behaviour of each party in the real protocol. We prove by simulation that our real functionality is as secure as the ideal functionality.

4.1 Properties

In our settings, we have n parties M_i. Compared to the original protocol, for each M_i the proposed solution has the following computational overheads.

- Let $p = \binom{n-2}{t-1}$. This is the number of t-sets in which one party participates as a verifier. If everyone is honest, then in order to verify M_j's honestness, M_i has to send the following messages:
  - In the initial protocol, send two signatures and $tp + p$ vectors of length $O(|C|)$ to each of the $n-1$ parties (tp for the randomness, and p for the inputs).
  - During the protocol execution, in addition to the original protocol communication, send $r \cdot (1 + \cdot(n-1)) = rn$ signatures to each of the $n-1$ receiver parties, where r is the number of rounds. Each receiver M_j produces rn more signatures of the same values. All these signatures are sent by each receiver M_j to the $n-1$ remaining parties (including M_i).
  - After the protocol execution, compute locally the auxiliary values for the proof in $O(|C| \log |C|)$ steps, as shown in [BSCG$^+$13]. Send to each of the other $n-1$ parties in parallel $2(n-1)$ signatures and $2p(n-1)$ vectors of length $O(|C|)$: $p(n-1)$ for intermediate variables (for each proof separately), and $p(n-1)$ for communication, each set of p vectors signed with one signature.

  As a verifier, in the verification process each M_i has to do the following:
  - Locally generate, sign and broadcast a random element of $\mathbb{F}$.
  - Locally generate p state informations $\mathbf{u}$ and $5p$ challenge vectors $\mathbf{q}_k$ of length $O(|C|)$ (according to the arithmetic circuit). This can be done in $O(|C|)$ steps, as shown in [BSCG$^+$13].
  - Locally sum up and concatenate pt vectors of length $O(|C|)$ that correspond to the randomness: for each of the p verifier sets, sum up all the t vectors.
  - Locally concatenate 4 vectors of total result length $O(|C|)$: the shares of the input, randomness, communication, and the intermediate values. This is done p times, for each verification set.
  - Locally compute $5p$ scalar products of vectors of length $O(|C|)$ and broadcast them (5 for each proof).
  - In the end, compute a constant number of local operations based on these scalar products: 2 multiplications, 3 additions, 1 scalar product of length $O(|\mathbf{v}|)$ for the part of the input $\mathbf{v}$ whose

value is public (which is in general just the constant 1), all operations in $\mathbb{F}$. Everything is done p times, for each proof.

- If something goes wrong with the proof of M_j's honestness, then in the worst case each sent message has to be sent in such a way that it is possible to prove afterwards what has been sent to whom. The $\mathcal{F}_{transmit}$ functionality from [DGN10] requires each message to be broadcast to all $n-1$ parties, and then this message should be delivered by each of the $n-2$ remaining parties to the receiver. No additional signatures are needed since we have already considered all of them in the case where everyone acts honestly.

According to the Linear PCP description from [BSCG+13], a dishonest prover may cheat with probability $\frac{2m}{|\mathbb{F}|}$ where m is the number of multiplication gates in the circuit. This means that either the field should be large enough, or the verification should be repeated k times, so that $\left(\frac{2m}{|\mathbb{F}|}\right)^k$ is negligible. All the k verifications can be done in parallel, by generating k sets of challenges instead of one, thus do not increasing the number of rounds at all, and increasing the communication in total by $p(n-1)k$ field elements and $p(n-1)k$ proof vector shares.

4.2 Notation

The circuit in general has the following variables:

- Input vectors: $\mathbf{x}_i$ for the input of each M_i.
- Randomness vectors: $\mathbf{r}_i$ for the randomness of each M_i.
- Public vectors: $\mathbf{v}_i$ for the input of each M_i. This is the part of M_i's input that may have to be public for some reason.
- Output vectors: $\mathbf{y}_i$ which is an output issued by M_i (not needed in verification unless involved in some published value).
- Intermediate gates: $\mathbf{z}_i$ computed by M_i.
- Communication values: $\mathbf{c}_{ij}$ for a total vector of all the values that have been sent from M_i to M_j.

Each party M_j is verified by each subset of t other parties. The idea is that each M_j has to prove that it knows appropriate $\mathbf{z}_j$ such that the computation is consistent with the signatures of the shares of inputs $\mathbf{x}_j$, randomness $\mathbf{r}_j$, and the communicated values $\mathbf{c}_{jk}$ and $\mathbf{c}_{kj}$ for $k \in \{1,\ldots,n\} \setminus \{j\}$.

4.3 Universal Composability

For each of the 5 queries (according to the PCP construction), we need to compute one scalar product of the form $\langle \pi, \mathbf{q}_k \rangle$ where π should be provided by the prover M_j. Here the prover is not permitted to see $\mathbf{q}_k$, and the verifier is not permitted to see π, although the result $\langle \pi, \mathbf{q}_k \rangle$ can by assumption be seen by anyone. Additionally, the vector π cannot be arbitrary, some of its parts should correspond to the committed randomness, input, and the communication variables. In general, this proof can be done by using homomorphic encryption of each coordinate of $\mathbf{q}_k$, but that would be very slow, and additionally we could lose some zero-knowledge assumptions from commitment verification.

We define an ideal functionality that we would like to have, and the functionality of each M_i in our protocol (all of them are symmetric). We prove that our real functionality is as secure as this ideal functionality by simulation.

Existing Ideal Functionalities

We will use some existing ideal functionalities as components.

- $\mathcal{F}_{ppp}$ implements a point-to-point secure channel between any two parties (and the adversary).
- $\mathcal{F}_{bc}$ implements broadcast channel between subsets of parties.
- $\mathcal{F}_{sign}$ allows signature generation and verification.

- $\mathcal{F}_{transmit}$ allows to prove to third parties which messages one received during the protocol, and to further transfer such revealed messages. We use the definition similar [DGN10], which works with message identifiers mid, encoding a sender $s(mid) \in \{1, \ldots, n\}$ and a receiver $r(mid) \in \{1, \ldots, n\}$, and assuming that no mid is used twice. Since in our case the signatures and the broadcasts are treated separately, our definition is formally a bit different but its essence remains the same. The functionality works as follows.
  - When receiving $(\mathbf{transmit}, mid, m)$ from an honest $M_{s(mid)}$, $\mathcal{F}_{transmit}$ sends (mid, m) to $M_{r(mid)}$.
  - On input $(\mathbf{reveal}, mid, T)$ from a party M_j which at any point has either sent or received $(\mathbf{transmit}, mid, m)$, if at least one of $M_{s(mid)}$ and $M_{r(mid)}$ is honest, output (mid, m) to the set of parties T.
  - On input $(\mathbf{reveal}, mid, T)$ from a party M_j, which at any point has either sent or received $(\mathbf{transmit}, mid, m)$, if all the parties $M_{s(mid)}$ and $M_{r(mid)}$ are dishonest, ask m' from the adversary and output (mid, m') to T.

  The accusations are based on the result of $(\mathbf{reveal}, mid, T)$, and they are defined inside the protocol. Differently from [DGN10], we do not send the accusations immediately in the $\mathcal{F}_{transmit}$ definition. The real implementation of this simplified $\mathcal{F}_{transmit}$ is very simple. The sender signs the mid and the contents. If the receiver has not received a message from the sender, or has received a message of wrong form (for example, without the signature), it sends a complaint to everyone. Now the sender has another attempt, but now its message has to be broadcast to all the other parties, so everyone may verify its correctness and send the message further to the receiver, or immediately accuse the sender. In this way, 3 additional rounds are introduced in case of a problem. Since all the messages are signed by the sender, the contents of the transmitted message can be revealed also later. Straightforwardly, this can be done only by the receiver (since only the sender signs the messages), but it is sufficient in our settings, since the sender may still reveal the message by publicly forcing the receiver to present the signatures, and if the receiver refuses to do it, it is claimed guilty. This implementation of $\mathcal{F}_{transmit}$ can be easily extended to broadcasting the message to several parties, just using broadcast channel and defining multiple receivers in $r(mid)$.
- $\mathcal{F}_{ver}$ implements all the algorithms of Linear PCP for verifiable computation (for a certain preagreed field $\mathbb{F}$ in which all the arithmetic circuits are implemented). This functionality is accessed by each party individually, and it does not involve any communication. $\mathcal{F}_{ver}$ reacts to the following commands.
  - $(\mathbf{proof}, \mathbf{v}, \mathbf{z}, \mathbf{x}, \mathbf{r}, \mathbf{c}, C_j)$: Given the arithmetic circuit and the set of all the public/intermediate/ input/randomness/communication variables, in constructs a proof $(\pi||\mathbf{d})$ where $\mathbf{d} = (\mathbf{x}||\mathbf{r}||\mathbf{c})$ corresponds to committed values and π to everything else.
  - $(\mathbf{challenge}, \tau, C_j)$: Given a randomness τ and a circuit C_j, it returns challenge vectors $\mathbf{q_1}, \ldots, \mathbf{q_5}$ such that for any valid proof $(\pi||\mathbf{d})$ generated by $(\mathbf{proof}, \mathbf{v}, \mathbf{z}, \mathbf{x}, \mathbf{r}, \mathbf{c}, C_j)$ for $\mathbf{d} = (\mathbf{x}||\mathbf{r}||\mathbf{c})$, the answer to the query $(\mathbf{verify}, C_j, \tau, \mathbf{v}, \langle(\pi||\mathbf{d}), \mathbf{q_1}\rangle, \ldots, \langle(\pi||\mathbf{d}), \mathbf{q_5}\rangle)$ is true with probability 1.
  - $(\mathbf{verify}, C_j, \tau, \mathbf{v}, a_1, \ldots, a_5)$: Given a randomness τ and some values $\mathbf{v}$ that should be public, check if $a_1, \ldots, a_5$ indeed correspond to a valid proof with respect to τ. A false proof may be accepted with probability $\frac{2m}{|\mathbb{F}|}$, where m is the number of multiplication gates in the circuit.

Transition Function for $\mathcal{F}_{ideal}$

The ideal functionality is secure against an ideal adversary $\mathcal{A}_S$. Let $\mathcal{M} = \{1, \ldots, n\}$ be the set of all parties.

- In the beginning, $\mathcal{F}_{ideal}$ gets from the environment all the arithmetic circuits $(\mathbf{init}, C_1^1, \ldots, C_n^r)$, where C_i^k corresponds to the computation of M_i on the k-th round. All C_i^k are sent to $\mathcal{A}_S$.
- If $\mathcal{F}_{ideal}$ receives $(\mathbf{corrupt}, i)$ from $\mathcal{A}_S$, it sets $\mathsf{evil}_i := \mathsf{true}$.
- A_S may send $(\mathbf{stop}, i_1, \ldots, i_\ell)$ to $\mathcal{F}_{ideal}$ for any corrupted parties $i_1, \ldots, i_\ell$. $\mathcal{F}_{ideal}$ sets $malicious[i_\ell] := 1$ for all i_ℓ. This models straightforward cheating in the initialization phase with the input and the randomness signatures.

- First, $\mathcal{F}_{ideal}$ computes all the messages of the honest parties of the next round by itself, based on the messages received in the previous rounds. If $\mathsf{evil_j} == \mathsf{true}$, then for any message $\mathbf{m}_k^{ij}$, $\mathcal{F}_{ideal}$ does the following:
  - Sends $\mathbf{m}_{ij}^k$ to $\mathcal{A}_S$.
  - Waits for $\mathbf{m}^{*k+1}_{ji_1}, \ldots, \mathbf{m}^{*k+1}_{ji_{|\mathcal{R}|}}$ from $\mathcal{A}_S$ for all the *honest* receivers $\mathcal{R}$. The communication between dishonest parties is not important at the moment. It should just be consistent in the proofs of the sender and the receiver, and that check will be performed in the end.
  - At any round, instead of $\mathbf{m}^{*k}_{ij}$, A_S may send $(\mathbf{stop}, i_1, \ldots, i_\ell)$ to $\mathcal{F}_{ideal}$ for any corrupted parties $i_1, \ldots, i_s$. $\mathcal{F}_{ideal}$ sets $malicious[i_\ell] := 1$ for all i_ℓ. This models straightforward cheating in the initial protocol that will be immediately detected.
- Upon receiving $(\mathbf{run}, \mathbf{x}_1, \ldots, \mathbf{x}_n)$ from the environment, $\mathcal{F}_{ideal}$ computes $C_i^1(\mathbf{x}_i)$ for all honest $i \in \mathcal{M}$, getting the vectors of values $\mathbf{m}_{ij}^1$ that will be used in the next round. On each round, it asks the adversary for the next values, as shown above (initially, it gets from the adversary the inputs $\mathbf{x}_i^*$ of malicious parties). All the $\mathbf{m}_{ij}^{k+1}$ that are computed by the functionality itself are computed based on the messages $\mathbf{m}^{*k}_{ij}$ that have been sent by the adversary and hence may be malicious. This ensures that no honest party will be accused just because someone else has sent to it wrong values.
- After the protocol has finished, $\mathcal{F}_{ideal}$ queries from the adversary all the remaining communication $\mathbf{m}^{*k}_{ji_1}, \ldots, \mathbf{m}^{*k}_{ji_{|\mathcal{R}|}}$ that corresponds to communication between dishonest parties. Then it may compute all the missing $\mathbf{m}_{k+1}^{ji}$ values. If some $\mathbf{m}^{*k+1}_{ji} \neq \mathbf{m}_{ji}^{k+1}$, set $malicious[j] := 1$. Here we assume that even if the computation requires some non-deterministic input, the communicated values are finally deterministic, and the non-determinism is just auxiliary. If is possible to make the security stronger by forcing $\mathcal{A}_S$ to decide on all the $\mathbf{m}^{*k+1}_{ji_1}, \ldots, \mathbf{m}^{*k+1}_{ji_{|\mathcal{R}|}}$ already during the computation, but this will require more communication, since all the shares of $\mathbf{c}_{ij}^\ell$ will have to be distributed already during the protocol execution.
- $\mathcal{F}_{ideal}$ also queries from the adversary a set of messages of the form $(\mathbf{corrupt}, i, j)$ for some i such that $\mathsf{evil_i} == \mathsf{true}$ and some j such that $\mathsf{evil_j} == \mathsf{false}$ (it may choose whether to send them or not). For all i such that $malicious[i] == 1$, the messages should be definitely sent to all j such that $\mathsf{evil_j} == \mathsf{false}$.
- After all the $(\mathbf{corrupt}, i, j)$ messages have been distributed, $\mathcal{A}_S$ sends to $\mathcal{F}_{ideal}$ the outputs $\mathbf{y_j^*}$ of malicious parties M_j. For the honest parties M_i, $\mathcal{F}_{ideal}$ outputs the real output $\mathbf{y_i}$ iff no $malicious[j] := 1$ has ever happened for any $j \in \mathcal{M}$, and otherwise it outputs $(\mathbf{output}, j_1, \ldots, j_k)$ such that for each j_ℓ the messages $(\mathbf{corrupt}, j_\ell, i)$ have been sent for at least t parties i.

Transition Function for M_i

We assume that each M_i maintains the current round in its state, so that in each round it will react just to those calls that are related to the current round. We assume that the number of rounds in the initial protocol is r.

- In the beginning, each M_i gets from the environment all the parts of the entire arithmetic circuit: $(\mathbf{init}, C_1^1, \ldots, C_n^r)$, where C_j^k corresponds to the computation of party M_j on the k-th round. All the C_j^k are also sent to $\mathcal{A}$.
- If any party M_i receives $(\mathbf{rand}, i)$ from environment, it sets $\mathsf{rand} := \mathsf{true}$. This means that given party will participate in randomness generation. The environment defines exactly t such parties.
- If any party M_i receives $(\mathbf{corrupt}, i)$ from $\mathcal{A}$, it sets $\mathsf{evil} := \mathsf{true}$.
- If $\mathsf{evil} == \mathsf{true}$, then upon receiving any message X, M_i does the following:
  - Sends X to $\mathcal{A}$.
  - Waits from $\mathcal{A}$ which messages should be sent to the receivers further, and in which way they should be shared and signed.
- Upon receiving $(\mathbf{preprocess}, \mathbf{x}_i)$ from the environment, the following happens.

– M_i represents its input $\mathbf{x}_i$ as $\binom{n-1}{t}$ distinct sums of the form $\mathbf{x}_i = \sum_{k \in T} \mathbf{x}_{ikT}$ for each subset T of $\mathcal{M} \setminus \{i\}$ of size t. Let $T_1^k, \ldots, T_p^k$ be all the sets to which M_k belongs. For each $k \neq i$, M_i generates a signature

$$sx_{ik} = Sign_{sk_i}(\mathbf{input\_shares}, i, k, \mathbf{x}_{ikT_1^k}, \ldots, \mathbf{x}_{ikT_p^k}) \ ,$$

and sends $(\mathbf{transmit}, (\mathbf{input\_shares}, i, k), (\mathbf{x}_{ikT_1^k}, \ldots, \mathbf{x}_{ikT_p^k}, sx_{ik}))$ to $\mathcal{F}_{transmit}$.

– If $\mathsf{rand} == \mathsf{true}$, M_i generates the randomness $\mathbf{r}_{ij}$ for each M_j. M_i represents its $\mathbf{r}_{ij}$ as $\binom{n-1}{t}$ distinct sums of the form $\mathbf{r}_{ij} = \sum_{k \in T} \mathbf{r}_{ijkT}$. For each $k \neq i$, M_i generates a signature

$$sr_{ijk} = Sign_{sk_i}(\mathbf{rand\_shares}, i, j, k, \mathbf{r}_{ijkT_1^k}, \ldots, \mathbf{r}_{ijkT_p^k}) \ .$$

Then M_i sends $(\mathbf{transmit}, (\mathbf{rand}, i, j), (\mathbf{r}_{ij1T_1^1}, \ldots, \mathbf{r}_{ijnT_p^n}, sr_{ij1}, \ldots, sr_{ijn}))$ to $\mathcal{F}_{transmit}$.

– Upon receiving all the $(\mathbf{transmit}, (\mathbf{rand}, i, j), (\mathbf{r}_{ij1T_1^1}, \ldots, \mathbf{r}_{ijnT_p^n}, sr_{ij1}, \ldots, sr_{ijn}))$, M_j checks if all the shares indeed correspond to the signatures, and if the shares that correspond to different sets T indeed all sum up to the same value. If everything is correct, M_j in turn creates the signatures $Sign_{sk_j}(sr_{ijk})$. For each k, it sends

$$(\mathbf{transmit}, (\mathbf{rand\_shares}, i, j, k), (\mathbf{r}_{ijkT_1^k}, \ldots, \mathbf{r}_{ijkT_p^k}, Sign_{sk_j}(sr_{ijk})))$$

to $\mathcal{F}_{transmit}$.
If something is wrong, $(\mathbf{reveal}, (\mathbf{rand}, i, j), \mathcal{M})$ is sent to $\mathcal{F}_{transmit}$, everyone checks the signatures, and the protocol immediately stops since one malicious party is detected (either M_i or M_j). Each honest party writes $malicious[i] := 1$ or $malicious[j] := 1$ for each cheated party, and goes to the last step of the protocol (the outputs).

– Upon receiving all $(\mathbf{transmit}, (\mathbf{input\_shares}, i, k), (\mathbf{x}_{ikT_1^k}, \ldots, \mathbf{x}_{ikT_p^k}, sx_{ik}))$ and $(\mathbf{transmit}, (\mathbf{rand\_shares}, i, j, k), (\mathbf{r}_{ijkT_1^k}, \ldots, \mathbf{r}_{ijkT_p^k}, Sign_{sk_j}(sr_{ijk})))$ from $\mathcal{F}_{transmit}$, M_k checks if the signatures are valid and correspond to the shares. If something is wrong, $(\mathbf{reveal}, mid, \mathcal{M})$ is sent to $\mathcal{F}_{transmit}$ for each wrong message mid, everyone checks the signatures, and the protocol immediately stops since one malicious party is detected (either the sender M_i was indeed wrong or M_k accused it without reason). Each honest party writes $malicious[i] := 1$ or $malicious[k] := 1$ for each cheated party, and goes to the last step of the protocol.

- Upon receiving $(\mathbf{run}, i)$ from the environment, M_i computes $C_i^1(\mathbf{x}_i, \mathbf{r}_i)$, getting the vectors of values $\mathbf{c}_{ij}^1$ that should be sent to M_j (for each receiver M_j). We will now generalize the behaviour of M_i to an arbitrary round ℓ. Let $\mathcal{R}$ be the set of all the receivers of M_i for the round ℓ. Let $\mathbf{c}_{ij}^\ell$ be vector of values sent by M_i to M_j on the round ℓ. These values are handled similarly to the randomness values, with the difference that the shares are not distributed yet, just their signatures.
  – Generate the shares $\mathbf{c}_{ij}^\ell = \sum_{k \in T} \mathbf{c}_{ijkT}^\ell$.
  – Generate the signature $sc_{ij}^\ell = Sign_{sk_i}(\mathbf{c}_{ij}^\ell)$.
  – For each $k \neq i$, M_i generates a signature

$$sc_{ijk}^\ell = Sign_{sk_i}(\mathbf{message\_shares}, \ell, i, j, k, \mathbf{c}_{ijkT_1^k}^\ell, \ldots, \mathbf{c}_{ijkT_p^k}^\ell) \ .$$

  – M_i sends $(\mathbf{transmit}, (\mathbf{message}, \ell, i, j), (\mathbf{c}_{ij}^\ell, sc_{ij}^\ell, sc_{ij1}^\ell, \ldots, sc_{ijn}^\ell))$ to $\mathcal{F}_{transmit}$
  – Upon receiving all the $(\mathbf{transmit}, (\mathbf{message}, \ell, i, j), (\mathbf{c}_{ij}^\ell, sc_{ij}^\ell, sc_{ij1}^\ell, \ldots, sc_{ijn}^\ell))$ for the current round ℓ, M_j checks if $\mathbf{c}_{ij}^\ell$ indeed corresponds to the signature sc_{ij}^ℓ. If everything is correct, M_j in turn creates the signatures $Sign_{sk_j}(sc_{ijk}^\ell)$. For each k, it sends

$$(\mathbf{transmit}, (\mathbf{message\_shares}, \ell, i, j, k), Sign_{sk_j}(sc_{ijk}^\ell))$$

to $\mathcal{F}_{transmit}$.
  – If something is wrong inside some message, the corresponding mid is revealed. At the moment no one can check to which shares the signatures $Sign_{sk_j}(sc_{ijk}^\ell))$ actually correspond. This check will be done later.

- During the verification phase, M_i acts at once as a prover, and as a verifier in each of the p subsets for each of the other $n-1$ parties. First, each subset T agrees on its own $\mathbf{q}_{1Tj}, \ldots, \mathbf{q}_{5Tj}$ for each prover M_j. For each subset T that is verifying honestness of some party M_j, $M_i \in T$ generates random τ_{Tij} and sends all $(\mathbf{bc}, T, (i, j, T, \tau_{Tij}, Sign_{sk_i}(\tau_{Tij})))$ to $\mathcal{F}_{bc}$. If some party sends an incorrect signature, or refuses to participate, any verifier of T is allowed to broadcast $(\mathbf{bc}, \mathcal{M}, (\mathbf{end}, j, T))$. Since either the signer or the sender is guilty, every honest party may now believe that continuing this T-set proof is senseless since it contains at least one malicious verifier, and the prover M_j is considered to have passed it.
- Upon receiving all $(\mathbf{bc}, T, (k, j, T, \tau_{Tkj}, Sign_{sk_k}(\tau_{Tkj})))$ from $\mathcal{F}_{bc}$, M_i sums up all the received τ_{Tkj} with its own τ_{Tij}. For each T, j, this will give a unique randomness $\tau_{T,j}$ (since at least one party is honest, it is indeed distributed uniformly and randomly). M_i sends $(\mathbf{challenge}, \tau_{T,j}, C_j)$ to $\mathcal{F}_{ver}$ and gets back $\mathbf{q}_{1Tj}, \ldots, \mathbf{q}_{5Tj}$.
- As a prover, each M_i sends $(\mathbf{proof}, \mathbf{v}_i, \mathbf{z}_i, \mathbf{x}_i, \mathbf{r}_i, \mathbf{c}_i, C_i)$ to $\mathcal{F}_{ver}$ and receives back a vector $(\pi_i || \mathbf{d}_i)$ with $\mathbf{d}_i = (\mathbf{x}_i || \mathbf{r}_i || \mathbf{c}_i)$ that contains all the necessary proof of C_j computation. For the prover's security, it is preferable to use new randomness in π_i in different proofs, so M_i generates a distinct π_{iT} for each subset T of the verifiers. M_i generates the shares such that $\pi_{iT} = \sum_{k \in T} \mathbf{p}_{ikT}$. It generates the signatures $sp_{ik} = Sign_{sk_i}(\mathbf{proof\_share\_1}, i, k, \mathbf{p}_{ikT_1^k}, \ldots, \mathbf{p}_{ikT_p^k})$. For each $k \in T$, M_i sends $(\mathbf{transmit}, (\mathbf{proof\_share\_1}, i, k), (\mathbf{p}_{ikT_1^k}, \ldots, \mathbf{p}_{ikT_p^k}, sp_{ik}))$ to $\mathcal{F}_{transmit}$.
- As a sender, M_i has to finally distribute amongst the corresponding verifiers all the shares $\mathbf{c}_{ijkT} = (\mathbf{c}^1_{ijkT} || \ldots || \mathbf{c}^r_{ijkT})$ for all the messages it has sent in the initial protocol. Here M_i has to broadcast them in such a way that one copy is sent to the receiver M_j, so that M_j may verify if the shares indeed correspond to $Sign_{sk_i}(\mathbf{c}^1_{ij}), \ldots, Sign_{sk_i}(\mathbf{c}^r_{ij})$ that it has already received during the computation. It generates a signature $sc_{ijk} = Sign_{sk_i}(sc^1_{ijk}, \ldots, sc^r_{ijk})$ from the same signatures sc^ℓ_{ijk} that he has already sent on the round ℓ. M_i sends $(\mathbf{transmit}, (\mathbf{proof\_share\_2}, i, j, k), (\mathbf{c}_{ijkT_1^k}, \ldots, \mathbf{c}_{ijkT_p^k}, sc_{ijk}))$ to $\mathcal{F}_{transmit}$, which knows that the message with such a *mid* should be broadcast to both j and k at once. Such a message is transmitted for each $k \neq i$.
- Upon receiving all $(\mathbf{transmit}, (\mathbf{proof\_share\_2}, i, j, k), (\mathbf{c}_{ijkT_1^k}, \ldots, \mathbf{c}_{ijkT_p^k}, sc_{ijk}))$ from $\mathcal{F}_{transmit}$, the corresponding receiver M_j checks if the shares correspond to the signatures that it has collected during the initial protocol computation. If at least one of the shares is wrong, then M_j sends $(\mathbf{reveal}, mid, \mathcal{M})$ for all the shares provided by M_i (not only the incorrect ones), and additionally broadcasts $(\mathbf{reveal}, (\mathbf{message}, \ell, i, j), \mathcal{M})$ for the corresponding round ℓ, so that everyone may compute the incorrect segment of $\mathbf{c}_{ij}$ from the shares and verify the signatures. All the parties may now decide whether the sender or the receiver is wrong.
  - If the receiver M_j is guilty:
    - The proof of M_j ends with failure, and each honest party sets $malicious[j] := 1$.
    - The proof of M_i continues as before, on the assumption that the committed communication is the shared one.
  - If the sender M_i is guilty:
    - The proof of M_i ends with failure, and each honest party sets $malicious[i] := 1$.
    - The verification for M_j continues on the assumption that $Sign_{sk_i}(\mathbf{c_{ij}})$ are the committed values, since it is the sender who has provided wrong signatures.
- Upon receiving all the

$$(\mathbf{transmit}, (\mathbf{proof\_share\_1}, i, k), (\mathbf{p}_{ikT_1^k}, \ldots, \mathbf{p}_{ikT_p^k}, sp_{ik}))$$

  and all the

$$(\mathbf{transmit}, (\mathbf{proof\_share\_2}, i, j, k), (\mathbf{c}_{ijkT_1^k}, \ldots, \mathbf{c}_{ijkT_p^k}, sc_{ijk}))$$

  from $\mathcal{F}_{transmit}$ from all the provers and all the senders, M_i verifies if all the shares are consistent with the signatures it has collected during the initial protocol computation. If something is wrong, $(\mathbf{reveal}, mid, \mathcal{M})$ is sent for all the inconsistent shares and signatures, so all the parties can check them. Each sender of the wrong messages M_i is malicious, and its proof ends with $malicious[i] := 1$. If some receiver M_j does not complain about wrong shares, then it is also malicious, and its proof

also ends with $malicious[j] := 1$ (this check can be actually performed already on the next round, then the other parties will not have to wait until M_j complains).

Let $\mathbf{d}_{ikT}$ denote the concatenation of all the shares of $\mathbf{x}_i$, $\mathbf{r}_i$, and $\mathbf{c}_i$ that are intended for the verifier M_k and the prover M_i, when it participates in the proof set T. More precisely, since each $\mathbf{r}_i = \mathbf{r}_{1i} + \ldots + \mathbf{r}_{ti}$, we may more formally define $\mathbf{d}_{ikT} = (\mathbf{x}_{ikT}||(\mathbf{r}_{1ikT} + \ldots + \mathbf{r}_{tikT})||\mathbf{c}_{ijkT})$. They are distributed in such a way that $\sum_{k \in T} \mathbf{d}_{ikT} = \mathbf{d}_i$.

If all the signatures have been correct, and no receiver has complained that the shares are wrong, M_k sends $(\mathbf{bc}, \mathcal{M}, (\mathbf{product\_share}, i, j, T, \langle(\mathbf{p}_{ikT}||\mathbf{d}_{ikT}), \mathbf{q}_{1Ti}\rangle, \ldots, \langle(\mathbf{p}_{ikT}||\mathbf{d}_{ikT}), \mathbf{q}_{5Ti}\rangle))$ to $\mathcal{F}_{bc}$.

- Upon receiving all the scalar products for all parties in T from $\mathcal{F}_{bc}$ for the prover M_j, each party M_i in T broadcasts $(\mathbf{bc}, \mathcal{M}, (\mathbf{publish\_challenge}, k, T, \tau_{T1j}, \ldots, \tau_{Tpj}, Sign_{sk_1}(\tau_{T1j}), \ldots, Sign_{sk_p}(\tau_{Tpj})))$.
- Upon receiving all **publish_challenge** messages, M_i selects the one in which all the signatures are correct (at least one should be since at least one party is honest). If there are several valid signatures, then the signer of multiple values M_k is definitely malicious and should be punished, so the protocol ends with $(\mathbf{bc}, \mathcal{M}, (\mathbf{end}, i, T))$, and the honest parties assume that M_i has passed the test since one of the verifiers was dishonest, so each honest party sets $malicious[k] := 1$ for each multiple signer M_k. After the correct τ is found, M_i verifies all the previously published scalar products. For any incorrect scalar product computed by some M_k, it sends $(\mathbf{reveal}, (\mathbf{proof\_share\_1}, j, i), \mathcal{M})$ and $(\mathbf{reveal}, (\mathbf{proof\_share\_2}, j, i, k), \mathcal{M})$ to $\mathcal{F}_{transmit}$, so now everyone in M may verify the signatures and compute the corresponding scalar products by itself.
- Each party M_i in $\mathcal{M}$ sums up the appropriate accepted scalar products (it may compute the scalar products for the published shares by itself), gets $a_1, \ldots, a_5$, and sends $(\mathbf{verify}, C_j, \tau, \mathbf{v}, a_1, \ldots, a_5)$ to $\mathcal{F}_{ver}$. If the answer is 1, then M_i accepts the proof of M_j for the given T. Otherwise it immediately sets $malicious[j] := 1$.
- After the protocol has finished for all T, j proofs, each M_i sees for which parties it has set $malicious[j] = 1$. If there is at least one party that is malicious, an honest M_i outputs $(\mathbf{output}, j_1, \ldots, j_k)$ for all parties j_ℓ that are malicious.

Transition Function for the Simulator S

In this subsection we prove that the real functionality is as secure as the ideal functionality. The S is located between $\mathcal{F}_{ideal}$ and $\mathcal{A}$, and it tries to convince $\mathcal{A}$ that it is a real functionality. At the same time for $\mathcal{F}_{ideal}$ it must be like if it communicated with an ideal adversary $\mathcal{A}_S$ who is not completely evil.

Since S simulates the communication between $\mathcal{A}$ and all the M_i, it should know their communication keys for $\mathcal{F}_{ppp}$ for all the M_i.

- In the beginning, $\mathcal{F}_{ideal}$ gets from the environment all the arithmetic circuits $(\mathbf{init}, C_1^1, \ldots, C_p^r)$, where C_i^k corresponds to the computation of party M_i the k-th round. All C_i^k are sent to S. The S just delivers them to $\mathcal{A}$.
- If S receives $(\mathbf{corrupt}, i)$ from $\mathcal{A}$ (that was intended for M_i), it sends $(\mathbf{corrupt}, i)$ to $\mathcal{F}_{ideal}$. $\mathcal{F}_{ideal}$ sets $\mathsf{evil_i} := \mathsf{true}$. The inputs of malicious parties are delivered by S to $\mathcal{A}$.
- In the real protocol, the parties should commit the input shares and generate the randomness. The $\mathcal{A}$ may propose its own inputs and signatures for dishonest parties. S checks by itself the signatures, as if it was an honest party. If anything is wrong, S sends $(\mathbf{stop}, i_1, \ldots, i_\ell)$ to $\mathcal{F}_{ideal}$, for all the detected parties. If the receiver is malicious, then the adversary decides whether anything should be revealed or not.
- From the name of honest parties, S should generate the shares of the randomness and the inputs by itself. Since the adversary gets up to $t-1$ shares of each value, and $t-1$ look completely random unless the last one is obtained, S may generate completely random shares, and they will not be inconsistent with the randomness and the inputs that should have been provided for the honest parties by $\mathcal{F}_{ideal}$. S signs all the shares by itself.
- For the dishonest parties M_i, $\mathcal{F}_{ideal}$ generates $\mathbf{r}_i$ by itself and shows it to the adversary. Here $\mathbf{r}_i$ should indeed be random. S needs to enforce the same $\mathbf{r}_i$ to be used in the real model. $\mathcal{A}$ may generate up to $t-1$ shares for the parties that it controls, but at least one share is generated by an honest party. From the name of the remaining honest parties M_j, S generates randomness $\mathbf{r}_{ij}$

in such a way that the sum of all the t shares equals $\mathbf{r}_i$ provided by the ideal functionality. If $\mathcal{A}$ has provided inconsistent shares for some $\mathbf{r}_{ki}$, the shares of $\mathbf{r}_{ji}$ should nevertheless be consistent. Hence S has to achieve $\mathbf{r}_i$ only for at least one all-honest t-set, and the others do not matter. Let that honest t-set be denoted H. S will now assume that H holds all the commitments, just to avoid confusion using several all-honest t-sets at once.

If we do not want to use the random oracle assumption, recovering $\mathbf{r}_{ki}$ from the shares is possible only if S receives the shares, not just the signatures (and in H all the shares indeed correspond to the signatures, so there are no contradictions). Since committing the inputs is just a preprocessing phase, the shares of $\mathbf{r}_i$ are distributed immediately.

- $\mathcal{F}_{ideal}$ starts running the protocol, and it immediately waits for the input $\mathbf{x}^*{}_i$ that should be queried from $\mathcal{A}_S$. S should take H and define $\mathbf{x}^*{}_i$ to be the vector committed to H as shares. This will be considered the proper committed input. If the shares of $\mathbf{x}^*{}_i$ are not distributed already in the beginning, there is no way for S to recover $\mathbf{x}^*{}_i$ from the signatures, it would only be possible in the random oracle model. Hence if we want to avoid this assumption, the shares should be distributed immediately in the preprocessing phase.
- At some moment $\mathcal{F}_{ideal}$ reaches the place where some $\mathbf{m}^{*k}_{ji}$ should be queried.
  - First, $\mathcal{F}_{ideal}$ computes all the messages of the next round by itself, based on the messages received in the previous rounds. If $\mathsf{evil_j} == \mathsf{true}$, then for any message $\mathbf{m}^k_{ij}$, $\mathcal{F}_{ideal}$ does the following:
    - Sends $\mathbf{m}^k_{ij}$ to S. For the honest parties, S composes all the necessary shares of this message by itself, and signs them from the name of M_i. It delivers the message shares to $\mathcal{A}$ through $\mathcal{F}_{ppp}$, pretending to be M_i.
    - $\mathcal{F}_{ideal}$ waits for $\mathbf{m}^{*k+1}_{ji_1}, \ldots, \mathbf{m}^{*k+1}_{ji_{|\mathcal{R}|}}$ from S for all the receivers $\mathcal{R}$. At the same time, $\mathcal{A}$ knows that in the real functionality M_i should wait for the shares and the signatures. It sends all the shares and the signatures to S, and it gives the orders to the malicious parties, which values they should sign by themselves. For each receiver i_ℓ, the signatures are not supposed to be valid and correspond to the message itself. From the name of honest receivers, S sends $(\mathbf{reveal}, mid, \mathcal{M})$ for all the messages that the honest parties would indeed reject, and since rejection is related just to checking the signatures of the malicious parties, S is able to do it itself. S sends $(\mathbf{stop}, i_1, \ldots, i_\ell)$ to $\mathcal{F}_{ideal}$, for all the detected parties of that round. If both the sender and the receiver are malicious, then the adversary decides whether the message should be accepted.
      Now S has to decide what the $\mathbf{m}^{*k+1}_{ij}$ should be, since $\mathcal{F}_{ideal}$ is waiting for it from $\mathcal{A}_S$.
      - Since each honest party uses $\mathbf{c}^k_{ij}$ it has received, and this value corresponds to the signature that each honest party presents to defend itself, S sets $\mathbf{m}^{*k+1}_{ij} = \mathbf{c}^k_{ij}$ for all honest receivers M_j.
      - For the dishonest receivers, S still takes $\mathbf{m}^{*k+1}_{ij} = \mathbf{c}^k_{ij}$ for all the honest senders since then $\mathbf{c}^k_{ij}$ indeed corresponds to both the shared and the non-shared commitment.
      - If both the sender and the receiver are dishonest, then then $\mathcal{F}_{ideal}$ does not wait for $\mathbf{m}^{*k+1}_{ij}$ since it does not need it in the next computation (the outputs of M_j in the next round are anyway defined by the adversary). $\mathcal{F}_{ideal}$ asks for these values later.

      Hence for the communications where at least one party is honest, $\mathbf{m}^{*k+1}_{ij} = \mathbf{c}^k_{ij}$ is sent to $\mathcal{F}_{ideal}$.
- After the simulation of the initial protocol has finished, $\mathcal{F}_{ideal}$ queries from the adversary a set of messages of the form $(\mathbf{corrupt}, i, j)$ for some i such that $\mathsf{evil_i} == \mathsf{true}$ and some j such that $\mathsf{evil_j} == \mathsf{false}$. It also waits for the final decision on $\mathbf{m}^{*1}_{ij}, \ldots, \mathbf{m}^{*r}_{ij}$ for malicious parties
  - As a verifier, $\mathcal{A}$ chooses the share τ_{Tij} for each corrupted i, for each other j who acts as a prover, for each verifier subset T. It sends these values to S. Now S may broadcast all $(\mathbf{bc}, T, (k, j, T, \tau_{Tkj}, Sign_{sk_i}(\tau_{Tij})))$, simulating the random shares of the remaining honest parties by itself. If $\mathcal{A}$ says that some M_k refuses to participate or presents incorrect signatures for M_j, S broadcasts $(\mathbf{bc}, \mathcal{M}, (\mathbf{end}, j, T))$ from the names of all honest parties in T, assuming that the prover M_j is honest. If there are no problems, S may send all $(\mathbf{challenge}, \tau_{Tj}, C_j)$ to $\mathcal{F}_{ver}$ and get back all $\mathbf{q}_{1Tj}, \ldots, \mathbf{q}_{5Tj}$.

– As a prover, for each T, $k \in T$, $\mathcal{A}$ selects

$$(\textbf{transmit}, (\textbf{proof\_share\_1}, i, k), (\mathbf{p}_{ikT_1^k}, \ldots, \mathbf{p}_{ikT_p^k}, sp_{ik}))$$

for M_k from the name of malicious M_i. As a sender, it selects

$$(\textbf{transmit}, (\textbf{proof\_share\_2}, i, j, k), (\mathbf{c}_{ijkT_1^k}, \ldots, \mathbf{c}_{ijkT_p^k}, sc_{ijk})) \ .$$

S waits until all of them come, or until timeout (since $\mathcal{A}$ may force some M_i to refuse to participate in verification). Then, from the name of each honest party, S checks if the shares correspond to the signatures. If all the signatures are correct, S sends

$$(\textbf{bc}, \mathcal{M}, (\textbf{product\_share}, i, k, T, \langle(\mathbf{p}_{ikT}||\mathbf{d}_{ikT}), \mathbf{q}_{1Ti}\rangle, \ldots, \langle(\mathbf{p}_{ikT}||\mathbf{d}_{ik1}), \mathbf{q}_{5Ti}\rangle))$$

to $\mathcal{F}_{bc}$, where $\mathbf{d}_{ik1}$ is constructed from the received values, as in description of M_i. If some signature is wrong, S sends $(\textbf{reveal}, mid, \mathcal{M})$ to $\mathcal{F}_{transmit}$ for a corresponding mid. From each honest M_i, S broadcasts $(\textbf{bc}, \mathcal{M}, (\textbf{end}, j, T))$, and stores $(\textbf{corrupt}, j, i)$. For a malicious receiver, the adversary decides whether it reveals the message or not.

– From the side of the honest provers, the verifier should generate all the shares of the proof by itself. All the inner communication shares between honest parties, and all the input/randomness shares that have never been seen by S, are generated randomly and signed by S from the names of corresponding honest parties. Since in any T there are at most $t-1$ dishonest parties, the adversary sees only up to $t-1$ shares which look completely random unless the last t-th share is obtained, and hence there are no inconsistencies with any real proof that S has never seen.

– The $\mathcal{A}$ decides on the scalar products for dishonest parties. The S just broadcasts these values from the names of the corresponding parties. If $\mathcal{A}$ decides that some party M_j refuses to broadcast, S simulates the end of the set T proof, writing $(\textbf{corrupt}, j, k)$ for all honest parties M_k.

– From the name of each honest party M_i, S has to generate scalar products by itself. The problem is that the sum of final shared scalar products should be equal to the real $\langle(\pi_{iT}||\mathbf{d}_{iT}), \mathbf{q}_{kTi}\rangle$. If $(\pi_{iT}||\mathbf{d}_{iT})$ belongs to an honest verifier, then it cannot be generated by S itself. However, it is known that, for an honest verifier, $\langle(\pi_{iT}||\mathbf{d}_{iT}), \mathbf{q}_k\rangle$ is statistical HVZK (the details can be seen in [BSCG+13]). Since S knows τ, it has access to a trapdoor, and it has enough information about how to generate $a_1, \ldots, a_5$ in such a way that the final proof on certain combinations of a_k would succeed, and their distribution is the same as for real proof. Hence S just generates some random a_k that satisfy this proof, and claims that $a_k = \langle(\pi_{iT}||\mathbf{d}_{iT}), \mathbf{q}_{kTi}\rangle$. Let the shares of the evil parties in the corresponding T-set be $\mathbf{s}_1, \ldots, \mathbf{s}_\ell$ for $\ell \leq t-1$. Their sum must now be equal to some vector $\mathbf{s}$ such that $r_1 + \langle\mathbf{s}, \mathbf{q}_{1Ti}\rangle = a_1, \ldots, r_5 + \langle\mathbf{s}, \mathbf{q}_{5Ti}\rangle = a_5$ where r_i are the sums of scalar products of the honest parties. S just has to distribute all $\mathbf{s}_i$ shares uniformly amongst the evil parties, and compute r_k according to a_k. Since in the real functionality the prover is not supposed to use the same randomness in π several times, π_{iT} is being generated as a completely new random value in each proof separately.

– If the protocol has not ended yet for some T, j, there is at least one party that has broadcast all the necessary scalar product shares, since by assumption at least one party is honest. However, some of the published scalar product shares may nevertheless be malicious. Now the shares of τ_{Tj} and their signatures have to be broadcast. S does that from the side of all the honest parties, and it waits until $\mathcal{A}$ decides on something for dishonest parties. Then S checks the signatures for inconsistencies from the name of each honest party, and broadcasts $(\textbf{bc}, \mathcal{M}, (\textbf{end}, j, T))$ in the case if something is wrong, setting $(\textbf{corrupt}, \ell, k)$ for all the honest parties M_k for all M_ℓ that provided multiple signatures.

  * If the prover has not been corrupted, then S verifies all the scalar products published by corrupted parties by itself. If some signature is wrong, it sends $(\textbf{reveal}, mid, \mathcal{M})$ to $\mathcal{F}_{transmit}$ for the corresponding messages. Since a wrong scalar product (or a wrong published $\mathbf{c}_{ijkT}$ share) comes from a corrupted party, the communication shares and the signatures are known by S. As in the real functionality, S simulates the end of T-set proof, storing $(\textbf{corrupt}, j, k)$ for all honest parties M_k.

* If the prover M_i has been corrupted, S waits until $\mathcal{A}$ announces which products should be claimed wrong. Again, the values from $(\mathbf{reveal}, mid, \mathcal{M})$ should be published, but S knows them since now they are already known by the malicious prover. S simulates the end of T-set proof, storing $(\mathbf{corrupt}, i, k)$ for all honest parties M_k.

– From the name of each honest receiver M_i, S has to send a complaint if the revealed communication shares do not correspond to the committed $\mathbf{c_{ij}}$. For the dishonest receivers, the adversary decides which complaints have to be sent. Now the communication $\mathbf{m}^{*k}_{ij}$ between the honest parties has finally been discovered. If $\mathcal{A}$ decided to reveal $\mathbf{c}_{ij}$, then $(\mathbf{m}^{*1}_{ij}||\ldots||\mathbf{m}^{*r}_{ij}) = \mathbf{c}_{ij}$. Otherwise $(\mathbf{m}^{*1}_{ij}||\ldots||\mathbf{m}^{*r}_{ij})$ is equal to the sum of the shares of H.

– After all the scalar products have been verified, S makes each honest verifier M_i act exactly like in the real protocol. It computes $a_1, \ldots, a_5$ from the shares, and then the certain combinations of these values, and checks if they match, storing $(\mathbf{corrupt}, j, i)$ for each honest party M_i if the proof fails. The decisions of malicious parties are made by the adversary.

– For each honest verifier M_i, S stores the corresponding counters of how many tests there have been in which each prover M_j succeeds. If there is some party M_j that has not succeeded in all the proofs, it stores $(\mathbf{corrupt}, j, i)$. The adversary sends the decisions of dishonest parties.

- $\mathcal{F}_{ideal}$ is still waiting from the adversary a set of messages of the form $(\mathbf{corrupt}, i, j)$ for some i such that $\mathsf{evil_i} == \mathsf{true}$ and some j such that $\mathsf{evil_j} == \mathsf{false}$ Now S should decide on that. Since all the $(\mathbf{corrupt}, j, i)$ have already been generated, S just delivers them to $\mathcal{F}_{ideal}$. During the simulation, S chose to claim corrupt only in the following cases:
  1. A party may be claimed corrupt in the initial protocol, if its sent shares do not correspond to its signatures. This can be done only by malicious parties.
  2. During the verification process, S corrupts only malicious parties, according to the protocol description.
  3. Additionally, we need to ensure that S has not accused any honest parties in the final check. Since a real prover M_j would never accept the scalar product shares computed by evils unless they are computed correctly, the scalar products that have reached the end of the proof are indeed $a_1, \ldots, a_5$, and they have been chosen by S in such a way that the test passes.
  4. The remaining place where honest users could be claimed malicious by $\mathcal{F}_{ideal}$ itself is the inconsistency of $\mathbf{m}^k_{ij}$ and $\mathbf{m}^{*k}_{ij}$, but S delivers malicious messages only from corrupted parties.
- For all i such that $malicious[i] == 1$, the messages should be definitely sent to all j such that $\mathsf{evil_j} == \mathsf{false}$. It is sufficient to show that if an inconsistency of $\mathbf{m}^k_{ij}$ and $\mathbf{m}^{*k}_{ij}$ happens in $\mathcal{F}_{ideal}$, then M_i does not pass the test of H.

  Since passing the test without actually finishing the verification happens only either if some of the verifiers acts dishonestly, or the protocol succeeds up to the final proof, the only way for M_j to pass the test for H is to make $a_1, \ldots, a_5$ accepted in the end. Hence it must have succeeded in generating $a_1, \ldots, a_5$ that correspond to $a_1 = \langle \mathbf{s}_{11}, \mathbf{q}_1\rangle + \ldots + \langle \mathbf{s}_{1t}, \mathbf{q}_1\rangle, \ldots, a_5 = \langle \mathbf{s}_{51}, \mathbf{q}_5\rangle + \ldots + \langle \mathbf{s}_{5t}, \mathbf{q}_5\rangle$. Since all the verifiers in that subset are honest, none of them could provide any information about any $\mathbf{q}_k$ to M_j. Since all the verifiers are honest, they use the shares distributed in the initial protocol consistently, and hence $\mathbf{s}_{1k} = \ldots = \mathbf{s}_{5k} =: (\mathbf{p}_{ikH}||\mathbf{d}_{ikH})$ for each verifier k, so we have $a_1 = \langle(\mathbf{p}_{i1H}||\mathbf{d}_{i1H}), \mathbf{q}_1\rangle + \ldots + \langle(\mathbf{p}_{itH}||\mathbf{d}_{itH}), \mathbf{q}_1\rangle, \ldots, a_5 = \langle(\mathbf{p}_{i1H}||\mathbf{d}_{i1H}), \mathbf{q}_5\rangle + \ldots + \langle(\mathbf{p}_{itH}||\mathbf{d}_{itH}), \mathbf{q}_5\rangle$ for valid challenges $\mathbf{q}_k$ that have not been seen by M_j before generating $(\mathbf{p}_1||\mathbf{d}_1), \ldots, (\mathbf{p}_t||\mathbf{d}_t)$. Denoting $\mathbf{p} := \mathbf{p}_{i1H} + \ldots + \mathbf{p}_{itH}$ and $\mathbf{d} := \mathbf{d}_{i1H} + \ldots + \mathbf{d}_{itH}$, we get $a_1 = \langle(\mathbf{p}||\mathbf{d}), \mathbf{q}_1\rangle, \ldots, a_5 = \langle(\mathbf{p}||\mathbf{d}), \mathbf{q}_5\rangle$. Since the honest parties would not accept communicated value shares that do not correspond to the signatures, the scalar products are actually of the form $\langle(\pi||\mathbf{x}||\mathbf{r}||\mathbf{c}), \mathbf{q}_1\rangle, \ldots, \langle(\pi||\mathbf{x}||\mathbf{r}||\mathbf{c}), \mathbf{q}_5\rangle$, where $\mathbf{x}$, $\mathbf{r}$, $\mathbf{c}$ indeed correspond to the values committed to the all-honest parties.

  Even if the other t-sets have received different commitments, it is important that these values have been committed to an all-honest-party set before the computation, according to the protocol. The vector $\mathbf{r}$ is indeed random since all-honest-parties have checked carefully all the signatures of the t generators of $\mathbf{r}$, and at least one of them was definitely honest, and hence the vector is indeed random. If the proof succeeds, then $\mathbf{p}$ is a valid proof (according to the PCP description). This implies proving that M_i performed its communication correctly with respect to the committed values. Here the situation with $\mathbf{c}$ is not so clear since it has been committed in two ways. We need to show that satisfying any of these two committed values by M_i implies all $\mathbf{m}^{*k}_{ij} = \mathbf{m}^k_{ij}$, where $\mathbf{m}^k_{ij}$ is computed by $\mathcal{F}_{ideal}$ from the previous round.

- Since S has chosen ${\mathbf{m}^*}_{ij}^{k-1} = \mathbf{c}_{ij}^{k-1}$ where $\mathbf{c}_{ij}^{k-1}$ is accepted by H (it does not matter now which of the two commitments it was), the value $\mathbf{m}_k^{ij}$ equals to the value computed from $\mathbf{c}_{ij}^{k-1}$, the inputs $\mathbf{x}_j$, and the randomness $\mathbf{r}_j$ which have been committed to H.
- Since the proof has succeeded, the computation is correct with respect to $\mathbf{c}_{ij}^{k-1}$, the inputs $\mathbf{x}_j$, and the randomness $\mathbf{r}_j$ committed to H. Hence the output of this computation ${\mathbf{m}^*}_{ij}^{k}$ is the same value $\mathbf{m}_{ij}^k$ that $\mathcal{F}_{ideal}$ would compute.

- After all the $(\mathbf{corrupt}, i, j)$ messages have been distributed, S waits from $\mathcal{A}$ the final outputs of the dishonest parties. It delivers them to $\mathcal{F}_{ideal}$. For the honest parties, $\mathcal{F}_{ideal}$ outputs real output iff no $malicious[j] := 1$ has ever happened in $\mathcal{F}_{ideal}$, and otherwise it outputs $(\mathbf{output}, j_1, \ldots, j_k)$ such that for each j_ℓ the messages $(\mathbf{corrupt}, j_\ell, i)$ has been sent. This is what $\mathcal{A}$ awaits from the output.

5 Using the Proposed Protocol in Secure Multiparty Computation Platforms

In this section we discuss how the proposed verification could be used in Secure Multiparty Computation Platforms. More precisely, here we should consider the case where in addition to *computing* parties (that participate in the protocol) we may have *input* parties (that provide the inputs, sharing them in some way amongst the computing parties) and the *result* parties (that receive the final output). In our protocol, the computing parties do commit the inputs before the computation starts, but we must ensure that these are indeed the same inputs that have been provided by the input parties.

5.1 Treating Inputs/Outputs as Communication

As a simpler solution, we may just handle the input and the output similarly to communication. Hence the following enhancements are made.

1. Let the number of input parties be N. In the beginning, each input party P_i generates the shares $\mathbf{x}_{i1}, \ldots, \mathbf{x}_{in}$ (according to an arbitrary sharing scheme) from all the computing parties $M_1, \ldots, M_n$, as it would do without the verification. Each M_j should now use the input vector $\mathbf{x}_j = (\mathbf{x}_{1j} || \ldots || \mathbf{x}_{Nj})$, where each $\mathbf{x}_{ij}$ is provided by an input party P_i. Now, for each $\mathbf{x}_{ij}$, P_i generates by itself all the $\binom{n-1}{t}$ shares $\mathbf{x}_{ijkT_\ell}$ such that $\sum_{k \in T_j} \mathbf{x}_{ijkT_\ell} = \mathbf{x}_{ij}$, signs all these shares, and sends them to M_j. As before, M_j should also sign all of these shares before redistributing them amongst all the verifier t-sets. The verifiers should now check both signatures, similarly to how it was done to communication.
2. In the end, each receiver party R_i gets the shares $\mathbf{y}_{i1}, \ldots, \mathbf{y}_{in}$ from all the computing parties $M_1, \ldots, M_n$. Now each M_j has to generate $\binom{n-1}{t}$ sums $\sum_{k \in T_j} \mathbf{y}_{ijkT_\ell} = \mathbf{y}_{ij}$, exactly in the same way as it would do with an ordinary communication value. M_j sends the shares and their signatures to P_i, and P_i redistributes them amongst the t-sets. In the verification process, they check both signatures, similarly to how it was done to communication.

Since the parties P_i and R_i do not participate in the computation, they do not have to participate in the verification. However, they will still be punished if they provide multiple signatures for the same value.

5.2 Possible Issues

The main drawback of the previous proposition is the numerous amount of signatures that the computing parties may have to check. While in the initial scheme each party M_j has to provide just one share $\mathbf{x}_{jkT_\ell}$ for each party M_k in each T_ℓ, now it has to provide N shares, where N is the number of input parties, and all their signatures have to be checked (for M_j it is still sufficient to use just one signature, but it does not help much). Depending on the settings, N can be very large. In the worst case, each input party provides only one bit, and hence $N \in O(|C|)$. However, each computing party would have to verify the source of all the inputs anyway. For P_i, sending $t \cdot \binom{n-1}{t}$ shares instead of one is not worse since all the values used by the same P_i may have the same signature. The problem comes when M_j wants to

redistribute the shares and the signatures to all T-sets, since each receiver will again have to check all N of them. Fortunately, this happens only in the beginning and in the end of the protocol.

Additionally, depending on the performed computation, the covert security may just not work with the input parties, especially in some anonymous statistical projects. Any participant may cheat without reason and complain afterwards. In our scheme, the verification of input share signatures is done already in the beginning, an hence the computing parties will not spend their time on clearly malicious parties whose shares do not correspond to their signatures. The problem still remains with the output, since the malicious output party R_i may sign wrong values just for fun, without fear of being detected. However, since such cheating would require just one additional broadcast (revealing the signatures to everyone), this is not too much different from the case if R_i has not complained. In any case, even if no one complains, it may still be some kind of attack where the input party is completely honest, but it just performs the computation without needing. In relation to statistic projects, some input parties may also produce unrealistic artificial inputs that harm the result in general, so some outlier detection would be needed in the beginning.

5.3 Deviations from the Initial Settings

In real Secure Multiparty Computation Platforms, it may happen that the number of input parties is initially unknown. For example, in the case of some statistical computation, the input parties may come and submit their inputs during the execution, and hence the shape of the computational circuit may be even unknown in the beginning, since the input length is undefined. Nevertheless, the proposed techniques still work. The coming input parties may commit the inputs as they come. In the end of the computation, the structure of the circuit will be known anyway. The formal proofs would have to be adjusted, but now we need to define a new ideal functionality that allows the adversary to introduce new input parties during the computation.

6 Conclusions and Future Work

In this paper we have proposed a scheme that allows to verify the computation of each party in a passively secure protocol, thus converting passive security to covert security. Each malicious party will be detected with probability close to 1, depending on the parameters of selected field.

While our verification is being done only after the entire computation has ended, it might be interesting to do something more similar to the active security model. Namely, we could require each party to prove the correctness after each round. If implemented straightforwardly, repeating our verification algorithm on each round, it multiplies the verification complexity by the number of rounds (actually, a bit less since in the beginning the vectors will be of smaller length). Doing it more cleverly, we could make use of the proofs of the previous rounds, making the next proof steps reliable on the proofs of the previous steps. The ideas can be taken for example from [CT10].

References

[AL10] Yonatan Aumann and Yehuda Lindell. Security against covert adversaries: Efficient protocols for realistic adversaries. *J. Cryptology*, 23(2):281–343, 2010.

[AO12] Gilad Asharov and Claudio Orlandi. Calling out cheaters: Covert security with public verifiability. In Xiaoyun Wang and Kazue Sako, editors, *ASIACRYPT*, volume 7658 of *Lecture Notes in Computer Science*, pages 681–698. Springer, 2012.

[BCI+13] Nir Bitansky, Alessandro Chiesa, Yuval Ishai, Rafail Ostrovsky, and Omer Paneth. Succinct non-interactive arguments via linear interactive proofs. In *TCC*, pages 315–333, 2013.

[BSCG+13] Eli Ben-Sasson, Alessandro Chiesa, Daniel Genkin, Eran Tromer, and Madars Virza. Snarks for c: Verifying program executions succinctly and in zero knowledge. In *CRYPTO (2)*, pages 90–108, 2013.

[Can01] Ran Canetti. Universally composable security: A new paradigm for cryptographic protocols. In *FOCS*, pages 136–145. IEEE Computer Society, 2001.

[CT10] Alessandro Chiesa and Eran Tromer. Proof-carrying data and hearsay arguments from signature cards. In Andrew Chi-Chih Yao, editor, *ICS*, pages 310–331. Tsinghua University Press, 2010.

[DGN10] Ivan Damgård, Martin Geisler, and Jesper Buus Nielsen. From passive to covert security at low cost. In Daniele Micciancio, editor, *TCC*, volume 5978 of *Lecture Notes in Computer Science*, pages 128–145. Springer, 2010.

[GGPR13] Rosario Gennaro, Craig Gentry, Bryan Parno, and Mariana Raykova. Quadratic span programs and succinct nizks without pcps. In Thomas Johansson and Phong Q. Nguyen, editors, *EUROCRYPT*, volume 7881 of *Lecture Notes in Computer Science*, pages 626–645. Springer, 2013.

[Lin13] Yehuda Lindell. Fast cut-and-choose based protocols for malicious and covert adversaries. In Ran Canetti and Juan A. Garay, editors, *CRYPTO (2)*, volume 8043 of *Lecture Notes in Computer Science*, pages 1–17. Springer, 2013.

[Lip13] Helger Lipmaa. Succinct non-interactive zero knowledge arguments from span programs and linear error-correcting codes. *IACR Cryptology ePrint Archive*, 2013:121, 2013.

Appendix F

Actively Secure Two-Party Computation: Efficient Beaver Triple Generation

The paper "Actively Secure Two-Party Computation: Efficient Beaver Triple Generation" [43] follows.

University of Tartu
Faculty of Mathematics and Computer Science
Institute of Computer Science

Pille Pullonen

Actively Secure Two-Party Computation: Efficient Beaver Triple Generation

Master's thesis (30 ETCS)

Supervisors: Sven Laur, Ph.D.
Tuomas Aura, Ph.D.
Instructor: Dan Bogdanov, Ph.D.

Author: ..... " ..... " august 2013

Supervisor: ..... " ..... " august 2013

Supervisor: ..... " ..... " august 2013

Instructor: ..... " ..... " august 2013

Allowed for defence

Professor: ..... " ..... " august 2013

Tartu 2013

Contents

Chapter 1

Introduction

1.1 Motivation

Information is among the most important resources of the modern world, allowing to make wiser business choices, estimate future trends or foresee natural catastrophes. The world wide web makes collecting and also sharing information fast and easy, so in an honest world everyone could just combine their data and analyse it as they like. However, as a resource information might mean business secrets or confidential data that should not be freely published, thus, we need methods to process this information without losing privacy.

Secure multi-party computation is a cryptographic tool that enables sharing data for analysis without actually revealing it. For example, consider a chain of food-stores that is interested in learning how well they do compared to other retailers. It could use secure computation techniques to collaborate with other stores in the area to get such aggregated results without leaking their client databases or their behaviours to their competitors.

Each store can expect the others to help the computation as long as they believe that other companies are also interested in getting accurate results. However, there might be a shop that deliberately fiddles with the computations, for example, to trick its competitors into thinking that others have some very popular products that actually no-one buys. The competitors are then likely to increase their supply of these goods and, hence, suffer an economical loss. *Passively* secure computation can be used in the first case, whereas *active* security is needed to ensure that all parties follow the computations correctly. Actively secure secure computation means that the misbehaving shop could not affect the computation outcomes with anything other than its inputs.

Secure computation is currently an active research field and has reached the state where is is efficient enough for practical applications. SHAREMIND is one of the more mature secure multi-party computation frameworks that currently offers *passive* security [11, 12]. This thesis uses the principles also combined in the SPDZ framework [26] to add an *actively* secure protocol set to SHAREMIND framework.

1.2 Contribution of the author

The aim of this work is to adapt the SPDZ general actively secure computation framework [26] for the two-party case and focus on optimising the precomputation phase.

An important distinction between our work and SPDZ is the usage of additively homomorphic cryptosystem instead of the somewhat-homomorphic cryptosystem for the precomputation phase. The resulting protocol set is implemented in SHAREMIND version 3 [53].

The author was responsible for working out the details of the proposed protocol sets, including the share representation, local operations on this representation, and the precomputation phase. In addition, the share representations required protocols to communicate with data donors and data analysts. The author also implemented and benchmarked the proposed asymmetric and symmetric protocol set, as well as the triple generation protocols as part of the SHAREMIND framework.

The main outcomes of this thesis are the implementation of the asymmetric and symmetric protection domains, as well as ideas for generating Beaver triples using additively homomorphic cryptosystem.

1.3 Structure of the thesis

In the following, Chapter 2 gives an overview of tools used to build following secure computation protocols. At first, it summarises the necessary cryptographic concepts and gives specific initialisations of schemes used further in this thesis. Secondly, it provides a short overview of secure multi-party computation and security related issues. Finally, it briefly describes the key aspects of SHAREMIND.

The SPDZ framework is introduced separately in Chapter 3 together with some insights to how the following work uses ideas from it. This chapter focuses on the key principles used in SPDZ and gives additional background for the rest of the thesis.

Our protocol set for two-party computation with asymmetric setup is introduced in Chapter 4. This includes both the necessary protocols and theoretical efficiency analysis.

Chapter 5 introduces some ideas to generate Beaver triples using Paillier cryptosystem. The main focus is on achieving triple generation for arbitrarily chosen modulus.

The online phase of the symmetric protocol set is described in Chapter 6. This chapter also gives hints about achieving suitable precomputation phase using the Beaver triple generation ideas from Chapter 5.

Chapter 7 focuses on the implementation details and benchmark results, giving an overview of the efficiency of our protocols. This helps to summarise and compare the ideas from the rest of the thesis.

Finally, Chapter 8 concludes this thesis and gives additional directions for further work.

Chapter 2

Preliminaries

This chapter introduces the necessary building blocks and background notions for the proposed protocols. At first, Section 2.1 introduces the cryptographic tools used throughout this thesis. Secondly, Section 2.2 gives an introduction to secure multi-party computation and related problems. Thirdly, Section 2.3 introduces the SHAREMIND framework where we set up our proposed protocols.

2.1 Cryptographic primitives

This section introduces different cryptographic notions used for building secure two-party protocols. We introduce the basic concepts and initialise them with the exact instantiations used in the following sections.

2.1.1 Additive secret sharing

Secret sharing schemes are methods of distributing data between participants so that some subsets of the participants are able to restore the initial information using their shares of the data [52]. A secret sharing scheme is a (t, n)-*threshold* scheme if the data is shared among n participants and any subset of $t \leq n$ or more participants is able to restore it. A secret sharing scheme is *correct* if t shares uniquely determine the secret. In addition, the scheme is *private* if any set of $t - 1$ or less shares does not give any information about the secret.

Additive secret sharing is usually defined in a ring $\mathbb{Z}_N$ for some integer $N > 0$. To share a secret $x \in \mathbb{Z}_N$, one defines shares $x_1, \ldots, x_n$ where $x_1 + \ldots + x_n = x$ and all arithmetic is performed in $\mathbb{Z}_N$. This defines an (n, n)-threshold scheme, where a secret is divided to n parts and all of those are needed to restore the secret. Restoring the secret given all the shares is straightforward and only requires summing the shares. Each computing participant $\mathcal{CP}_i$ only receives the value x_i for secret x. Moreover, additive secret sharing is information-theoretically secure, meaning that having access to less than n shares does not reveal any information about the secret value. Such share representation also allows us to perform computations on the shares.

In the following, we use a two-party protocol where a secret $x \in \mathbb{Z}_N$ is distributed to two additive shares where $x_1 + x_2 \equiv x \bmod N$. Shared values will be denoted as $[\![x]\!]_N$. We omit the modulus if it is clearly inferred from the context.

2.1.2 Chinese remainder theorem

The Chinese remainder theorem (CRT) is a number theoretic result that has found many usages also in cryptography. The CRT states that a set of equations in the form

$$x \equiv a_i \bmod m_i \ , \ \ i \in \{1, \ldots, k\}$$

where all m_i are pairwise coprime has a solution to x and it is uniquely fixed modulo $M = m_1 \cdot \ldots \cdot m_k$. In addition, the solution can be found as

$$x \equiv \sum_{i=1}^{k} a_i \cdot b_i \cdot \frac{M}{m_i} \bmod M$$

where

$$b_i \equiv \left(\frac{M}{m_i}\right)^{-1} \bmod m_i \ .$$

CRT is commonly used to reduce computations modulo M to many smaller computations modulo m_i and restore the result, or to unify computations for different m_i by computing them modulo M and then dividing back to separate moduli.

2.1.3 Universal composability

The framework of universal composability (UC) was proposed to provide a unified method for proving that protocols are secure even if executed sequentially or in parallel with other protocols [16, 47]. Security of a protocol is described in terms of securely implementing an ideal black-box behaviour of the protocol. An ideal functionality is described by a trusted third party (TTP), who securely collects all the inputs and then computes and returns the outputs of the protocol.

A protocol is said to be UC if for every adversary and computational context where the protocol is executed there exists an ideal world adversary that has the same computational advantage against the ideal world protocol. A universally composable protocol keeps its security guarantees in every context, provided that is is used in a black box manner. Thus, the protocol can receive inputs and give outputs, but all the intermediate messages are not used after the execution. UC security definitions give very strong security guarantees, but are, in general, also very restrictive.

Many two-party protocols can not be realized in the universally composable manner in the plain model, where the only setup assumption is the existence of authenticated communication [17]. The feasibility of universally composable two-party computation in the common reference string model (CRS) was shown in [18]. In addition, it is possible to avoid the assumption for trusted setup and base the protocols on the assumption of existence of public-key infrastructure like setup where each party has registered a public key, but no registration authority needs to be fully trusted [2].

2.1.4 Paillier cryptosystem

The Paillier public-key cryptosystem [45] uses an RSA modulus $N = pq$ where the secret components p and q are large primes of equal bit length. The requirement for equal bit length ensures that N is co-prime with the Euler totient function $\phi(N)$, namely

$$\gcd(pq, (p-1)(q-1)) = 1$$

where gcd is the *greatest common divisor*. The public key pk is (N, g), where $g \in \mathbb{Z}^*_{N^2}$ and the private key sk is the Carmichael function of N which can be computed as

$$\lambda = \text{lcm}(p-1, q-1)$$

where lcm denotes the *least common multiple*.

For a shorthand, we define the Paillier cryptosystem as a set of algorithms for key generation, encryption and decryption as $(\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$. The setup algorithm Gen is used to generate the keys pk and sk. The encryption function $\mathsf{Enc}_{pk}(m, r)$, where $m \in \mathbb{Z}_N$, requires a randomness $r \in \mathbb{Z}^*_N$ and defines the ciphertext as

$$c = \mathsf{Enc}_{pk}(m, r) = g^m r^N \bmod N^2 \ ,$$

where $c \in \mathbb{Z}^*_{N^2}$. In addition, the ciphertext space of the Paillier cryptosystem is equal to $\mathbb{Z}^*_{N^2}$ as Paillier showed that encryption is a bijection $\mathbb{Z}_N \times \mathbb{Z}^*_N \mapsto \mathbb{Z}^*_{N^2}$. This means that for each element c parties can publicly verify if it is a valid ciphertext. The decryption function

$$\mathsf{Dec}_{sk}(c) = \frac{L(c^\lambda \bmod N^2)}{L(g^\lambda \bmod N^2)} \bmod N$$

uses a helper function

$$L(x) = \left\lfloor \frac{x-1}{N} \right\rfloor \ .$$

The Paillier cryptosystem is additively homomorphic, allowing to compute the sum of the messages under encryption

$$\mathsf{Enc}_{pk}(m_1 + m_2, r_1 \cdot r_2) = \mathsf{Enc}_{pk}(m_1, r_1) \cdot \mathsf{Enc}_{pk}(m_2, r_2) \ .$$

This property also allows to evaluate the multiplication of an encrypted message and a plain value k under encryption $\mathsf{Enc}_{pk}(km) = \mathsf{Enc}_{pk}(m)^k$. We omit the randomness if its exact value is not important and in such case it is chosen uniformly during encryption.

Additive homomorphism also allows to re-randomize ciphertexts. Given a valid encryption $c = \mathsf{Enc}_{pk}(x)$ and $\mathsf{Enc}_{pk}(0)$ then $\hat{c} = c \cdot \mathsf{Enc}_{pk}(0)$ has the same distribution as $\mathsf{Enc}_{pk}(x, r)$, $r \leftarrow \mathbb{Z}^*_N$ and nothing else than x can be learned from $\hat{c}$. In addition, for valid ciphertexts $c = \mathsf{Enc}_{pk}(x)$ and $d = \mathsf{Enc}_{pk}(y)$, the combination $c \cdot d \cdot \mathsf{Enc}_{pk}(0)$ reveals nothing else than $x + y$.

We say that a cryptosystem is (t, ε)-indistinguishable under a chosen plaintext attack (IND-CPA) if, for two known messages m_0 and m_1 and any t-time adversary $\mathcal{A}$, the probability of distinguishing between the encryptions of these messages is

$$Adv^{IND-CPA}(\mathcal{A}) = \left| \Pr[G^{\mathcal{A}}_{0,IND-CPA} = 1] - \Pr[G^{\mathcal{A}}_{1,IND-CPA} = 1] \right| \leq \varepsilon \ .$$

where

$$G^{\mathcal{A}}_{0,IND-CPA} \left[\begin{array}{l} pk, sk \leftarrow \mathsf{Gen} \\ m_0, m_1 \leftarrow \mathcal{A}(pk) \\ \textbf{return } \mathcal{A}(\mathsf{Enc}_{pk}(m_0)) \end{array} \right. \qquad G^{\mathcal{A}}_{1,IND-CPA} \left[\begin{array}{l} pk, sk \leftarrow \mathsf{Gen} \\ m_0, m_1 \leftarrow \mathcal{A}(pk) \\ \textbf{return } \mathcal{A}(\mathsf{Enc}_{pk}(m_1)) \end{array} \right. .$$

The decisional composite residuosity assumption (DCRA) assumes that it is difficult to distinguish a random element of $\mathbb{Z}^*_{N^2}$ from a random N-th power in $\mathbb{Z}^*_{N^2}$. Let $\mathcal{N}$ be a

randomised algorithm for generating Paillier moduli. Then $\mathcal{N}$ is considered to be (t, ε)-secure against DCRA if for any t-time adversary $\mathcal{A}$ the probability of distinguishing random elements from random N-th residues is at most

$$Adv^{DCRA}(\mathcal{A}) = \left| \Pr[G^{\mathcal{A}}_{0,DCRA} = 1] - \Pr[G^{\mathcal{A}}_{1,DCRA} = 1] \right| \leq \varepsilon \ .$$

where

$$\begin{array}{ll} G^{\mathcal{A}}_{0,DCRA} & G^{\mathcal{A}}_{1,DCRA} \\ \left[\begin{array}{l} N \leftarrow \mathcal{N} \\ x \leftarrow \mathbb{Z}^*_{N^2} \\ \textbf{return } \mathcal{A}(x, N) \end{array} \right. & \left[\begin{array}{l} N \leftarrow \mathcal{N} \\ x \leftarrow \mathbb{Z}^*_{N^2} \\ \textbf{return } \mathcal{A}(x^N \bmod N^2, N) \end{array} \right. \end{array} \ .$$

Paillier cryptosystem is indistinguishable under chosen plaintext attacks (IND-CPA) under DCRA, which implies that the modulus N is hard to factor.

There are several known efficiency improvements to the basic definition of the Paillier cryptosystem, for example [45] and [22]. The decryption can be simplified by precomputing constant values in function $L(x)$ and using CRT. More precisely, we at first compute the decryption separately modulo p and q and use CRT to restore the decryption result modulo N [45].

In addition, we can define $g = N + 1$ which results in faster encryption function

$$\mathsf{Enc}_{pk}(m, r) = (Nm + 1)r^N \bmod N^2 \ .$$

This simplification results from the Binomial theorem which gives a general expression

$$(a + b)^c = \sum_{i=0}^{c} \binom{c}{i} a^i b^{c-i} \ ,$$

and from the fact that the generator does not need to be randomly chosen [22]. For our case, we need to compute

$$(N + 1)^m = \sum_{i=0}^{c} \binom{c}{i} N^i = 1 + cN + \binom{c}{2} N^2 + \cdots \ .$$

Encryption is a modular operation, thus, we can discard all the elements that have high powers of N since they are divisible by N^2 and we obtain

$$\mathsf{Enc}_{pk}(m, r) = (N + 1)^m r^N = (Nm + 1)r^N \bmod N^2 \ .$$

Furthermore, the encryption function can be computed more efficiently using the private key and CRT to compute separately modulo p^2 and q^2. Someone in possession of the private key might, for example, compute the encryption as a commitment to the encrypted value or to enable the other party to evaluate a function on the encrypted inputs.

2.1.5 Elliptic curves

Elliptic curves are plane curves with the general equation

$$y^2 = x^3 + ax + b$$

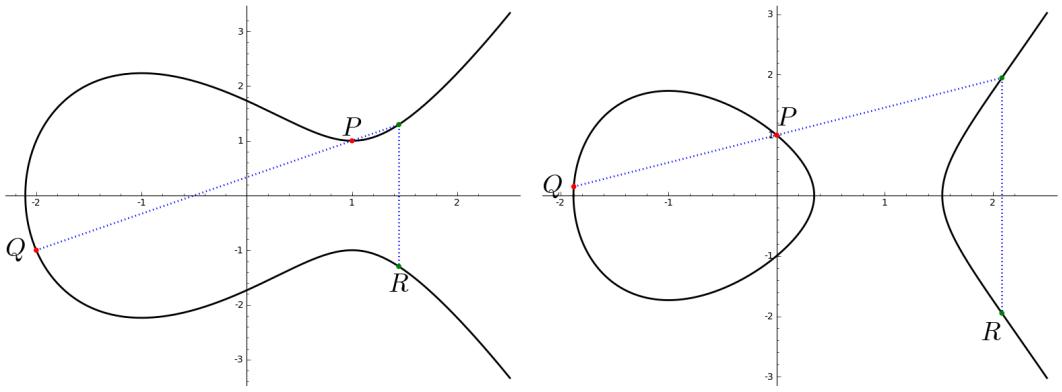


Figure 2.1: Elliptic curve addition $R = P + Q$ for curves $y^2 = x^3 - 3x + 1$ and $y^2 = x^3 - 3x + 1$ over real numbers.

where a and b are constants that define a specific curve. Cryptography usually only consider curves where $x, y, a, b \in \mathbb{F}$, for some finite field $\mathbb{F}$. In addition, there is a specific infinity point ∞ and a definition of a group operation so that elliptic curves form an Abelian group, where ∞ is the identity element. We usually use additive notation for operations between elliptic curve points which extends to multiplication of a point and an integer.

Figure 2.1 illustrates two different shapes of elliptic curves over real numbers. The geometric idea is that to find the sum of two points we must draw a line through them and find the third place where this line cuts the curve. The sum is this point mirrored by the x-axis or ∞ if there is no such point. Elliptic curves are much used in cryptography because their structure makes computing the discrete logarithm hard. Cryptography uses elliptic curves over finite fields where we can not draw illustrations as Figure 2.1, but the formulas derived from this planar interpretation still hold.

Elliptic curves are used as a basis for public key cryptosystems to be able to use shorter keys. For example, in our implementation, we can use 256-bit elliptic curve to achieve the same security level as 2048-bit Paillier key [3, 33]. To fix a curve, we must fix the field $\mathbb{F}$, constants a and b, base point (generator) of the elliptic curve and the order of the base point. There are several standard curves, where the parameters have been optimized for security and computational efficiency, for example see [43].

We use the elliptic curve *P-256* recommended by NIST [43] that is given in the Weierstrass form. Weierstrass form elliptic curves have a potential side-channel vulnerability because the addition of two different points and doubling one point have different algorithms. This is a serious threat as the point and scalar multiplication in elliptic curves is often implemented with an additive analogue of the *square-and-multiply* exponentiation algorithm and, therefore, analysing the power trace may leak the scalar used in the multiplication [15]. However, this risk can be mitigated with a common method of unifying the computation in these algorithms. We use Crypto++ [20] implementation of elliptic curves that avoids this vulnerability.

2.1.6 Lifted Elgamal cryptosystem

The Elgamal cryptosystem [29] is defined for a cyclic group $\mathbb{G}$ with generator g of prime order q. The setup phase defines the public key as $h = g^x$ and private key as x, where the latter can be chosen randomly. To encrypt a message $m \in \mathbb{G}$ we compute

$$\mathsf{E.Enc}_h(m, r) = (g^r, h^r \cdot m) \ ,$$

where $r \leftarrow \mathbb{Z}_q$. The decryption is defined as

$$\mathsf{E.Dec}_x(c_1, c_2) = c_2 \cdot c_1^{-x} \ ,$$

which is correct as

$$\mathsf{E.Dec}_x(\mathsf{E.Enc}_h(m, r)) = h^r \cdot m \cdot g^{-x \cdot r} = h^{r-r} \cdot m = m \ .$$

The Elgamal cryptosystem is multiplicatively homomorphic meaning that

$$\mathsf{E.Enc}_h(m) \cdot \mathsf{E.Enc}_h(n) = \mathsf{E.Enc}_h(m \cdot n) \ .$$

The Lifted Elgamal cryptosystem shares the setup phase with the aforementioned Elgamal cryptosystem, but differs from it as the encrypted message is used as an exponent rather than a multiplicand. In the following, we will use the Lifted Elgamal cryptosystem defined over elliptic curves. It is used as part of the commitment scheme and we do not use the expensive decryption operation. For a shorthand, we denote Lifted Elgamal cryptosystem as $(\mathsf{LE.Gen}, \mathsf{LE.Enc}, \mathsf{LE.Dec})$. The encryption function is defined as

$$\mathsf{LE.Enc}_h(m, r) = (g^r, h^r \cdot g^m)$$

and decryption requires computing a discrete logarithm on base g, denoted as $\log_g$, as follows

$$\mathsf{LE.Dec}_x(c_1, c_2) = \log_g(c_2 \cdot (c_1^{-x})) \ ,$$

which is correct as

$$\mathsf{LE.Dec}_x(\mathsf{LE.Enc}_h(m)) = \log_g(g^m) = m \ .$$

The Lifted Elgamal cryptosystem is additively homomorphic, as

$$\mathsf{LE.Enc}_h(m) \cdot \mathsf{LE.Enc}_h(n) = \mathsf{LE.Enc}_h(m + n) \ .$$

More precisely, homomorphism is correct because

$$\begin{aligned} \mathsf{LE.Enc}_h(m, r_m) \cdot \mathsf{LE.Enc}_h(n, r_n) &= (g^{r_m} \cdot g^{r_n}, h^{r_m} \cdot g^m \cdot h^{r_n} \cdot g^n) \\ &= (g^{r_m + r_n}, h^{r_m + r_n} \cdot g^{m+n}) = \mathsf{LE.Enc}_h(m + n) \ . \end{aligned}$$

We can also blind the ciphertext with $\mathsf{LE.Enc}_h(0)$ and having $\mathsf{LE.Enc}_h(m)$, $\mathsf{LE.Enc}_h(n)$ and $\mathsf{LE.Enc}_h(0)$ only allows us to learn $\mathsf{LE.Enc}_h(m + n)$.

Elgamal and Lifted Elgamal cryptosystems are IND-CPA secure under the Decisional Diffie-Hellman assumption (DDH). We say that a group $\mathbb{G}$ with generator g of order q is a (t, ε)-secure DDH-group if for any t-time adversary $\mathcal{A}$ the advantage

$$Adv^{DDH}(\mathcal{A}) = \left| \Pr[G^{\mathcal{A}}_{0,DDH} = 1] - \Pr[G^{\mathcal{A}}_{1,DDH} = 1] \right| \leq \varepsilon \ .$$

where

$$\begin{array}{ll} G^{\mathcal{A}}_{0,DDH} & G^{\mathcal{A}}_{1,DDH} \\ \left[\begin{array}{l} x, y \leftarrow \mathbb{Z}_q \\ \textbf{return } \mathcal{A}(g, g^x, g^y, g^{xy}) \end{array} \right. & \left[\begin{array}{l} x, y, z \leftarrow \mathbb{Z}_q \\ \textbf{return } \mathcal{A}(g, g^x, g^y, g^z) \ . \end{array} \right. \end{array}$$

2.1.7 Zero-knowledge proofs

Zero-knowledge proofs are interactive proofs for the correctness of a statement, whereas the proofs should not reveal any additional information. Zero-knowledge proofs have three important properties: completeness, soundness and zero-knowledge. Completeness means that in case of an honest verifier and prover the verifier accepts the proof. Soundness, on the other hand, gives the guarantee that a malicious prover can not make the verifier accept a faulty statement. Zero-knowledge property ensures that in case of an honest prover and a correct statement, the verifier learns nothing aside from the proof outcome.

We need a zero-knowledge protocol to prove that we have three ciphertexts of elements $x_1,\ x_2,\ x_3$ in multiplicative relation $x_3 = x_1 \cdot x_2$ without revealing these elements, but assuming that we have a valid public key pk. This proof is defined in Algorithm 1 and is built from a conditional disclosure of secrets protocol based on [40]. The encryption scheme defines the randomness space $\mathcal{R}$ and the space of suitable messages $\mathcal{M}$. The space of secrets $\mathcal{S}$ of the conditional disclosure of secrets protocol serves as a randomness to check, if the prover received the correct message.

We use a randomised encoding function

$$\mathsf{Encode}(s, r) = s + 2^\ell \cdot r$$

for ℓ-bit secrets s and a randomness $r \leftarrow \mathbb{Z}_{\lfloor N/2^\ell \rfloor}$. Therefore, the randomness space $\mathcal{R}_1$ is defined by the encoding function as $\mathcal{R}_1 = \mathbb{Z}_{\lfloor N/2^\ell \rfloor}$. The corresponding decoding function is defined as

$$\mathsf{Decode}(s + 2^\ell \cdot r) = s + 2^\ell \cdot r \bmod 2^\ell = s$$

meaning that

$$\mathsf{Decode}(\mathsf{Encode}(s)) = s \ .$$

The latter ensures that the secret can be correctly restored if the queries q_1, q_2, q_3 were valid.

The encoding is said to be ε-secure if for any secret s and an additive non-zero subgroup $\mathcal{G} \subseteq \mathcal{M}$ the distribution of $\mathsf{Encode}(s)+\mathcal{G}$ is statistically ε-close to the uniform distribution over $\mathcal{M}$. This encoding is $\frac{2^{\ell-1}}{p}$-secure where p is the smallest factor of N [39]. The encoding function is used to add noise to hide the secret if the queries are not in the multiplicative relation.

The general idea of the protocol is that the prover should only be able to correctly get the secret $s' = s$ if the initial queries were in the multiplicative relation. This results from the conditional disclosure of secrets protocol that enables the prover to only learn the secret if the multiplicative condition is satisfied. Round 3 clearly gives correct result for multiplicative elements as

$$s' = \mathsf{Decode}(x_3 \cdot e_1 + x_2 \cdot e_2 + \mathsf{Encode}(s) - (x_1 \cdot e_1 + e_2) \cdot x_2) = \mathsf{Decode}(\mathsf{Encode}(s)) \ .$$

Nothing is leaked to the verifier in the honest case, as the prover only sends encryptions and the element $s' = s$ and an honest verifier already knows s. The verifier releases all used randomness to the prover in round 4 to show that it behaved honestly and really knows s. This ensures that the verifier can not learn new information from the provers secret s'. For security, we need a hiding and binding commitment scheme, secure randomised encoding function, and an IND-CPA secure cryptosystem.

Algorithm 1 CdsZkMul - zero-knowledge protocol for multiplicative relation of ciphertexts for an additively homomorphic cryptosystem

Setup: Commitment parameters ck,
$Prover$ has a keypair (pk, sk),
$Verifier$ has pk
Data: $Prover$ has x_1, x_2, x_3
Result: $True$ for successful proof of $\mathsf{Enc}_{pk}(x_1) \cdot \mathsf{Enc}_{pk}(x_2) = \mathsf{Enc}_{pk}(x_3)$,
$False$ in case of any failure or abort

1: Round: 1
2: $Prover$ sets $q_1 = \mathsf{Enc}_{pk}(x_1)$, $q_2 = \mathsf{Enc}_{pk}(x_2)$, $q_3 = \mathsf{Enc}_{pk}(x_3)$
3: $Prover$ sends $q = (q_1, q_2, q_3)$ to $Verifier$
4: Round: 2
5: $Verifier$ checks that q_1, q_2, q_3 are valid ciphertexts
6: $Verifier$ generates $e_1, e_2 \leftarrow \mathcal{M}$, $r_1, r_2 \leftarrow \mathcal{R}$, $r \leftarrow \mathcal{R}_1$, $s \leftarrow \mathcal{S}$
7: $Verifier$ computes $a_1 = q_1^{e_1} \cdot \mathsf{Enc}_{pk}(e_2, r_1)$
8: $Verifier$ computes $a_2 = q_3^{e_1} \cdot q_2^{e2} \cdot \mathsf{Enc}_{pk}(\mathsf{Encode}(s, r), r_2)$
9: $Verifier$ sends $a = (a_1, a_2)$ to $Prover$
10: Round: 3
11: $Prover$ computes $s' = \mathsf{Decode}(\mathsf{Dec}_{sk}(a_2) - \mathsf{Dec}_{sk}(a_1) \cdot x_2)$
12: $Prover$ computes $(c, d) = \mathsf{Com}_{ck}(s')$
13: $Prover$ sends commitment c to $Verifier$
14: Round: 4
15: $Verifier$ sends $(s, e_1, e_2, r_1, r_2, r)$ to $Prover$
16: Round: 5
17: $Prover$: **if** $a_1 \neq q_1^{e_1} \cdot \mathsf{Enc}_{pk}(e_2, r_1)$ **return** $False$
18: $Prover$: **if** $a_2 \neq q_3^{e_1} \cdot q_2^{e2} \cdot \mathsf{Enc}_{pk}(\mathsf{Encode}(s, r), r_2)$ **return** $False$
19: $Prover$ sends decommitment d to $Verifier$
20: Round: 6
21: $Verifier$: **if** $\mathsf{Open}_{ck}(c, d) \neq s$ **return** $False$
22: **return** $True$

In the following, we will use this protocol with the Paillier cryptosystem and a commitment scheme defined in Section 2.1.8. We will obtain two special cases of this protocol defined in Algorithm 12 and Algorithm 10.

2.1.8 Dual-mode commitment schemes

Commitment schemes allow one participant to commit to a value, but keep it private from other participants. Afterwards, the participant can open the commitment to prove that he initially committed to the value that he claims to have committed to.

We define a commitment scheme as the set of protocols, namely setup, commitment and opening, $(\mathsf{Gen}, \mathsf{Com}, \mathsf{Open})$. In general, they rely on an additively homomorphic IND-CPA cryptosystem. More precisely, we will use Lifted Elgamal on elliptic curves in our initialisation.

The commitment parameters of a binding commitment are

$$ck = (pk, e)$$

where $e = \mathsf{LE.Enc}_{pk}(1)$. We use the homomorphic properties of the cryptosystem to compute the commitment and decommitment to m as

$$\mathsf{Com}_{ck}(m) = (c, d)$$

where commitment is $c = e^m \cdot \mathsf{LE.Enc}_h(0, r) = \mathsf{LE.Enc}_{pk}(m)$ and decommitment $d = (m, r)$. A commitment is opened by releasing all values used to compute c and the opening function just recalculates the commitment and verifies the correctness as

$$\mathsf{Open}_{ck}(c, d) = \begin{cases} m, & \textbf{if } c = e^m \cdot \mathsf{LE.Enc}_h(0, r) \wedge c \in \mathcal{C} \\ \perp, & \text{otherwise} \end{cases}$$

where $\mathcal{C}$ is the set of valid ciphertexts.

Commitment schemes have two important properties: hiding and binding. A commitment scheme is (t, ε)-hiding if, for any t-time adversary $\mathcal{A}$, the probability of distinguishing commitments to two different messages is

$$Adv^{hiding}(\mathcal{A}) = \Big| \Pr[G^{\mathcal{A}}_{0,hiding} = 1] - \Pr[G^{\mathcal{A}}_{1,hiding} = 1] \Big| \leq \varepsilon \ ,$$

where

$G^{\mathcal{A}}_{0,hiding}$

$ck \leftarrow \mathsf{Gen}$
$(m_0, m_1) \leftarrow \mathcal{A}(ck)$
$(c, d) \leftarrow \mathsf{Com}_{ck}(m_0)$
return $\mathcal{A}(c)$

$G^{\mathcal{A}}_{1,hiding}$

$ck \leftarrow \mathsf{Gen}$
$(m_0, m_1) \leftarrow \mathcal{A}(ck)$
$(c, d) \leftarrow \mathsf{Com}_{ck}(m_1)$
return $\mathcal{A}(c)$.

A commitment scheme is (t, ε)-binding if for any t-time adversary $\mathcal{A}$ the probability of creating a double opening is

$$Adv^{binding}(\mathcal{A}) = \Pr[G^{\mathcal{A}}_{0,binding} = 1] \leq \varepsilon \ ,$$

where

$G^{\mathcal{A}}_{0,binding}$

$ck \leftarrow \mathsf{Gen}$
$(c, d_0, d_1) \leftarrow \mathcal{A}(ck)$
if $\mathsf{Open}_{ck}(c, d_0) \neq \perp \wedge \mathsf{Open}_{ck}(c, d_1) \neq \perp$
    return $\mathsf{Open}_{ck}(c, d_0) \neq \mathsf{Open}_{ck}(c, d_1)$
else
    return 0 .

We call a commitment perfectly equivocal, if a valid commitment can be efficiently opened to any message given some trapdoor information. We say that a commitment scheme is (t, ε)-equivocal, if for any t-time adversary $\mathcal{A}$ the advantage of distinguishing real and equivocal commitments is bounded as

$$Adv^{equivocal}(\mathcal{A}) = \Big| \Pr[G^{\mathcal{A}}_{0,equivocal} = 1] - \Pr[G^{\mathcal{A}}_{1,equivocal} = 1] \Big| \leq \varepsilon \ ,$$

where

$G^{\mathcal{A}}_{0,equivocal}$

$ck \leftarrow \mathsf{Gen}$
$\mathcal{A}(ck)$
for as long as $\mathcal{A}$ wants
  $m \leftarrow \mathcal{A}$
  $(c, d) \leftarrow \mathsf{Com}_{ck}(m)$
  $\mathcal{A}(c, d)$
end for
return $\mathcal{A}(ck)$

$G^{\mathcal{A}}_{1,equivocal}$

$ck, ek \leftarrow \mathsf{FakeGen}$
$\mathcal{A}(ck)$
for as long as $\mathcal{A}$ wants
  $m \leftarrow \mathcal{A}$
  $c, r \leftarrow \mathsf{FakeCom}_{ck,ek}()$
  $d \leftarrow \mathsf{Equivocation}_{ck,ek}(d, m, r)$
  $\mathcal{A}(c, d)$
end for
return $\mathcal{A}(ck)$.

According to the requirements of the aforementioned zero-knowledge proof protocols, we need our commitment to be an equivocal and binding dual-mode commitment [40]. We need computationally indistinguishable setup phases that yield either statistically binding or perfectly equivocal commitment scheme to fulfil the requirements of the dual-mode commitment.

The previously described commitment scheme allows us to define a suitable equivocal setup. In addition, the ideal setup and defined algorithms $(\mathsf{Gen}, \mathsf{Com}, \mathsf{Open})$ yield a computationally hiding and unconditionally binding commitment as proved in Theorem 2.1.1. The security proofs in this section give guarantees up to a constant factor in terms of the running time.

Theorem 2.1.1. *The commitment scheme in algorithms* $(\mathsf{Gen}, \mathsf{Com}, \mathsf{Open})$ *yields a* (t, ε)*-hiding and unconditionally binding commitment given a* (t, ε)*-IND-CPA secure cryptosystem.*

Proof Sketch. According to the commitment algorithm, the commitment to message m is $c = \mathsf{Enc}_{pk}(m)$. Hence, the hiding property of the commitment scheme directly follows from the definition of IND-CPA security of the cryptosystem.

Similarly, the binding property follows from the fact that the public key pk defines a valid secret key and, thus, it is theoretically possible to decrypt the commitment, whereas the decryption always succeeds and yields m. □

We can use an altered setup so that the commitment key $ck = (h, e)$ and equivocation key ek are fixed according to $\mathsf{FakeGen}$ to obtain a perfectly equivocal commitment. $\mathsf{FakeGen}$ defines $ck = (h, \mathsf{LE.Enc}_h(0, r_*))$ and $ek = r_*$. Equivocal commitment is computed as

$$\mathsf{FakeCom}_{ck} = (c, r')$$

where r' is the trapdoor for equivocation and $c = \mathsf{LE.Enc}_h(0, r')$. Such a commitment and opened to any chosen message m by

$$\mathsf{Equivocation}_{ck,r_*}(m, c, r') = (m, r' - r_* \cdot m) \ .$$

The result can be verified using Open as defined previously. The correctness of this equivocal setup is shown in Theorem 2.1.2.

Theorem 2.1.2. *Algorithms* $(\mathsf{FakeGen}, \mathsf{FakeCom}, \mathsf{Equivocation}, \mathsf{Open})$ *yield a perfectly equivocal commitment of a random message given a* (t, ε)*-IND-CPA-secure Lifted Elgamal cryptosystem.*

Proof. According to the definition of Com, any commitment would be $c = \mathsf{LE.Enc}_h(0)$ and knowing r_* it can be opened to any message m. If we initially computed the commitment as $\mathsf{FakeCom}_{ck}$ and obtained

$$c = \mathsf{LE.Enc}_h(0, r') = (g^{r'}, h^{r'}) \ ,$$

then we can easily compute the decommitment to any m as $r = r' - r_* \cdot m$ as defined by $\mathsf{Equivocation}$. This is accepted by Open because

$$\begin{aligned} e^m \cdot \mathsf{LE.Enc}_h(0, r) &= \mathsf{LE.Enc}_h(0, r_*)^m \cdot \mathsf{LE.Enc}_h(0, r) = \mathsf{LE.Enc}_h(0, r_* \cdot m + r) \\ &= \mathsf{LE.Enc}_h(0, r_* \cdot m + r' - r_* \cdot m) = \mathsf{LE.Enc}_h(0, r') = c \ . \end{aligned}$$

Secondly, we need that the distributions of $(c, d) \leftarrow \mathsf{Com}_{ck}(m)$ and (c_*, d_*) where $c_* \leftarrow \mathsf{FakeCom}_{ck}$ and $d_* \leftarrow \mathsf{Equivocation}_{ck,r_*}(m)$ coincide. The distributions of c and c_* coincide because they are both random encryptions of zero. If we compute the commitment according to Com_{ck} with setup from Gen, we obtain

$$\begin{aligned} c &= e^m \cdot \mathsf{LE.Enc}_h(0, r) = ((g^{r_*})^m, (h^{r_*} g^0)^m) \cdot (g^r, h^r g^0) \\ &= (g^{r_* \cdot m + r}, h^{r_* \cdot m + r} g^0) = (g^{r_* \cdot m + r}, h^{r_* \cdot m + r}) = (g^{r'}, h^{r'}) \\ &= \mathsf{LE.Enc}_h(0, r') \ . \end{aligned}$$

In addition, fixing a message m and having a commitment c uniquely fixes the only possible decommitment $d = (m, r)$ as the value r is uniquely fixed. Thus, if the distributions of commitments c and c_* coincide then so do the joint distributions of (c, d) and (c_*, d_*).

Thus, we have a perfectly equivocal commitment. □

Theorem 2.1.3. *The setups for binding* (Gen) *and equivocal* ($\mathsf{FakeGen}$) *commitment schemes are (t, ε)-indistinguishable for parties who only see the commitment parameters $ck = (pk, e)$, given a (t, ε)-IND-CPA secure cryptosystem.*

Proof Sketch. Assume that there is an adversary $\mathcal{A}$ who can distinguish between these two setup phases. This means that $\mathcal{A}$ can differentiate between $(pk, \mathsf{LE.Enc}_h(0))$ and $(h, \mathsf{LE.Enc}_h(1))$. Hence, we can build an adversary $\mathcal{B}$ for the IND-CPA security of the cryptosystem. At first $\mathcal{B}$ sends the messages $m_0 = 0$ and $m_1 = 1$ and receives the ciphertext $\mathsf{LE.Enc}_h(m_b)$. $\mathcal{B}$ then forwards the message $(h, \mathsf{LE.Enc}_h(m_b))$ to $\mathcal{A}$ and outputs the result as $\mathcal{A}$. Hence, $\mathcal{B}$ breaks the IND-CPA security exactly when $\mathcal{A}$ successfully distinguishes the setups and the success of t-time adversary $\mathcal{A}$ is bounded by ε. The running time of $\mathcal{B}$ needs only to be constant time longer than $\mathcal{A}$ to forward the messages. □

Corollary 2.1.4. *According to Theorems 2.1.2 and 2.1.3 commitment setups are (t, ε)-indistinguishable even if the adversary $\mathcal{A}$ sees pairs of correct commitments (c, d) or fake commitments (c_*, d_*).*

For practical purposes we need a protocol to implement the setup phase. It can be combined from the Diffie-Hellman key exchange [28], homomorphic properties of Lifted Elgamal cryptosystem and Schnorr Σ-protocols [51]. It is important that the setup should yield perfectly binding commitment in case computing parties execute it, but there also has to exist a simulator who can achieve equivocal setup. Full specification of this protocol is out of the scope of this thesis.

2.1.9 Message authentication codes

A message authentication code (MAC) is an extra piece of information about a message that enables to detect modifications to the initial message. MACs are often described as keyed hash functions that output a tag from a message and a secret key. A MAC is secure, if an adversary can not substitute messages or generate valid message and tag pairs. A substitution attack means changing message and tag pair (x, z) with a different pair $(\overline{x}, \overline{z})$ where $\overline{z}$ is a valid tag for $\overline{x}$. An impersonation attack means generating a valid pair $(\overline{x}, \overline{z})$ without seeing any authentication pairs before.

We define a MAC for secret-shared elements as follows. We have a key k and a secret $[\![x]\!]$, where $x = x_1 + x_2$ and $x_1,\ x_2$ are the additive shares. We define $z = k \cdot x$ as the authentication code for x and keep it as shares $z_{1,x}, z_{2,x}$, hence

$$z_{1,x} + z_{2,x} = k \cdot (x_1 + x_2) \ .$$

Verifying the code is trivial for anyone in possession of the secret key k. The key should be chosen from the same algebraic structure as used for the secret sharing.

Keeping MAC in shares allows us think of it as $z_{2,x} = k \cdot (x_1 + x_2) - z_{1,x}$ where $z_{2,x}$ is the tag for secret x and $z_{1,x}$ is part of the secret key which becomes $(k, z_{1,x})$. Hence, the view of $\mathcal{CP}_2$ is like having tags for unknown messages x where all these pairs share the sub-key k, but differ by the second part of the key. Such construction was introduced by Rabin and Ben-Or as *Information Checking* or *Check Vectors* for verifiable secret sharing [49].

Our attack scenario results from the opening of the shares where $\mathcal{CP}_2$ might receive x_1 and therefore learn x before sending x_2 and z_2 to $\mathcal{CP}_1$. Hence, we need that $\mathcal{CP}_2$ must not be able to come up with $\hat{x}_2$ and $\hat{z}_2$ such that $\mathcal{CP}_1$ would accept the MAC. This is a special substitution attack, because the attacker only gets to see one message and tag pair before the attack, however it is sufficient as the second part of the key is always different and the the attacker can not see tags for more messages of the same key. We can more precisely define it as a security game for $[\![x]\!]_N$ in ring R as $G^{\mathcal{A}}_{MAC}$. We say that a MAC is statistically ε-secure, if for any adversary $\mathcal{A}$ the probability of winning in $G^{\mathcal{A}}_{MAC}$ is bounded by ε:

$$\Pr[G^{\mathcal{A}}_{MAC} = 1] \leq \varepsilon$$

where

$G^{\mathcal{A}}_{MAC}$

- $k \leftarrow R$
- **for** as long as $\mathcal{A}$ wants
  - $x \leftarrow \mathcal{A}$
  - $z_1, x_2 \leftarrow R$
  - $z_2 = k \cdot x - z_1$
  - $\mathcal{A}(z_2, x - x_2, x_2)$
- **end for**
- $x \leftarrow \mathcal{A}$
- $z_1, x_2 \leftarrow R$
- $z_2 = k \cdot x - z_1$
- $x_1 = x - x_2$
- $\hat{z}_2, \hat{x}_2 \leftarrow \mathcal{A}(z_2, x_1, x_2)$
- **return** $\hat{z}_2 == (\hat{x}_2 + x_1) \cdot k - z_1 \wedge x_2 \neq \hat{x}_2$.

Theorem 2.1.5. *The adversaries success in $G^{\mathcal{A}}_{MAC}$ for a finite field $\mathbb{F}_{p^n}$ is bounded by $\frac{1}{p^n}$.*

Proof Sketch. First, the views of the adversary $\mathcal{A}$ in $G^{\mathcal{A}}_{MAC}$ and $G'^{\mathcal{A}}_{MAC}$ are indistinguishable and the advantage is the same, where

$G'^{\mathcal{A}}_{MAC}$

for as long as $\mathcal{A}$ wants
  $x \leftarrow \mathcal{A}$
  $z_2, x_2 \leftarrow \mathbb{F}_{p^n}$
  $\mathcal{A}(z_2, x - x_2, x_2)$
end for
$x \leftarrow \mathcal{A}$
$z_2, x_2 \leftarrow \mathbb{F}_{p^n}$
$x_1 = x - x_2$
$\hat{z}_2, \hat{x}_2 \leftarrow \mathcal{A}(z_2, x_1, x_2)$
$k \leftarrow \mathbb{F}_{p^n}$
$z_1 = k \cdot x - z_2$
return $\hat{z}_2 == (\hat{x}_2 + x_1) \cdot k - z_1 \wedge x_2 \neq \hat{x}_2$

The values of z_2 in $G^{\mathcal{A}}_{MAC}$ are randomly uniform, because z_1 is chosen uniformly and, therefore, $kx - z_1 \bmod p$ is also uniformly random element of $\mathbb{F}_{p^n}$. In addition, the values that adversary sees do not depend on the key k, so, it can be chosen later.

Clearly, the advantage in $G'^{\mathcal{A}}_{MAC}$ is the same as the possibility of coming up with a pair $\hat{z}_2, \hat{x}_2$ such that $\hat{z}_2 == (\hat{x}_2 + x_1) \cdot k - z_1$. After $\mathcal{A}$ picks $\hat{z}_2, \hat{x}_2$ there is exactly one

$$k_* = (z_1 + \hat{z}_2) \cdot (\hat{x}_2 + x_1)^{-1} \ ,$$

such that $\hat{z}_2 == (\hat{x}_2 + x_1) \cdot k_* - z_1$. To conclude, the possibility of picking k such that the verification succeeds is $\frac{1}{p^n}$ because $\mathbb{F}_{p^n}$ has p^n different elements and only one uniquely fixed k_*. □

Our message space $\mathcal{M}$ may have a composite order and, therefore, this is not as secure MAC as it would be in case of finite fields. However, we will have $R = \mathbb{Z}_N$, where $N = pq$ is the Paillier modulus and both of its factors are large and the security is the same as it would be for either of the prime factors.

Theorem 2.1.6. *The success of adversary $\mathcal{A}$ in $G^{\mathcal{A}}_{MAC}$ with a Paillier modulus N that defines $R = Z_N$, where $N = pq$, p and q are primes, and $p < q$ is bounded by $\frac{1}{p}$.*

Proof Sketch. Assume that there is an adversary $\mathcal{A}$ against $G^{\mathcal{A}}_{MAC}$ for some modulus $N = pq$ who is very successful. Then, there exists an adversary $\mathcal{B}$ in G_{MAC} for modulus p, who can use this $\mathcal{A}$ to win in its game. Adversary $\mathcal{B}$ has a fixed modulus p, picks a prime $q = N/p$ and uses the adversary $\mathcal{A}$ for modulus N. For every x that $\mathcal{A}$ picks $\mathcal{B}$ forwards it to the challenger and gives the result pack to $\mathcal{A}$. It behaves similarly with the final challenge. According to the Chinese remainder theorem, if $\mathcal{A}$ gives a correct result modulo N then it also holds modulo p and $\mathcal{B}$ wins it its game exactly when $\mathcal{A}$ is successful. The runtime of $\mathcal{B}$ is only a constant factor longer than $\mathcal{A}$. However, the maximal success of $\mathcal{B}$ is limited by Theorem 2.1.5 as $\mathbb{Z}_p$ is a finite field and, thus, the maximal success of $\mathcal{A}$ is also $\frac{1}{p}$. From this we know that for any Paillier modulus $N = pq$, the maximal success is $\frac{1}{p}$ where $p < q$. □

This MAC also has homomorphic properties making it possible to compute the new MAC and new key for the sum of two messages given the tags of the initial messages.

2.2 Secure multi-party computation

Secure multi-party computation (SMC) is a mechanism that allows several participants to evaluate a function without revealing their inputs. A classical SMC problem known as the Millionaires' problem was proposed by Yao [56]. There are two millionaires who wish to know who has more money without revealing their wealth to the other millionaire.

2.2.1 Overview of SMC techniques

Garbled circuits

Together with the Millionaires' problem Yao proposed a solution for securely evaluating boolean circuits [56]. This approach is known as garbled circuit evaluation and has developed a lot since it was first proposed. Garbled circuits are commonly used for two-party computations in the passive model, but this approach can be extended to more parties [5, 55].

The general idea of garbled circuits is that the original circuit of a function is transformed so that the wires only contain random bitstrings. Each gate is encoded so that its output bitstring can be computed from the inputs and only the random bitstrings of output gates can be mapped back to actual results. This way the evaluation computes the function, but does not leak information about the values on separate wires. The main drawback of the garbled circuit technique are inefficient evaluation and inability to reuse the circuit. However, they have been used in SMC frameworks [42, 35].

Secret sharing

Secret sharing was introduced by Shamir [52] and Blakley [8]. Since then Shamir's scheme has provided a basis for different verifiable secret sharing [30, 46, 49], SMC and threshold encryption ideas [31]. SMC frameworks can be obtained from the secret sharing schemes based on the homomorphic properties of the schemes and by defining protocols for operations not directly supported by the homomorphism.

This thesis and the Sharemind framework are based on additive secret sharing as introduced in Section 2.1.1. Share computation is mainly aimed at securely evaluating arithmetic circuits. For many use-cases arithmetic circuits are more efficient than boolean circuits and, therefore, share computation is likely to be more efficient than garbled circuits.

Homomorphic encryption

Homomorphic encryption is especially useful for building secure client-server model applications, but it can be extended to more general settings. For example, the a sends encrypted inputs to a server who then computes the desired function on the ciphertexts.

Currently we know of several additively homomorphic cryptosystems, such as Paillier or Lifted Elgamal, or multiplicatively homomorphic cryptosystems, for example Elgamal. We can use them to obtain frameworks where one side can compute some

operations locally, but others require collaboration. These difficulties can be overcome with *fully homomorphic* cryptosystems where both multiplication and addition can be performed locally [32]. However, at current state, fully homomorphic encryption is too inefficient for practical SMC frameworks.

The main limitation of SMC frameworks based on homomorphic cryptosystems is the inability to use common data types. The cryptosystem defines a modulus and all the arithmetic is with respect to these moduli. However, to achieve reasonable security, we commonly need moduli that are thousands of bits long. We use some of the ideas from this setup in the precomputation phase of our protocol sets and, therefore, we also suffer from this restriction on our modulus.

2.2.2 General SMC threat model

An important privacy goal of SMC is that all inputs and outputs should remain private, unless specifically declassified. However, we can not avoid that the output of a function may leak information about the inputs. Furthermore, we often require that the correctness of the outputs is guaranteed or at least verifiable.

The passive or *honest-but-curious* security model defines an adversary who always follows the protocol specification, but may try to extract additional information from its view of the protocol. An active or *malicious* security model proposes no restrictions to the behaviour of the adversary whereas the security aim is to catch the adversary with overwhelming probability. Consequently, the adversary can control all aspects of the corrupted parties and communication channels. *Covert* security model is somewhere in between the previous two, as the adversary can behave maliciously and must be caught with certain arbitrarily fixed probability. However, there is a bigger risk of leaking secrets than when achieving security against an active adversary.

In addition, adversarial behaviour can either be *static*, *adaptive*, or *mobile*. A static adversary picks the set of corrupted parties in the beginning of the protocol and is unable to change it later. An adaptive adversary can increase the set of corrupted parties over time. Finally, a mobile adversary can adaptively corrupt and release parties, thus varying the set of corrupted parties during the protocol execution.

Although we mostly concentrate on the computing parties, it is possible that the adversary is not any of the computing nodes, but, for example, someone in the network. Such adversary can eavesdrop or modify the network and disrupt the communication. We can use classical techniques to secure the channels against eavesdropping or modification, but we can not solve different denial of service attacks on the network.

A commonly used threshold limitation of SMC is that achieving unconditional security against a passive adversary is only possible if less than $\frac{n}{2}$ parties of n are corrupted or correspondingly $\frac{n}{3}$ for active adversary [19, 6]. These results are special cases of more general result with adversary structures that allow for stronger results [36, 37]. An adversary structure consists of sets of parties where the adversary is allowed to corrupt any of these sets. Let $Q^{(2)}$ ($Q^{(3)}$) be the conditions that no two (three) of these sets cover the whole set of parties. Every function can then be unconditionally securely computed by an active adaptive adversary if it is in $Q^{(3)}$. Analogous result holds for adaptive passive adversary for $Q^{(2)}$.

2.2.3 Achieving actively secure two-party computation

The active security model allows the adversary to behave maliciously and do anything it likes, for example, send incorrect messages. Hence, we need to ensure the correctness of computations and the privacy of the inputs. In addition, we also require universal composability to use the basic protocols as building blocks for more general functions. However, we restrict our adversary so that only one of the computing parties can be statically compromised and the two computing parties can not collude. The properties of the additive secret sharing scheme clearly define that the two computing parties can not collude and, hence, the adversary can not corrupt both of the computing parties.

We take the same approach as SPDZ [26] to ensure the correctness of computation results, where we only verify the correctness when we publish a result and not during the computation. Here we rely of the security properties of the used verification mechanisms. In addition, we assume that the setup of the protection domain has been securely fixed and other protocols must be secure in the shared setup model.

In general, we require universal composability, but for brevity we do not give full proofs for this. In the following, security means both privacy of the inputs and the correctness of the outputs. The security is achieved using protection mechanisms on the shares. All protocols with only local computation trivially protect privacy, but we need to ensure privacy in collaborative protocols.

In the following, we prove security of the protocols in the stand-alone model with shared setup so that it also implies security in sequential compositions. For some protocols, we only show the simulatability of the communication so that the adversary can not distinguish between simulated and real protocol run.

We actually assume that there are three conditions that a protocol needs to fulfil to achieve security. Firstly, the communication of the protocol should be simulatable. Secondly, the parties can notice if others cheat. Thirdly, after the protocol, the parties are convinced about the consistency of the share. However, proving that these are sufficient, is out of the scope of this thesis.

2.3 The SHAREMIND SMC framework

SHAREMIND is an SMC framework [11, 12, 9] with three main goals: (1) it must be usable for securely processing confidential data, (2) it must be efficient enough for practical applications, and (3) it must be usable by non-cryptographers. This thesis is based on version 3 of the SHAREMIND framework where we can easily define new secure computation schemes in addition to the traditional one with three miners and a passive security guarantee.

2.3.1 Application model

SHAREMIND is designed as a general tool for SMC and privacy preserving data mining. The model has three different kinds of parties: (1) computing parties, (2) input parties, and (3) result parties. One participant can belong to all of these classes.

Input parties use secret sharing to distribute their inputs between the computing parties and are denoted as $\mathcal{IP}_i$. Input parties correspond to the data donors or owners of the data and can often be the same as the result parties. Computing parties, a.k.a miners, perform computations on the shared data following the protocols specific to

the sharing method. Computing party will be denoted as $\mathcal{CP}_i$ and can be thought of as a dedicated server, we denote the set of computing parties by $\mathcal{CP}$. Finally, result parties map to data analysts who initiate the queries and computations and learn the final public outcomes. They will be denoted as $\mathcal{RP}_i$. Result parties get to aggregate the data from the input parties without actual access to confidential inputs.

The number of input and result parties is not limited, but the number of computing parties is often defined by the computation protocols. For example, classical Sharemind protocols use three computing parties, but this thesis focuses on the case with two miners.

The important trust requirements are that the input parties must believe that the computing miners do not collude and the result parties must believe that the computing parties give correct results. The latter can be ensured at a protection domain level as long as the miners do not collude. In addition, the miners should either believe or check the consistency of inputs. However, in any case we can not avoid the attack where input party decides to classify false information.

2.3.2 Computation primitives

In general, a query from the result party means that the computing parties must execute some algorithm to compute the result. These algorithms are collections of operations such as addition or multiplication. Each of these operations in turn correspond to a secure computation protocol. The computing parties execute the corresponding protocols in order to securely evaluate the query.

A protection domain kind (PDK) is a set of algorithms that define the data representation and computation protocols. Different protection domain kinds can define different elementary operations. For example, some may support division, but others do not have to. A protection domain (PD) is a concrete initialisation of a protection domain kind. A protection domain is defined by the algorithms from the corresponding protection domain kind and its configuration, for example, the keys of the participants. In addition, a protection domain also consists of the protected data. Common Sharemind PDK is based on additive secret sharing among three miners and is secure in the passive adversary model, where a PD is fixed by the computing nodes.

All elementary protocols of the PDK must be reusable and composable with each other to achieve provably secure query evaluation. It has been shown that following simple rules when designing protocols for elementary operations yields a provably secure composition of protocols for the traditional three miner PDK [9]. In general we require universal composability of the computation protocols.

2.3.3 Programming applications

Sharemind 3 uses the SecreC 2 programming language to specify the query algorithms. SecreC is a Sharemind specific C-like language designed to be privacy-preserving and easy to use. SecreC is strongly typed whereas the type of the private variables includes the PD. In fact, all public values can be seen as belonging to some public PD and also including this in their type. The programmer does not need to be aware of the underlying PDK protocols for operations on the private data and can call them as any predefined functionality.

SecreC can be used for different protection domain kinds and for writing code that

is not specific to any fixed PDK [10]. The domain-polymorphic code clearly only works if the PDK defines all the necessary protocols. Polymorphism means that integrating new PDKs to applications is simple and one can easily test their application against several PDs or develop libraries independently of the PDK. In addition, it is possible to implement a general function and then specify a special version of it for some PDK where, for example, the required functionality can be achieved more efficiently than in the generic code.

Algorithm 2 Example of SecreC

```
1  kind additive2pa;
2  kind additive2paSym;
3  domain pd_a2a additive2pa;
4  domain pd_a2a_sym additive2paSym;
5
6  template <domain D>
7  D uint32 sum (D uint32 [[1]] arr) {
8      D uint32 out = 0;
9      for (uint64 i = 0; i < size(arr); i++){
10         out = out + arr[i];
11     }
12     return out;
13 }
14
15 void main () {
16     uint64 n = 10;
17     pd_a2a uint32 [[1]] arr1 (n) = 2;
18     pd_a2a uint32 s1 = sum(arr1);
19     assert (declassify(s1) == (20 :: uint32));
20
21     pd_a2a_sym uint32 [[1]] arr2 (n) = 3;
22     pd_a2a_sym uint32 s2 = sum(arr2);
23     assert (declassify(s2) == (30 :: uint32));
24
25     return;
26 }
```

Algorithm 2 gives an example of SecreC code that uses two different PDK where the domain fixes the setup of given PD. The main function defines one dimensional matrix (vector) of length n with secret shared elements equal to either 2 or 3 and computes the sum of the vector elements for both of these PD. The function sum is defined independently of the used PDK and can be used as long as the PDK defines type *uint32* and an addition operation. Finally the main function publishes the result and verifies that it has the value that was expected.

Chapter 3

Principles of the SPDZ framework

This chapter introduces the main aspects of SPDZ (pronounced *Speedz*) that is an actively secure SMC framework [26] that also has a covertly secure version [23]. An important characteristic of SPDZ is the usage of precomputations, which separate the protocols to two parts as also used in [24, 21, 25, 7, 44]. Firstly, the precomputation phase is independent of the secret information and produces some random shares. Secondly, the online phase uses the secrets and precomputation results to evaluate necessary functions.

The SPDZ framework utilises three important tools: (1) Oblivious Message Authentication Codes, (2) Beaver triples, and (3) vectorized homomorphic encryption. The first two are used to ensure security against an active adversary and the second as precomputation for multiplication. These two have been previously used together for SMC in BDOZ [7]. However, SPDZ adds an important idea that MAC is used to authenticate the shared secret as a whole and not for authenticating independent shares. Thirdly, vectorised somewhat-homomorphic encryption is used to generate Beaver triples in a communication-efficient way and is a SPDZ-specific property.

It can be seen that SPDZ is a mixture of different previously existing ideas. We actually omit the usage of the somewhat homomorphic encryption, which is the most specific idea of SPDZ, but we use the idea of authenticating the secret, not the separate shares. In the following, we, in general, use the name SPDZ to refer to the collection of these ideas and reference the origins separately as we introduce the concepts. Our vision on the development of SPDZ is given on Figure 3.1.

3.1 Precomputation model

The precomputation and online phases are essentially independent and can be optimized separately, as long as they have consistent share representations. However, having a separate precomputation phase is meaningful only as long as preprocessing gives some benefit to the online phase.

Currently, SPDZ precomputes Beaver triples and single random shares. The covertly secure extension [23] also precomputes squaring pairs analogous to Beaver triples and shared bits for comparison, bit-decomposition, fixed point and floating point operations. It is an open question if other operations can be efficiently precomputed. The precomputation model originates from Beaver [4] and has found wider usage in SMC after [24].

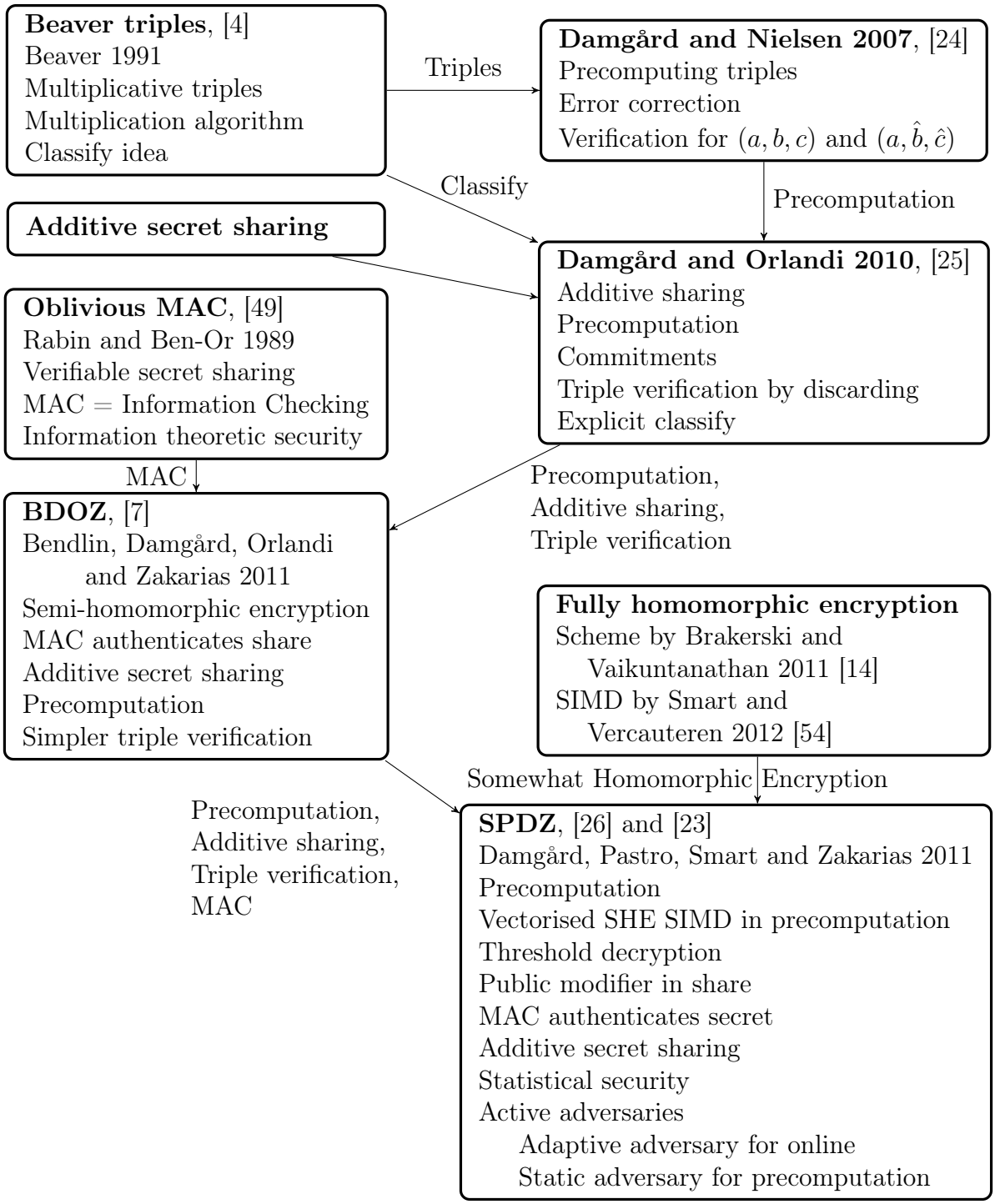


Figure 3.1: The development of SPDZ

Although precomputation is used to achieve efficient online computations, it may mean that the overall cost of the protocols increases. For example, it could be possible to use an expensive multiplication protocol M to precompute Beaver triples and then use the triples to do online multiplication in protocol O. In such case, we use computation time for both M and O, whereas, in theory, only the time of the precomputation M suffices for multiplication. However, dividing it to two parts allows for more efficient online phase. Thus, precomputation model allows us to gain online performance but may not reduce total workload. This model is usable if O is reasonably more efficient

than M or if we can do precomputations without actually defining a multiplication protocol M.

Precomputation is meaningful in situations where the overall system has uneven workload so it can do precomputations in the background. Precomputations could be performed when the users are not active or in parallel with online computations. The latter is reasonable if it does not significantly reduce online performance. For example, we can consider a data analyst who likes to get fast results during the workday, but does not use the system outside common working hours. The latter indicating that the night-time can be easily used for precomputations.

The precomputation model assumes that the online phase always has sufficient precomputed values to proceed. However, it is not trivial to ensure this in practice. Therefore, it is important to consider the desired behaviour of online protocols when they can not retrieve all necessary precomputation products. One possibility is to signal the precomputing process and then compute the protocol incrementally as the precomputation results become available. The other case would be to define a separate slower protocol for the same functionality that does not require precomputations and use it instead. In addition, depletion of precomputation results can be avoided if the algorithms to be executed as well as the input sizes are well known beforehand.

The difficulty from dependence on the precomputation protocol speed indicates that this model is not well designed for all SMC use-cases. For example, the classical Millionaires' problem would have the best solution if the millionaires can set up the framework, insert their input and get the output at once. The alternative with the precomputation is unsatisfactory, as they may not wish to wait a while to allow the machine to perform all kinds of precomputations. In conclusion, the precomputation model is best suited for applications where the data is used for an extended time period.

3.2 Oblivious MAC

Oblivious MAC algorithms resemble threshold cryptosystems in a way that no party can check the MAC tag independently of others as no party can decrypt alone. Furthermore, the MAC tag value of a secret shared input will be stored as a secret shared value. For example, a secret value $[\![x]\!]$ might be protected using a MAC, where the tag is in turn kept in shares as $[\![z]\!]$. Together, these requirements indicate that the MAC key k must be a secret value. In addition, the MAC algorithm must have homomorphic properties to be able to compute the tag for the computation result from the tags of the inputs. The idea of producing MAC tags to unknown values originates from Rabin and Ben-Or as *Information Checking* for verifiable secret sharing [49].

The SPDZ framework uses unconditionally secure MAC to verify the correctness of shared value instead of the correctness of each share. The idea of checking the shared value and not each share results in less storage for MAC tags, but also means that we can not check the validity of the computations before declassifying the value. MAC key is shared using additive secret sharing together with meta-information so that all parties can verify the correctness of the key. Each party has a share of the value and a share of the tag on that value for each secretly shared element. An analogous algorithm to MAC from Section 2.1.9 is also used by SPDZ. All their arithmetic is in a finite field $\mathbb{F}_{p^k}$ for a prime p and integer k and thus, according to Theorem 2.1.5 the MAC is secure.

The security requirements from Section 2.1.9 apply also to SPDZ when extended to more than two parties. In addition, SPDZ proposes an efficiency improvement that either verifies that all the elements in a vector or none of them. The idea is to combine the MAC tags to reduce network communication when checking the tags. In this case, we do not exactly learn, which share was faulty, but in practice we only need to learn the fact that some party might be malicious. Besides, shares could be verified separately to sort out the false ones.

3.3 Beaver triples

Beaver triples are multiplicative triples $\langle a, b, c \rangle$ such that $c = a \cdot b$ proposed to simplify multiplication on secret shared inputs [4]. The initial idea was to randomize every input of an arithmetic circuit and evaluate the circuit on these random shared values to obtain $\hat{y} = f(r_1, \ldots, r_k)$. Afterwards, the difference δ_x of the random input r_i and real input x_i is computed as $\delta_i = x_i - r_i$ and made public. The second time the circuit is evaluated using public differences and initial random inputs to find the difference δ_y for the output $y = f(x_1, \ldots, x_k)$. The real output of the circuit is $y = \hat{y} + \delta_y$. The main question is correctly fixing the difference δ_y.

This idea is used in the following by the Multiplication protocol in Algorithm 3. For multiplication, the difference is computed as $\delta_y = \delta_1 \cdot r_2 + \delta_2 \cdot r_1 + \delta_1 \cdot \delta_2$, which follows trivially from the definition $y = (r_1 + \delta_1) \cdot (r_2 + \delta_2)$ as $x_i = r_i + \delta_i$ and $\hat{y} = r_1 \cdot r_2$.

The idea was proposed for secret sharing schemes that allow local addition and, actually, the randomization can be avoided during the addition step. However, computing multiplication results requires collaboration and computing the multiplication of the random inputs. Thus, Beaver triples are actually two random inputs a and b used to hide the protocol inputs and their multiplication $c = a \cdot b$ used together with public differences to restore the correct multiplication result.

Beaver triples are currently a common precomputation mechanism for SMC, as they can be computed before the inputs are known. The triples are used as helper values in multiplication according to the original proposal as shown in Multiplication protocol. Beaver triples were originally described for linearly shared secrets, but can easily be extended to shares with linear protection mechanisms such as the aforementioned MAC tags.

3.4 Basic protocols

The description of some SPDZ protocols is independent from the secret sharing method as long as the scheme defines protocols for publishing shares privately to each computing or result party, generating a random share, and generating random Beaver triples. There are three main protocols: (1) classifying the secret input, (2) opening the secret, and (3) multiplication of shared values.

Classifying and multiplication both require protocols to publish shares, which are dependant on the share representation. In general, we require three versions of the Publish protocol: (1) to declassify shares to all computing parties at the same time, (2) to publish to all computing parties separately, and (3) to declassify to non-computing parties. The declassification protocols are not specified here as they depend on the share

description. However, the general idea is that a party receives information about the secret from others and verifies its correctness.

We assume that the share representation enables a local addition operation, meaning that to obtain $[\![x]\!] + [\![y]\!]$ all computing parties $\mathcal{CP}$ only need to compute on their own shares. In addition, this implies local operations for subtraction and multiplication with a public value. The multiplication Algorithm 3 assumes the existence of precomputed and verified Beaver triples and is directly based on Beaver's ideas [4]. It is derived from triples and local computations together with publishing a value, combining those to obtain $[\![xy]\!]$ from $[\![x]\!]$ and $[\![y]\!]$.

Algorithm 3 Multiplying two secret values (Multiplication)

Data: Shared secrets $[\![x]\!]$ and $[\![y]\!]$
Result: Shared result $[\![w]\!]$, where $w = x \cdot y$

1: $\mathcal{CP}$ collaboratively choose a triple $[\![a]\!]$, $[\![b]\!]$, $[\![c]\!]$, where $c = a \cdot b$
2: $\mathcal{CP}$ compute $[\![e]\!] = [\![x]\!] - [\![a]\!]$ and $[\![d]\!] = [\![y]\!] - [\![b]\!]$
3: $\mathcal{CP}$ collaboratively open $[\![e]\!]$ and $[\![d]\!]$ to all $\mathcal{CP}$
4: $\mathcal{CP}$ compute $[\![w]\!] = [\![c]\!] + e \cdot [\![b]\!] + d \cdot [\![a]\!] + e \cdot d$
5: **return** $[\![w]\!]$

The classifying protocol in Algorithm 4 enables input and computing parties to share their secret value among all computing parties. This is a straightforward extension of the circuit randomization idea from Beaver [4]. It assumes the existence of precomputation that produces random shared values to all parties.

Algorithm 4 Classifying a private input Classify-$\mathcal{IP}_i$

Data: Input party $\mathcal{IP}_i$ has a secret x
Result: Computing parties $\mathcal{CP}$ have $[\![x]\!]$

1: $\mathcal{CP}$ collaboratively choose a precomputed randomness $[\![r]\!]$
2: $\mathcal{CP}$ open $[\![r]\!]$ to $\mathcal{IP}_i$
3: $\mathcal{IP}_i$ computes $e = x - r$ and sends e to $\mathcal{CP}$
4: $\mathcal{CP}$ compute $[\![x]\!] = [\![r]\!] + e$
5: **return** $[\![x]\!]$

These online protocols are claimed to be statically secure against an adaptive active adversary, if we have an ideal precomputation phase. However, only the case of static adversaries is proved as only this can be achieved by the precomputation protocols [26]. The adversary is allowed to corrupt at most $n - 1$ parties out of n.

Although the precomputation phase is not defined here due to dependencies on the share representation, we can still define one important step to verify the correctness of multiplicative triples. This ensures that the triple really has multiplicative relation. The triple verification process in Algorithm 5 takes two multiplicative triples and performs computations analogously to multiplication. For a finite field $\mathbb{Z}_p$ where p is prime, the probability of cheating in the verification is $\frac{1}{p}$ assuming ideal opening phase [25].

The behaviour of these protocols somewhat depends on either working with an honest minority or majority. Everything is the same in case all the protocols succeed—everyone communicates and all checks in the opening phase succeed. However, the difference comes when something fails. For example, if a two-party protocol fails then none of the parties can continue and they also can not restore the secrets. However,

Algorithm 5 Verifying the correctness of multiplicative triples

Data: Secret shared random triple $[\![x]\!]$, $[\![y]\!]$, $[\![w]\!]$
Result: $True$ if $w = x \cdot y$, $False$ in case any check fails

1: $\mathcal{CP}$ collaboratively choose a random value $[\![r]\!]$ and open it to all parties
2: $\mathcal{CP}$ collaboratively choose a triple $[\![a]\!]$, $[\![b]\!]$, $[\![c]\!]$, where presumably $c = a \cdot b$
3: $\mathcal{CP}$ compute $[\![e]\!] = r \cdot [\![x]\!] - [\![a]\!]$ and $[\![d]\!] = [\![y]\!] - [\![b]\!]$
4: $\mathcal{CP}$ open $[\![e]\!]$ and $[\![d]\!]$ to all $\mathcal{CP}$
5: $\mathcal{CP}$ compute $[\![h]\!] = r \cdot [\![w]\!] - [\![c]\!] - e \cdot [\![b]\!] - d \cdot [\![a]\!] - e \cdot d$
6: $\mathcal{CP}$ open $[\![h]\!]$ to all $\mathcal{CP}$
7: **return** $h == 0$

for (t, n)-threshold scheme a set of t honest parties could point out the malicious participants and continue the computations without them.

3.5 Initialising actively secure two-party computation

Frameworks analogous to SPDZ can be used with two computing parties and could have three considerably different initial setups. The main difference between them is how the MAC keys are defined and who knows them. In all cases, we require a homomorphic MAC. In addition, we would like the secret sharing method and homomorphic MAC to have the same operations that can be computed locally. Two parties are denoted by $\mathcal{CP}_1$ and $\mathcal{CP}_2$.

3.5.1 Asymmetric setup

Asymmetric setup differentiates the computing parties so that one gets the role of a master node ($\mathcal{CP}_1$) who defines the MAC key and the client ($\mathcal{CP}_2$) is using the keys from the master. Using the MAC to either authenticate the secret value or the share of the other party enables $\mathcal{CP}_1$ to easily verify the correctness of the declassification result. However, $\mathcal{CP}_2$ is unable to verify the MAC as it must not know the MAC secret key. It is up to the master to also define something that $\mathcal{CP}_2$ can check.

For example, the master can publish a homomorphic commitment to its input shares. That way the homomorphic properties of the commitment enable $\mathcal{CP}_2$ to compute valid commitments to all computation results and validate the declassification result. In addition, the master node must have a way to compute the openings for all commitments derived during the computation.

The MAC tag for the whole value or the share of $\mathcal{CP}_2$ can not be kept by the master node. MAC algorithms are not designed to protect the privacy of the message, thus, seeing the whole tag might leak the secret to the master node who also knows the MAC secret key. In addition, storing it on the side of $\mathcal{CP}_2$ might also leak some information about the secret or the key. Hence, for best security we need to store the tag z in a secret shared manner as $[\![z]\!]$ and both parties must be able to update their parts of the tags during computation. We can use the MAC from Section 2.1.9 where the key k is defined by $\mathcal{CP}_1$.

The used commitment has to be binding so that the client node can believe that it received a correct share in the opening phase. In addition, the hiding property of the commitment ensures the privacy of secret information in all phases but the

declassifying. A complete initialization of a protocol set with asymmetric setup is described in Chapter 4.

3.5.2 Symmetric setup

A symmetric setup means that both computing parties define similar parameters. A direct continuation of the previous asymmetric setting would be that both parties $\mathcal{CP}_i$ in the symmetric setting define their own MAC keys k_i. This would mean that on top of the secret sharing method we have two MAC tags $z^{(1)},\ z^{(2)}$ where both parties can verify one of them during the declassification phase. As in the asymmetric, case we need a to keep the tags in shares $[\![z^{(1)}]\!]$ and $[\![z^{(2)}]\!]$ to avoid revealing the secrets. For example, we can use the MAC from Section 2.1.9 where both parties define their own key.

The main benefit of this setup over the asymmetric one is that the protocol descriptions would also become symmetric. This simplifies the notation and also means that the parties can do exactly the same workload in parallel. In some sense, this enables us to gain more efficient time usage. More precisely, it is unlikely to have protocols where one party has to wait between sending and receiving network message without having any computations to perform. Furthermore, we can only use the cheap MAC algorithm and do not have a need for more expensive homomorphic commitments that we used in the asymmetric case.

Our specification of a protocol set with symmetric setup can be found in Chapter 6. Symmetric setup with MACs was also used by BDOZ [7]. A setup with using only commitments to the secrets was introduced in [25].

3.5.3 Shared key setup

The shared key model is a further extension changing the symmetric setup so that instead of both parties defining a key they share one key $[\![k]\!]_*$ between them. This defines a threshold MAC algorithm where all parties must participate in the verification of the tag. It can give additional efficiency gains as now the parties only have to update a single tag $[\![z]\!]$ during the computations. However, the sharing $[\![k]\!]_*$ is special as it has to define some additional information, allowing parties to verify the correctness of the restored key and checked tags. The shared key setup is the approach currently used by the SPDZ framework.

However, there are well-known difficulties with this approach as the knowledge of the secret key is usually needed to verify the MAC tags. One possible solution is to not verify any opened results before all computation is done. Afterwards, it is possible to restore the MAC key and verify all the results at once. However, there are drawbacks because parties can only notice cheating very late and they must agree on a new key before next computations. In addition, changing the key means that after verifying the correctness of opened values, the shares of the outputs or intermediate results from the checked computations can not be reused.

The first version of SPDZ used the previous approach but they substituted it to a way to collaboratively check the MAC without revealing the keys [23]. The idea is that if the secret value is made public and the tag is a linear combination of this public value and the MAC key then it can be checked by computing on the shares and publishing only the verification result.

Chapter 4

Asymmetric two-party computation

This section introduces our initialisation of an asymmetric two-party secure computation scheme. It includes the share representation as well as protocols specific to this representation, including precomputation.

4.1 Protection domain setup

We consider an additive secret sharing scheme in $\mathbb{Z}_N$ where N is a Paillier modulus and we have a Paillier keypair (pk, sk) corresponding to this modulus. The party $\mathcal{CP}_1$ knows this keypair, while $\mathcal{CP}_2$ only knows the public key pk. In addition, $\mathcal{CP}_2$ must be convinced that it is a valid key. $\mathcal{CP}_1$ uses $\mathsf{Enc}_{pk}(x)$ to commit to a value x and stores the encryption randomness as the decommitment. $\mathcal{CP}_1$ can also define a key $k \leftarrow \mathbb{Z}_N$ for message authentication together with a commitment $\mathsf{Enc}_{pk}(k)$, which is also known by $\mathcal{CP}_2$. To distinguish a commitment from encrypting, we denote $(\!|k|\!)_{pk} = \mathsf{Enc}_{pk}(k, r_k)$ which is a fixed value depending on the randomness r_k that $\mathcal{CP}_1$ chose when initially encrypting it. We use the same notation to represent encryptions that $\mathcal{CP}_1$ has published as commitments during the computation. These encryption and MAC keys must be usable throughout the life of the shares computed with them.

Each secret value x is is represented by a tuple

$$[\![x]\!]_N = \langle \Delta, x_1, x_2, r, (\!|x_1|\!)_{pk}, z_1, z_2 \rangle$$

such that $x = x_1 + x_2 + \Delta$ and $z_1 + z_2 = k \cdot (x_1 + x_2)$. The values Δ and $(\!|x_1|\!)_{pk} = \mathsf{Enc}_{pk}(x_1, r)$ are public whereas $\mathcal{CP}_i$ has private values z_i and x_i. The public modifier Δ is always 0 for random values and is used to enable fast addition of a share and public constant. Value r is kept by $\mathcal{CP}_1$ to open the commitment to $(\!|x_1|\!)_{pk}$ of share $[\![x]\!]_N$. This randomness also enables us to write protocols so that actually only $\mathcal{CP}_2$ computes $(\!|x_1|\!)_{pk}$ and $\mathcal{CP}_1$ recomputes the encryption if needed. This is a reasonable step because, in reality, $\mathcal{CP}_1$ only needs the encryption result during the zero-knowledge proofs in the precomputation and avoiding computation on ciphertexts enables faster online computation. We sometimes use labels as $z_1^{(x)}$ and $\Delta^{(x)}$ to denote that these part of the share representation $[\![x]\!]$. For security, we need to rely on the security of MAC as showed in Theorem 2.1.6 and security of the commitment shown by Theorem 4.1.1.

Theorem 4.1.1. *The commitment scheme based on (t, ε)-IND-CPA secure cryptosystem where the commitment $c = \mathsf{Enc}_{pk}(m, r)$ is the encryption and opening $d = (m, r)$*

is the message together with the encryption randomness is (t, ε)-hiding and unconditionally binding if the public key of the cryptosystem is publicly verifiable or proved to be valid using a zero-knowledge proof.

Proof Sketch. The perfect binding property follows from the fact that public key uniquely fixes the secret key and, hence, it is possible to decrypt the commitment.

The hiding property follows from the definition of the IND-CPA security of a cryptosystem. □

Furthermore, addition is a local operation as we can just sum the additive share elements pairwise and use the homomorphic properties of Paillier cryptosystem. In addition, the existence of an addition protocol (Addition) also defines a subtraction protocol (Subtraction) and a protocol for multiplying the shared value with a public constant (Constant Multiplication). Moreover, adding a public value to the shared secret (Constant Addition) only requires modifying the value Δ. In a way, public value v can also be thought of as having a fixed share representation

$$[\![v]\!]_N = \langle \Delta = v, v_1 = 0, v_2 = 0, r = 1, (\![v_1]\!)_{pk} = 1, z_1 = 0, z_2 = 0 \rangle \ .$$

Every time when $\mathcal{CP}_2$ receives a ciphertext and uses it to compute a response to $\mathcal{CP}_1$, it has to verify that it is valid. For the Paillier cryptosystem, the validity means that the ciphertext c belongs to $\mathbb{Z}^*_{N^2}$. Checking that $c \in \mathbb{Z}^*_{N^2}$ is equivalent to checking that $\gcd(c, N) = 1$. However, it is actually unlikely for either party to send invalid ciphertexts. If $\mathcal{CP}_2$ sends an invalid ciphertext, it means that $\mathcal{CP}_2$ can actually factor N and break the security of this setup, thus, it is as likely as factoring. On the other hand, if $\mathcal{CP}_1$ sends an invalid ciphertext then it deliberately leaks its secret key to $\mathcal{CP}_2$.

Computing parties must also be able to communicate with result parties $\mathcal{RP}_i$ and input parties $\mathcal{IP}_i$. We need to enable the computing parties to receive inputs from input parties and to make sure that results parties can learn the correct outputs of the protocol. Input and result parties know the Paillier public key N and have received the commitment $(\![k]\!)_{pk}$ of the MAC key, thus they are in a similar role to $\mathcal{CP}_2$.

In the security proofs of this section, the computational security of the protocols results from the computational security of the Paillier cryptosystem. Therefore, by choosing suitable keys, we can make the protocols as secure as we require. However, we specially stress the probability $\frac{1}{p}$ that a party can cheat against the MAC (Theorem 2.1.6), as for some cases, picking securer keys may not mean that also this probability $\frac{1}{p}$ lessens. Therefore, it could be seen as a fixed value rather than a value that we can make arbitrarily negligible. However, when using the Paillier cryptosystem, increasing the modulus would also increase the value of p.

We assume the existence of secure authenticated communication channels and exclude eavesdropping and modification of network messages from the security analysis. In practice, secure network channels are achieved using standard secure channel implementations like TLS [27].

4.2 Publishing shared values

The secret value may either be made public to $\mathcal{CP}_1$, $\mathcal{CP}_2$ or $\mathcal{RP}_i$ and we need to have different protocols for these cases. We can use a combination of the first two to publish the value to both computing participants at the same time (Publish-both-$\mathcal{CP}_i$).

Clearly, sending the corresponding share to other party is a similar step in all of these protocols. However, the mechanisms for verifying the correctness of the given share value are different.

Algorithm 6 Publishing a shared value to $\mathcal{CP}_1$ (Publish-$\mathcal{CP}_1$)

Data: Shared secret $[\![x]\!]_N$
Result: $\mathcal{CP}_1$ learns the value x

1: $\mathcal{CP}_2$ sends x_2 and z_2 to $\mathcal{CP}_1$
2: $\mathcal{CP}_1$ verifies $z_1 + z_2 = k \cdot (x_1 + x_2)$
3: **return** $\mathcal{CP}_1$ outputs $x_1 + x_2 + \Delta$

It is easy to see that $\mathcal{CP}_1$ can perform the verification in Publish-$\mathcal{CP}_1$ (Algorithm 6) because $\mathcal{CP}_1$ knows all the plain values in the verification equation. The security of the MAC algorithm ensures that $\mathcal{CP}_2$ is unlikely to pass the verification when submitting a share or a tag not obtained from correct computations.

As the value $(\![x_1]\!)_{pk}$ is a commitment to value x_1 from $\mathcal{CP}_1$ then $\mathcal{CP}_2$ can verify the correctness of received x_1 in Publish-$\mathcal{CP}_2$ (Algorithm 7) by successfully opening the commitment. The perfectly binding property of the commitment ensures that $\mathcal{CP}_1$ can only pass the verification with the unique correct share and randomness pair.

Algorithm 7 Publishing a shared value to $\mathcal{CP}_2$ (Publish-$\mathcal{CP}_2$)

Data: Shared secret $[\![x]\!]_N$
Result: $\mathcal{CP}_2$ learns the value x

1: $\mathcal{CP}_1$ sends x_1 and r to $\mathcal{CP}_2$
2: $\mathcal{CP}_2$ verifies $(\![x_1]\!)_{pk} = \mathsf{Enc}_{pk}(x_1, r)$
3: **return** $\mathcal{CP}_2$ outputs $x_1 + x_2 + \Delta$

In the following, we give a full proof for the security of Publish-$\mathcal{CP}_i$ protocol, later in the thesis we give a more brief overview about the ideal world and simulator for the security proofs.

Theorem 4.2.1. *Algorithms* Publish-$\mathcal{CP}_1$ *and* Publish-$\mathcal{CP}_2$ *for publishing the value to one computing party are correct.* Publish-$\mathcal{CP}_1$ *is computationally secure against cheating* $\mathcal{CP}_2$ *with an additional statistical* $\frac{1}{p}$*-error probability, where* p *is the smaller prime factor of* N *and computationally secure against cheating* $\mathcal{CP}_1$*. Protocol* Publish-$\mathcal{CP}_2$ *is perfectly secure against a cheating* $\mathcal{CP}_1$ *and computationally secure against a cheating* $\mathcal{CP}_2$*.*

Proof sketch. For correctness, we need that $x = x_1 + x_2 + \Delta$ or the protocol aborts. The former is trivially true by the definition of the share representation and, in case of honest participants, the verification always succeeds. In the following, we show the security in the stand-alone setting.

We describe the ideal secure execution of this protocol using the model with a trusted third party (TTP) who always behaves honestly. The ideal functionality of publishing to $\mathcal{CP}_i$ is such that the TTP notifies $\mathcal{CP}_j$ that it is about to declassify x. On input *Continue* it sends x and Δ to $\mathcal{CP}_i$ and on input *Abort* it cancels the publishing. The real setup where the publish protocol is executed is such that the parties have executed some protocols and received the output x and correction value Δ and then $\mathcal{CP}_j$ sends the declassification values to $\mathcal{CP}_i$. However, the previous protocol runs are

secure and can be replaced by a TTP, who gives the shares of x to the computing parties. These two execution models are illustrated on Figure 4.1 for Publish-$\mathcal{CP}_1$, the case for Publish-$\mathcal{CP}_2$ is analogous.

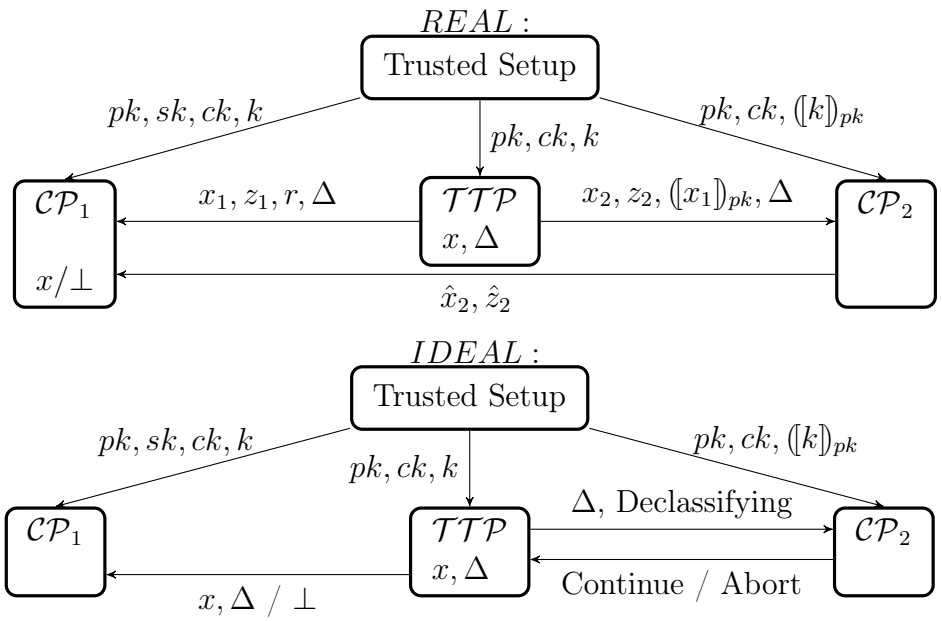


Figure 4.1: Ideal and real protocol execution of Publish-$\mathcal{CP}_1$

Therefore, for both protocols, we need to show a simulator, such that the output distributions of the simulated adversary and the $\mathcal{CP}_i$ of the ideal world coincide with the outputs of the adversary and $\mathcal{CP}_i$ in the real world.

Firstly, consider a corrupted $\mathcal{CP}_2$ in Publish-$\mathcal{CP}_2$. The corresponding simulator at first receives x and Δ from TTP if the ideal publishing succeeds and can easily create suitable values $x_1, x_2, z_2, (\![x_1]\!)_{pk} = \mathsf{Enc}_{pk}(x_1, r), r$, where $x_1 + x_2 = x$. The simulator can forward these to the corrupted $\mathcal{CP}_2$ as two messages in the real protocol execution. The outputs clearly coincide as the corrupted $\mathcal{CP}_2$ always gets the same x and honest $\mathcal{CP}_1$ has seen the same Δ.

Analogously, the simulator for corrupted $\mathcal{CP}_1$ in Publish-$\mathcal{CP}_1$ is straightforward. It receives x and Δ from TTP and can prepare $x_1, x_2, z_2, (\![x_1]\!)_{pk} = \mathsf{Enc}_{pk}(x_1, r), r$, where $x_1 + x_2 = x$. However, it can not fix a correct z_1, but it can compute $\mathsf{Enc}_{pk}(z_1) = \mathsf{Enc}_{pk}(k)^x \cdot \mathsf{Enc}_{pk}(-z_2)$. Therefore, it can simulate the protocol for the equivalent of $\mathcal{CP}_1$, who expects z_1 in an encrypted form.

In the following, for publishing to $\mathcal{CP}_i$ we only consider the case where the other party $\mathcal{CP}_j$ is corrupted. We assume that there has been a setup phase beforehand, where all parties and the simulator learned $(\![k]\!)_{pk}$ and N. In addition, $\mathcal{CP}_1$ learned the private key for the Paillier cryptosystem and the TTP also has k. The setup is shared between protocol runs and is not a part of this protocol execution.

Publishing to $\mathcal{CP}_1$. The simulator $\mathcal{S}$ at first picks a MAC key $k_{\mathcal{S}}$ for itself. It then generates the share $x_2, z_2, (\![x_1^*]\!)_{pk}$ to the adversary $\mathcal{A}$ who responds with $\hat{x}_2, \hat{z}_2$. The simulator either finishes with *Abort* or *Continue* where the output Ψ_1 of $\mathcal{CP}_1$ will be $\perp$ in the former and x in the latter case as given by the ideal world TTP. The general game for this is specified as $G_1^{\mathcal{A}}(x)$ and $G_2^{\mathcal{A}}(x)$ rewrites this with the specific details of the simulator.

The corresponding real protocol execution can be seen from $G_3^{\mathcal{A}}(x)$ with the exact details of the honest $\mathcal{CP}_1$ and TTP in $G_4^{\mathcal{A}}(x)$. It can be seen that the games $G_2^{\mathcal{A}}(x)$

$G_1^{\mathcal{A}}(x, \Delta)$

$sk, pk, k, ck, (\![k]\!)_{pk} \leftarrow \text{Setup}$
$\Delta, x_2, z_2, c^* \leftarrow \mathcal{S}(pk, (\![k]\!)_{pk})$
$\hat{x}_2, \hat{z}_2 \leftarrow \mathcal{A}(\Delta, x_2, z_2, c^*)$
$\Psi_2 \leftarrow \mathcal{A}$
$b \leftarrow \mathcal{S}(\hat{x}_2, \hat{z}_2)$
if $b = Continue$
  then $\Psi_1 = x$
  else $\Psi_1 = \perp$
return (Ψ_1, Ψ_2)

$G_2^{\mathcal{A}}(x, \Delta)$

$sk, pk, k, ck, (\![k]\!)_{pk} \leftarrow \text{Setup}$
$\Delta \leftarrow \mathcal{TTP}(pk, ck, k)$
$x_2, z_2, x_1^*, k_{\mathcal{S}} \leftarrow \mathbb{Z}_N$
$z_1 = k_{\mathcal{S}} \cdot (x_1^* + x_2) - z_2$
$c^* \leftarrow \mathsf{Enc}_{pk}(x_1^*)$
$\hat{x}_2, \hat{z}_2 \leftarrow \mathcal{A}(\Delta, x_2, z_2, c^*)$
$\Psi_2 \leftarrow \mathcal{A}$
if $k_{\mathcal{S}} \cdot (x_1^* + \hat{x}_2) = z_1 + \hat{z}_2$
  then $\Psi_1 = x$
  else $\Psi_1 = \perp$
return (Ψ_1, Ψ_2)

Security game 4.2.1: Publishing to $\mathcal{CP}_1$ with simulator

and $G_4^{\mathcal{A}}(x)$ are almost equivalent, except for the usage of a different key, different commitments $(\![x_1]\!)_{pk}$, $(\![x_1^*]\!)_{pk}$ and some additional computations in $G_4^{\mathcal{A}}(x)$. By IND-CPA security we know that $(\![x_1]\!)_{pk}$ and $(\![x_1^*]\!)_{pk}$ are computationally indistinguishable. We know that starting $\mathcal{A}$ with the same randomness ϕ_2 will always result in the same output Ψ_2.

As the simulator does not know the real key k, it may falsely accept when the real protocol run rejects the inputs. However, according to the MAC security, the simulator falsely accepts with probability less than $\frac{1}{p}$ which is the same as in the real protocol run. From the IND-CPA security of the cryptosystem we know that $(\![k]\!)_{pk}$ hides k, therefore using $k_{\mathcal{S}}$ instead of k is computationally indistinguishable. However, in the ideal protocol run we know that $\mathcal{CP}_1$ always gets the correct output x, but due to the possible cheating in the MAC algorithm there is a $\frac{1}{p}$ possibility that the real protocol run finishes with $\hat{x}$. Therefore, the outputs of the ideal and real world coincide except with probability $\frac{1}{p}$.

$G_3^{\mathcal{A}}(x, \Delta)$

$sk, pk, k, ck, (\![k]\!)_{pk} \leftarrow \text{Setup}$
$\Delta, x_1, z_1, r, x_2, z_2, c \leftarrow \mathcal{TTP}(x, \Delta, pk, ck, k)$
$\mathcal{CP}_1(\Delta, x_1, z_1, r)$
$\hat{x}_2, \hat{z}_2 \leftarrow \mathcal{A}(\Delta, x_2, z_2, c)$
$\Psi_2 \leftarrow \mathcal{A}$
$\Psi_1 \leftarrow \mathcal{CP}_1(\hat{x}_2, \hat{z}_2)$
return (Ψ_1, Ψ_2)

$G_4^{\mathcal{A}}(x, \Delta)$

$sk, pk, k, ck, (\![k]\!)_{pk} \leftarrow \text{Setup}$
$x_1, z_1 \leftarrow \mathbb{Z}_N, r \leftarrow \mathbb{Z}_N^*$
$c \leftarrow \mathsf{Enc}_{pk}(x_1, r)$
$x_2 = x - x_1 - \Delta$
$z_2 = k \cdot x - z_1$
$\hat{x}_2, \hat{z}_2 \leftarrow \mathcal{A}(\Delta, x_2, z_2, c)$
$\Psi_2 \leftarrow \mathcal{A}$
if $k \cdot (x_1 + \hat{x}_2) = z_1 + \hat{z}_2$
  then $\Psi_1 = x$
  else $\Psi_1 = \perp$
return (Ψ_1, Ψ_2)

Security game 4.2.2: Publishing to $\mathcal{CP}_1$ in real protocol run

Publishing to $\mathcal{CP}_2$. The simulator $\mathcal{S}$ picks the shares that $\mathcal{CP}_1$ should receive and sends them to $\mathcal{A}$ to simulate the TTP in the real protocol execution. Then $\mathcal{A}$ sends the declassification message $\hat{x}_1, \hat{r}$. The simulator outputs *Continue* in case $\mathsf{Enc}_{pk}(x_1, r) = \mathsf{Enc}_{pk}(\hat{x}_1, \hat{r})$ which is the same check that an honest $\mathcal{CP}_2$ would do to check the commitment. The simulated protocol run can be seen in the game $G_5^{\mathcal{A}}(x)$ and the simulator specifics have been written out in $G_6^{\mathcal{A}}(x)$.

$G_5^{\mathcal{A}}(x, \Delta)$

$sk, pk, k, ck, (\![k]\!)_{pk} \leftarrow \text{Setup}$
$\Delta, x_1, z_1, r \leftarrow \mathcal{S}(pk, (\![k]\!)_{pk})$
$\hat{x}_1, \hat{r} \leftarrow \mathcal{A}(\Delta, x_1, z_1, r)$
$\Psi_1 \leftarrow \mathcal{A}$
$b \leftarrow \mathcal{S}(\hat{x}_1, \hat{r})$
if $b = Continue$
  then $\Psi_2 = x$
  else $\Psi_2 = \perp$
return (Ψ_1, Ψ_2)

$G_6^{\mathcal{A}}(x, \Delta)$

$sk, pk, k, ck, (\![k]\!)_{pk} \leftarrow \text{Setup}$
$\Delta \leftarrow \mathcal{TTP}(x, \Delta)$
$x_1, z_1 \leftarrow \mathbb{Z}_N$
$r \leftarrow \mathbb{Z}_N^*$
$\hat{x}_1, \hat{r} \leftarrow \mathcal{A}(\Delta, x_1, z_1, r)$
$\Psi_1 \leftarrow \mathcal{A}$
if $\mathsf{Enc}_{pk}(\hat{x}_1, \hat{r}) = \mathsf{Enc}_{pk}(x_1, r)$
  then $\Psi_2 = x$
  else $\Psi_2 = \perp$
return (Ψ_1, Ψ_2)

Security game 4.2.3: Publishing to $\mathcal{CP}_2$ with simulator

An analogous game of the real protocol run with the TTP representing the previous computations and a real honest $\mathcal{CP}_2$ is shown in $G_7^{\mathcal{A}}(x)$. Finally, the game $G_8^{\mathcal{A}}(x)$ shows the real execution with the exact workings of TTP and $\mathcal{CP}_2$.

$G_7^{\mathcal{A}}(x, \Delta)$

$sk, pk, k, ck, (\![k]\!)_{pk} \leftarrow \text{Setup}$
$x_1, z_1, r, x_2, z_2, c \leftarrow \mathcal{TTP}(x, \Delta)$
$\mathcal{CP}_2(\Delta, x_2, z_2, c)$
$\hat{x}_1, \hat{r} \leftarrow \mathcal{A}(\Delta, x_1, z_1, r)$
$\Psi_1 \leftarrow \mathcal{A}$
$\Psi_2 \leftarrow \mathcal{CP}_2(\hat{x}_1, \hat{r})$
return (Ψ_1, Ψ_2)

$G_8^{\mathcal{A}}(x, \Delta)$

$sk, pk, k, ck, (\![k]\!)_{pk} \leftarrow \text{Setup}$
$x_1, z_1 \leftarrow \mathbb{Z}_N$
$r \leftarrow \mathbb{Z}_N^*$
$c \leftarrow \mathsf{Enc}_{pk}(x_1, r)$
$x_2 = x - x_1 - \Delta$
$z_2 = k \cdot x - z_1$
$\hat{x}_1, \hat{r} \leftarrow \mathcal{A}(\Delta, x_1, z_1, r)$
$\Psi_1 \leftarrow \mathcal{A}$
if $\mathsf{Enc}_{pk}(\hat{x}_1, \hat{r}) = c$
  then $\Psi_2 = x$
  else $\Psi_2 = \perp$
return (Ψ_1, Ψ_2)

Security game 4.2.4: Publishing to $\mathcal{CP}_2$ in real protocol run

We can see that besides some additional computations, the games and outputs of $G_6^{\mathcal{A}}(x)$ and $G_8^{\mathcal{A}}(x)$ coincide and the simulation is perfect. The simulator always accepts the same cases as the real protocol run because the commitment is perfectly binding. Therefore, the outputs of the real and ideal world coincide. □

There are two special cases when result parties $\mathcal{RP}_i$ may need to learn the computation outcomes. In one scenario, the computing parties $\mathcal{CP}_i$ are allowed to also learn the outcome, whereas in the other case, only the output party $\mathcal{RP}_i$ can learn the declassification result.

It is straightforward to satisfy the first case. The computing parties just run the declassification protocol Publish-both-$\mathcal{CP}_i$ and they both forward the declassified result to $\mathcal{RP}_i$. The result party only has to verify that both computing parties sent the same declassified result. Differently from either Publish-$\mathcal{CP}_i$, in Publish-$\mathcal{CP}\&\mathcal{RP}_i$, the result party $\mathcal{RP}_i$ can not easily check which of the computing parties has tried to cheat, if the verification does not succeed.

Theorem 4.2.2. *Publishing values to both computing parties and to result parties* (Publish-$\mathcal{CP}$&$\mathcal{RP}_i$) *is correct and as secure against corrupted* $\mathcal{CP}_j$ *as* Publish-$\mathcal{CP}_i$ *and perfectly secure against a corrupted* $\mathcal{RP}_i$.

Proof sketch. These properties result from the correctness and security of publishing to either of the computing parties in Publish-$\mathcal{CP}_i$. The fact that the result party verifies that both computing parties sent the same same result also detects cheating after finishing the predefined publishing protocols.

It is trivial to simulate the protocol run for a corrupted $\mathcal{RP}_i$ as the simulator can just forward the value x from the TTP to $\mathcal{RP}_i$. □

The second case requires more work on the side of the result party as given in Publish-$\mathcal{RP}_i$ (Algorithm 8). The idea is that the result party $\mathcal{RP}_i$ can verify the commitment similarly to $\mathcal{CP}_2$, but it can not verify the MAC tag as it does not know the secret key k. However, $\mathcal{RP}_i$ can verify that it has the right share using the proof of correct share representation similarly to the Singles protocol. The main drawback of this protocol is that the corresponding zero-knowledge proof protocol, that we instantiate with a version of CdsZkMul protocol (CdsZKTags), can be quite expensive. However, in real life we would like to avoid heavy computational needs on the side of the input and result parties in order to make this usable in a variety of different settings, including, for example, those where the input and result parties are using mobile devices.

Algorithm 8 Publishing a shared value to $\mathcal{RP}_i$ (Publish-$\mathcal{RP}_i$)

Data: Shared secret $[\![x]\!]_N$
Result: Result party $\mathcal{RP}_i$ learns the value x

1: $\mathcal{CP}_1$ sends Δ, x_1 and r to $\mathcal{RP}_i$
2: $\mathcal{CP}_2$ sends Δ, x_2, z_2, and $(\![x_1]\!)_{pk}$ to $\mathcal{RP}_i$
3: $\mathcal{RP}_i$ verifies that $\mathcal{CP}_1$ and $\mathcal{CP}_2$ sent the same Δ
4: $\mathcal{RP}_i$ verifies that $(\![x_1]\!)_{pk} = \mathsf{Enc}_{pk}(x_1, r)$
5: $\mathcal{CP}_1$ proves $z_1 + z_2 = k \cdot (x_1 + x_2)$ to $\mathcal{RP}_i$ using CdsZKTags
6: **return** $\mathcal{RP}_i$ computes $x = x_1 + x_2 + \Delta$

Intuitively, the protocol is secure if neither party can make $\mathcal{RP}_i$ accept a faulty x.

Theorem 4.2.3. *Algorithm* Publish-$\mathcal{RP}_i$ *for declassifying shared secrets to result parties is correct. Protocol* Publish-$\mathcal{RP}_i$ *is computationally secure against corrupted* $\mathcal{CP}_2$ *with possible* $\frac{1}{p}$ *error probability, where* p *is the smaller prime factor of* N. *Protocol* Publish-$\mathcal{RP}_i$ *is perfectly secure against a corrupted* $\mathcal{CP}_1$. *It is also computationally secure against malicious* $\mathcal{RP}_i$, *assuming a simulatable* CdsZKTags *protocol.*

Proof sketch. The correctness means that in case of honest participants, $\mathcal{RP}_i$ receives the result x. It follows trivially from the share description and correctness of the used ZK proof.

For security in the stand-alone setting, we are interested in the cases where either of the parties is corrupted alone. We show the simulator construction for these cases. The ideal model and real world execution scenarios are analogous to those of the Publish-$\mathcal{CP}_i$, except that $\mathcal{RP}_i$ is supposed to learn the final outcome.

The simulation in both $\mathcal{CP}_j$ cases begins by using the corresponding simulator from Publish-$\mathcal{CP}_i$ where $\mathcal{CP}_j$ is corrupted. It at first acts on behalf of the previous protocols

and gives the shares of a secret x^* to the corrupted $\mathcal{CP}_j$. The $\mathcal{CP}_j$ has to release the same values as in Publish-$\mathcal{CP}_i$ with the additional values Δ on both sides and $(\![x_1]\!)_{pk}$ from $\mathcal{CP}_2$. The simulator checks the correctness of these by either opening the commitment $(\![x_1]\!)_{pk}$ or verifying the tags with respect to its own key as in Publish-$\mathcal{CP}_i$.

In addition, the simulator of corrupted $\mathcal{CP}_2$ verifies the correctness of $(\![x_1]\!)_{pk}$ as an honest $\mathcal{RP}_i$ would by checking the opening $(\![x_1]\!)_{pk} = \mathsf{Enc}_{pk}(x_1, r)$. Finally, the simulator also has to check the validity of Δ, which it can do by storing the same Δ that the simulator learned from the TTP and forwarded to the corrupted $\mathcal{CP}_j$.

For the CdsZKTags part of the corrupted $\mathcal{CP}_1$, the simulator can define $x_2 = -x_1$ and $z_2 = -z_1$. This way the proof has the statement $0 = k \cdot 0$, which is correct independently of the modulus and the key, and the simulator can behave as an honest $\mathcal{RP}_i$ who is the verifier in this proof. Hence, we know that CdsZKTags has the correct inputs and that the proof does not leak z_2 and x_2, which mean that this special case of the proof is indistinguishable from the real case for the corrupted $\mathcal{CP}_1$. The simulations of corrupted computing parties give the same output distribution as the real protocol run, as the corrupted party has the same view and the result party learns either x or $\perp$, depending on the computing parties correctly participating in the protocol.

For corrupted $\mathcal{RP}_i$, the simulator receives x and Δ from the TTP and can fix x_1, x_2, z_2, r, $(\![x_1]\!)_{pk}$ that it forwards to the result party. In addition, the simulator can compute $\mathsf{Enc}_{pk}(z_1) = \mathsf{Enc}_{pk}(k)^x \cdot \mathsf{Enc}_{pk}(-z_2)$ for the CdsZKTags. Hence, it can act as a simulator of the proof because it has all the correct queries $(\![x_1]\!)_{pk}$, $(\![k]\!)_{pk}$ and $(\![z_1]\!)_{pk}$. The output distributions coincide as the corrupted $\mathcal{RP}_i$ has the same view and the output of the computing parties depends on the final decision of $\mathcal{RP}_i$. □

An interesting aspect is that cheating in CdsZKTags can not help $\mathcal{CP}_1$ to make $\mathcal{RP}_i$ accept a wrong x. According to our assumptions, the $\mathcal{CP}_i$ are not allowed to collude and this proof can only make RPi to accept a faulty value from $\mathcal{CP}_2$. However, it can easily make the publishing protocol fail.

On the downside, as in Publish-$\mathcal{CP}\&\mathcal{RP}_i$ the Publish-$\mathcal{RP}_i$ also doest not allow $\mathcal{RP}_i$ to easily verify which of the parties $\mathcal{CP}_i$ tried to cheat if any of the checks fails. Theoretically, it would be possible to achieve by having both parties prove the correctness of all their previous computations.

4.3 Random share generation

The random share protocol (Singles) must generate a valid share representation of $[\![x]\!]_N = \langle \Delta, x_1, x_2, r, (\![x_1]\!)_{pk}, z_1, z_2 \rangle$ for a random x where the participants do not know the value of x. This is a necessary protocol for sharing the inputs and producing random multiplicative triples. Both parties choose a random additive share and collaborate to fix the MAC tag as described in Algorithm 9. In addition, this gives a uniformly distributed random value $[\![x]\!]_N$ as a result as the sum of two uniformly distributed values is uniform. Furthermore, the value is uniformly distributed even if only one of the participants generated its share correctly.

Our initialisation of the proof of $z_1 + z_2 = k \cdot (x_1 + x_2)$ follows CdsZkMul (Algorithm 1), except for the initial messages, because a part of the query can be computed from the share by the verifier $\mathcal{CP}_2$. However, this actually means that the security of this protocol does not follow easily from the CdsZkMul. The best possibility would be to make $\mathcal{CP}_2$ to prove that q_3 is computed correctly. This way we could ensure the

Algorithm 9 Generating a random share (Singles)

Data: No inputs

Result: Shares $[\![x]\!]_N$ of random value x

Round: 1

  $\mathcal{CP}_i$ sets $\Delta = 0$

  $\mathcal{CP}_1$ generates $x_1 \leftarrow \mathbb{Z}_N$

  $\mathcal{CP}_1$ generates $r \leftarrow \mathbb{Z}_N^*$

  $\mathcal{CP}_1$ sends $(\![x_1]\!)_{pk} = \mathsf{Enc}_{pk}(x_1, r)$ to $\mathcal{CP}_2$

  $\mathcal{CP}_2$ generates $x_2, z_2 \leftarrow \mathbb{Z}_N$

  $\mathcal{CP}_2$ computes $c = (\![k]\!)_{pk}^{x_2} \cdot \mathsf{Enc}_{pk}(-z_2)$

  $\mathcal{CP}_2$ sends c to $\mathcal{CP}_1$

Round: 2

  $\mathcal{CP}_1$ computes $z_1 = k \cdot x_1 + \mathsf{Dec}_{sk}(c)$

  $\mathcal{CP}_2$ verifies that $(\![x_1]\!)_{pk}$ is a valid ciphertext

Verification:

  $\mathcal{CP}_1$ proves the correctness of MAC tags $z_1 + z_2 = k \cdot (x_1 + x_2)$ to $\mathcal{CP}_2$

return $[\![x]\!]_N$

simulatability and, hence, zero-knowledge property of this protocol, but would lose a lot of efficiency for the additional zero-knowledge proof. For now, we just assume that $\mathcal{CP}_1$ verifies that q_3 contains the right plaintext, which leaves a small hole that $\mathcal{CP}_2$ might use $\mathcal{CP}_1$ as kind of a decryption oracle, to check if it formed the q_3 correctly to contain the multiplication of the plaintexts from q_1 and q_2. We keep this protocol in hopes that we can define a simulatable protocol with the same form queries. In the future we should specify a simulatable version of CdsZKTags. A simple way to add some additional verification would be that occasionally the parties decide to discard the random value and open all values used for computing this or proving the correctness.

The idea of CdsZKTags is to ensure to $\mathcal{CP}_2$ that $\mathcal{CP}_1$ has all the values to accept this share during the opening phase in Publish-$\mathcal{CP}_1$. More precisely, after this proof, the $\mathcal{CP}_2$ knows that the share is correctly formed and that, if $\mathcal{CP}_2$ uses it correctly in the following computations, then $\mathcal{CP}_1$ should be able to open all the following results. Thus, $\mathcal{CP}_2$ can avoid malicious $\mathcal{CP}_1$ framing $\mathcal{CP}_2$ it as a malicious party. An honest $\mathcal{CP}_1$ already has this property because the commitment $(\![x_1]\!)_{pk}$ is public. This property is important as the Singles protocol is the basis for input sharing protocol Classify-$\mathcal{CP}_i$ which is the first step of all computations. Thus, verifying the correctness of $[\![x]\!]$ in Singles can be a basis for showing the correctness of all outputs.

Theorem 4.3.1. *Algorithm* Singles *for generating random shares is correct.*

Proof. The correctness of $(\![x_1]\!)_{pk} = \mathsf{Enc}_{pk}(x_1, r)$ is trivial in case of honest $\mathcal{CP}_1$. It is also trivial that $x_1 + x_2 + \Delta = x$ as the value of x is not predefined. For correctness we need to show that $z_1 + z_2 = k \cdot (x_1 + x_2)$ so that the verification succeeds:

$$\begin{aligned} z = z_1 + z_2 &= k \cdot x_1 + \mathsf{Dec}_{sk}(c) + z_2 = k \cdot x_1 + \mathsf{Dec}_{sk}((\![k]\!)_{pk}^{x_2} \cdot (\![-z_2]\!)_{pk}) + z_2 \\ &= k \cdot x_1 + k \cdot x_2 - z_2 + z_2 = k \cdot x_1 + k \cdot x_2 = k \cdot (x_1 + x_2) \ . \end{aligned}$$

□

The basic ideal functionality of the Singles protocol would be such that both parties notify the TTP that they are interested in sharing a random value. Then, the TTP

Algorithm 10 CdsZkMul for correctness of MAC tags (CdsZKTags)

Setup: Commitment parameters ck,
Paillier keypair (pk, sk) from the protection domain setup
Data: Shared secret $[\![w]\!]_N$
Result: $True$ for successful proof of $z_{1,w} + z_{2,w} = k(w_1 + w_2)$, $False$ in case of any failure

Round: 1
  $\mathcal{CP}_1$ computes and sends $(\![z_1^{(w)}]\!)_{pk} = \mathsf{Enc}_{pk}(z_1^{(w)})$ to $\mathcal{CP}_2$
Round: 2
  $\mathcal{CP}_2$ checks that $(\![z_1^{(w)}]\!)_{pk}$ is a valid ciphertext
  $\mathcal{CP}_i$ sets $q_1 = (\![w_1]\!)_{pk}$, $q_2 = (\![k]\!)_{pk}$, $q_3 = (\![z_1^{(w)}]\!)_{pk} \cdot ((\![k]\!)_{pk}^{w_2} \cdot \mathsf{Enc}_{pk}(-z_2^{(w)}))^{-1}$
  $\mathcal{CP}_2$ generates $e_1, e_2 \leftarrow \mathcal{M}$, $r_1, r_2 \leftarrow \mathcal{R}$, $r \leftarrow \mathcal{R}_1$, $s \leftarrow \mathcal{S}$
  $\mathcal{CP}_2$ computes and sends $a_1 = q_1^{e_1} \cdot \mathsf{Enc}_{pk}(e_2, r_1)$ to $\mathcal{CP}_1$
  $\mathcal{CP}_2$ computes and sends $a_2 = q_3^{e_1} \cdot q_2^{e2} \cdot \mathsf{Enc}_{pk}(\mathsf{Encode}(s, r), r_2)$ to $\mathcal{CP}_1$
Round: 3
  $\mathcal{CP}_1$ computes $s' = \mathsf{Decode}(\mathsf{Dec}_{sk}(a_2) - \mathsf{Dec}_{sk}(a_1) \cdot k)$
  $\mathcal{CP}_1$ computes $(c, d) = \mathsf{Com}_{ck}(s')$ and sends c to $\mathcal{CP}_2$
Round: 4
  $\mathcal{CP}_2$ sends $(s, e_1, e_2, r_1, r_2, r, q_3)$ to $\mathcal{CP}_1$
Round: 5
  $\mathcal{CP}_1$ verifies that $\mathsf{Dec}_{sk}(q_3) = k \cdot w_1$
  $\mathcal{CP}_1$: **if** $a_1 \neq q_1^{e_1} \cdot \mathsf{Enc}_{pk}(e_2, r_1)$ **return** $False$
  $\mathcal{CP}_1$: **if** $a_2 \neq q_3^{e_1} \cdot q_2^{e2} \cdot \mathsf{Enc}_{pk}(\mathsf{Encode}(s, r), r_2)$ **return** $False$
  $\mathcal{CP}_1$ sends d to $\mathcal{CP}_2$
Round: 6
  $\mathcal{CP}_2$: **if** $\mathsf{Open}_{ck}(c, d) \neq s$ **return** $False$
return $True$

would generate the value x and give the share representation back to the computing parties. However, it is straightforward to see that we can not achieve this, as in our protocol both parties can choose their own x_i. We would like to consider a slightly different case where the TTP takes x_i as inputs from $\mathcal{CP}_i$. However, in this case we can also only fully simulate the case of corrupted $\mathcal{CP}_1$ that sends $\mathsf{Enc}_{pk}(x_1)$ in the real protocol where the simulator could learn x_1. However, for corrupted $\mathcal{CP}_2$ the only message c that it sends is independent of the input x_2 and, therefore, the corresponding simulator could not learn x_2. For now we show the simulatability of the communication assuming that x_i are private inputs of the protocol. For achieving fully simulatable protocol we should include the proof that $\mathcal{CP}_2$ knows x_2 and z_2 that it uses to compute the massage c.

Theorem 4.3.2. *The communication in algorithm* Singles *for generating random shares is computationally simulatable and the final shared value x is computationally uniformly distributed in $\mathbb{Z}_N$ given a computationally IND-CPA secure cryptosystem.*

Proof sketch. We show that there exists a non-rewinding simulator for the steps before the verification phase that manages to compute all the values needed for the verification. Cheating can be discovered during the verification, if the checks pass then the sharing is valid. We assume that x_i are actually the inputs of this protocol that the honest

party would choose uniformly.

If $\mathcal{CP}_1$ is corrupted, then the simulator has to simulate the reply c from $\mathcal{CP}_2$. By definition $c = \mathsf{Enc}_{pk}(k)^{x_2} \cdot \mathsf{Enc}_{pk}(-z_2)$ is independent of x_2, therefore the simulator can simulate this efficiently by picking random values x_2^* and z_2^* and computing c according to the definition. If the adversary $\mathcal{A}$ sends $(\![x_1]\!)_{pk}$ that is not a valid ciphertext, then the simulator aborts, otherwise it finished successfully. In the end, the simulation has all the values x_2^*, z_2^*, $\mathsf{Enc}_{pk}(x_1)$ and $\mathsf{Enc}_{pk}(k)$ that it needs to continue with the zero-knowledge proof as an honest verifier. In can do continue with the proof as it has all the values for the queries and because the proof does not leak information about x_2^* and z_2^*, meaning that the proof with these value in indistinguishable from the one with real x_2 and z_2 for corrupted $\mathcal{CP}_1$.

If $\mathcal{CP}_2$ is corrupted, then the simulator must simulate the message $\mathsf{Enc}_{pk}(x_1)$. It can do this efficiently by publishing an encryption of a random value x_1^* because by the IND-CPA security of the cryptosystem, this in computationally indistinguishable from the encryption of the real input. The simulator sends *Abort* to the ideal functionality, if the message c from the adversary is not a valid ciphertext. Finally, the simulator can compute $\mathsf{Enc}_{pk}(z_1) = \mathsf{Enc}_{pk}(k)^{x_1^*} \cdot c$ for the zero-knowledge proof as its initial input to CdsZKTags. It could continue as a simulator of the proof if the simulator is defined.

Clearly, the result x is uniformly random, if at least one of the computing parties is honest. An honest party chooses its share uniformly as $x_i \leftarrow \mathbb{Z}_N$ and we know that in a ring the sum $x + r$ is uniformly distributed if x is uniform, independently of the distribution of r. However, we only achieve computationally uniform because $\mathcal{CP}_2$ also knows $(\![x_1]\!)_{pk}$ and might choose x_2 based on that. However, for a computationally IND-CPA secure cryptosystem, the probability of x_2 depending on x_1 is bounded by the computational indistinguishability. □

4.4 Beaver triples generation

Beaver triples [4] are multiplicative triples, so we need $[\![w]\!]_N = [\![xy]\!]_N$ from $[\![x]\!]_N$ and $[\![y]\!]_N$, where x and y are random values. We can easily find random shares $[\![x]\!]_N$ and $[\![y]\!]_N$ with Singles, so the main task of Triples protocol in Algorithm 11 is to correctly obtain $[\![w]\!]_N$. Random multiplicative triples are necessary to perform multiplication of the shares (Multiplication).

Verifying the correctness of $\mathcal{CP}_2$ requires using another unverified triple and thus, it can not be used after all runs of this protocol. However, in practice the protocols are commonly run on vectorised inputs and we could use half of the triples from generation to verify the other half.

Theorem 4.4.1. *Algorithm* Triples *for generating random triples is correct.*

Proof. The correctness of the commitment $(\![w_1]\!)_{pk}$ is trivial for an honest $\mathcal{CP}_1$. For correctness, we need to show that if both parties are following the protocol, then

Algorithm 11 Generating a random multiplicative triple (Triples)

Data: No inputs

Result: $[\![x]\!]_N$, $[\![y]\!]_N$, $[\![w]\!]_N$, where $w = x \cdot y$

1: $\mathcal{CP}_i$ generate $[\![x]\!]_N$ and $[\![y]\!]_N$ with Singles
2: Round: 1
3: $\mathcal{CP}_i$ sets $\Delta^{(w)} = 0$
4: $\mathcal{CP}_2$ generates r, $z_2^{(w)} \leftarrow \mathbb{Z}_N$
5: $\mathcal{CP}_2$ computes $w_2 = x_2 \cdot y_2 - r$
6: $\mathcal{CP}_2$ sends $v = (\![x_1]\!)_{pk}^{y_2} \cdot (\![y_1]\!)_{pk}^{x_2} \cdot \mathsf{Enc}_{pk}(r)$ to $\mathcal{CP}_1$
7: $\mathcal{CP}_2$ sends $t = (\![k]\!)_{pk}^{w_2} \cdot \mathsf{Enc}_{pk}(-z_2^{(w)})$ to $\mathcal{CP}_1$
8: Round: 2
9: $\mathcal{CP}_1$ computes $w_1 = x_1 \cdot y_1 + \mathsf{Dec}_{sk}(v)$
10: $\mathcal{CP}_1$ computes $z_1^{(w)} = k \cdot w_1 + \mathsf{Dec}_{sk}(t)$
11: $\mathcal{CP}_1$ generates $r^{(w)} \leftarrow \mathbb{Z}_N^*$
12: $\mathcal{CP}_1$ sends $(\![w_1]\!)_{pk} = \mathsf{Enc}_{pk}(w_1, r^{(w)})$ to $\mathcal{CP}_2$
13: Verification: Verify that $\mathcal{CP}_1$ is correct
14: $\mathcal{CP}_2$ verifies that $(\![w_1]\!)_{pk}$ is a valid ciphertext
15: $\mathcal{CP}_1$ proves $w = x \cdot y$ to $\mathcal{CP}_2$ using CdsZKMult
16: $\mathcal{CP}_1$ proves $z_1^{(w)} + z_2^{(w)} = k \cdot (w_1 + w_2)$ to $\mathcal{CP}_2$ using CdsZKTags
17: Verification: Verify that $\mathcal{CP}_2$ is correct
18: $\mathcal{CP}$ collaboratively run Triple Verification to learn h
19: $\mathcal{CP}_1$: if $h \neq 0$ **return** $\perp$
20: **return** $[\![x]\!]_N, [\![y]\!]_N, [\![w]\!]_N$

$w = x \cdot y$ and $z^{(w)} = k \cdot (w_1 + w_2)$. Is is straightforward, as

$$\begin{aligned} w &= w_1 + w_2 + \Delta^{(w)} = x_1 \cdot y_1 + \mathsf{Dec}_{sk}(v) + x_2 \cdot y_2 - r + 0 \\ &= x_1 \cdot y_1 + x_1 \cdot y_2 + y_1 \cdot x_2 + r + x_2 \cdot y_2 - r \\ &= x_1 \cdot (y_1 + y_2) + x_2 \cdot (y_1 + y_2) = (x_1 + x_2 + 0) \cdot (y_1 + y_2 + 0) = x \cdot y \\ z^{(w)} &= z_1^{(w)} + z_2^{(w)} = k \cdot w_1 + \mathsf{Dec}_{sk}(t) + z_2^{(w)} = k \cdot w_1 + k \cdot w_2 - z_2^{(w)} + z_2^{(w)} \\ &= k \cdot (w_1 + w_2) \ . \end{aligned}$$

□

Analogously to the Singles protocol, the basic ideal functionality should be such that the parties notify TTP that they want a set of triples and the TTP then gives them the share. However, in the Singles protocol we are actually more likely to achieve the case where parties can pick their own inputs x_i and the TTP gives the other elements in the share representation. In the triples protocol the party $\mathcal{CP}_2$ can also actually pick w_2 for itself, therefore we also consider this as an input to the triple generation in the ideal world. Hence, the ideal functionality of the triple generation protocol is such that the TTP receives x_1, x_2, y_1, y_2, w_2 and computes w_1 such that the multiplicative relation holds, as well as fixes the tags and commitments. However, for now we can only show the simulatability of the communication of this protocol.

Theorem 4.4.2. *The communication of the* Triples *protocol for generating random multiplicative triples is computationally simulatable.*

Proof sketch. We show that the communication to either side is simulatable. We except the verification as it is straightforward to see that Triple Verification is simulatable and we have specially addressed the problems with CdsZKTags and CdsZKMult. However, we know that if the verification succeeds then the triple has multiplicative relation and that the shares of the triple elements are correctly formed. We assume that the shares of the singles and w_2 are the inputs to this protocol.

The simulator can use the simulation for protocol Singles from Theorem 4.3.2 for generating random x and y. In case these protocols should abort, the simulator also aborts. Otherwise, it continues simulation.

The simulation for a malicious $\mathcal{CP}_1$ behaves exactly as an honest $\mathcal{CP}_2$ would, except that it has to pick w_2^* at random. The simulator can simulate v and t efficiently by picking a random w_2^* and using the values x_2^*, y_2^* that it picked for the simulation of the Singles. It succeeds unless $(\![w_1]\!)_{pk}$ is an invalid ciphertext. This simulation is perfect as the sent messages are independent of the input. By definition it has honestly computed the values needed in the zero-knowledge proofs and can behave as an honest verifier in the proof, because the proof does not leak its private input w_2^*.

We actually assume that the simulator for corrupted $\mathcal{CP}_2$ behaves slightly differently from the previous simulators. Namely, it also modifies the trusted setup, by defining a simulator of the setup, that picks $k_{\mathcal{S}}$ as a MAC key of the simulator and gives $(\![k_{\mathcal{S}}]\!)_{pk}$ instead of $(\![k]\!)_{pk}$ to $\mathcal{CP}_2$. Due to the IND-CPA security, this is computationally indistinguishable from the real setup. However, the limitation is that this simulated setup has to occur before any protocol runs, because the setup is shared between protocols. Therefore, all simulated runs for a corrupted $\mathcal{CP}_2$ must use the same $k_{\mathcal{S}}$ if they also contain the Triples protocol.

The simulator for a malicious $\mathcal{CP}_2$ can compute the only message $(\![w_1]\!)_{pk}$ that it has to simulate as $c = \mathsf{Enc}_{pk}(x_1^* \cdot y_1^*) \cdot v$. It aborts, if v or t are invalid ciphertexts. This simulation is computationally indistinguishable from the real protocol run, given a computationally IND-CPA secure cryptosystem. This holds because the maximal advantage that adversary $\mathcal{A}$ might have for distinguishing $\mathcal{S}$ from honest $\mathcal{CP}_1$ occurs if it actually knows w_1 and can distinguish c from $(\![w_1]\!)_{pk}$. Finally, the simulator has $(\![w_2]\!)_{pk}$ and $(\![k_{\mathcal{S}}]\!)_{pk}$ as the queries to the zero-knowledge proof. It can define also the message $(\![z_1]\!)_{pk} = (\![w_1]\!)_{pk}^{k_{\mathcal{S}}} \cdot t$. Therefore it has all the correct values for inputs of CdsZKTags. In addition, the simulator has all the values $(\![x_1^*]\!)_{pk}$, $(\![y_1^*]\!)_{pk}$ and $(\![w_1]\!)_{pk} \cdot v^{-1}$ that it needs as queries in CdsZKMult. Hence, it can run as a simulator for the proofs if the simulator is defined. □

The verification in Triples proves the correctness of $\mathcal{CP}_2$ to $\mathcal{CP}_1$ and vice versa. In general, we need that if $\mathcal{CP}_i$ has behaved correctly in the triple generation protocol, then this verification convinces $\mathcal{CP}_i$ that the other party $\mathcal{CP}_j$ also behaved correctly.

Our initialisation of the correctness proof of $\mathcal{CP}_1$ uses the special cases of CdsZkMul (Algorithm 1) and fails, if the multiplicative relation does not hold (CdsZKMult) or the MAC tag is not correctly formed (CdsZKTags). Their main difference from CdsZkMul is that the *prover* $\mathcal{CP}_1$ does not send the full initial query, but the query messages are fixed by the *verifier* $\mathcal{CP}_2$. Previously, we stressed that CdsZKTags can not be simulated because we need a zero-knowledge proof for the correctness of q_3 for that. The same holds for CdsZKMult, because, also in there, $\mathcal{CP}_1$ has to be convinced that q_3 is computed correctly. However, for now we do not specify this proof and it is up to the follow-up work to define a simulatable initialisation for CdsZKMult and CdsZKTags. As for the CdsZKTags, we expect the simulatable version of CdsZKMult to

also have the same input queries as the current version and use this assumption as a basis for our security proofs.

In addition, the verification step should ensure to $\mathcal{CP}_2$ that at this stage the sharing $[\![w]\!]$ is correct and $\mathcal{CP}_1$ can not frame it later. Analogously to the Singles protocol, anti-framing property holds for $\mathcal{CP}_1$ because of the commitment it made to w_1.

Proving the correctness of $\mathcal{CP}_2$ uses Triple Verification and needs to pick another unverified triple $[\![a]\!]_N$, $[\![b]\!]_N$, $[\![c]\!]_N$. If any of the two triples in the protocol are faulty then the correctness proof of $\mathcal{CP}_2$ fails, because $h \neq 0$. Actually, by the original definition, Triple Verification is used to prove the multiplicative relation to both $\mathcal{CP}_i$. However, here we only use this to prove the correctness of $\mathcal{CP}_2$ and similar proof about $\mathcal{CP}_1$ is done using CdsZKMult. This is due to the fact that using Triple Verification to also prove the multiplicative relation to $\mathcal{CP}_2$ could avoid CdsZKMult, but would introduce additional CdsZKTags. Currently $\mathcal{CP}_2$ does not have any knowledge about the correctness of the verification triple $[\![a]\!]_N, [\![b]\!]_N, [\![c]\!]_N$ and, therefore, we should use CdsZKTags to also prove that the tag of the third element of the verification triple is correct, meaning $z_1^{(c)} + z_2^{(c)} = k \cdot (c_1 + c_2)$. The latter is needed to prove to $\mathcal{CP}_2$ that the commitments used to open h in Triple Verification were computed correctly. However, CdsZKMult can be implemented slightly more efficiently than CdsZKTags and, therefore, our current specification is reasonable if using these proofs. A different approach should be considered if a more efficient analogue of CdsZKTags is used.

Algorithm 12 CdsZkMul for proving the multiplicative relation of the shares (CdsZKMult)

Setup: Commitment parameters ck,
Paillier keypair (pk, sk) from the protection domain setup
Data: Shares $[\![x]\!]_N, [\![y]\!]_N, [\![w]\!]_N$
Result: $True$ for successful proof of $x \cdot y = w$, $False$ in case of any failure

Round: 1
  $\mathcal{CP}_i$ sets $q_1 = (\![x_1]\!)_{pk}$, $q_2 = (\![y_1]\!)_{pk}$
  $\mathcal{CP}_2$ sets $q_3 = (\![w_1]\!)_{pk} \cdot \mathsf{Enc}_{pk}(w_2) \cdot ((\![x_1]\!)_{pk}^{y_2} \cdot (\![y_1]\!)_{pk}^{x_2} \cdot \mathsf{Enc}_{pk}(x_2 \cdot y_2))^{-1}$
  $\mathcal{CP}_2$ generates $e_1, e_2 \leftarrow \mathcal{M}$, $r_1, r_2 \leftarrow \mathcal{R}$, $r \leftarrow \mathcal{R}_1$, $s \leftarrow \mathcal{S}$
  $\mathcal{CP}_2$ computes and sends $a_1 = q_1^{e_1} \cdot \mathsf{Enc}_{pk}(e_2, r_1)$ to $\mathcal{CP}_1$
  $\mathcal{CP}_2$ computes and sends $a_2 = q_3^{e_1} \cdot q_2^{e2} \cdot \mathsf{Enc}_{pk}(\mathsf{Encode}(s, r), r_2)$ to $\mathcal{CP}_1$
Round: 2
  $\mathcal{CP}_1$ computes $s' = \mathsf{Decode}(\mathsf{Dec}_{sk}(a_2) - \mathsf{Dec}_{sk}(a_1) \cdot y_1)$
  $\mathcal{CP}_1$ computes $(c, d) = \mathsf{Com}_{ck}(s')$ and sends c to $\mathcal{CP}_2$
Round: 3
  $\mathcal{CP}_2$ sends $(s, e_1, e_2, r_1, r_2, r, q_3)$ to $\mathcal{CP}_1$
Round: 4
  $\mathcal{CP}_1$ verifies that $\mathsf{Dec}_{sk}(q_3) = x_1 \cdot y_1$
  $\mathcal{CP}_1$: **if** $a_1 \neq q_1^{e_1} \cdot \mathsf{Enc}_{pk}(e_2, r_1)$ **return** $False$
  $\mathcal{CP}_1$: **if** $a_2 \neq q_3^{e_1} \cdot q_2^{e2} \cdot \mathsf{Enc}_{pk}(\mathsf{Encode}(s, r), r_2)$ **return** $False$
  $\mathcal{CP}_1$ sends d to $\mathcal{CP}_2$
Round: 5
  $\mathcal{CP}_2$: **if** $\mathsf{Open}_{ck}(c, d) \neq s$ **return** $False$
return $True$

Theorem 4.4.3. *The verification steps in triple generation algorithm* Triples *are correct.*

Proof. The verification is correct, if it accepts correctly formed triples. We only need to show the correctness of the computations by $\mathcal{CP}_2$ as the correctness of $\mathcal{CP}_1$ is verified by two versions of the correct and universally composable CdsZkMul and the validity of the ciphertext.

Hence, we need to show that for a correct triple $[\![x]\!]_N$, $[\![y]\!]_N$, $[\![z]\!]_N$ and a verification triple $[\![a]\!]_N$, $[\![b]\!]_N$, $[\![c]\!]_N$ we get $h = 0$ from Triple Verification (Algorithm 5). We assume the correctness of necessary subprotocols of Triple Verification: Constant Multiplication, Subtraction, Constant Addition and Publish. By definition, we have

$$\begin{aligned} h &= s \cdot [\![w]\!]_N - [\![c]\!]_N - d \cdot [\![a]\!]_N - g \cdot [\![b]\!]_N - gd \\ &= s \cdot x \cdot y - a \cdot b - (y - b) \cdot a - (sx - a) \cdot b - (sx - a) \cdot (y - b) \\ &= sxy - ab - ya + ab - sxb + ab - sxy + sxb + ay - ab = 0 \ . \end{aligned}$$

Therefore, in the case of a correctly formed triple the verification succeeds. □

Theorem 4.4.4. *The verification steps in triple generation algorithm* Triples *are statistically* $\frac{1}{p}$*-secure against a cheating* $\mathcal{CP}_2$ *and as secure against a cheating* $\mathcal{CP}_1$ *as the proofs* CdsZKTags *and* CdsZKMult*, where* $N = pq$*,* p, q *are primes and* $p < q$*.*

Proof sketch. The security of $\mathcal{CP}_2$ depends on the security of CdsZKTags and CdsZKMult, that we should improve in the future to make them simulatable. However, the soundness property from CdsZkMul still holds, therefore, a successful proof indicates that $\mathcal{CP}_1$ has computed correctly.

Hence, we need to show the security of verification of the correctness of $\mathcal{CP}_2$. The verification algorithm for multiplicative relation is information-theoretically secure for finite fields [25]. According to CRT, breaking the security of the verification in case of modulus $N = pq$ also means breaking it separately modulo primes p and q, therefore the verification phase is $\frac{1}{p}$ secure for $\mathcal{CP}_1$ where $p < q$. □

4.5 Receiving inputs from the input party

The previously described Publish-$\mathcal{RP}_i$ (Algorithm 8) can be combined with the SPDZ classification protocol Classify-$\mathcal{IP}_i$ (Algorithm 4) in a straightforward manner to obtain a protocol for sharing the input of any third party. However, this version of the algorithm requires heavy computation and communication from the input party, who has to take part in a zero-knowledge proof. This is not efficient in many practical settings where we could have a variety of input devices, for example, smartphones and tablets.

There is a different protocol Classify-$\mathcal{IP}_i^\star$ in Algoritm 13 that uses ideas from the Singles protocol. In a way, the input party runs the single generation by itself and sends the corresponding share parts to the computing parties. Of course, instead of making a random share, it creates a share for its secret. The only addition is that the computing parties have to notify the input party if they accept this share. This means that the input party should wait and check that its input was accepted before it can know that it has correctly inserted the data.

Algorithm 13 Receiving an input from (non-computing) $\mathcal{IP}_i$ (Classify-$\mathcal{IP}_i^\star$)

Data: Input party $\mathcal{IP}_i$ has a secret x
Result: Computing parties $\mathcal{CP}_i$ have a valid share representation $[\![x]\!]_N$

Round: 1
  $\mathcal{CP}_i$ fixes $\Delta = 0$
  $\mathcal{IP}_i$ generates $x_1, z_2 \leftarrow \mathbb{Z}_N$, $r, t \leftarrow \mathbb{Z}_N^*$
  $\mathcal{IP}_i$ computes $x_2 = x - x_1$
  $\mathcal{IP}_i$ computes $c = (\![k]\!)_{pk}^{x_2} \cdot \mathsf{Enc}_{pk}(-z_2, t)$
  $\mathcal{IP}_i$ sends x_1, c, r to $\mathcal{CP}_1$
  $\mathcal{IP}_i$ sends x_2, z_2, $(\![x_1]\!)_{pk} = \mathsf{Enc}_{pk}(x_1, r)$, t to $\mathcal{CP}_2$
Round: 2
  $\mathcal{CP}_1$ computes $z_1 = k \cdot x_1 + \mathsf{Dec}_{sk}(c)$
  $\mathcal{CP}_2$ computes $c^* = (\![k]\!)_{pk}^{x_2} \cdot \mathsf{Enc}_{pk}(-z_2, t)$
  $\mathcal{CP}_2$ sends $(\![x_1]\!)_{pk}, c^*$ to $\mathcal{CP}_1$
Verification:
  $\mathcal{CP}_i$ verifies that $(\![x_1]\!)_{pk}$ is a valid ciphertext
  $\mathcal{CP}_1$ verifies that $c^* = c$
  $\mathcal{CP}_1$ verifies that it received $(\![x_1]\!)_{pk} = \mathsf{Enc}_{pk}(x_1, r)$
  $\mathcal{CP}_1$ proves $z_1 + z_2 = k \cdot (x_1 + x_2)$ to $\mathcal{CP}_2$ using CdsZKTags
  $\mathcal{CP}_1$ and $\mathcal{CP}_2$ notify $\mathcal{IP}_i$ about the verification outcome

Theorem 4.5.1. *Protocol* Classify-$\mathcal{IP}_i^\star$ *in Algorithm 13 for collecting inputs from input parties is correct.*

Proof sketch. The correctness of additive shares is clear by the definition as $x = x_1 + x_2 + \Delta = x_1 + x - x_1 + 0 = x$. If $\mathcal{CP}_1$ accepts the commitment then it is valid. Finally, we need that the MAC tag is correct:

$$z = z_1 + z_2 = k \cdot x_1 + k \cdot x_2 - z_2 + z_2 = k \cdot (x_1 + x_2) \ ,$$

thus, in case of honest participants the zero-knowledge proof succeeds and the share is correctly formed. □

There are three potential security risks in this protocol: (1) $\mathcal{CP}_i$ might modify the share, (2) $\mathcal{CP}_2$ might modify the share, and (3) $\mathcal{IP}_i$ might try to give inconsistent share representation. However, $\mathcal{IP}_i$ can always affect the outcome by inputting a maliciously chosen value x instead of the valid input value. Still, by security definition regardless of the choice of x it can only input a valid representation $[\![x]\!]$. This version of the protocol is only usable if $\mathcal{IP}_i$ is not one of the computing parties and does not collude with them. The former is not a restriction as we have a simpler protocol for $\mathcal{CP}_i$ to classify inputs, but the latter may be too restrictive for practical applications.

Theorem 4.5.2. *Protocol* Classify-$\mathcal{IP}_i^\star$ *for collecting inputs from input parties is perfectly secure against corrupted* $\mathcal{CP}_1$ *and* $\mathcal{IP}_i$ *and computationally secure against corrupted* $\mathcal{CP}_2$ *with additional* $\frac{1}{p}$ *error probability, assuming a computationally IND-CPA secure cryptosystem and modulus* $N = pq$*, where* p *is the smaller of its prime factors.*

Proof sketch. The principal ideal functionality of this protocol would be such that the $\mathcal{IP}_i$ gives x to the TTP and TTP gives shares of x to the computing parties. However, we can not achieve this as the distribution of the shares of x is controlled

by $\mathcal{IP}_i$. Hence, we define an ideal model where $\mathcal{IP}_i$ gives x, x_1, z_2 and r to TTP, who creates the remaining z_1 and forwards these to the computing parties. After that, the computing parties notify the TTP about accepting or rejecting these shares and the TTP forwards the outcome as *Success* or *Failure* to all parties.

Corrupted $\mathcal{CP}_1$. The simulator at first receives x_1, z_1, r from the TTP. It then computes $c = \mathsf{Enc}_{pk}(z_1) \cdot (\![k]\!)_{pk}^{-x_1} = \mathsf{Enc}_{pk}(z_1 - k \cdot x_1)$ and sends x_1, c and r to the corrupted $\mathcal{CP}_1$. By definition $\mathcal{CP}_1$ gets the output z_1 as the one given by the TTP. In the following, the simulator simulates the messages from $\mathcal{CP}_2$ as $\mathsf{Enc}_{pk}(x_1, r)$ and $c^* = c$. For the zero-knowledge proof, the simulator can define $x_2^* = -x_1$ and $z_2^* = -z_1$ to behave as an honest verifier for the case $0 = k \cdot 0$. This can be done as the proof does not leak x_2^* and z_2^* and, therefore, the corrupted $\mathcal{CP}_1$ sees the same view that it would in the real world. Finally, the simulator receives *Continue* or *Abort* from the corrupted $\mathcal{CP}_1$ and forwards this to the TTP. The outputs of the real and simulated ideal world coincide as in case the sharing succeeds both computing parties have the same output shares in both worlds and in case the sharing does not succeed they have both seen the same shares in these two versions of the protocol.

Corrupted $\mathcal{CP}_2$. In case of the simulation for corrupted $\mathcal{CP}_2$, we assume that the shared setup step was also simulated, so that the simulator has $k_{\mathcal{S}}$, whereas the $\mathcal{CP}_2$ has $(\![k_{\mathcal{S}}]\!)_{pk}$. The simulator receives x_2, z_2 and $(\![x_1]\!)_{pk}$ from the TTP. It has to specify the value t, that it can do by generating it as honest $\mathcal{IP}_i$ would. It then forwards x_2, z_2, $(\![x_1]\!)_{pk}$ and t to $\mathcal{CP}_2$. On messages c^* and $(\![x_1]\!)_{pk}$ from the corrupted $\mathcal{CP}_2$, the simulator verifies that the $(\![x_1]\!)_{pk}$ is the same as sent to $\mathcal{CP}_2$ and that $c^* = (\![k_{\mathcal{S}}]\!)_{pk}^{x_2} \cdot \mathsf{Enc}_{pk} - z_2, t$. If these are incorrect, then the simulator outputs *Failure*, otherwise it continues the simulation. For the zero-knowledge proof, the simulator has $(\![x_1]\!)_{pk}$ and $(\![k_{\mathcal{S}}]\!)_{pk}$ meaning that it can also compute $(\![z_1]\!)_{pk} = ((\![x_1]\!)_{pk} \cdot \mathsf{Enc}_{pk}(x_1))^{k_{\mathcal{S}}} \cdot \mathsf{Enc}_{pk}(-z_2) = \mathsf{Enc}_{pk}(x \cdot k_{\mathcal{S}} - z_2)$. Therefore the simulator has all the correct inputs for CdsZKTags and it could behave as a simulator for the proof. The outputs coincide as for a corrupted $\mathcal{CP}_1$.

Corrupted $\mathcal{IP}_i$. The simulator for a corrupted $\mathcal{IP}_i$ at first receives x_1, c, r, x_2, z_2, $(\![x_1]\!)_{pk}$ and t from the corrupted $\mathcal{IP}_i$. In then verifies that $(\![x_1]\!)_{pk} = \mathsf{Enc}_{pk}(x_1, r)$ and $c = (\![k]\!)_{pk}^{x_2} \cdot \mathsf{Enc}_{pk}(-z_2, t)$. If these hold, then it sends x, x_1, z_2 and r to the TTP and forwards the *Success* or *Failure* to the corrupted $\mathcal{IP}_i$ as messages from $\mathcal{CP}_i$. If the initial check does not verify, then it sends *Failure* to the $\mathcal{IP}_i$ and notifies the TTP that it does not participate in the protocol, which means that the protocol is a failure for all parties. The output distributions coincide because the simulator performs the same checks as the real functionality would to ensure the correctness of the $\mathcal{IP}_i$. □

Actually, Classify-$\mathcal{IP}_i^\star$ is also secure if $\mathcal{CP}_1$ and $\mathcal{IP}_i$ are corrupted by the same adversary, but insecure if $\mathcal{CP}_2$ and $\mathcal{IP}_i$ are corrupted together. In the latter case, the adversary could use one run of the protocol to check if some ciphertext that it sends as c is an encryption of some fixed message m. In practice, this protocol can be fairly securely used if $\mathcal{CP}_1$ only publishes encryptions of uniformly distributed elements in $\mathbb{Z}_N$ and the number of expected inputs is small, or if the input party is authenticated and one party should input a limited number of elements. In this case, the probability of an input party guessing the correct m within the expected number of input attempts can be made arbitrarily small. For full security, we could define a zero-knowledge proof where $\mathcal{IP}_i$ or $\mathcal{CP}_2$ proves to $\mathcal{CP}_1$ that c or c^* is computed correctly. It would be more reasonable to define this for c^* because we are more likely to have miners with bigger computing capability. Besides, we can always use Classify-$\mathcal{IP}_i$ with Publish-$\mathcal{RP}_i$, if we expect $\mathcal{IP}_i$ to have enough computational power to efficiently participate in an

analogous proof.

We have the anti-framing property, as after this protocol, all parties know that the share was correctly formed. Intuitively, as in the Singles protocol, framing $\mathcal{CP}_2$ is infeasible because of the zero-knowledge proof, which shows that in this step $\mathcal{CP}_1$ could correctly open this share. Analogously, framing $\mathcal{CP}_1$ is impossible because it can convince itself that the commitment $([\![x_1]\!])_{pk}$ is correct. However, if any of the checks fail during the protocol, then the computing parties can not easily verify whether the input party or the other computing party is acting maliciously.

4.6 Efficiency of the protocols

This section analyses the theoretical cost of the proposed protocols. We have two important criteria: (1) computational cost and (2) communication cost. These allow to compare these protocols as well as to estimate the cost of future protocols that are built from these existing blocks. As an overview, Figure 4.2 illustrates the current state of existing primitive protocols and protocols combined from them. The protocol Classify-$\mathcal{IP}_i$ has actually two versions, one that is derivated from publishing algorithm Publish-$\mathcal{RP}_i$ (which is equal to Publish $\mathcal{IP}_i$), and the second, that is a standalone protocol Classify-$\mathcal{IP}_i^\star$.

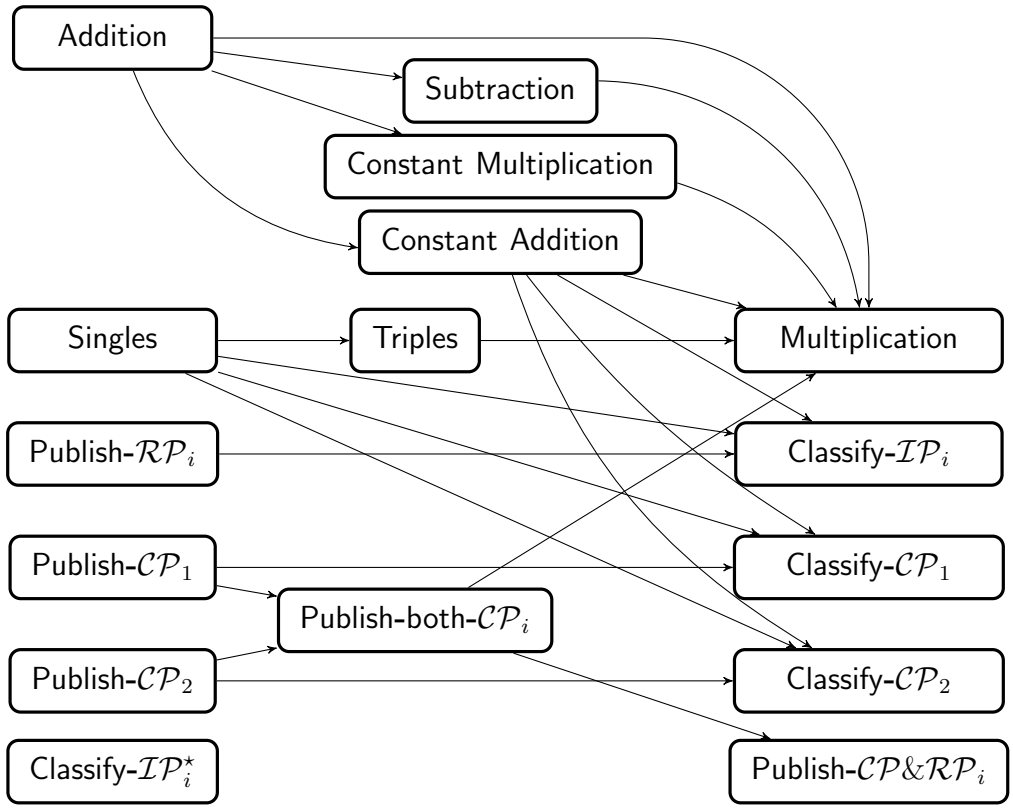


Figure 4.2: The hierarchy of protocols for the asymmetric setup

For a shorthand we define Add for Addition, Subtract for Subtraction, ConstMult for Constant Multiplication and Multiply for Multiplication.

4.6.1 Computational cost

This section analyses the computational requirements if the protocols are applied to one element, the extension to vectors in most cases just requires that amount of computation for each element. We focus on the multiplication and exponentiation operations as addition and subtraction are always considerably more efficient.

For a $|N|$-bit exponent we assume that we need to perform approximately $|N|$ multiplications, as using square-and-multiply it can definitely be done with $2|N|$. Furthermore, Paillier encryption and decryption have approximately the cost of one exponentiation, where exponent has length $|N|$, but multiplied elements have length $2 \cdot |N|$. Finally, modular inversion has the cost of a few multiplications, denoted by i in the following. For simplicity, we also assume that computing gcd for Paillier ciphertext validity check also has cost i as both of these can be done by Euclidean algorithm. Hence, we can estimate the computational complexity as the number of multiplication on either values of $\mathbb{Z}_N$ or Paillier ciphertexts in $\mathbb{Z}_{N^2}$ of length $2|N|$. Because of this we divide the analysis to two parts and give separate results results for both of these lengths. In total, the following should be taken as a rough estimate for comparing these protocols and the total cost is the sum of both length with more relative cost for length $2|N|$.

Party	Length	Publish	Add	Subtract	ConstMult	Multiply
$\mathcal{CP}_1$	$\|N\|$	1	1	$i+1$	$\|N\|+3$	$2\|N\|+2i+13$
	$2\|N\|$	0	0	0	0	0
$\mathcal{CP}_2$	$\|N\|$	0	0	0	3	7
	$2\|N\|$	$\|N\|$	1	$i+1$	$\|N\|$	$4\|N\|+2i+4$

Table 4.1: The computational cost of computation protocols as a number of multiplications

Table 4.1 summarises the multiplicative cost of our basic computation protocols. The length denotes the bitlength of the multiplication operands and other fields stand for separate protocols. For Publish-$\mathcal{CP}_i$ protocol, we consider the work that either party $\mathcal{CP}_i$ has to do in order to verify the result if the value is opened to it. We omit Classify-$\mathcal{CP}_i$ as it has exactly the same multiplicative cost as Publish-$\mathcal{CP}_i$. The results are obtained by counting the corresponding operations in the protocols.

Party	Length	Singles	Triples	TripleVerif
$\mathcal{CP}_1$	$\|N\|$	1	2	$4\|N\|+5i+22$
	$2\|N\|$	$2\|N\|$	$3\|N\|$	0
$\mathcal{CP}_2$	$\|N\|$	0	1	13
	$2\|N\|$	$2\|N\|+i+1$	$5\|N\|+i+3$	$8\|N\|+5i+5$

Table 4.2: The computational cost of precomputation as a number of multiplications

The cost of precomputation can be seen from Table 4.2. The costs of zero-knowledge proofs have been omitted from the precomputation protocols and can be found in Table 4.3. We omit the cost of our commitment scheme from the analysis of the zero-knowledge protocols as we use a lot smaller field for elliptic curves than in our general computations. Likewise, the cost of Singles values has been omitted from the Triples protocol, which here only illustrates the cost of obtaining one unverified triple as given in Algorithm 11. Triple Verification is an exceptional protocol as its amortized cost per vector element is slightly less than given in the table because we can pick one random element to verify a set of triples.

Party	Length	CdsZKTags	CdsZKMult
Prover	$\|N\|$	2	2
	$2\|N\|$	$9\|N\|+3$	$8\|N\|+3$
Verifier	$\|N\|$	0	1
	$2\|N\|$	$7\|N\|+i+5$	$9\|N\|+i+7$

Table 4.3: The computational cost of zero-knowledge proofs as a number of multiplications

Table tbl:zk-comp-requirements illustrates the computational requirements of the zero-knowledge proofs. It can be seen that CdsZKMult is more efficient on the side of $\mathcal{CP}_1$ and actually it can easily be implemented more efficiently also for $\mathcal{CP}_2$. The main complexity on the side of $\mathcal{CP}_2$ results from the computation of the third query q_3, which actually does several computation already performed in Triples and we could reduce $9|N|+i+7$ to approximately $5|N|+i+4$. Therefore, in our context, CdsZKMult is more efficient than CdsZKTags when used to verify the triple generation procedure.

Table 4.4 summarises the cost of protocols used to communicate with input and result parties. Similarly to precomputation, the zero-knowledge proofs have been omitted from this analysis. It can be seen that with additional proofs, the workload of $\mathcal{RP}_i$ in Publish-$\mathcal{RP}_i$ is approximately the same as that of $\mathcal{CP}_1$ whereas $\mathcal{CP}_2$ does not need to do any computations. However, Classify-$\mathcal{IP}_i^\star$ adds the complexity of the proof to the computing parties. Thus, in total $\mathcal{IP}_i$ has lower workload than $\mathcal{RP}_i$.

Party	Length	Classify-$\mathcal{IP}_i^\star$	Publish-$\mathcal{RP}_i$
$\mathcal{CP}_1$	$\|N\|$	1	0
	$2\|N\|$	$2\|N\|+i$	0
$\mathcal{CP}_2$	$\|N\|$	0	0
	$2\|N\|$	$2\|N\|+i+1$	0
$\mathcal{IP}_i \setminus \mathcal{RP}_i$	$\|N\|$	0	0
	$2\|N\|$	$3\|N\|+1$	$\|N\|$

Table 4.4: The computational cost of protocols for communicating with a third party as a number of multiplications

In total, the precomputation and zero-knowledge proofs form the most expensive part of this protection domain. Therefore, they remain the most important point for further optimisations.

4.6.2 Communication cost

This section analyses the cost of communication per protocol output in terms of the number and length of the sent messages. As in the previous section, we have two clear classes of messages with different length: plain elements of $\mathbb{Z}_N$ and Paillier ciphertext. In addition, we now include commitments for zero-knowledge protocols. These commitments are pairs of elliptic curve elements and for a prime P each element can be encoded as $|P|+1$ bits. Decommitments consist of two values that are both at most $|P|$ bits. In addition, we can assume that the secret value in zero-knowledge proofs is at most length $|P|$, and the randomness of the encoding function is bounded by N.

All the local computation protocols Addition, Subtraction, Constant Addition and Constant Multiplication do not require any communication. In addition, protocols Triple Verification and Multiplication only use communication during Publish protocols.

Party	Length	CdsZKTags	CdsZKMult	Singles	Triples	Publish
$\mathcal{CP}_1$	$\|N\|$	0	0	0	0	2
	$2\|N\|$	1	0	1	1	0
	$\|P\|$	4	4	0	0	0
$\mathcal{CP}_2$	$\|N\|$	5	5	0	0	2
	$2\|N\|$	3	3	1	2	0
	$\|P\|$	1	1	0	0	0

Table 4.5: The communication cost of computation protocols as number of messages

Therefore, it is sufficient to only analyse the communication cost of the zero-knowledge protocols, precomputation and publishing.

Table 4.5 summarises the communication cost for each class of messages. As previously, the precomputation protocols only include the cost of their specific functions and not proofs or other precomputation protocols. The amount or messages for a player means how much messages of which length he has to send for each protocol. Publish protocols only have a communication cost for one of the participants and the table should be read so that $\mathcal{CP}_j$ has to send that many messages in Publish-$\mathcal{CP}_i$. It can be seen that the precomputations and Publish-$\mathcal{CP}_i$ are quite efficient compared to the zero-knowledge protocols.

Party	Length	Classify-$\mathcal{IP}_i^\star$	Publish-$\mathcal{RP}_i$
$\mathcal{CP}_1$	$\|N\|$	0	3
	$2\|N\|$	0	0
$\mathcal{CP}_2$	$\|N\|$	0	3
	$2\|N\|$	2	1
$\mathcal{IP}_i \setminus \mathcal{RP}_i$	$\|N\|$	5	0
	$2\|N\|$	2	0

Table 4.6: The communication cost of protocols for communicating with a third party as number of messages

Table 4.6 gives an analogous overview for protocols including $\mathcal{IP}_i$ or $\mathcal{RP}_i$. Separately, they also prove more communication efficient than zero knowledge proofs. Thus, also communication-wise, the zero-knowledge proofs are currently the main bottleneck of this protection domain.

Chapter 5

Protocols for Beaver triple generation

This section describes our efforts to efficiently generate Beaver triples for various moduli using the additively homomorphic Paillier cryptosystem. This section only considers the *semi-honest* security setting as the main goal of this section is to propose new ideas for precomputation and it is easier to reason about them in the passive model. Besides, it is reasonable as we often can do precomputation by firstly fixing unprotected shares, then protection mechanisms and finally, we can verify that the sharing is correct. More specifically, we only consider the case of semi-honest static adversary in our security proofs.

5.1 Setup for triple generation protocols

This chapter considers the case where we have additively shared secrets $[\![x]\!]_M$ and $[\![y]\!]_M$ for some modulus M and the goal is to obtain $[\![w]\!]_M$ where $w = x \cdot y \bmod N$. Hence, $[\![x]\!]_M = \langle x_1, x_2 \rangle$.

Algorithm 14 Multiplication of additively shared secrets based on the Paillier cryptosystem (Paillier Multiplication)

Setup: Paillier keypair with modulus N, where $\mathcal{CP}_1$ knows the secret key
Data: Shared secrets $[\![x]\!]_M,\ [\![y]\!]_M$
Result: Shared result $[\![w]\!]_N$, where $w = (x_1 + x_2) \cdot (y_1 + y_2) \bmod N$

1: $\mathcal{CP}_1$ encrypts and sends $\mathsf{Enc}_{pk}(x_1)$, $\mathsf{Enc}_{pk}(y_1)$ to $\mathcal{CP}_2$
2: $\mathcal{CP}_2$ generates $r \leftarrow \mathbb{Z}_N$
3: $\mathcal{CP}_2$ computes and sends $t = \mathsf{Enc}_{pk}(x_1 \cdot y_2 + y_1 \cdot x_2 + r)$ to $\mathcal{CP}_1$
4: $\mathcal{CP}_2$ computes $w_2 = x_2 \cdot y_2 - r \bmod N$
5: $\mathcal{CP}_1$ computes $w_1 = x_1 \cdot y_1 + \mathsf{Dec}_{sk}(t) \bmod N$

Protocol Paillier Multiplication in Algorithm 14 shows how to do simple share multiplication using the Paillier cryptosystem and additive secret sharing. This will be an important tool throughout this chapter. This protocol is actually all that is needed to generate triples $[\![x]\!]_N, [\![y]\!]_N, [\![w]\!]_N$ for a Paillier modulus N.

Theorem 5.1.1. *The* Paillier Multiplication *protocol for share multiplication using the Paillier cryptosystem is correct.*

Proof. It remains to show that indeed $w = x \cdot y \bmod N$, which can be seen trivially, as

$$\begin{aligned} w &= w_1 + w_2 = x_1 \cdot y_1 + \mathsf{Dec}_{sk}(t) + x_2 \cdot y_2 - r \\ &= x_1 \cdot y_1 + x_1 \cdot y_2 + x_2 \cdot y_1 + r + x_2 \cdot y_2 - r = (x_1 + x_2) \cdot (y_1 + y_2) = x \cdot y \ . \end{aligned}$$

□

Theorem 5.1.2. *The* Paillier Multiplication *protocol for share multiplication using the Paillier cryptosystem is computationally secure against corrupted $\mathcal{CP}_2$ and statistically secure against a corrupted $\mathcal{CP}_1$ in the passive model.*

Proof. The ideal functionality of this protocol is such that it receives x_1, y_1 from $\mathcal{CP}_1$ and x_2, y_2 from $\mathcal{CP}_2$. It gives back w_1 to $\mathcal{CP}_1$ and w_2 to $\mathcal{CP}_2$.

In semi-honest case the simulator knows the inputs x_i and y_i of the corrupted party and can easily forward them to the TTP. In both cases it gets w_i back from the TTP and must now make sure that the output of corrupted $\mathcal{CP}_i$ is finally w_i.

In case of a corrupted $\mathcal{CP}_1$, the output is fixed as $w_1 = x_1 \cdot y_1 + \mathsf{Dec}_{sk}(t)$ where the simulator must fix t. Knowing w_1, x_1 and y_1, the simulator can easily compute $c = w_1 - x_1 \cdot y_1$ and fix $t = \mathsf{Enc}_{pk}(c)$. In is straightforward to see that the corrupted $\mathcal{CP}_1$ will output w_1. The outputs of the real and simulated world coincide.

In case of a corrupted $\mathcal{CP}_2$, we know that the output $w_2 = x_2 \cdot y_2 - r$, depends on the randomness r chosen by the corrupted $\mathcal{CP}_2$. However, as it is semi-honest, we know that it chooses $r \leftarrow \mathbb{Z}_N$, therefore, r is affected by the initial randomness of $\mathcal{CP}_2$. Knowing the desired output w_2 and inputs x_2, y_2 the simulator can compute $r = x_2 y_2 - w_2$ and pick a suitable randomness to run $\mathcal{CP}_2$ with. Therefore, the simulated run in the ideal world and the corresponding run in the real world always have coinciding output distributions. □

As stated in the algorithm description, we can actually obtain something more general, namely $[\![w]\!]_N$ for $w = (x_1 + x_2) \cdot (y_1 + y_2) \bmod N$ from $[\![x]\!]_M$ and $[\![y]\!]_M$, which is actually closer to what we will use in the following. If we use a modulus M such that $M^2 < N$, then we actually have $x \cdot y < N$. Therefore taking the final $w \bmod M$ should give us a valid triple. Actually, it is slightly more difficult as we have

$$w_1 + w_2 \geq N \text{ or } w_1 + w_2 < N \ .$$

In the latter case, this conversion works by reducing both shares separately. However, in the former case a simple modulo reduction gives invalid results. This remaining issue is discussed in Section 5.3.

For now, we can assume that we can get correct results for at least half of the executions of Paillier Multiplication with modulus M. For example, if we always toss a fair coin after the generation and then either try to correct the error by computing $w = w_1 + w_2 - N \bmod M$ or do not try to correct it, then we get the right triple half of the times. A similar algorithm has been used for precomputations also by predecessors of SPDZ [7, 25], but with restrictions to the size of the randomness r to avoid the overflow.

We say that the yield of the protocol is the ratio of the length of the produced triple elements to the length of the Paillier modulus and denote it by γ_{len}. Analogously, by γ_{net} we denote the ratio of triple length to the cumulative length of exchanged messages on the network. Finally, by γ_{comp} we denote the ratio of outputs bits to multiplications

of ciphertexts that are elements of length $2|N|$. As in the asymmetric case, we assume that encryption and decryption require $|N|$ multiplications and that the ciphertexts have length $2|N|$. In general, we would like to maximise all these parameters for the best efficiency. For packing the best achievable bound is 1, for others there is no fixed limit. For now, we focus on maximising γ_{len} and give other for additional comparison in hope that they are easier to improve on using clever implementation tricks.

Clearly, for modulus N we have

$$\gamma_{len} = 1, \;\; \gamma_{net} = \frac{|N|}{3 \cdot 2|N|} = \frac{1}{6} \text{ and } \gamma_{comp} = \frac{|N|}{6|N| + 2} \approx \frac{1}{6} \;\; .$$

For an arbitrarily chosen modulus M, where $M^2 < N$, we achieve on average

$$\gamma_{len} = \frac{1}{2} \cdot \frac{|M|}{|N|} \leq \frac{1}{4}, \;\; \gamma_{net} = \frac{1}{2} \cdot \frac{|M|}{3 \cdot 2|N|} \leq \frac{1}{24}, \;\; \gamma_{comp} = \frac{1}{2} \cdot \frac{|M|}{4|N| + 2|M| + 2} \leq \frac{1}{20} \;\; ,$$

if we assume that we get half of the triples correctly.

5.2 Packing several shares into one generation

The introduced Paillier Multiplication was used for a general modulus M with a relation to the used Paillier modulus N, approximately $M^2 < N$, or for $M = N$. It could then be used for any such modulus M, but is clearly most efficient in terms of γ_{len} if the size of M is close to the bound $M^2 < N$. However, for practical sizes of N, this results in a very long $|M| \geq 1024$. This section explores how shorter types could be packed inside elements of $\mathbb{Z}_M$ so that we can most efficiently use the triple generation protocols and learn meaningful triples for shorter moduli.

The efficiency of packing is mainly shown by γ_{len}. The values for γ_{net} and γ_{comp} reflect more the communication and computation cost that we need to compute with given packing.

5.2.1 Packing as base-B numbers

Packing as B-ary numbers means that each element modulo $M < B$ represents a digit and we pack them as a numbers of base B. A three-digit base-B number could be written out as

$$x = B^2 \cdot x_3 + B \cdot x_2 + x_1 \;\; ,$$

where $x_i < B$ are digits. If we assume, that y is written out in a similar manner as

$$y = B^2 \cdot y_3 + B \cdot y_2 + y_1 \;\; ,$$

then the corresponding multiplication becomes

$$xy = B^4 x_3 y_3 + B^3(x_3 y_2 + x_2 y_3) + B^2(x_2 y_2 + x_1 y_3 + x_3 y_1) + B(x_2 y_1 + x_1 y_2) + x_1 y_1 \;\; .$$

This shows that we could get a triple x_1, y_1, $x_1 y_1$ assuming that $x_1 y_1$ does not overflow the B-ary digit. With restrictions to the initial x_i and y_i values we can ensure that this nor other combinations do not overflow and xy is a five-digit number. Thus, we could also get a triple x_3, y_3 and $x_3 y_3$.

Packing as straightforward B-ary numbers is, therefore, not very beneficial as we did not receive a triple x_2, y_2, x_2y_2. However, we could consider another example, with x as before, but y is modified, giving

$$\begin{aligned} x &= B^2 \cdot x_3 + B \cdot x_2 + x_1 \\ y &= B^6 y_3 + B^3 \cdot y_2 + y_1 \ . \end{aligned}$$

The resulting multiplication is

$$\begin{aligned} xy =& B^8 x_3 y_3 + B^7 x_2 y_3 + B^6 y_3 x_1 + B^5 x_3 y_2 + B^4 x_2 y_2 + B^3 x_1 y_2 + \\ &+ B^2 x_3 y_1 + B x_2 y_1 + x_1 y_1 \ . \end{aligned}$$

Here we can see that xy contains all the triples x_i, y_i and $x_i y_i$, but also some elements $x_i y_j,\ i \neq j$ that we do not need. Picking the powers in y as multiples of the number of digits in x always gives analogous results. More specifically, if both pack n elements, then the product would have n^2 digits where we are only interested in n of them in form $x_i y_i$. Thus, in this packing we only get the same number of triples as the square-root of the number digits in the base-B representation of xy.

What we need to achieve is actually that every result $x_i y_i$ is multiplied by a unique power of B. On the other hand, we do not care about the other $x_i y_j$, which could share the same powers of B between them. This observation allows us to always make small adjustments to special cases. For example, using

$$y = B^4 y_3 + B^2 y_2 + y_1$$

would also enable us to get all the $x_i y_i$ pairs with the gain of of two B-ary digits. The result xy is only a seven digit number as

$$\begin{aligned} xy =& B^6 x_3 y_3 + B^5 x_2 y_3 + B^4 (x_1 y_3 + x_3 y_2) + B^3 x_2 y_2 + B^2 (x_1 y_2 + x_3 y_1) + \\ &+ B x_2 y_1 + x_1 y_1 \ . \end{aligned}$$

However, we can not get rid of all the unnecessary products $x_i y_j$ in this packed multiplication result.

The problem with using this approach in a straightforward manner is that we have to assume that $y_i x_i < B$, which essentially means that the result xy contains integer form results of all triples as digits. We can not use this directly with Paillier Multiplication, as the randomisation there would ruin this structure. However, we can not define it without any randomisation either because otherwise seeing $y_i x_i$ and knowing x_i also leaks the secret y_i. A possible solution is that we actually redefine the randomising element as

$$r = \sum B^i r_i$$

and make sure that at least for all $r_i + x_i y_i$, there is no overflow from B from either side. On the downside, this is not completely secure because the values $r_i + x_i y_i$ will not be uniformly distributed and therefore may leak information about $x_i y_i$. Commonly, we would define a security parameter σ so that r is σ bits longer than $x_i y_i$ to hide it with probability $1 - 2^{-\sigma}$. The hiding properties of this randomisation are addressed by Theorem 5.2.2.

Therefore, in Paillier Multiplication with base-B numbers we achieve

$$\gamma_{len} = \frac{|M| \cdot m}{|N|} \leq \frac{1}{2}, \ \ \gamma_{net} = \frac{|M| \cdot m}{3 \cdot 2|N|} \leq \frac{1}{12}, \ \ \gamma_{comp} = \frac{|M| \cdot m}{4|N| + 2|M| + 2} \leq \frac{1}{10} \ ,$$

because we always get the correct outcome $w_1 + w_2 < N$. However, the bounds are only achievable in very insecure settings where we do not use the randomisers.

The main benefit of this approach is that there are no restrictions to the length of the packed type because we can always pick a suitable B. The following Algorithm 15 uses a version of this linear packing.

5.2.2 Triple generation with partial base-B packing

Protocol B-Triples in Algorithm 15 is the protocol by Thomas Schneider as used in [48]. In uses the ideas of B-ary packing, however, these are used slightly differently from the previous initialisation. One party sends x_i separately and the other responds with $\sum B^i x_i y_i + r_i$. This is not very communication-efficient, but avoids the occurrence of elements $x_i y_j$ in the packed response.

The main drawback of this protocol is that although we use randomness to blind the encrypted response, we actually have secret sharing over integers in the response v and it may leak some information about the shares. The main idea of this protocol is to reduce the network communication compared to the basic Paillier Multiplication. The gain comes from the fact that the responder does not need to send back a ciphertext for each of the triple, but packs elements into one ciphertext.

Let σ denote a statistical security parameter. The efficiency of this protocol depends on σ and the length $k = |M|$ of the initial single values. Variable ℓ stands for the length of the packed values. We have $\ell = 2k + 2 + \sigma$ and thus, it is possible to pack $\lfloor |N|/\ell \rfloor$ responses into one ciphertext. This length also defines the randomness that is used to hide the actual value of the inputs.

Algorithm 15 Generating $m < |N|/\ell$ triples with B-ary packing (B-Triples)

Setup: Security parameter σ, modulus length $k = |M|$, $\ell = 2k + 2 + \sigma$
Data: Arrays of shared secrets $[\![x_1]\!]_M, \ldots, [\![x_m]\!]_M$ and $[\![y_1]\!]_M, \ldots, [\![y_m]\!]_M$
Result: Array $[\![w_1]\!]_M, \ldots, [\![w_m]\!]_M$, where $[\![w_i]\!]_M = [\![x_i \cdot y_i]\!]_M$

1: $\mathcal{CP}_1$ sends $\mathsf{Enc}_{pk}(x_{1,1}), \ldots, \mathsf{Enc}_{pk}(x_{1,m})$ to $\mathcal{CP}_2$
2: $\mathcal{CP}_1$ sends $\mathsf{Enc}_{pk}(y_{1,1}), \ldots, \mathsf{Enc}_{pk}(y_{1,m})$ to $\mathcal{CP}_2$
3: $\mathcal{CP}_2$ fixes $r = 0$, $e = 0$, $\mathsf{Enc}_{pk}(e)$
4: **for** $i \in \{1, \ldots, m\}$ **do**
5: $\quad$ $\mathcal{CP}_2$ generates a random $r_i \leftarrow \{0,1\}^{2 \cdot k + 1 + \sigma}$
6: $\quad$ $\mathcal{CP}_2$ computes $\mathsf{Enc}_{pk}(t_i) = \mathsf{Enc}_{pk}(x_{1,i} \cdot y_{2,i} + y_{1,i} \cdot x_{2,i})$
7: $\quad$ $\mathcal{CP}_2$ computes $r = r \cdot 2^{\ell} + r_i$
8: $\quad$ $\mathcal{CP}_2$ computes $\mathsf{Enc}_{pk}(e) = \mathsf{Enc}_{pk}(e \cdot 2^{\ell} + t_i)$
9: $\quad$ $\mathcal{CP}_2$ computes $w_{2,i} = x_{2,i} \cdot y_{2,i} - r_i \bmod M$
10: **end for**
11: $\mathcal{CP}_2$ encrypts $\mathsf{Enc}_{pk}(r)$
12: $\mathcal{CP}_2$ sends $\mathsf{Enc}_{pk}(v) = \mathsf{Enc}_{pk}(e + r)$ to $\mathcal{CP}_1$
13: $\mathcal{CP}_1$ decrypts and unpacks single values $v_1 || v_2 || \ldots || v_m = \mathsf{Dec}_{sk}(\mathsf{Enc}_{pk}(v))$
14: **for** $i \in \{1, \ldots, m\}$ **do**
15: $\quad$ $\mathcal{CP}_1$ computes $w_{1,i} = x_{1,i} \cdot y_{1,i} + v_i \bmod M$
16: **end for**
17: **return** $[\![c]\!]$

For example, choosing $|N| = 2048$, $\sigma = 112$ and $k = 32$ as in [48] allows us to pack 11 elements of 32-bits to 2048-bit modulus. The main strength of this approach

is that there is no need for share conversion as the fixed length of elements also ensures that the response does not overflow N and the algorithm always yields correct triples modulo M.

Theorem 5.2.1. *Protocol* B-Triples *in Algorithm 15 for generating Beaver triples with partial B-ary packing is correct.*

Proof. We need to show that $w_i = x_i \cdot y_i$ for all $i \in 1, \ldots, m$. For this, it is crucial to analyse what happens in the packing. We define $B = 2^\ell$ and the cycle computes the response

$$\begin{aligned} e =& B^{m-1} \cdot (x_{1,1} \cdot y_{2,1} + x_{2,1} \cdot y_{1,1}) + \ldots + B \cdot (x_{1,m-1} \cdot y_{2,m-1} + x_{2,m-1} \cdot y_{1,m-1}) + \\ &+ (x_{1,m} \cdot y_{2,m} + x_{2,m} \cdot y_{1,m}) \end{aligned}$$

and a randomness

$$r = B^{m-1} \cdot r_1 + \ldots + B \cdot r_{m-1} + r_m \ .$$

Finally, the sum of these is computed and sent back to $\mathcal{CP}_1$ who can decrypt and disassemble it to blocks

$$v_i = x_{1,i} \cdot y_{2,i} + x_{2,i} \cdot y_{1,i} + r_i \ .$$

We assume that the elements x_i and y_i have a length of k bits, thus, each $x_{1,i} \cdot y_{2,i} + x_{2,i} \cdot y_{1,i}$ is at most $2k + 1$ bits long. The randomness r is defined as $2k + 1 + \sigma$ bits, which means that the value v_i is at most $2k + 2 + \sigma$ bits and if we define $\ell > 2k + 2 + \sigma$ then each of these values fits into an ℓ-bit slot. Thus, the packed value can always be restored correctly if this value is smaller than the encryption modulus, which is ensured as $m \cdot \ell < |N|$.

It remains to show that $w_i = x_i \cdot y_i$, which can be easily seen, as

$$\begin{aligned} w_i &= w_{1,i} + w_{2,i} = x_{1,i} \cdot y_{1,i} + v_i + x_{2,i} \cdot y_{2,i} - r_i \\ &= x_{1,i} \cdot y_{1,i} + x_{1,i} \cdot y_{2,i} + x_{2,i} \cdot y_{1,i} + r_i + x_{2,i} \cdot y_{2,i} - r_i \\ &= (x_{1,i} + x_{2,i}) \cdot (y_{1,i} + y_{2,i}) = x_i \cdot y_i \ . \end{aligned}$$

□

Theorem 5.2.2. *Protocol* B-Triples *in Algorithm 15 for generating Beaver triples with B-ary packing is statistically secure against a corrupted $\mathcal{CP}_1$ and computationally secure against a corrupted $\mathcal{CP}_2$, given a (t, ε)-IND-CPA secure cryptosystem and statistical security constant σ for packing.*

Proof sketch. The simulator can adjust the outputs of either party to correspond to the results from the TTP as in the Paillier Multiplication. In the following we consider, how the simulator can simulate the communication of this protocol.

Corrupted $\mathcal{CP}_1$. The simulator can replace each v_i with an encryption of a random element $r^* \leftarrow R$ where $R = \{t_{max}/2, \ldots, 2^{2k+\sigma+1} + t_{max}/2\}$ where $t_{max} = 2^{2k+1} - 2^{k+2} + 2 < 2^{2k+1}$ is the maximal value that t_i might have. It is easy to see that the minimal value that t_i might have is 0, therefore, $t_{max}/2$ is the median value of t_i. The advantage of the adversary in this case is bounded by the statistical distance of $v = t_i + r_i$ and

r^*. The value $v = t_i + r_i$ is uniformly distributed in $T = \{t_i, \ldots, t_i + 2^{2k+\sigma+1}\}$ where $0 \leq t \leq t_{max}$. Hence, the statistical distance is

$$\begin{aligned}
\mathsf{sd}(r^*, v) &= \frac{1}{2} \cdot \sum_{x \in R \cup T} \Big| \Pr[r^* = x] - \Pr[v = x] \Big| \\
&= \frac{1}{2} \cdot \Big(\sum_{x \in R \cap T} \Big| \Pr[r^* = x] - \Pr[v = x] \Big| + \sum_{x \in T \setminus R} \Big| \Pr[r^* = x] - \Pr[v = x] \Big| + \\
&+ \sum_{x \in R \setminus T} \Big| \Pr[r^* = x] - \Pr[v = x] \Big| \Big) \\
&= \frac{1}{2} \cdot \Big(\sum_{x \in T \cap R} \Big| \frac{1}{|R|} - \frac{1}{|T|} \Big| + \sum_{x \in T \setminus R} \Big| 0 - \frac{1}{|T|} \Big| + \sum_{x \in R \setminus T} \Big| \frac{1}{|R|} - 0 \Big| \Big) \ .
\end{aligned}$$

It is possible to continue this evaluation as we know that $|R| = |T| = 2^{2k+\sigma+1} + 1$ and that $0 \leq |R \setminus T| = |T \setminus R| \leq t_{max}/2$. Therefore we can give an upper bound to $\mathsf{sd}(r^*, v)$ as

$$\begin{aligned}
\mathsf{sd}(r^*, v) &= \frac{1}{2} \cdot \Big(\sum_{x \in T \cap R} 0 + \sum_{x \in T \setminus R} \frac{1}{2^{2k+\sigma+1} + 1} + \sum_{x \in R \setminus T} \frac{1}{2^{2k+\sigma+1} + 1} \Big) \\
&\leq \frac{1}{2} \cdot t_{max} \cdot \frac{1}{2^{2k+\sigma+1} + 1} = \frac{t_{max}}{2^{2k+\sigma+2} + 2} \leq \frac{2^{2k+1}}{2^{2k+\sigma+2} + 2} \leq 2^{-\sigma} \ .
\end{aligned}$$

Therefore, sending an encryption of a random element from R is statistically $2^{-\sigma}$ indistinguishable from a correctly computed reply. It follows that the simulator can pick r^* such that the output of the corrupted $\mathcal{CP}_1$ is as desired.

Corrupted $\mathcal{CP}_2$. The simulator should send the encryptions of $x_{1,i}$ and $y_{1,i}$ to $\mathcal{CP}_2$, which it can do efficiently by sending the encryptions of random values. Due to the IND-CPA security, the simulation is at distance $2m \cdot \varepsilon$ from the real protocol run for any t-time adversary. □

The efficiency of this protocol depends on the chosen parameters, but for now, we write it out in terms of $m < |N|/\ell$ and $|M|$, where $\ell = 2|M| + \sigma + 2$. In total, we learn $|M| \cdot m$ bits of valid triples which gives

$$\gamma_{len} = \frac{|M| \cdot m}{|N|} \leq \frac{1}{2} - \frac{\sigma \cdot m}{2|N|} - \frac{m}{|N|} < \frac{1}{2} \ ,$$

where we get the estimate as we know that $\ell \cdot m < |N|$ which gives

$$|M| \cdot m < \frac{|N|}{2} - \frac{\sigma \cdot m}{2} - m \ .$$

Communication-wise this protocol gives

$$\gamma_{net} = \frac{|M| \cdot m}{(2m+1) \cdot 2|N|} \leq \frac{1}{4 \cdot (2m+1)} \ .$$

Finally, in terms of computation this protocol is also quite expensive due to the separate encryption operations resulting in

$$\gamma_{comp} = \frac{|M| \cdot m}{(2m+2) \cdot |N| + 2m \cdot |M| + m \cdot \ell + 2m + 1} \leq \frac{1}{4m+4} \ .$$

5.2.3 Packing using the Chinese remainder theorem

We can also use the Chinese remainder theorem for packing several elements into one ciphertext. However, the used mechanism is quite different from the previously described B-ary packing and can only be used, if we are using elements with pairwise coprime moduli p_i. By definition, CRT can be used to combine all those single random values modulo p_i for a modulus

$$M = p_1 \cdot \ldots \cdot p_k$$

and execute the triple generation protocol to obtain the corresponding third triple elements modulo M.

For example, we start with values x_i and y_i and we interpret them as

$$\begin{cases} x = x_i \bmod p_i \\ y = y_i \bmod p_i \end{cases} .$$

Then, we combine them using CRT to learn $x \bmod M$ and $y \bmod M$ which are inputs to the triple generation protocol for learning $xy \bmod M$. We know that, by definition, we have

$$\Big\{ xy \quad = x_i y_i \bmod p_i \ ,$$

where we are interested in learning the shares for $x_i y_i$. The CRT allows us to reduce the final result respectively for all moduli p_i to learn the third triple element for all initial random value pairs. Therefore, we can learn $x_i y_i$ from $xy \bmod M$ as

$$x_i y_i = xy \bmod p_i \ .$$

Packing with CRT enables us to get exactly $|M|$-bit triples from one execution of the triple generation protocol. However, we could also use this with the idea to later convert all these shares of different moduli to one shared modulus as discussed in Section 5.3.2. This would result in approximately $\frac{|M|}{2}$-bit triple elements.

This packing can be trivially well used with the Paillier Multiplication protocol because all we need is to learn a valid triple modulo M to be able to get all separate triples modulo p_i. However, we have to ensure that, in this case, $M^2 < N$. Therefore, we can get approximately $\frac{|N|}{2}$-bit triples when using a modulus M. Using Paillier Multiplication with CRT packing gives exactly the same yields as it does for any arbitrarily chosen modulus M as given in Section 5.1. We need to do additional computations for packing and unpacking, but these do not significantly affect our computational cost as they are not operations on ciphertexts.

5.3 Share conversion

Share conversion is the process of transforming a shared value $[\![x]\!]_M$ for one fixed modulus M to a valid share $[\![v]\!]_{M^*}$ of the same secret value $x = v$ under a different modulus M^*. It will be necessary as we use the Paillier cryptosystem that has homomorphic properties modulo N for generating triples of a generic modulus M. This section focuses on transforming $[\![x]\!]_2$ to $[\![v]\!]_M$ and $[\![x]\!]_N$ to $[\![v]\!]_M$, where $N > M$ that are needed for triple generation. In addition, we stress additional restrictions that must be met in order to successfully convert the triple from Paillier Multiplication so that the multiplicative relation still holds after the conversion.

5.3.1 Converting binary shares to any modulus

A protocol for obtaining $[\![v]\!]_M$ from $[\![x]\!]_2$ is a simpler subcase of all conversion protocols as x in $[\![x]\!]_2$ has only two potential values. Thus, if $x = 0$ we should have $[\![v]\!]_M$ as $v_1 \leftarrow \mathbb{Z}_M$ and $v_2 = M - v_1$ or, correspondingly, $v_2 = M - v_1 + 1$ for $x = 1$.

		$\mathcal{CP}_1$ input	
		$x_1 = 0$	$x_1 = 1$
$\mathcal{CP}_2$ input	$x_2 = 0$	$v_2 = M - v_1$	$v_2 = M - v_1 + 1$
	$x_2 = 1$	$v_2 = M - v_1 + 1$	$v_2 = M - v_1$

Table 5.1: Oblivious transfer for share conversion $[\![x]\!]_2$ to $[\![v]\!]_M$

Such a replacement of the shares can be achieved using oblivious transfer (OT). The *1-out-of-2* OT is a communication protocol for transporting information so that the sender does not know which of its two inputs it forwarded to the receiver. In addition, the receiver is only able to learn one of the sender's inputs per protocol. The idea for share conversion is that the sender defines $v_1 \leftarrow \mathbb{Z}_M$ and sends $v_2 = M - v_1$ or $v_2 = M - v_1 + 1$ to the receiver based on the value of x. More precisely, we do not open the value x, but do OT based on the shares of $[\![x]\!]_2$ being equal ($x = 0$) or not ($x = 1$). $\mathcal{CP}_2$ learns the outcomes as specified in Table 5.1 for each potential input combination and $\mathcal{CP}_1$ always outputs v_1.

For the efficiency analysis, we use the well known AIR OT protocol [1] for the *1-out-of-2* case. We use it with the Paillier cryptosystem as defined in Algorithm 16. Any OT protocol could be used in future implementations, but we will use this due to its simplicity to analyse the efficiency of using this share conversion in the triple generation protocols.

Algorithm 16 Aiello-Ishai-Reingold oblivious transfer

Setup: *Receiver* has defined a Paillier keypair (pk, sk), which defines a modulus N
Data: *Receiver* has input $x \in \{0, 1\}$, *Sender* has two secrets s_0 and s_1
Result: *Receiver* learns s_x

1: *Receiver* computes and sends $c = \mathsf{Enc}_{pk}(x)$ to the *Sender*
2: *Sender* generates $r_0, r_1 \leftarrow \mathbb{Z}_M$
3: *Sender* computes and sends $c_0 = c^{r_0} \cdot \mathsf{Enc}_{pk}(s_0)$ to the *Receiver*
4: *Sender* computes and sends $c_1 = (c \cdot \mathsf{Enc}_{pk}(-1))^{r_1} \cdot \mathsf{Enc}_{pk}(s_1)$ to the *Receiver*
5: *Receiver* decrypts $s_x = \mathsf{Dec}_{sk}(c_x)$

The idea of AIR OT is straightforward, as $c_0 = \mathsf{Enc}_{pk}(0 \cdot r_0 + s_0)$, if $x = 0$ and analogously, if $x = 1$ then $c_1 = \mathsf{Enc}_{pk}(0 \cdot r_1 + s_1)$. The role of the randomiser r_i is to ensure that the other secret is not leaked. For example, in case the query was $x = 0$, then c_1 is an encryption of a random value $\mathsf{Enc}_{pk}(-1 \cdot r_1 + s_1)$ and does not reveal s_1.

5.3.2 Problems with converting the third triple element

We can expect the first two elements x and y of the triple to be generated according to some fixed modulus M. However, the triple generation protocol may change the modulus for the outcome $w = x \cdot y$. The possibility to convert this outcome back to the original modulus means that we can actually generate triples for any modulus. However, there are losses in how many bits we use in the multiplication and afterwards receive as triples.

One place where such share conversion is needed is from the Paillier modulus to our chosen modulus M. At some point in triple generation, we use Paillier Multiplication, where the result will be given modulo N. Namely, if we have some uniformly generated values $[\![x]\!]_M$ and $[\![y]\!]_M$ modulo M, then we know that $x \cdot y < N$ if $M^2 < N$. Hence, we can avoid modular reductions in the product. However, working with additive share representation requires more care, as we actually have

$$(x_1 + x_2) \cdot (y_1 + y_2) < N \ ,$$

where $x_1 + x_2 < 2M$ and

$$(x_1 + x_2) \cdot (y_1 + y_2) < 4M^2 \ .$$

Hence, we require that $4M^2 < N$. With this restriction to initial values, we can apply general share conversion to the third triple element and achieve a multiplicative triple with respect to modulus M.

The main challenge in the share conversion is to differentiate between having either

$$w_1 + w_2 \geq N \text{ or } w_1 + w_2 < N \ .$$

The latter case means that if $x \cdot y = w_1 + w_2 \bmod N$ then also $x \cdot y = w_1 + w_2 \bmod M$ and share conversion can be obtained by converting both $w_i \bmod M$ separately. However, the former gives us $x \cdot y = w_1 + w_2 - N \bmod M$ and means that we have to check for this error when converting triples. Achieving this error correction is one of the important goals of the triple generation algorithms based on Paillier Multiplication. The main idea is that the parties can collaboratively decide if they have $w_1 + w_2 \geq N$ or $w_1 + w_2 < N$ and in the former case they can fix the shares as $w = w - N$ before modular reduction.

There are many different ways for performing this check. For example, one possibility is to do computations as in Triple Verification and check for the value of h. It is straightforward to fix values of h that mean that a triple was correct or had $w_1 + w_2 \geq N$. For security, h should not be published and the correction could be done based on the value of h using oblivious transfer. A more efficient method is introduced as part of Algorithm 17 and a more general idea analogous to that is also specified later in Chapter 6 as Algorithm 20. The ideas in Algorithm 17 and Algorithm 20 can be used to perform general conversion from $[\![x]\!]_N$ to $[\![v]\!]_M$, independently of the fact that we are working with a multiplication result.

In addition, share conversion of the third triple element can be used with CRT packing to get results that all use the same modulus. For example, the initial shares of x_i and y_i are fixed with relation to some modulus P and then we choose primes $p_i > P^2$ and interpret these shares each for a different modulus p_i. After the triple generation we learn $xy \bmod p_i$, but as previously, we know that $xy < p_i$ and now converting them from modulus p_i to P is the same as converting from N to M after Paillier Multiplication. This approach decreases the efficiency of the CRT packing by half, but helps to get rid of the need to use different moduli.

5.3.3 Triple generation with share conversion

The problem with using Paillier Multiplication for triple generation in a straightforward manner was mentioned in Section 5.3.2. It means that occasionally this protocol gives a valid triple for moduli other than the Paillier modulus, but sometimes the result is

invalid. This section introduces a way that uses the Paillier Multiplication algorithm, but adds additional checks to always get the correct result for a chosen modulus.

The idea of ShareConv-Triples in Algorithm 17 is to use a modulus 2 to test if the shares of the third element $w_1 + w_2 < N$ or overflow N and base the share conversion on the result of this check. This clearly leaks some information about the result as we need to declassify the least significant bit of the inputs, but this may not be an issue for all use-cases. The idea is that by declassifying the least significant bits of x and y we learn what the parity of w should be without any modular reductions. It could be used with CRT packing trivially as long as we do not use modulus 2 in the packing. Using this idea exactly like this is actually not secure as it leaks the least significant bits of x and y.

However, we can easily avoid the leakage, by actually using an odd modulus P that is one bit shorter that maximum length of M in Paillier Multiplication. In such case we just define

$$x_i = 0 \bmod 2 \text{ and } y_i = 0 \bmod 2$$

and use the CRT to compute the representation of x and y for modulus $2P$. We can learn the correct triple modulo P by reducing the final shares of $[\![w]\!]_{2P}$ modulo P.

According to the CRT, the multiplicative relation modulo $2P$ holds exactly, if it holds for all its prime divisors, including 2. The latter means that by checking the relation modulo 2, we actually verify that it holds for modulus $2P$.

Algorithm 17 Triple generation with share conversion (ShareConv-Triples)

Setup: Paillier keypair (pk, sk) with modulus N
$2|P| + 5 < |N|$ and P is odd
Data: Shared secrets $[\![x]\!]_P, [\![y]\!]_P$
Result: Third triple element $[\![w]\!]_P$ where $w = xy \bmod P$

1: $\mathcal{CP}_i$ uses CRT with inputs $x_i = 0 \bmod 2$ and $x_i = x_i \bmod P$ to learn $x_i \bmod 2P$
2: $\mathcal{CP}_i$ uses CRT with inputs $y_i = 0 \bmod 2$ and $y_i = y_i \bmod P$ to learn $y_i \bmod 2P$
3: $\mathcal{CP}$ compute $[\![w]\!]_{2P}$ from $[\![x]\!]_{2P}, [\![y]\!]_{2P}$ with Paillier Multiplication
4: $\mathcal{CP}_i$ computes $c_i = w_i \bmod 2$
5: $\mathcal{CP}$ convert $c = c_1 + c_2$ from $[\![c]\!]_2$ to $[\![c]\!]_{2P}$
6: $\mathcal{CP}_i$ computes $w_i = w_i - N \cdot c_i \bmod 2P$ to correct the potential mistakes
7: $\mathcal{CP}_i$ fixes $w_i = w_i \bmod P$ to get the correct final modulus

Theorem 5.3.1. *The* ShareConv-Triples *protocol in Algorithm 17 for generating Beaver triples with simple share conversion is correct assuming the correctness of share conversion from* $[\![c]\!]_2$ *to* $[\![c]\!]_{2P}$ *and* Paillier Multiplication.

Proof. For correctness, we need to analyse the meaning of c in this algorithm. We assume that the share conversion from $[\![c]\!]_2$ to $[\![c]\!]_{2P}$ is correct and Paillier Multiplication always gives either $w = w_1 + w_2$ or $w = w_1 + w_2 - N$. By definition, $c \in \{0, 1\}$ and $c = c_1 + c_2$, where

$$c_1 = w_1 \bmod 2 \text{ and } c_2 = w_2 \bmod 2 \ .$$

We know that as both x and y are defined as being even then w also has to be even and we have

$$c = w_1 + w_2 \bmod 2 \ .$$

We also know that N is odd and therefore w can be even if either $w_1 + w_2$ is even which means $w_1 + w_2 < N$ or if $w_1 + w_2$ is odd which gives $w_1 + w_2 \geq N$. Hence, if $c = 0$, then both w_i have the same parity and $w_1 + w_2$ is even. Knowing that w has to be even gives us $w = w_1 + w_2 < N$. On the other hand, if $c = 1$, then w_i have different parity and $w_1 + w_2 \geq N$ as integer is odd. Thus, we have $w = w_1 + w_2 - N$.

This clearly corresponds to how we compute w as $w_i = w_i - N \cdot c_i \bmod 2P$ gives us

$$w = w_1 + w_2 = w_1 - N \cdot c_1 + w_2 - N \cdot c_2 = w_1 + w_2 - c \cdot N \bmod 2P \ .$$

The final modular conversion $w_i = w_i \bmod P$ is correct according to the CRT. □

Theorem 5.3.2. *The* ShareConv-Triples *protocol in Algorithm 17 for generating Beaver triples with simple share conversion is secure, assuming the security of binary share conversion and* Paillier Multiplication.

Proof sketch. Te security follows trivially, as this protocol is just a combination of share conversion, Paillier Multiplication and local operations. □

Protocol ShareConv-Triples works trivially well with CRT packing, if the packing does not include modulus 2, because we require P to be odd. There is no need to use this with B-ary packing as in that case, the randomisation in Paillier Multiplication is defined so that we always get $w_1 + w_2 < N$ and there is no need for additional correction.

This protocol is more efficient than the previous, enabling us to get one $\frac{|N|}{2} - 2$-bit triple at the cost of one Paillier multiplication and error correction. However, it proposes additional restrictions to the choice of P, which has to be odd. In terms of achieved bits we clearly have

$$\gamma_{len} = \frac{|P|}{|N|} \leq \frac{1}{2} \ .$$

In terms of communication we have to take into account that we do Paillier Multiplication and share conversion, we currently use share conversion with AIR OT for comparison. This gives us

$$\gamma_{net} = \frac{|P|}{6 \cdot 2|N|} \leq \frac{1}{24}$$

because AIR OT has the same communication cost as Paillier Multiplication. Finally, in terms of computation, we also require ciphertext operations in both Paillier Multiplication and OT, which gives

$$\gamma_{comp} = \frac{|P|}{10|N| + 2(|P| + 1) + 5} \leq \frac{1}{22} \ .$$

5.4 Comparison of proposed triple generation ideas

In a real life setting, we would like to generate a set of triples for some fixed size. This section gives a comparison of the ideas from this chapter for the case where we are interested in learning triples for $M = 2^{32}$, we analyse the case for $|N| = 2048$. For packing, we assume that $M = 2^{32}$ for B-Triples and that we use 33-bit primes for CRT packing. In addition, for B-Triples we define $\sigma = 112$, which gives $\ell = 178$ and $m = 11$.

In B-ary packing we assume the same setup and basic packing with

$$x = B^{m-1}x_m + \ldots + x_1$$
$$y = B^{(m-1)\cdot m}y_m + B^m y_2 + \ldots + y_1 \ .$$

We need $|B| = 2 \cdot |M| + \sigma = 176$ and $B^{m^2} < N$, which enables us to pack $m = 3$ elements, because we need $m^2 < 11$. With CRT packing, we can pack at most 31 elements if we choose small primes, or 30 for general 33-bit primes, because we need that $4M^2 < N$. In addition, for ShareConv-Triples we can not use exactly $P = 2^{32}$, but we expect that $|P| = 32$ and therefore $|2P| = 33$.

Protocol	γ_{len}	γ_{net}	γ_{comp}
Paillier Multiplication	$\frac{1}{128} \approx 0.008$	$\frac{1}{768} \approx 0.001$	$\frac{8}{4129} \approx 0.002$
B-ary packing	$\frac{3}{64} \approx 0.047$	$\frac{1}{128} \approx 0.008$	$\frac{48}{4129} \approx 0.012$
CRT packing	$\frac{31}{128} \approx 0.242$	$\frac{31}{768} \approx 0.04$	$\frac{248}{4129} \approx 0.06$
B-Triples	$\frac{11}{64} \approx 0.171$	$\frac{11}{2944} \approx 0.004$	$\frac{352}{51837} \approx 0.007$
ShareConv-Triples	$\frac{1}{64} \approx 0.016$	$\frac{1}{768} \approx 0.001$	$\frac{32}{20551} \approx 0.002$
CRT packing	$\frac{1023}{2048} \approx 0.5$	$\frac{341}{8192} \approx 0.042$	$\frac{992}{22471} \approx 0.044$

Table 5.2: Comparison of Beaver triple generation protocols

The approximate values in Table 5.2 are given simply for making the comparison of these results more straightforward. Trivially, Paillier Multiplication with B-ary packing is three times more efficient than without as we can pack exactly 3 elements. However, there is an additional 2 times gain as the output triples are always correct. Packing with CRT is exactly 31 times more efficient that plain Paillier Multiplication. Though, the main trouble with these two cases is that although these are the average ratios assuming that the triple is rightfully corrected, we should also verify that they are correct. For example, if we use Triple Verification then learning one correct triple actually has the cost of two unverified triples.

Partial packing in B-Triples has significantly better packing count than using basic B-ary packing which results in better ratio of γ_{len}. The loss in other parameters is small enough to give this algorithm precedence over Paillier Multiplication with packing.

Basic ShareConv-Triples is close to Paillier Multiplication as expected, as we always get the correct triple, but the correction has approximately the same cost as the multiplication. The ratio for γ_{len} of ShareConv-Triples with CRT packing is actually ideal because $\frac{1}{2}$ is the best limit we can achieve with our current ideas about arbitrary modulus in Paillier Multiplication. This packing ratio also affects the efficiency of network usage as well as computations and clearly makes this the most efficient of our ideas resulting in bounds close to the theoretical ones.

In Chapter 7, we also give results for the implementation of ShareConv-Triples with CRT packing and B-Triples, as they are the more efficient and easier to use protocols according to given comparison. However, Table 5.2 also indicates that we possibly should consider only using Paillier Multiplication and CRT packing in cases where we can increase the probability of receiving a correct triple so that we are more likely to pass the Triple Verification check. It is especially meaningful if we need to perform Triple Verification to check for some other possible errors as well. In the follow-up work, we should compare the efficiency of this approach to ShareConv-Triples with CRT packing and different OT protocols.

Chapter 6

Symmetric two-party computation

This section introduces our ideas for setting up symmetric two-party computation. Currently, this section consists of the online phase, which is derived quite directly from the share representation and ideas from the asymmetric protocol set in Chapter 4. The question of achieving reasonably efficient precomputation of Beaver triples is currently unsolved, but this section give hints on how we might use the protocols from Chapter 5.

6.1 Protection domain setup

We consider additive secret sharing in a ring $\mathbb{Z}_p$ for some modulus p. Party $\mathcal{CP}_i$ defines a MAC key k_i so that $z^{(i)} = k_i \cdot x \bmod p$. It is clear from Section 2.1.9 that we can obtain a secure protection scheme for a prime p and moduli with only suitably large prime divisors. In case we use a modulus with a short bitlength, we can allow each party to define several keys to enhance the security. This way, we could achieve the necessary security level for any desired threshold, independently of the modulus. However, each additional key will make the computations less efficient. It is currently an open question, if a suitable efficient MAC algorithm could be obtained for other moduli. All arithmetic in this scheme is with respect to the modulus p. In the following, the security proofs give security guarantees with respect to using a prime modulus p.

We propose a share representation as

$$[\![x]\!]_p = \langle \Delta, x_1, x_2, z_1^{(1)}, z_2^{(1)}, z_1^{(2)}, z_2^{(2)} \rangle \ ,$$

where $x = x_1 + x_2 + \Delta$ and Δ is the public modifier. The remaining values belong to the MAC tags as $z_1^{(1)} + z_2^{(1)} = k_1 \cdot (x_1 + x_2)$ and $z_1^{(2)} + z_2^{(2)} = k_2 \cdot (x_1 + x_2)$. Both parties know Δ and, in addition, $\mathcal{CP}_i$ has values x_i, $z_i^{(1)}$ and $z_i^{(2)}$.

It is straightforward to obtain an addition protocol (Addition) as both parties can just locally add their shares to get their share of the sum. Analogously, we get protocols for subtraction (Subtraction) and multiplication with a public value (Constant Multiplication). Addition with a public value (Constant Addition) still only requires modifying the common value Δ. Thus, a public value v can be seen as

$$[\![v]\!]_p = \langle \Delta = v, v_1 = 0, v_2 = 0, z_1^{(1)} = 0, z_2^{(1)} = 0, z_1^{(2)} = 0, z_2^{(2)} = 0 \rangle \ .$$

For the sake of achieving protocols for communication with non-computing parties $\mathcal{IP}_i$ and $\mathcal{RP}_i$ and precomputation, we also assume that both computing parties $\mathcal{CP}_i$ have defined their own Paillier keypair (pk_i, sk_i) where pk_i is also known by the other

parties and defines a modulus N_i. In addition, they have published a commitment $\mathsf{Enc}_{pk_i}(k_i) = (\![k_i]\!)_{pk_i}$. The inconvenience in this is that our otherwise statistically secure setup becomes computationally secure, depending on the IND-CPA security of the cryptosystem that hides k_i.

We occasionally use the notation $\mathcal{CP}_i$ and $\mathcal{CP}_j$ where the idea is that $i \neq j$. For example, to specify that $\mathcal{CP}_i$ sends something to the other party $\mathcal{CP}_j$ where the meaning is that both computing parties send something to the other. We occasionally use an abbreviation $\mathcal{CP}$, that should be read as *computing parties*, to denote that both $\mathcal{CP}_i$ execute some sub-protocol together.

6.2 Publishing shared values

Due to the symmetric setup of the protection domain, we can give a general publishing protocol Publish-$\mathcal{CP}_i$ (Algorithm 18) to open share to party $\mathcal{CP}_i$. Party $\mathcal{CP}_i$ learns the correct result if the verification succeeds and should otherwise abort the protocol. $\mathcal{CP}_i$ is the party who should receive the output and by $\mathcal{CP}_j$ we mean the other party who sends its shares. We can combine two instances of this protocol to simultaneously declassify to both computing parties (Publish-both-$\mathcal{CP}_i$).

Algorithm 18 Publishing a shared value to $\mathcal{CP}_i$ (Publish-$\mathcal{CP}_i$)

Data: Shared secret $[\![x]\!]_p$
Result: $\mathcal{CP}_i$ learns the value x
1: $\mathcal{CP}_j$ sends x_j and $z_j^{(i)}$ to $\mathcal{CP}_i$
2: $\mathcal{CP}_i$ verifies $z_1^{(i)} + z_2^{(i)} = k_i \cdot (x_1 + x_2)$
3: **return** $\mathcal{CP}_i$ outputs $x_1 + x_2 + \Delta$

Theorem 6.2.1. *Protocol* Publish-$\mathcal{CP}_i$ *for publishing a shared value to one party is correct.*

Proof. For correctness, we need that $x = x_1 + x_2 + \Delta$, which is trivially true in case the verification process is correct. In the case of honest participants, we know that the verified equations must hold by the definition of the shares. □

Theorem 6.2.1. *Protocol* Publish-$\mathcal{CP}_i$ *for publishing a shared value to one party is computationally secure with additional statistical* $\frac{1}{p}$ *error probability.*

Proof sketch. This proof is analogous to the part Publish-$\mathcal{CP}_1$ in Theorem 6.2.1. In the asymmetric setting, $\mathcal{CP}_1$ also verified the correctness using MAC tags. The computational requirement follows from the fact that $\mathcal{CP}_j$ knows $(\![k_i]\!)_{pk_i}$. □

As in the asymmetric case, we have a simple possibility that we declassify an element to the computing parties, who check the MAC and then forward the declassified results to the result parties $\mathcal{RP}_i$. The result parties only have to verify that both computing parties forwarded them the same declassification result and accept the output. Thus, we can easily obtain the protocol Publish-$\mathcal{CP}\&\mathcal{RP}_i$. The security and correctness of this protocol result from those of Publish-$\mathcal{CP}_i$ protocol. This is an easy way to make the results publicly known, a way to open shares only to $\mathcal{RP}_i$ is discussed later in Section 6.4.

6.3 Receiving inputs from the input party

This section defines a standalone protocol to receive inputs from $\mathcal{IP}_i$. We do not yet have a Publish-$\mathcal{RP}_i$ protocol, therefore we can not use the common Classify-$\mathcal{IP}_i$ protocol from Chapter 3 and need to define something independent from other protocols. Our Classify-$\mathcal{IP}_i^\star$ protocol is given in Algorithm 19.

Algorithm 19 Receiving inputs from $\mathcal{IP}_i$ (Classify-$\mathcal{IP}_i^\star$)

Data: $\mathcal{IP}_i$ has a secret x
Result: $\mathcal{CP}_i$ have a shared secret $[\![x]\!]_p$

1: Round: 1
2: $\quad\mathcal{CP}_i$ fixes $\Delta = 0$
3: $\quad\mathcal{IP}_i$ generates $x_1 \leftarrow \mathbb{Z}_p$, $z_1^{(2)} \leftarrow \mathbb{Z}_{N_2}$, $z_2^{(1)} \leftarrow \mathbb{Z}_{N_1}$, $r_1, r_2 \leftarrow \mathbb{Z}_N^*$
4: $\quad\mathcal{IP}_i$ computes $x_2 = x - x_1 \bmod p$
5: $\quad\mathcal{IP}_i$ computes $c_1 = \mathsf{Enc}_{pk_1}(k_1)^{x_2} \cdot \mathsf{Enc}_{pk_1}(-z_2^{(1)}, r_1)$
6: $\quad\mathcal{IP}_i$ computes $c_2 = \mathsf{Enc}_{pk_2}(k_2)^{x_1} \cdot \mathsf{Enc}_{pk_2}(-z_1^{(2)}, r_2)$
7: $\quad\mathcal{IP}_i$ sends $x_i, z_i^{(j)}, c_i, r_j$ to $\mathcal{CP}_i$
8: Round: 2
9: $\quad\mathcal{CP}_i$ computes $z_i^{(i)} = k_i \cdot x_i + \mathsf{Dec}_{pk_i}(c_i) \bmod N_i$ to learn $[\![z^{(i)}]\!]_{N_i}$
10: $\quad\mathcal{CP}_i$ computes $c_j^* = \mathsf{Enc}_{pk_j}(k_j)^{x_i} \cdot \mathsf{Enc}_{pk_j}(-z_i^{(j)}, r_j)$ and send to $\mathcal{CP}_j$
11: Round: 3 (share conversion)
12: $\quad\mathcal{CP}$ collaboratively convert $[\![z^{(1)}]\!]_{N_1}$ to $[\![z^{(1)}]\!]_p$ and $[\![z^{(2)}]\!]_{N_2}$ to $[\![z^{(2)}]\!]_p$ using ShareConv
13: Verification:
14: $\quad\mathcal{CP}_i$ checks that $c_i = c_i^*$
15: $\quad\mathcal{CP}_i$ notifies $\mathcal{IP}_i$ about the verification outcome

The idea of this algorithm is similar to some tricks from the triple generation algorithms. Namely, the input party uses the encryptions of the MAC keys to share the tags modulo N and the computing parties convert them to correct modulus M.

Theorem 6.3.1. *The protocol* Classify-$\mathcal{IP}_i^\star$ *for receiving inputs from* $\mathcal{IP}_i$ *is correct, assuming the correctness of* ShareConv.

Proof sketch. The correctness of $x = x_1 + x_2 \bmod p$ is trivial from the definition. Similarly, the correctness of $z^{(i)} = k_i \cdot x \bmod N_i$ is straightforward from the algorithm description. Furthermore, the correctness of the verification is trivial, as by definition $c = c^*$ if all parties are honest. □

For security, we need that neither the computing parties nor the input party can cheat during the protocol. Cheating on the side of input party means that it tries to give inconsistent share representations. On the side of computing parties cheating means that they try to modify the shares they received. However, as in the asymmetric case, we still have the limitation that $\mathcal{IP}_i$ can not collude with either $\mathcal{CP}_i$.

Theorem 6.3.2. *The protocol* Classify-$\mathcal{IP}_i^\star$ *for receiving inputs from* $\mathcal{IP}_i$ *is computationally secure against corrupted* $\mathcal{CP}_i$ *and perfectly secure against corrupted* $\mathcal{IP}_i$*, if the adversary is allowed to corrupt at most one party, assuming a computationally secure share conversion protocol.*

Proof sketch. Similarly to the asymmetric case, the ideal functionality of this protocol receives x, x_1, $z_1^{(2)}$ and $z_2^{(1)}$ from the $\mathcal{IP}_i$. It then fixes the remaining $z_1^{(1)}$ and $z_2^{(2)}$ and forwards the shares to the computing parties. The computing parties can either accept or reject the shares. If either party rejects then the output of all parties is $\perp$, otherwise the computing parties learn their shares and $\mathcal{IP}_i$ learns that the input was received correctly.

The simulator for a corrupted $\mathcal{CP}_i$ at first receives x_i, $z_i^{(1)}$, and $z_i^{(2)}$ from the TTP. It then computes $c_i = \mathsf{Enc}_{pk}(z_i^{(i)}) \cdot (\![k_i]\!)_{pk_i}^{-x_i} = \mathsf{Enc}_{pk}(z_i^{(i)} - k_i \cdot x_i)$ and picks a r_j as an honest $\mathcal{IP}_i$ would. The simulator forwards x_i, $z_i^{(j)}$, c_i and r_j to the corrupted $\mathcal{CP}_i$ as a message from the $\mathcal{IP}_i$. By definition, $\mathcal{CP}_i$ can compute $z_i^{(i)}$ as originally defined by the TTP. In addition, it forwards c_i also as a message from the other computing party $\mathcal{CP}_j$. For the share conversion, it can act as an honest party by picking a random input for the case where it has to send the initial query. In addition, it can simulate the conversion for the case where it is the sender. Finally, the simulator gets the output *Continue* or *Failure* from the corrupted $\mathcal{CP}_i$ and forwards this to the TTP. Clearly, the simulator can make the output shares of $\mathcal{CP}_i$ the same as they would be in the ideal world and the output of $\mathcal{CP}_j$ is also the same, therefore, the outputs of the real and simulated ideal world coincide.

The simulator for the corrupted $\mathcal{IP}_i$ receives all the values x_i, $z_i^{(j)}$, c_i, r_i from the $\mathcal{IP}_i$. It checks that $c_i = (\![k_i]\!)_{pk_i}^{x_j} \mathsf{Enc}_{pk}(-z_j^{(i)}, r_j)$ for both c_i and forwards x, x_1, $z_1^{(2)}$ and $z_2^{(1)}$ to the TTP, if the check succeeds. In this case, it gives the output *Continue* or *Failure*, that it receives from the TTP, back to the corrupted $\mathcal{IP}_i$. Otherwise, if the check did not pass, it gives *Failure* to both the corrupted $\mathcal{IP}_i$ and the TTP. Clearly, the check that the simulator does for c_i is sufficient to check that the $\mathcal{IP}_i$ gives correct inputs. In addition, the final states of the ideal and real world coincide as in case the sharing succeeds the shares are chosen by the $\mathcal{IP}_i$ and parties have also seen these shares in case the sharing does not succeed. □

This protocol does not give the anti-framing property, because in the end the computing parties have not verified that the other party has the value $z_i^{(i)}$ that it needs to very the tag. To achieve this property we must include corresponding zero-knowledge proofs as a part of this protocol. We should also include the proofs that c_i is correctly computed in order to achieve security against collaborating pairs $\mathcal{CP}_j$ and $\mathcal{IP}_i$.

We can use the ShareConv protocol in Algorithm 20 for share conversion. This is intended for the case introduced in Section 5.3.2 where we either have $z = z_1 + z_2 \bmod p$ or $z = z_1 + z_2 - N \bmod p$, which is exactly what we have in the Classify-$\mathcal{IP}_i^\star$ protocol.

Algorithm 20 Share conversion from $[\![z]\!]_N$ to $[\![v]\!]_p$ (ShareConv)

Data: Shared secret $[\![z]\!]_N$
Result: Shared secret $[\![v]\!]_p$, where $z = v$

1: $\mathcal{CP}_i$ computes $t_i = 2 \cdot z_i \bmod N$
2: This ensures that $t = t_1 + t_2$ as an integer is even
3: $\mathcal{CP}_i$ computes $t_i^* = t_i \bmod 2$
4: $\mathcal{CP}$ collaboratively perform simple share conversion from $[\![t^*]\!]_2$ to $[\![c]\!]_p$,
5: This can be done with OT, as discussed in Section 5.3
6: $\mathcal{CP}_i$ computes $t_i = t_i - c_i \cdot N \bmod p$
7: $\mathcal{CP}_i$ computes $v_i = 2^{-1} \cdot t_i \bmod p$

Theorem 6.3.3. *The protocol* ShareConv *in Algorithm 20 for converting* $[\![z]\!]_N$ *to* $[\![v]\!]_p$ *is correct and secure, assuming secure share conversion from binary to any modulus.*

Proof sketch. The idea of computing $t_i = 2 \cdot z_i \bmod N$ is to ensure that $t_1 + t_2$ share an even number $[\![2z]\!]_N$. This enables us to do the share conversion using OT from Section 5.3 where, in the end, computing parties have a shared secret $c = 0$, if the parity of t_1 and t_2 was the same and $c = 1$ in other case. Here, different parity indicates that $t_1 + t_2 \geq N$ and same parity ensures $t_1 + t_2 < N$. By computing $t_i = t_i - c_i \cdot N \bmod p$ the parties learn $[\![2z]\!]_p$, where $2z = t_1 + t_2 \bmod p$. Trivially, computing $v_i = 2^{-1} \cdot t_i \bmod p$ gives $[\![v]\!]_p$, where $v = z \bmod p$. The correctness of this protocol follows from the correctness of the general share conversion idea.

The security clearly results from the security of the conversion from $[\![t^*]\!]_2$ to $[\![c]\!]_p$. According to Section 5.3.1, it depends on the security of the oblivious transfer. □

6.4 Publishing a secret to the result party

Previously, we defined a protocol for declassifying a secret to both computing and result parties (Publish-$\mathcal{CP}\&\mathcal{RP}_i$). However, we also have to satisfy the case where we need to open the secret privately only to $\mathcal{RP}_i$. Algorithm 21 defines protocol Publish-$\mathcal{RP}_i$ that achieves this by combining Classify-$\mathcal{IP}_i^\star$ and Publish-$\mathcal{CP}\&\mathcal{RP}_i$ in a very general manner.

Algorithm 21 Publishing a shared value to $\mathcal{RP}_i$ (Publish-$\mathcal{RP}_i$)

Data: secret $[\![x]\!]_p$
Result: $\mathcal{RP}_i$ learns x

1: $\mathcal{RP}_i$ shares a uniformly random value y using Classify-$\mathcal{IP}_i^\star$
2: $\mathcal{CP}$ compute $[\![w]\!]_p = [\![x]\!]_p + [\![y]\!]_p$
3: $\mathcal{CP}$ and $\mathcal{RP}_i$ execute Publish-$\mathcal{CP}\&\mathcal{RP}_i$ to learn $w = x + y$
4: **return** $\mathcal{RP}_i$ corrects $x = w - y$

We could probably use a simpler version of Classify-$\mathcal{IP}_i^\star$ for this purpose, because the verification of correct sharing is actually a part of Publish-$\mathcal{CP}\&\mathcal{RP}_i$. Hence, we could omit all the verification steps from Classify-$\mathcal{IP}_i^\star$. The idea of this protocol is very simple, as the value y randomises the published result w and, thus, w reveals nothing about x to $\mathcal{CP}_i$.

Theorem 6.4.1. *The* Publish-$\mathcal{RP}_i$ *protocol for declassifying shared secrets to result parties is correct and as secure against a cheating* $\mathcal{CP}_i$ *as* Publish-$\mathcal{CP}\&\mathcal{RP}_i$.

Proof sketch. Both, correctness and security, result from the properties of the used sub-protocols Addition, Classify-$\mathcal{IP}_i^\star$ and Publish-$\mathcal{CP}\&\mathcal{RP}_i$. The correctness of the output $x = w - y$ follows trivially from the definition $w = x + y$. □

6.5 Precomputation

We do not have a full precomputation phase for this protection domain at the moment. We introduce a protocol for generating random shares and discuss how the Beaver triple protocols from Chapter 5 could be used to achieve the necessary share representation and security guarantees.

6.5.1 Random share generation

Generating MAC tags is actually the same task as generating Beaver triples, only the inputs are slightly different. An additively shared secret that needs the tag can be used as one of the random inputs. The second is clearly the key so that the third element of the triple is actually the tag value.

Although the key is not kept in the form of additive shares, we can assume that $\mathcal{CP}_i$ has the share as the value of the key k_i and $\mathcal{CP}_j$ has the share value as 0. This, as well as the fact that we always use the same k_i for all shares, allows us to somewhat simplify the triple generation protocols, but the core ideas remain the same. All in all, Beaver triple generation protocols can also be the key for generating protection mechanisms to shares and thus, to generating single random values as well as triples.

More specifically, Classify-$\mathcal{IP}_i^\star$ already introduced a way, how a third party can generate a valid share representation. The value x is not known when the computing parties generate a random value, but they could collaborate to share the tags modulo N as shown in Algorithm 22. Afterwards, they could do the same conversion as in Classify-$\mathcal{IP}_i^\star$.

Algorithm 22 Generating a random share (Singles)

Data: No input

Result: $\mathcal{CP}_i$ have a shared secret $[\![x]\!]_p$ for uniformly random $x \leftarrow \mathbb{Z}_p$

1: Round: 1
2: $\quad\mathcal{CP}_i$ fixes $\Delta = 0$
3: $\quad\mathcal{CP}_i$ generates $x_i \leftarrow \mathbb{Z}_p$, $z_i^{(j)} \leftarrow \mathbb{Z}_{N_j}$
4: $\quad\mathcal{CP}_i$ computes and sends $c_j = \mathsf{Enc}_{pk_j}(k_j \cdot x_i - z_i^{(j)})$ to $\mathcal{CP}_j$
5: Round: 2
6: $\quad\mathcal{CP}_i$ computes $z_i^{(i)} = k_i \cdot x_i + \mathsf{Dec}_{pk_i}(c_i) \bmod N_i$ to learn $[\![z^{(i)}]\!]_{N_i}$
7: $\quad\mathcal{CP}$ collaboratively perform ShareConv to get $[\![z^{(1)}]\!]_p$ from $[\![z^{(1)}]\!]_{N_1}$ and $[\![z^{(2)}]\!]_p$ from $[\![z^{(2)}]\!]_{N_2}$

Theorem 6.5.1. *The* Singles *protocol for generating random shares with correct tags is correct.*

Proof sketch. We need that $z_1^{(i)} + z_2^{(i)} = k_i \cdot (x_1 + x_2) \bmod p$. We assume the correctness of the share conversion and only show

$$z_1^{(i)} + z_2^{(i)} = k_i \cdot (x_1 + x_2) \bmod N \ .$$

In addition, we assume that $k_i \cdot (x_1 + x_2) < N$. We know that, by definition

$$z_i^{(i)} = k_i \cdot x_i + k_i \cdot x_j - z_j^{(i)} \bmod N \ ,$$

where it trivially follows that $z^{(i)} = k_i \cdot x \bmod N$. □

Similarly to the asymmetric version of the Singles protocol, we would actually need a zero-knowledge proof that the messages c_i are correctly formed. Currently, we could not define a simulator for the ideal version of the Singles protocol where the parties input x_i because the messages that the parties send are independent of their inputs. The ideal functionality that we would like to achieve is that the computing parties input x_i to the TTP who computes all the tag values and gives them back to the computing parties.

Theorem 6.5.2. *The communication of the* Singles *protocol for generating random shares with correct tags is simulatable and the final value of x is uniformly distributed in $\mathbb{Z}_p$, assuming computationally secure share conversion.*

Proof sketch. The communication in the generation part of this protocol is clearly simulatable as by definition c_i is an encryption of a random value that does not depend on the protocol inputs and can be simulated by sending an encryption of a random value. The share conversion part can be simulated using the corresponding simulator.

Finally, if at least one participant is honest, then x is a uniformly random element in $\mathbb{Z}_p$. This holds, because if one participant $\mathcal{CP}_i$ is honest, then $x_i \leftarrow \mathbb{Z}_p$ is uniformly distributed in $\mathbb{Z}_p$ and so is $x_1 + x_2$, because party $\mathcal{CP}_j$ does not receive any information about the value of x_i. □

Differently from the asymmetric case this Singles protocol does not specify verification and therefore we can not be sure that the share $[\![x]\!]_p$ is correctly formed. This means that we can not get the anti-framing property, but does not make the protocol less secure as we would discover the wrongly formed share during the publishing. We could add some verification as, for example, we could generate the values in pairs. For each pair, the parties would randomly choose one value that they open and where both $\mathcal{CP}_i$ show that they know x_i and $z_i^{(j)}$ together with the randomness that they used to compute c_j. That means that with probability $\frac{1}{2}$ we could notice cheating in this protocol. A more general solution would be to add a zero-knowledge protocol about the correctness of c_i.

6.5.2 Beaver triples generation

As previously, we mainly check the correctness of the computations during the opening phase, where the information-theoretic security of the MAC ensures the correctness of the verification. We can use the protocol Triple Verification to ensure that both the multiplicative relation of the triple holds and the MAC tags have been computed correctly. The security of this check results from the security of the triple verification and the security of the MAC algorithm.

One way to generate valid Beaver triples with all protection mechanisms would be to at first generate two random values x and y together with MAC tags as in Singles protocol. Then we could choose a Beaver triple protocol from Chapter 5 and run this with input shares as additive shares of x and y to learn the additive shares of $w = x \cdot y$. It would require some extra care to ensure that the Beaver triple protocol defined in the semi-honest model protects the privacy of the inputs even in the presence of active adversaries. Finally, the tags could be generated for w similarly to the tag generation the the Singles protocol. This process should be finished with a full Triple Verification protocol to ensure that the triples really have the multiplicative relation.

It currently seems most beneficial to use the basic Paillier Multiplication protocol with CRT packing for modulus M that is k-bits shorter than the maximal length allowed by $M^2 < N$. This way, we can actually always optimistically correct the result w from the Paillier Multiplication as $w = w_1 + w_2 - N \bmod M$ and with probability more than $1 - 2^{-2k}$ we have a correct w modulo M. This probability results from the fact that with probability $1 - 2^{-2k}$ we pick $w_2 \leftarrow \mathbb{Z}_N$ such that $w_2 > xy$ and compute $w_1 > xy$, giving $w_1 + w_2 > N$. After generating the protection mechanisms, we anyway have to perform Triple Verification the check the tags which also checks if the parties received

a correct $w \bmod p_i$. The other possibility would be to use ShareConv-Triples with an efficient OT protocol. It would require testing the corresponding implementation in order to verify which can be more efficient.

6.6 Efficiency of the protocols

This section analyses the theoretical cost of the proposed protocols. We have two important criteria: (1) computational cost and (2) communication cost. Figure 6.1 illustrates all our protocols for the symmetric protocol set and also marks the place for the Triples protocol that was not specified.

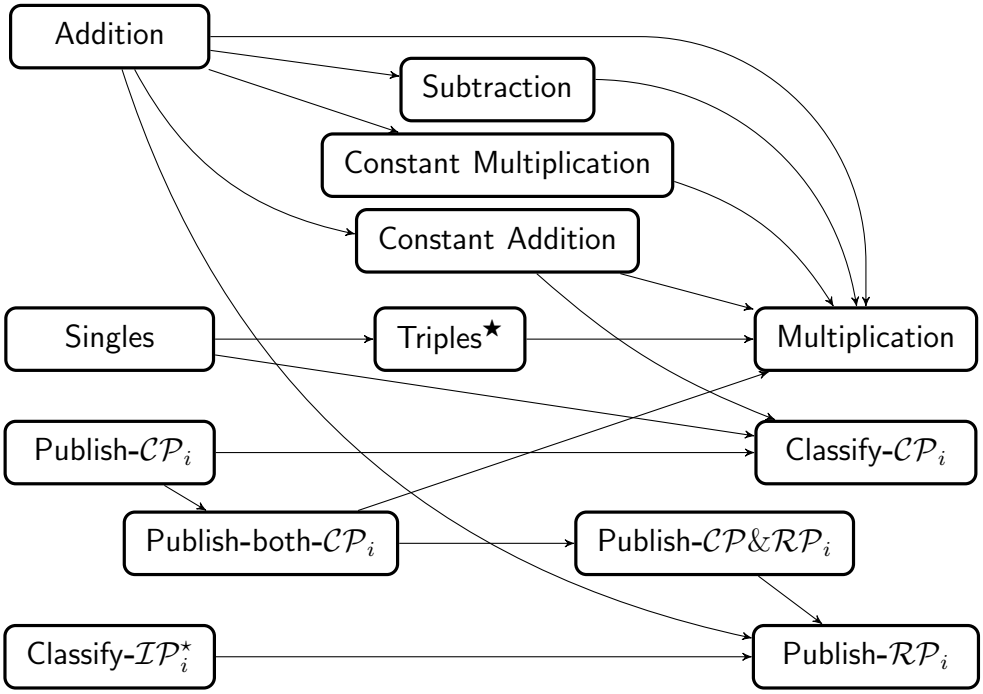


Figure 6.1: The hierarchy of the protocols for the symmetric setup

For simplicity of the analysis, we assume that both parties have chosen Paillier keys of the same length $|N_1| \approx |N_2|$. We need to consider three different classes of operations: (1) operations on additive shares of length $|p|$ bits, (2) operations on Paillier ciphertexts of length $2|N|$ bits, and (3) operations of Paillier plaintext space of length $|N|$ bits.

6.6.1 Computational cost

Differently from the asymmetric setting, most of the online protocols do not need any multiplications. Actually, the only computation protocols requiring multiplication operations are Multiplication, Constant Multiplication and Publish-$\mathcal{CP}_i$, whereas Multiplication only uses multiplication operations from the Constant Multiplication and Publish-$\mathcal{CP}_i$ protocols. In all these cases, we only use $|P|$ bit operands and get the result of the same length. However, the protocols to communicate with third parties add some complexity because there we also have to operate with ciphertexts of length $2|N|$ bits and plaintext space of $|N|$ bits. Out of these, we only analyse Classify-$\mathcal{IP}_i^\star$ because Publish-$\mathcal{RP}_i$ combines this with Publish-both-$\mathcal{CP}_i$ and uses no additional multiplications. Finally, we also include the Singles precomputation protocol.

Party	Length	ConstMult	Publish-$\mathcal{CP}_i$	Classify-$\mathcal{IP}_i^\star$	Singles
$\mathcal{CP}_i$	$\|p\|$	4	1	0	0
	$\|N\|$	0	0	1	1
	$2\|N\|$	0	0	$3\|N\| + \|p\| + 1$	$2\|N\| + \|p\| + 1$
$\mathcal{IP}_i$ $\mathcal{RP}_i$	$\|p\|$	-	-	0	-
	$\|N\|$	-	-	0	-
	$2\|N\|$	-	-	$2\|N\| + 2\|p\| + 2$	-

Table 6.1: The computational cost of protocols as a number of multiplications

As in the asymmetric setting, we assume that the Paillier encryption and decryption have the computational complexity of $|N|$ multiplications on elements of length $2|N|$. Addition under encryption has the cost of one multiplication and $\mathsf{Enc}_{pk}(m)^k$ has the cost of $|k|$ multiplications.

Table 6.1 summarises the computational complexity of our independent protocols. The complexity for Constant Multiplication results from the fact that all share elements have to be multiplied with the public value. These protocols are symmetric for the computing parties and the third party only participates in Classify-$\mathcal{IP}_i^\star$. However, for Publish-$\mathcal{CP}_i$, we mean that only $\mathcal{CP}_i$, to who the value is opened to, has to do this amount of work. From this we can see that actually the protocol Multiplication would only require 11 multiplications of length p.

The computational complexity of Classify-$\mathcal{IP}_i^\star$ for $\mathcal{IP}_i$ results from the computation of tag values under encryption where $\mathsf{Enc}_{pk}(k_1)^x$ requires approximately $|x|$ multiplications, which we estimate by $|p|$. It is similar for the Singles protocol, only that the work is done by $\mathcal{CP}_i$. We exclude the cost of share conversion from these protocols as it mainly depends on the complexity of the chosen oblivious transfer protocol.

It is easy to see that the need to use encryption makes Classify-$\mathcal{IP}_i^\star$ and Singles very expensive compared to the online computation protocols. In addition, compared to the asymmetric protocol set we have a very cheap Publish-$\mathcal{CP}_i$ protocol.

6.6.2 Communication cost

The protocols that require communication are Publish-$\mathcal{CP}_i$, Classify-$\mathcal{CP}_i$, Multiplication, Singles, Classify-$\mathcal{IP}_i^\star$ and Publish-$\mathcal{RP}_i$. However, Multiplication and Classify-$\mathcal{CP}_i$ only require communication as part of the Publish-$\mathcal{CP}_i$ protocol and Publish-$\mathcal{RP}_i$ is just a combination of Publish-$\mathcal{CP}_i$ and Classify-$\mathcal{IP}_i^\star$. Thus, it is reasonable to only analyse the communicational complexity of Publish-$\mathcal{CP}_i$, Classify-$\mathcal{IP}_i^\star$ and Singles.

Party	Length	Publish-$\mathcal{CP}_i$	Classify-$\mathcal{IP}_i^\star$	Singles
$\mathcal{CP}_j$	$\|p\|$	2	2	2
	$\|N\|$	0	0	0
	$2\|N\|$	0	1	1
$\mathcal{IP}_i$ $\mathcal{RP}_i$	$\|p\|$	-	2	-
	$\|N\|$	-	4	-
	$2\|N\|$	-	2	-

Table 6.2: The communication cost of the protocols as the number of messages of different length

Table 6.2 gives an overview of the communication complexity of the Publish-$\mathcal{CP}_i$, Classify-$\mathcal{IP}_i^\star$ and Singles protocols. The lines for Publish-$\mathcal{CP}_i$ should be read so, that in

this protocol, $\mathcal{CP}_j$ has to send that many messages. The workload of the computing parties is equal for the other protocols and they both have to send the given amount of messages. Third parties have to send a given amount of messages in total and exactly half of those are sent to each $\mathcal{CP}_i$.

Protocols Classify-$\mathcal{IP}_i^\star$ and Singles exclude the cost of the OT as the protocol could be used with different initialisations of OT or possibly with other binary share conversion ideas than our idea with OT.

Analogously to the computational cost the usage of a cryptosystem also makes the communication requirements of Classify-$\mathcal{IP}_i^\star$ and Singles to stand out. Especially, since the additional cost of the share conversion further increases the amount of messages. However, from the straightforward relatively low cost of Publish-$\mathcal{CP}_i$ we can also derive that multiplication and Classify-$\mathcal{CP}_i$ are communication efficient protocols in this protection domain.

Chapter 7

Implementation

This chapter introduces the details of our implementation as well as the benchmark results of the proposed protocol sets and some precomputation ideas.

7.1 Implementation platform

Our protocols are part of SHAREMIND 3 which is implemented in C++ as are our protection domains. SHAREMIND currently uses RakNet [50] as a network layer and Boost [13] for multi-threading and configuration. We used a popular free C++ cryptography library Crypto++ [20] for the functionality of the elliptic curves. In addition, the GNU Multiple Precision library (GMP) [34] was used to get unbounded integers needed to represent shares and ciphertexts in our implementation. The implementation of the Paillier cryptosystem is similar to [48], but ported to GMP.

7.2 Secure computation capabilities

The asymmetric protocol set is implemented as a SHAREMIND protection domain kind defining share operations for addition, subtraction and multiplication. In addition, there are special protocols to multiply a share with a public value and to add a public value to the share. All these protocols have been implemented as operations on vectors applying the suitable function component-wise. Besides these online protocols, the asymmetric protection domain also contains protocols for precomputing random values and multiplicative triples. These protocols are executed as needed to keep some threshold of precomputations available. In the future we might also consider the restricted configuration where all triples needed to execute a previously define algorithm are generated beforehand and not replaced during the computations.

Currently, the controller side of this PD, consisting of communication with non-computing parties, has not been implemented because the infrastructure of SHAREMIND 3 does not yet fully support this functionality. It is future work to implement these as well as a full setup phase. Current implementation is sufficient to give insights to the usability of this PD.

The symmetric protocol set only includes the online protocols for working with the data. This includes classifying, publishing, addition, subtraction, and multiplication on the shares, as well as adding or multiplying a shared value and a public constant. This online phase is accompanied by an insecure precomputation phase to produce singles

and triples necessary for testing online computations. It is future work to specify and implement a real precomputation phase. It is reasonable to test the online phase independently of the precomputations to see if it proves efficient enough to continue with these ideas. Similarly to the asymmetric case, the protocols to communicate with non-computing parties have not been implemented.

Two of the most promising protocols for Beaver triple generation were also implemented for testing purposes. At the moment they do not form a full precomputation phase for any PD. The B-Triples protocol is implemented with packing as built to the algorithm description. Protocol ShareConv-Triples is implemented in a general manner that is independent of the packing and tested using packing with CRT. We use a computationally private information retrieval (CPIR) [38] protocol to perform share conversion in ShareConv-Triples. CPIR is suitable for replacing OT in the semi-honest setting where our triple generation currently works. We used a *1-out-of-2* initialisation from [41] with the Paillier cryptosystem that is more communication efficient than AIR OT from Algorithm 16.

7.3 Performance measurements

The tests were executed on the SHAREMIND cluster where each miner ran in a different machine and they were communicating over LAN. Each of the cluster machines had 48 GB of RAM, two Intel Xeon X5670 CPUs and were connected with 1 GB/s LAN connection.

All the given results are average running times of the operations over at least ten repeated tests, more tests were used for faster operations. Column *length* denotes the length of the input and output vectors, other columns in the following tables denote various implemented protocols.

All the experiments were executed using a SECREC script. We recorded the running times of each independent execution of separate operations. These results are fixed at a miner level, thus allowing us to get separate measurements from both miners. The latter is mostly important for the online protocols of the asymmetric protocol set. It is important to note that the precomputations are running in parallel with online operations during the measurements of the online phase. This mostly affects the multiplication operation because it uses up a lot of precomputed triples that need to be replaced.

7.3.1 Online protocols

This section analyses the time requirements of the online phase of the asymmetric and symmetric protocol sets. We use the asymmetric setting with 2048-bit key and give the symmetric setting for 2048-bit prime as a comparison to that, as they represent similar data types. In addition, we compare the efficiency of the two computing parties in the asymmetric setting and give a 65-bit version of the symmetric PD.

Tables 7.1 and 7.2 illustrate the time requirements of the two computing parties in the asymmetric setting. Theoretical analysis in Section 4.6 indicated that this setup results in unbalanced workload for the two computing parties, and our measurements also reflect this. Local protocols of $\mathcal{CP}_1$ are two to three times faster than the same protocols for $\mathcal{CP}_2$ who also has to compute with ciphertexts. There is less difference for publishing or multiplication protocols as those are collaborative and it is likely that

Length	Publish	Add	Subtract	ConstAdd	ConstMult	Multiply
1	21.28	0.03	0.06	0.002	0.15	218.57
10	197.67	0.10	0.41	0.008	1.37	572.57
100	1974.02	0.62	3.93	0.037	13.49	4135.15
1000	19 732.16	6.27	38.87	0.170	134.19	39 866.97
10000	197 276.02	72.75	400.92	3.652	1343.81	392 461.09

Table 7.1: Time requirements of asymmetric computation protocols for party $\mathcal{CP}_1$ in SHAREMIND (milliseconds)

Length	Publish	Add	Subtract	ConstAdd	ConstMult	Multiply
1	24.76	0.02	0.11	0.003	0.47	222.54
10	210.76	0.15	0.98	0.004	4.65	599.92
100	2103.54	1.38	9.64	0.025	46.19	4399.50
1000	20 919.80	13.92	96.69	0.227	461.83	42 510.33
10000	209 190.81	172.94	989.36	5.749	4613.70	418 776.28

Table 7.2: Time requirements of asymmetric computation protocols for party $\mathcal{CP}_2$ in SHAREMIND (milliseconds)

$\mathcal{CP}_1$ has to wait until $\mathcal{CP}_2$ finishes some computations and answers on the network, before the parties can continue. Time requirements of both miners demonstrate a linear growth as the test inputs increase, illustrating that we actually do not gain much from vectorisation and that the computations are more likely to be CPU than network bounded.

The asymmetric setting can be compared to the symmetric setting with a 2048-bit modulus. Comparing the asymmetric results in Tables 7.1 and 7.2 to those of the symmetric protocols in Table 7.3 reveals that the gain from the symmetric protocol is significant. The declassifying and, thus, also multiplication protocols have gained most as there are no more encryption operations involved in the symmetric setting.

Length	Publish	Add	Subtract	ConstAdd	ConstMult	Multiply
1	10.28	0.01	0.01	0.003	0.01	110.48
10	10.56	0.03	0.05	0.004	0.03	112.36
100	9.94	0.26	0.39	0.024	0.24	127.89
1000	11.27	2.71	3.89	0.175	1.41	223.09
10000	22.65	34.83	48.63	2.534	12.27	1147.97

Table 7.3: Time requirements of symmetric computation protocols for 2048-bit modulus in SHAREMIND (milliseconds)

A new trend in the symmetric setting is that the times to declassify a value or multiply shares do not increase linearly as the input size grows, at least for small input sizes. This probably indicates that these protocols depend more on the network speed than computation power. The sudden growth in multiplication cost for *length* 10000 can be explained by the fact it has to perform several Publish operations and the network capacity may become a bottleneck. In addition, it requires as many triples as the input length and, thus, there is continuous precomputation in the background to replace those triples. These trends can be especially well seen from Table 7.4 which also includes longer input lengths.

The comparison of Table 7.3 to Table 7.4 shows, that the considerable differences in the data type size affect the running time less than we might expect. According

Length	Publish	Add	Subtract	ConstAdd	ConstMult	Multiply
1	10.51	0.02	0.01	0.005	0.01	55.79
10	10.27	0.04	0.02	0.007	0.01	56.76
100	10.16	0.23	0.19	0.023	0.05	54.33
1000	11.01	1.37	1.75	0.188	0.62	65.84
10000	24.77	13.56	17.84	0.886	4.49	203.48
100000	102.27	146.48	185.64	10.462	46.20	1880.76
1000000	846.05	1467.25	1682.50	97.189	460.60	14 084.73

Table 7.4: Time requirements of symmetric computation protocols for 65-bit modulus in SHAREMIND (milliseconds)

to Tables 7.3 and 7.4, computation with 65-bit modulus in only two to three times faster than computing with 2048-bit modulus. The difference between using 65-bit and 33-bit modulus illustrated the same trend where 33-bit modulus is only slightly faster than 65-bit. The surprising result that ConstMult is faster than Add results from the specifics of our setup where the public value is a uniformly random 32-bit element, which is small compared to general tested values. Measuring the symmetric setup with 33-bit prime gives a better estimate where ConstMult is actually approximately three times slower than Add.

These results clearly show that the symmetric setting can be more efficient than the asymmetric one, as expected. However, the symmetric PD can only be made usable if there also exists a reasonably efficient precomputation phase. In conclusion, the protocol set for the symmetric setup is a reasonable focus for future developments.

For simple comparison, in traditional SHAREMIND three miners PD multiplication of vectors of length 10000 took less than 100 milliseconds and was close to that also for shorter input lengths of 32-bit secrets [12]. Our asymmetric protocol set is significantly slower than that, but actually our symmetric protocol set can show similar speeds for 65 or 33-bit moduli. The main difference here is of course that [12] does not do precomputations. Covertly secure SPDZ [23] for two-parties reports doing 64-bit multiplications of input length 10000 in about 76 milliseconds for one thread and vectorised inputs. Our symmetric protocol set is currently slightly slower than that, but seems to be a good step from the asymmetric version.

7.3.2 Precomputation protocols

This section analyses the behaviour of our precomputation protocols. Table 7.5 gives the results of the time requirements of the precomputation of the asymmetric protection domain. The precomputation phase of the asymmetric protocol set is clearly less efficient than the online phase. In addition, measured results also indicate that the zero-knowledge proofs are the most expensive part of these protocols as also noted in Section 4.6. The proofs take approximately $\frac{4}{5}$ of time in the singles protocol and $\frac{3}{4}$ of total time in the triples protocol. We need approximately 1.6 seconds for one 2048-bit triple, whereas SPDZ [23] can prepare one 128-bit triple in 0.4 seconds.

Protocol B-Triples is used exactly as given in its protocol description in Algorithm 15 as packing smaller data types was native to this algorithm. The ShareConv-Triples from Algorithm 17 is benchmarked using the packing idea based on the Chinese remainder theorem. We only consider packings where all packed moduli are of equal bit length for simpler exposition and comparison. We chose 65-bit and 33-bit moduli as they are

Length	Singles with ZK	Triples with ZK	Singles	Triples
1	315	1852	42	529
10	2335	15 786	402	4699
100	22 492	154 853	4014	46 487
1000	226 923	1 544 571	40 257	464 853
10000	2 233 351	15 464 414	402 658	4 678 799

Table 7.5: Time requirements of asymmetric precomputation protocols in SHAREMIND (milliseconds)

sufficient to keep traditional 32-bit or 64-bit integers in them.

The CRT packing enables us to pack 15 elements of length 65-bits and 31 elements of length 33-bits into one ciphertext for 2048-bit modulus. This also explains the phenomena in Table 7.6 that lengths 1 and 10 take the same time for Algorithm 17—in both cases they are packed into one ciphertext and the main algorithm has the same workload. Difference between packing efficiency results in the approximately double difference between efficiency of 33-bit and 65-bit versions of these algorithms. Theoretical analysis in Section 5.4 showed that ShareConv-Triples is the most efficient of our proposals and the measurements clearly illustrate this. ShareConv-Triples can prepare about 186 packed 65-bit triples in a second, which is approximately 12 triple generation operations. In comparison, this means that ShareConv-Triples can prepare a semi-honestly secure 65-bit triple in 0.005 seconds, and SPDZ can prepare an actively secure 64-bit triple in 0.027 seconds [23].

	B-Triples		ShareConv-Triples	
Length	33-bit	65-bit	33-bit	65-bit
1	63	64	152	155
10	287	311	153	153
100	2617	2767	398	661
1000	25 686	27 199	2789	5458
10000	256 775	270 903	26 948	53 674

Table 7.6: Time requirements of Beaver triple protocols with packing in SHAREMIND (milliseconds)

For linear packing in B-Triples, we use a security constant $\sigma = 112$, which enabled us to pack 11 elements of 33-bits and 8 elements of length 65-bit into 2048-bit of plaintext space. Both this packing inefficiency and considerably higher requirements on the network made this less efficient than ShareConv-Triples. These packing counts also explain the relatively small difference in runningtimes for 33 and 65-bit cases. For both of these moduli, $\mathcal{CP}_1$ has to encrypt all *length* elements and the gain of packing only comes from a shorter result it gets back from $\mathcal{CP}_2$ which also lessens the amount of decryptions. Hence, the effect the packing has on the overall performance is substantially smaller than for packing with CRT, but the latter gain most from reducing the amount of necessary encryption and decryption functions.

In conclusion, it seems realistic to combine one of our Beaver triple protocols with CRT packing and share conversion to use it as full precomputation in the symmetric setting. The main open issue is defining efficient general share conversion that applies to additive shares and protection mechanisms.

Chapter 8

Conclusions

Secure multi-party computation is a general solution for privacy preserving data processing tasks. This thesis explores the subcase of SMC for two computing parties with the additional benefit that the parties can detect faults in the computation results. The main tools used to achieve this are an additively homomorphic cryptosystem, additive secret sharing and message authentication codes. We introduced a popular computation model that divides work to preprocessing and online phase. The latter is used to prepare some randomness that helps to speed up computations in the online phase, that performs all desired computations.

The goal of this thesis is to propose and implement new protocols for secure two-party computation for both online and precomputation phase. We concentrate mostly on common operations as sharing and publishing secret data as well as addition and multiplication. The latter is commonly implemented using Beaver triples, that are prepared in the offline phase. One of the important goals of our protocol sets is to define efficient generation of Beaver triples using an additively homomorphic cryptosystem.

The main result of this thesis is the introduction of three different flavours of setup for secure two-party computation, including asymmetric, symmetric and shared key setup. Their theoretical differences are stressed by the exact initialisation and implementation of the first two. For our initialisation, the symmetric setup is both more efficient and more flexible than the asymmetric setting. The shared key setup is presumably more efficient than the symmetric one, but adds additional complexity to verify the correctness of both computing parties.

The main goal of the Beaver triple generation protocols is to maximise the total bit length of the triples we can obtain from one multiplication using the Paillier cryptosystem. The main difficulties are coming up with a good way to pack smaller elements into the plaintext space of the Paillier cryptosystem and modifying the multiplication with the Paillier cryptosystem to give correct results for other moduli than the Paillier modulus. Two possibilities to pack smaller values into the plaintext space include linear packing and packing using the Chinese remainder theorem. The former is useful because it proposes no limits to the packed types, but the latter can be more efficient. We can also correct the results of the Paillier multiplication by analysing the potential outcomes of the protocol and collaboratively deciding which of those happened.

Current results show that actively secure multi-party computation is significantly slower than passively secure versions. However, our results indicate that fully implemented symmetric protocol set could be close to the performance of the SPDZ framework that is the current leader in actively secure multi-party computation frameworks. In addition, achieving security against malicious adversaries can be very important for

data mining tasks that have important economical or societal outcomes. Therefore, in many cases the extra time consumption is a reasonable trade-off for the additional layer of security.

Future work should extend the symmetric setup to include a full precomputation phase and add new operations to both introduced protocol sets. In addition, an implementation of the shared key setup using precomputation with Paillier cryposystem would provide an interesting comparison to the existing asymmetric and symmetric setups. Furthermore, the protocols for collecting inputs or returning outputs should be implemented to allow us to use these protection domains in real world applications. Likewise, it would be important to fully specify the universally composability of each protocol as well as define protocols for setting up the necessary keys of the protection domains.

Kahe osapoolega turvaline ühisarvutus: efektiivne Beaveri kolmikute genereerimine

Magistritöö

Pille Pullonen

Resümee

Turvaline ühisarvutus võimaldab salajaste sisenditega funktsioone väärtustada ning seeläbi lahendada turvaliselt mitmeid andmetöötlusülesandeid. Passiivselt turvaline ühisarvutus kindlustab, et kui kõik osapooled järgivad protokolli, siis jäävad sisendid salajaseks ning väljundid on õiged. Aktiivne turvamudel tagab privaatsuse ka siis, kui osapooled ei käitu ausalt ning võimaldab kontrollida saadud tulemuste korrektsust.

Käesolev töö uurib turvaliste ühisarvutuste erijuhtu, kus on kaks arvutavat osapoolt. Neile lisaks võib olla ka kolmandaid osapooli, kes annavad arvutusele sisendeid või soovivad saada tulemusi. Töö peamiseks eesmärgiks on kirjeldada aktiivses mudelis turvalisi kahe osapoolega protokollistike ning implementeerida need turvalise ühisarvutuse raamistikus Sharemind. Meie protokollid on jagatud kahte osasse: ettearvutamine ning tööfaas. Efektiivse ettearvutamise saavutamiseks vaatleme eraldi, kuidas genereerida Beaveri kolmikuid, mis võimaldavad tööfaasis teha kiiret korrutamist.

Kahe osapoolega ühisarvutuse ülesseadmiseks on vähemalt kolm erinevat võimalust: asümmeetriline, sümmeetriline ja jagatud konfiguratsioon. Käesolev töö keskendub kahele esimesele ning defineerib kummagi jaoks konkreetse protokollistiku näite. Kolmas on olemas meie tööd oluliselt mõjutanud SPDZ protokollistikus. Meie põhiline tööriist aktiivses mudelis turvalisuse saavutamiseks on sõnumiautentimiskood, mille abil kontrollitakse salastatud väärtuste korrektsust. Ebasümmeetrilises protokollistikus kasutame lisaks ka kinnistusskeeme ja nullteadmustõestusi. Mõlemad protokollistikud põhinevad aditiivsel ühissalastusel. Nii meie teoreetiliste arutluste kui implementatsiooni järgi on sümmeetriline protokollistik efektiivsem ning paindlikum kui ebasümmeetriline. Eelkõige on sümmeetriline praktilisem, sest võimaldab vähese vaevaga defineerida erineva suurusega andmetüüpe.

Ettearvutamise osas keskendusime eelkõige Beaveri kolmikute ehk juhusliku väärtusega multiplikatiivsete kolmikute (a, b, c) genereerimisele, kusjuures a, b on juhuslikud, ning $c = a \cdot b$. Kasutame selleks aditiivselt homomorfset Paillier' krüptosüsteemi ning klassikalist algoritmi aditiivselt jagatud andmete korrutamiseks Paillier' krüptosüsteemi kasutades. Peamiseks väljakutseks on selle algoritmi kohandamine erinevatele andmetüüpidele sõltumata krüptosüsteemi jaoks defineeritud moodulist. Eelkõige vaatame, kuidas garanteerida, et korrutamisprotokoll annaks sõltumata moodulist korrektseid tulemusi. Selgub, et võimalikud tekkivad vead on hästi defineeritud ning arvutavad osapooled saavad turvaliselt kontrollida, kas viga esines või mitte.

Efektiivsuse tõstmiseks analüüsime ka erinevaid viise, kuidas väiksemaid andmetüüpe Paillier' avateksti sisse pakkida nii, et lõpptulemusena saame iga pakitud elemendi jaoks korrektse kolmiku. Elemente saab pakkida nii lineaarselt kui ka Hiina jäägiteoreemi kasutades. Meie tulemuste kohaselt on viimane neist pakkimise mõttes efektiivsem, kuid seab lisapiiranguid pakitud elementide moodulitele. Praktikas tähendab see, et Hiina jäägiteoreemi järgi pakkimisele lisaks võime me vajada ka algoritme jagatud andmete mooduli vahetamiseks.

Realiseerisime nii asümmeetrilise kui sümmeetrilise protokollistiku tööfaasi ja asümmeetrilise protokollistiku ettearvutamise faasi. Lisaks realiseerisime ühe lineaarse pakkimisega ning ühe ühe Hiina jäägiteoreemil põhineva pakkimisega Beaveri kolmikute genereerimise protokolli. Katsed näitavad, et aktiivselt turvalise sümmeetrilise protokollistiku tööfaas on rohkem kui kaks korda ajamahukam kui traditsiooniline kolme osapoolega passiivselt turvaline SHAREMINDi protokollistik. Samas on jõudluse vahe piisavalt väike selleks, et sümmeetriline protokollistik oleks praktikas kasutatav. Lisaks võivad tugevamated turvagarantiid paljude kriitilise tähtsusega andmetöötlusülesannete lahendamisel kaaluda üles jõudluse puudujäägid.

Ettearvutamise osas on selgelt näha, et asümmeetriline protokollistik jääb oluliselt alla SPDZ protokollistiku täishomomorfsel krüposüsteemil põhinevale ettearvutamisele. Samas on meie Hiina jäägiteoreemil põhinev pakkimismeetod koos sobiva kolmikute genereerimise meetodiga piisavalt efektiivne, et oleks võimalik selle alusel defineerida ettearvutusfaas sümmeetrilisele protokollistikule.

Bibliography

[1] Aiello, W., Ishai, Y., and Reingold, O. Priced oblivious transfer: How to sell digital goods. In *Proceedings of the International Conference on the Theory and Application of Cryptographic Techniques: Advances in Cryptology* (London, UK, UK, 2001), EUROCRYPT '01, Springer-Verlag, pp. 119–135.

[2] Barak, B., Canetti, R., Nielsen, J. B., and Pass, R. Universally composable protocols with relaxed set-up assumptions. In *Proceedings of the 45th Annual IEEE Symposium on Foundations of Computer Science* (Washington, DC, USA, 2004), FOCS '04, IEEE Computer Society, pp. 186–195.

[3] Barker, E., Barker, W., Burr, W., Polk, W., and Smid, M. Recommendation for key management – part 1: General (revision 3). Tech. rep., National Institute of Standards and Technology, 2012. NIST Special Publication 800-57.

[4] Beaver, D. Efficient multiparty protocols using circuit randomization. In *Proceedings of the 11th Annual International Cryptology Conference. CRYPTO '91* (1991), J. Feigenbaum, Ed., vol. 576 of *Lecture Notes in Computer Science*, Springer, pp. 420–432.

[5] Beaver, D., Micali, S., and Rogaway, P. The round complexity of secure protocols. In *Proceedings of the twenty-second annual ACM symposium on Theory of computing* (New York, NY, USA, 1990), STOC '90, ACM, pp. 503–513.

[6] Ben-Or, M., Goldwasser, S., and Wigderson, A. Completeness theorems for non-cryptographic fault-tolerant distributed computation. In *Proceedings of the twentieth annual ACM symposium on Theory of computing* (New York, NY, USA, 1988), STOC '88, ACM, pp. 1–10.

[7] Bendlin, R., Damgård, I., Orlandi, C., and Zakarias, S. Semi-homomorphic encryption and multiparty computation. In *Proceedings of the 30th Annual international conference on Theory and applications of cryptographic techniques: advances in cryptology* (Berlin, Heidelberg, 2011), EUROCRYPT'11, Springer-Verlag, pp. 169–188.

[8] Blakley, G. R. Safeguarding cryptographic keys. In *Proceedings of the 1979 AFIPS National Computer Conference* (1979), vol. 48, pp. 313–317.

[9] Bogdanov, D. *Sharemind: programmable secure computations with practical applications*. PhD thesis, University of Tartu, 2013. `http://hdl.handle.net/10062/29041`.

[10] Bogdanov, D., Laud, P., and Randmets, J. Domain-polymorphic programming of privacy-preserving applications.

[11] Bogdanov, D., Laur, S., and Willemson, J. Sharemind: A framework for fast privacy-preserving computations. In *Proceedings of the 13th European Symposium on Research in Computer Security - ESORICS'08* (2008), S. Jajodia and J. Lopez, Eds., vol. 5283 of *Lecture Notes in Computer Science*, Springer Berlin / Heidelberg, pp. 192–206.

[12] Bogdanov, D., Niitsoo, M., Toft, T., and Willemson, J. High-performance secure multi-party computation for data mining applications. *Int. J. Inf. Sec. 11*, 6 (2012), 403–418.

[13] Boost - C++ libraries. `http://www.boost.org/`. Last accessed 2013-04-02.

[14] Brakerski, Z., and Vaikuntanathan, V. Fully homomorphic encryption from ring-lwe and security for key dependent messages. In *Proceedings of the 31st annual conference on Advances in cryptology* (Berlin, Heidelberg, 2011), CRYPTO'11, Springer-Verlag, pp. 505–524.

[15] Brier, E., and Joye, M. Weierstraß elliptic curves and side-channel attacks. In *Public Key Cryptography - PKC 2003, 6th International Workshop on Theory and Practice in Public Key Cryptography* (2002), vol. 2274 of *Lecture Notes in Computer Science*, Springer, pp. 335–345.

[16] Canetti, R. Universally composable security: a new paradigm for cryptographic protocols. In *Foundations of Computer Science, 2001. Proceedings. 42nd IEEE Symposium on* (oct. 2001), pp. 136 – 145.

[17] Canetti, R., Kushilevitz, E., and Lindell, Y. On the limitations of universally composable two-party computation without set-up assumptions. In *Proceedings of the 22nd international conference on Theory and applications of cryptographic techniques* (Berlin, Heidelberg, 2003), EUROCRYPT'03, Springer-Verlag, pp. 68–86.

[18] Canetti, R., Lindell, Y., Ostrovsky, R., and Sahai, A. Universally composable two-party and multi-party secure computation. In *Proceedings of the thiry-fourth annual ACM symposium on Theory of computing* (New York, NY, USA, 2002), STOC '02, ACM, pp. 494–503.

[19] Chaum, D., Crépeau, C., and Damgård, I. Multiparty unconditionally secure protocols. In *Proceedings of the twentieth annual ACM symposium on Theory of computing* (New York, NY, USA, 1988), STOC '88, ACM, pp. 11–19.

[20] Dai, W. Crypto++ library. `http://www.cryptopp.com/`. Last accessed 2013-04-02.

[21] Damgård, I., Geisler, M., Krøigaard, M., and Nielsen, J. B. Asynchronous multiparty computation: Theory and implementation. In *Proceedings of the 12th International Conference on Practice and Theory in Public Key Cryptography: PKC '09* (Berlin, Heidelberg, 2009), Irvine, Springer-Verlag, pp. 160–179.

[22] Damgård, I., and Jurik, M. A generalisation, a simplification and some applications of Paillier's probabilistic public-key system. In *Proceedings of the 4th International Workshop on Practice and Theory in Public Key Cryptography: Public Key Cryptography* (London, UK, UK, 2001), PKC '01, Springer-Verlag, pp. 119–136.

[23] Damgård, I., Keller, M., Larraia, E., Pastro, V., Scholl, P., and Smart, N. P. Practical covertly secure MPC for dishonest majority – or: Breaking the SPDZ limits. Cryptology ePrint Archive, Report 2012/642, 2012. `http://eprint.iacr.org/`.

[24] Damgård, I., and Nielsen, J. B. Scalable and unconditionally secure multiparty computation. In *Proceedings of the 27th annual international cryptology conference on Advances in cryptology* (Berlin, Heidelberg, 2007), CRYPTO'07, Springer-Verlag, pp. 572–590.

[25] Damgård, I., and Orlandi, C. Multiparty computation for dishonest majority: from passive to active security at low cost. In *Proceedings of the 30th annual conference on Advances in cryptology* (Berlin, Heidelberg, 2010), CRYPTO'10, Springer-Verlag, pp. 558–576.

[26] Damgård, I., Pastro, V., Smart, N., and Zakarias, S. Multiparty computation from somewhat homomorphic encryption. Cryptology ePrint Archive, Report 2011/535, 2011. `http://eprint.iacr.org/`.

[27] Dierks, T., and Rescorla, E. RFC 5246 - The transport layer security (TLS) protocol version 1.2. `http://tools.ietf.org/html/rfc5246`, August 2008. Last accessed 2013-05-14.

[28] Diffie, W., and Hellman, M. E. New directions in cryptography. *IEEE Transactions on Information Theory 22*, 6 (1976), 644–654.

[29] El Gamal, T. A public key cryptosystem and a signature scheme based on discrete logarithms. In *Proceedings of CRYPTO'84 on Advances in cryptology* (New York, NY, USA, 1985), Springer-Verlag New York, Inc., pp. 10–18.

[30] Feldman, P. A practical scheme for non-interactive verifiable secret sharing. In *Proceedings of the 28th Annual Symposium on Foundations of Computer Science* (Washington, DC, USA, 1987), SFCS '87, IEEE Computer Society, pp. 427–438.

[31] Fouque, P.-A., Poupard, G., and Stern, J. Sharing decryption in the context of voting or lotteries. In *Proceedings of the 4th International Conference on Financial Cryptography* (London, UK, UK, 2001), FC '00, Springer-Verlag, pp. 90–104.

[32] Gentry, C. Fully homomorphic encryption using ideal lattices. In *Proceedings of the 41st annual ACM symposium on Theory of computing* (New York, NY, USA, 2009), STOC '09, ACM, pp. 169–178.

[33] Giry, D. BlueKrypt - cryptographic key length recommendation. `http://www.keylength.com`. Last accessed 2013-04-02.

[34] Granlund, T. GMP: The GNU multiple precision arithmetic library. http://gmplib.org/. Last accessed 2013-04-02.

[35] Henecka, W., Kögl, S., Sadeghi, A.-R., Schneider, T., and Wehrenberg, I. TASTY: tool for automating secure two-party computations. In *Proceedings of the 17th ACM conference on Computer and communications security* (New York, NY, USA, 2010), CCS '10, ACM, pp. 451–462.

[36] Hirt, M., and Maurer, U. Complete characterization of adversaries tolerable in secure multi-party computation (extended abstract). In *Proceedings of the sixteenth annual ACM symposium on Principles of distributed computing* (New York, NY, USA, 1997), PODC '97, ACM, pp. 25–34.

[37] Hirt, M., and Maurer, U. Player simulation and general adversary structures in perfect multiparty computation. *JOURNAL OF CRYPTOLOGY 13* (2000), 31–60.

[38] Kushilevitz, E., and Ostrovsky, R. Replication is not needed: single database, computationally-private information retrieval. In *Proceedings of the 38th Annual Symposium on Foundations of Computer Science* (Washington, DC, USA, 1997), FOCS '97, IEEE Computer Society, pp. 364–.

[39] Laur, S., and Lipmaa, H. A new protocol for conditional disclosure of secrets and its applications. *Applied Cryptography and Network Security* (2007), 1–19.

[40] Laur, S., and Zhang, B. Lightweight zero-knowledge proofs for cryptocomputing protocols. Cryptology ePrint Archive, Report 2013/064, 2013. http://eprint.iacr.org/.

[41] Lipmaa, H. First cpir protocol with data-dependent computation. In *Proceedings of the 12th international conference on Information security and cryptology* (Berlin, Heidelberg, 2010), ICISC'09, Springer-Verlag, pp. 193–210.

[42] Malkhi, D., Nisan, N., Pinkas, B., and Sella, Y. Fairplay - a secure two-party computation system. In *Proceedings of the 13th conference on USENIX Security Symposium - Volume 13* (Berkeley, CA, USA, 2004), SSYM'04, USENIX Association, pp. 20–20.

[43] Recommended elliptic curves for federal government use. Tech. rep., National Institute of Standards and Technology, 1999. http://csrc.nist.gov/groups/ST/toolkit/documents/dss/NISTReCur.pdf.

[44] Nielsen, J. B., Nordholt, P. S., Orlandi, C., and Burra, S. S. A new approach to practical active-secure two-party computation. In *Advances in Cryptology – CRYPTO 2012*, R. Safavi-Naini and R. Canetti, Eds., vol. 7417 of *Lecture Notes in Computer Science*. Springer Berlin Heidelberg, 2012, pp. 681–700.

[45] Paillier, P. Public-key cryptosystems based on composite degree residuosity classes. In *Proceedings of the 17th international conference on Theory and application of cryptographic techniques* (Berlin, Heidelberg, 1999), EUROCRYPT'99, Springer-Verlag, pp. 223–238.

[46] Pedersen, T. P. Non-interactive and information-theoretic secure verifiable secret sharing. In *Proceedings of the 11th Annual International Cryptology Conference on Advances in Cryptology* (London, UK, UK, 1992), CRYPTO '91, Springer-Verlag, pp. 129–140.

[47] Pfitzmann, B., and Waidner, M. A model for asynchronous reactive systems and its application to secure message transmission. In *Proceedings of the 2001 IEEE Symposium on Security and Privacy* (Washington, DC, USA, 2001), SP '01, IEEE Computer Society, pp. 184–.

[48] Pullonen, P., Bogdanov, D., and Schneider, T. The design and implementation of a two-party protocol suite for Sharemind 3. Tech. rep., Cybernetica AS Institute of Information Security, 2012. `http://research.cyber.ee`.

[49] Rabin, T., and Ben-Or, M. Verifiable secret sharing and multiparty protocols with honest majority. In *Proceedings of the twenty-first annual ACM symposium on Theory of computing* (New York, NY, USA, 1989), STOC '89, ACM, pp. 73–85.

[50] RakNet - multiplayer game network engine. `http://www.jenkinssoftware.com/`. Last accessed 2013-04-02.

[51] Schnorr, C.-P. Efficient identification and signatures for smart cards. In *Advances in Cryptology - EUROCRYPT '89*, J.-J. Quisquater and J. Vandewalle, Eds., vol. 434 of *Lecture Notes in Computer Science*. Springer Berlin / Heidelberg, 1990, pp. 688–689.

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

[53] Sharemind. `http://sharemind.cyber.ee`. Last accessed 2013-04-19.

[54] Smart, N., and Vercauteren, F. Fully homomorphic SIMD operations. *Designs, Codes and Cryptography* (2012), 1–25.

[55] Tate, S. R., and Xu, K. On garbled circuits and constant round secure function evaluation. Tech. rep., University of North Texas, 2003.

[56] Yao, A. C. Protocols for secure computations. In *Proceedings of the 23rd Annual Symposium on Foundations of Computer Science* (Washington, DC, USA, 1982), SFCS '82, IEEE Computer Society, pp. 160–164.

Non-exclusive licence to reproduce thesis and make thesis public

I, Pille Pullonen, (date of birth: 06.01.1989),

1. herewith grant the University of Tartu a free permit (non-exclusive licence) to:

   1.1. reproduce, for the purpose of preservation and making available to the public, including for addition to the DSpace digital archives until expiry of the term of validity of the copyright, and

   1.2. make available to the public via the web environment of the University of Tartu, including via the DSpace digital archives until expiry of the term of validity of the copyright,

   Actively secure two-party computation: Efficient Beaver triple generation

   supervised by Sven Laur, Tuomas Aura, and Dan Bogdanov.

2. I am aware of the fact that the author retains these rights.

3. I certify that granting the non-exclusive licence does not infringe the intellectual property rights or rights arising from the Personal Data Protection Act.

Tartu, 20.05.2013

Appendix G

A Practical Analysis of Oblivious Sorting Algorithms for Secure Multi-party Computation

The paper "A Practical Analysis of Oblivious Sorting Algorithms for Secure Multi-party Computation" [11] follows.

A Practical Analysis of Oblivious Sorting Algorithms for Secure Multi-party Computation

Anonymous submission to ASIACCS 2014

ABSTRACT

Cryptographic secure computing methods like secure multi-party computation, circuit garbling and homomorphic encryption are becoming practical enough to be usable in applications. Such applications need special data-independent sorting algorithms to preserve privacy. In this paper, we describe the design and implementation of four different oblivious sorting algorithms. We improve two earlier designs based on sorting networks and quicksort with the capability of sorting matrices. We also propose two new designs—a naive comparison-based sort with a low round count and an oblivious radix sort algorithm that does not require any private comparisons. We implement all algorithms and present a thorough complexity and performance analysis, including runtime, network use and memory use analysis.

Categories and Subject Descriptors

K.6.5 [**Management of Computing and Information Systems**]: Security and Protection; F.2.2 [**Analysis of Algorithms and Problem Complexity**]: Nonnumerical Algorithms and Problems—*Sorting and searching*

General Terms

Algorithms, Experimentation, Performance

Keywords

Privacy, algorithms, sorting, implementation, performance analysis, secure multi-party computation

1. INTRODUCTION

One of the main challenges in data analysis is getting the best data for solving the problem. However, the data are not always available, because their owners do not want to share information because of privacy concerns. We need a solution that allows controlled data mining that preserves the privacy of the data owners.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.
Copyright 20XX ACM X-XXXXX-XX-X/XX/XX ...$15.00.

Several solutions have been proposed for privacy-preserving data mining. Anonymization techniques like k-anonymity [22], l-diversity [18] and others [10] transform data by adding noise so that certain statistical properties remain, but individual records do not disclose private information. However, statistical de-anonymization is an efficient counter-measure against anonymization, especially if auxiliary information is present [19].

Differential privacy [8] is another algorithm transformation approach that adds noise to preserve privacy. It is an efficient method for building private data analysis tools when initialised with correct threshold parameters.

Cryptographic techniques like secure multi-party computation (SMC) have gained popularity as they have become more efficient. Initial cryptographic protocols were tailored for particular tasks like the scalar product [12], but programmable SMC has gained efficiency and has become as fast or even faster.

Some of the fastest SMC systems [4, 3, 7] are based on secret sharing [21]. Secret sharing is a cryptographic primitive for securely sharing confidential information among several parties. There exist SMC protocols that allow secret-shared data to be processed without reconstructing the original secrets. There are also encryption schemes that allow such processing [20, 11], but they are less efficient at this time.

Sorting is an important operation in privacy-preserving data analysis and data mining. In addition to its obvious use in ordering data, sorting is used for finding ranked elements (top-k, quantiles), performing group-level aggregations and implementing statistical tests.

1.1 Related work

Several oblivious sorting methods use oblivious shuffling [17]. In [13], Hamada *et al.* propose a generic blueprint for converting any comparison-based sorting algorithm into an oblivious sorting algorithm.

Sorting networks [16] are another suitable tool for designing oblivious sorting algorithms, as they have a fixed structure and are easy to implement on secure computation systems. Previously, oblivious sorting networks in SMC context are used in [15, 23].

In [24], Zhang proposes multiple constant-round sorting schemes for secret sharing schemes. These include counting sort, arrayless bead sort and sorting key-indexed data. These schemes assume a known range of inputs and the author proposes to use radix sort on top of these protocols to deal with larger ranges.

1.2 Our contribution

In this paper we propose two new sorting algorithms — a naive comparison-based sorting algorithm with a low round count that is suitable for sorting short vectors and an oblivious radix sorting algorithm that does not declassify comparison results and works in data-independent time. Additionally, our radix sorting algorithm does not require oblivious comparisons, making it compatible with more SMC systems. We give a construction for using these algorithms and previous work on sorting networks and comparison-based algorithms to sort matrices by values in one or more columns. Finally, we implement all of the mentioned sorting algorithms and give a thorough analysis of their running time, network communication complexity and memory consumption. Our benchmark results are measured on implementation developed for the SHAREMIND secure computation platform [1]. The algorithms are implemented using the SECREC programming language [2]. We implemented all the algorithms on the same platform to perform a fair comparison of the implementations.

2. REQUIREMENTS FOR OBLIVIOUS SORTING ALGORITHMS

Most oblivious sorting algorithms in this paper are not designed for a particular secure computation technology. However, we analyze the efficiency of the algorithms in an SMC environment based on secret sharing. We focus on solutions based on secret sharing, as they currently provide the fastest practical implementations. In this setting, n parties evaluate a function $f(x_1, \ldots, x_n) = (y_1, \ldots, y_n)$ so that party i will learn its input x_i, output y_i and nothing else.

Secret sharing [21] allows us to hide secret values by splitting them into random shares that are distributed among the parties. In additive secret sharing, a secret value x is split into n random addends $x_1, \ldots, x_n$ held by parties $\mathcal{P}_1, \ldots, \mathcal{P}_n$ so they add up to the original value (modulo some p):

$$x = x_1 + x_2 + \ldots + x_n \mod p.$$

A bitwise secret-sharing scheme works similarly, except instead of sum we use the bitwise exclusive or operation to calculate and reconstruct the shares:

$$x = x_1 \oplus x_2 \oplus \ldots \oplus x_n.$$

In this paper, we denote a secret-shared value x by $[\![x]\!]$.

Arguing about the security of secure multi-party computation protocols often includes demonstrating that an adversary cannot distinguish between the secure processing of actual inputs from the similar evaluation of random inputs. We suggest that the reader studies the work of Canetti on proving the security of SMC protocol suites [5]. It is important that the SMC protocols are *composable*, so they could be used to build algorithm implementations.

Algorithms must fulfill the following requirements to be *data-independent*. First, the intermediate and output values of the algorithm must not leak anything about the secret inputs. This prevents information leakage through observing the memory during execution. Second, for a fixed number of private inputs, the algorithm's execution time should not depend on the input values. This prevents information leakage through observations of the algorithm's running time.

Hiding the number of inputs is rarely required in practice. While this is possible, such techniques still require an upper bound for the number of inputs, resulting in a leak of information on the maximum number of records and a waste of computing resources on unused elements.

In this paper, we consider algorithms that are data-independent and algorithms with a small data dependence that allows for a greater efficiency in secure evaluation. We still consider these algorithms oblivious, if the leakage is well-defined so that the user is informed of the risks.

Algorithm 1: NaiveCompSort

Data: Input array $[\![\mathcal{D}]\!] \in \mathbb{Z}_{2^k}^n$
Result: Sorted array $[\![\mathcal{D}']\!]$
1 Let $[\![T]\!] = \mathsf{Shuffle}([\![T]\!])$
`// All comparisons here are done in parallel.`
2 **for** $i < j \in \{1, 2, \ldots, n\}$ **do**
3 | Let $[\![g_{i,j}]\!] = [\![\mathcal{D}_i]\!] \leq [\![\mathcal{D}_j]\!]$
4 **end**
5 Declassify the values $[\![g_{i,j}]\!]$ and sort $[\![\mathcal{D}]\!]$ according to them, obtaining $[\![\mathcal{D}']\!]$
6 **return** $[\![\mathcal{D}']\!]$

3. OBLIVIOUS SORTING TECHNIQUES

3.1 Constructions based on oblivious shuffling

Comparison-based sorting algorithms use the comparison operation to determine the correct sequence of elements of a given array. Such algorithms are inherently data-dependent, as the execution flow depends on the outcomes of the comparison operations. Hence, changes in the input data will affect the running time of the algorithm. A simple solution would be to evaluate all the branches of the sorting algorithm and obliviously select the correct output in the end, but this dramatically reduces efficiency.

Hamada *et al.* [13] propose a generic solution to *obliviously shuffle* the inputs before performing a comparison-based sort. Then, as we are comparing values in a randomly shuffled vector, any declassified (published) comparison results are also random. However, the pattern of comparisons in the algorithm can still leak information, such as the number of equal elements in the input vector.

Because many SMC implementations have highly efficient vector operations, vectorized naive protocols may sometimes be more efficient than protocols with a lower computational complexity and a lower degree of vectorization. In this paper, we propose a naive sorting protocol based on shuffle and vectorized comparisons called NaiveCompSort (Algorithm 1). In this algorithm, we first shuffle the input array and then compare every element with every other element in the array in one big vector operation. Finally, we rearrange the elements according to the declassified comparison results. This algorithm always works in the worst case time of $\mathcal{O}(n^2)$ and its runtime is, therefore, data-independent.

3.2 Constructions based on sorting networks

A sorting network is a structure that consists of several stages of compare-and-exchange (CompEx) functions. A CompEx function takes two inputs, compares them according to a given condition and exchanges them, if the comparison result is true. An example CompEx function for sorting two

Algorithm 2: Basic algorithm for sorting with a sorting network.

Data: Input array $\mathcal{D} \in \mathbb{Z}_{2^k}^n$ and a sorting network $\mathcal{N} \in \mathbb{L}^m$.
Result: Sorted output array $\mathcal{D} \in \mathbb{Z}_{2^k}^n$.

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

values in ascending order is defined as

$$\mathsf{CompEx}(x, y) = (\mathsf{Min}(x, y), \mathsf{Max}(x, y)). \quad (1)$$

When all CompEx functions in the stages 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, see [16].

The inputs of each CompEx function can be encoded with their indices in the input data array. Therefore, we will represent an m-stage sorting network as a tuple $\mathbb{L}^m$, consisting of stages in the form $\mathbb{L}_i = (\mathbb{N} \times \mathbb{N})^{\ell_i}$. Each stage $\mathbb{L}_i$ contains ℓ_i CompEx functions. For efficiency, we prefer sorting networks where no index appears more than once in each individual stage as this lets us vectorize the implementation.

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

As the structure of the sorting network is the same for all inputs, Algorithm 2 is trivially data-independent, given a data-independent implementation of the CompEx function. Such implementations are also easy to construct, following the minimum-maximum blueprint shown in Equation (1).

3.3 Constructions specific for bitwise secret sharing schemes

If data is secret-shared using a bitwise secret-sharing scheme, access to individual bits is cheap. This allows us to design a very efficient count/radix sorting algorithm. Counting sort [6, 9] is a sorting algorithm that can sort an array of integers in a small range by first constructing a frequency table and then rearranging items in the array according to this table. Algorithm 3 describes a counting sort algorithm for binary data.

Radix sort [14] sorts an array of integers by rearranging them based on counting sort results on digits in the same positions. Radix sort sorts data one digit position at a time, starting with the least significant digit. This works as the underlying counting sort is a stable sorting algorithm. Algorithm 4 shows the full protocol of oblivious radix sort that uses binary counting sort as a subroutine. The underlying counting sort is made data-independent by obliviously updating counters $[\![c_0]\!]$, $[\![c_1]\!]$ and the order vector $[\![ord]\!]$. Such a data-independent counting sort is sufficient to make our radix sort data-independent as well.

As our radix sort algorithm does not use comparison operations, it is not bound by the computational complexity lower bound of $\Omega(n \log n)$ for comparison-based sorting algorithms. Counting sort has a complexity of $\mathcal{O}(n)$ and radix sort on k-digit elements that uses counting sort as a subroutine, has a computational complexity of $\mathcal{O}(kn)$. However, the data-independent counting sort protocol also uses addition and multiplication operations which are expensive protocols on bitwise shared data. Therefore, after creating a vector with bits on a given position, we convert it to additively shared data and work in this domain. The output of the algorithm is still in a bitwise form.

Algorithm 3: Counting sort algorithm for binary arrays.

Data: Binary input array $\mathcal{D} \in \mathbb{Z}_2^n$.
Result: Array $\mathcal{D}' \in \mathbb{Z}_2^n$ with elements of $\mathcal{D}$ in increasing order.

```
n_0 ← n − sum(𝒟); // Count number of zeros.
c_0 ← 0; c_1 ← 0; // Keep counters for processed zeros and ones.
// Put each element in right position:
foreach i ∈ 1...n do
    if 𝒟_i == 0 then
        c_0 = c_0 + 1
        𝒟'_{c_0} = 𝒟_i
    else
        c_1 = c_1 + 1
        𝒟'_{n_0+c_1} = 𝒟_i
    end
end
return 𝒟'
```

4. OPTIMIZATION METHODS

4.1 Vectorization

Data parallelization (SIMD operations) allows us to reduce the number of communication rounds and optimize the running time of algorithms for SMC. We designed the NaiveCompSort especially with vectorization in mind. Also, the quicksort design of [13] is vectorized by performing all comparisons at each depth of the quicksort algorithm at once.

Sorting network evaluation in Algorithm 2 can be vectorized by evaluating all CompEx functions of a given stage together. As mentioned previously, this is possible because of the assumption on the uniqueness of indices we made while describing the structure of the sorting network.

Similarly, we vectorize all secure operations in our counting sort algorithm design. We could apply counting sort on chunks of 2 or more bits and reduce the number of rounds for radix sort. However, this requires substituting the cheap oblivious choice subprotocol for a more expensive comparison protocol.

4.2 Changing the share representation

Both comparison-based sorting algorithms and sorting networks rely on the comparison operation. Comparison is a bit-level operation and works faster on bitwise shared data. Therefore, we can convert additively shared inputs into bitwise shared form and run the intended algorithm on the converted shares. The results can be converted back to additively shared form at the end of the algorithm.

Converting additive shares to bitwise shares requires a bit extraction protocol. However, for algorithms that perform

Algorithm 4: Data-independent radix sort.

Data: Input array $[\![\mathcal{D}]\!] \in \mathbb{Z}_{2^k}^n$.
Result: Sorted array $[\![\mathcal{D}]\!] \in \mathbb{Z}_{2^k}^n$.
// Iterate over all digits starting with the least significant digit:
foreach $m \in 1 \dots k$ **do**
  // Construct a binary vector consisting of m-th digits.
  // Convert it to additively shared data.
  $[\![d]\!] \leftarrow \mathsf{ShareConv}(([\![\mathcal{D}_1]\!]_m, [\![\mathcal{D}_2]\!]_m, \dots, [\![\mathcal{D}_n]\!]_m))$
  $[\![n_0]\!] \leftarrow n - sum([\![d]\!])$; // Count number of zeros.
  $[\![c_0]\!] \leftarrow 0$; $[\![c_1]\!] \leftarrow 0$; // Keep counters for processed zeros and ones.
  $[\![ord]\!]$; // Keep n-element shared order vector.
  // Put each element in the right position:
  foreach $i \in 1 \dots n$ **do**
    $[\![c_0]\!] = [\![c_0]\!] + 1 - [\![d_i]\!]$
    $[\![c_1]\!] = [\![c_1]\!] + [\![d_i]\!]$
    // Obliviously update order vector:
    $[\![ord_i]\!] = (1 - [\![d_i]\!]) * [\![c_0]\!] + [\![d_i]\!] * ([\![n_0]\!] + [\![c_1]\!])$
  end
  $([\![\mathcal{D}]\!], [\![ord]\!]) \leftarrow \mathsf{Shuffle}([\![\mathcal{D}]\!], [\![ord]\!])$; // Shuffle two column database.
  $ord \leftarrow \mathsf{Declassify}([\![ord]\!])$
  Rearrange elements in $[\![\mathcal{D}]\!]$ according to ord.
end
return $[\![\mathcal{D}]\!]$

many comparisons after one another, the benefits of many fast comparisons outweigh one costly conversion.

4.3 Optimizations specific to sorting networks

In software implementations, the generation of sorting networks can take a significant amount of time. As the sorting network structure is data-independent, we can store the sorting network after generation to re-use it later.

If we shuffle the inputs before sorting, we can optimize the CompEx function implementations by declassifying comparison results and performing the exchanges non-obliviously. The running time of the resulting algorithm is data-independent because of the constant structure of the sorting network.

5. SORTING SECRET-SHARED MATRICES

5.1 Comparison-based sorting and radix sort

Comparison-based algorithms can be easily modified to support matrix sorting. Assume that our input data is in the form of a matrix $\mathcal{D}_{i,j}$ where $i = 1 \dots n$ and $j = 1 \dots m$. Let us also fix a column k by which we want to sort the rows.

First, we obliviously shuffle the rows in the whole matrix[1]. Next, we extract the k-th column from the matrix and pass it to the sorting algorithm of our choice together with an n-element index vector $(1, 2, \dots, n)$.

The sorting protocol now swaps elements in the data vector and the index vector together. After sorting these two vectors, we declassify the output index vector and use it as a permutation to rearrange rows in the matrix. Declassifying the index vector leaks information on how the elements were rearranged. However, as the input matrix was obliviously shuffled, this leaks no information on the original placement of rows in the initial matrix.

[1]Note that shuffling is already a part of comparison-based sorting protocols like quicksort and NaiveCompSort. However, this extra step has to be added for radix sort.

5.2 Sorting networks

Oblivious sorting based on sorting networks does not declassify any values that could help us reorder other columns. Instead, we redefine the CompEx operation to work on the rows of the matrix. We need a CompEx function that compares and exchanges two input arrays $\mathcal{A}$ and $\mathcal{B}$ according to the comparison result from column k. A suitable algorithm for this purpose is presented in Appendix A. The algorithm is trivial to optimize using vector operations.

6. EXPERIMENTAL EVALUATION OF OBLIVIOUS SORTING ALGORITHMS

6.1 Overview of algorithm implementations

Table 1 gives an overview of sorting algorithms implemented for this paper. We implemented all algorithms in the SecreC programming language [2] to run on the Sharemind secure multi-party computation system [1].

Our quicksort implementation is based on the work in [13] and personal communication with its authors. We implemented the algorithm as similarly as possible to achieve a fair comparison. The naive comparison sort and radix sort algorithms are implemented straightforwardly from Algorithms 1, 3 and 4.

The sorting network implementation consists of two parts. We implemented sorting network generation using Florian Forster's `libsortnetwork` library[2]. Sharemind generates and caches sorting networks and encodes them for delivery to SecreC programs. The evaluation of the sorting network is implemented in SecreC. Our implementation generates Batcher's bitonic mergesort networks, as they were the fastest to generate. See Appendix B for details.

All algorithms are vectorized to optimize the running time using techniques described in Section 4. While this may lead to extensive memory usage, it allows us to demonstrate the performance of the algorithms. The memory usage of nearly all the algorithms can be reduced by breaking large vector operations to smaller pieces.

We implemented vector sorting and matrix sorting for all algorithms, following the techniques described in Section 5. The performance results for matrix sorting are given in Appendix C.

6.2 Experimental setup

The experiments were conducted using a Sharemind installation consisting of three servers connected by a 1 Gbps local area network. Each server was equipped with 48 GB of memory and a 12-core 3 GHz Intel processor.

We measured the running time and network usage using the profiling mechanism built into Sharemind. We marked code sections and Sharemind measured and logged the running time and network usage of each section invocation. We sampled the memory use reported for the Sharemind server

[2]Available from `http://verplant.org/libsortnetwork/` in December, 2013.

Algorithm	Data-independence and leakage	References
Quicksort	Comparison results are declassified. Running time is data-dependent. May leak the number of equal elements.	[13]
Naive comparison sort (Algorithm 1)	Comparison results are declassified. Running time is data-independent. Leaks the number of equal elements.	this paper
Sorting network sort (Algorithm 2)	Fully data-independent	[23, 15]
Radix sort (Algorithms 3 and 4)	Reordering decisions are declassified. Running time is data-independent. Does not leak the number of equal elements	this paper

Table 1: An overview of oblivious sorting algorithms implemented for this paper.

process every one second and aligned this data with the running time data to find the peak memory usage for each experiment.

We ran each algorithm with bitwise-shared 64-bit unsigned integer data. We used worst-case data (all equal values) for each algorithm and additionally, used random data for the quicksort algorithm similarly to the experiments of [13].

6.3 Results and analysis

6.3.1 *Theoretical complexities.*

Before presenting the benchmark results, we give the secure computation complexities of all the implemented functions. We express the complexities in the number of subprotocol invocations. For example, $m\mathsf{Protocol}(n)$ means that the SMC protocol $\mathsf{Protocol}$ is invoked m times with n parallel elements for each invocation. Where necessary, oblivious addition operations are done on additively shared values. Hence, they do not require any network communication and are omitted from Table 2.

6.3.2 *Timing profiles of individual sorting algorithms.*

Figure 1 shows the breakdown of the running times of oblivious sorting algorithms. We see that most of the time in naive sorting and quicksort is spent on comparisons. The time taken for sorting network evaluation begins with mostly comparisons and oblivious choice, but their importance is reduced as the time needed for generating the network increases. This is a strong motivator for the precomputing and caching of sorting networks. Radix sort has the most interesting profile, as it does not use comparisons. Instead, its most expensive part is oblivious choice.

6.3.3 *Comparison of different sorting algorithms.*

We now present comparisons of all the algorithms. Note that the axes of the comparison figures are on a logarithmic scale. Figure 2 shows the comparison of the running time. Naive comparison sort is very fast on small inputs, but its high complexity makes it infeasible for larger inputs. Quicksort is the fastest of all algorithms, but only in the random data vector experiment. When we run quicksort on data vectors with all equal values, it performs significantly slower. This can be explained by the need to actually go through all the subsets of the data. On randomized data, our implementation of quicksort achieves the same performance as reported in [13].

Radix sorting is not the most efficient on small inputs, but its use of cheap secure operations ensures that its running time does not grow as quickly as that of the other algorithms. Sorting networks are efficient early, but the time needed to generate the network starts to grow significantly as the data size grows. If the sorting network structure is cached, sorting network evaluation is almost as fast as radix sorting.

We see the network usage measurements in Figure 3. Naive sorting and quicksort on worst-case data require a lot of network communication. The other algorithms form a more efficient group, with quicksort on random data requiring the least communication and radix sort taking the second place.

Finally, Figure 4 shows the memory usage. The memory usage of the naive implementation grows squared in the size of data, making it infeasible for large inputs. The sorting network algorithm uses significant amounts of memory during the generation of the sorting network and reduced amounts after that. The memory requirements of oblivious radix and quicksort are low in comparison.

7. CONCLUSION

We describe four designs for oblivious versions of known sorting algorithms—naive comparison-based sort, quicksort, radix sort and sorting network based sort. The first three perform some declassifications to improve efficiency while the use of sorting networks results in a fully data-independent algorithm.

Our performance analysis shows that even though naive comparison-based sorting is fast on small inputs, its $\mathcal{O}(n^2)$ complexity makes it slow for practical input sizes. While the oblivious version of quicksort is very efficient on random data, it performs poorly when the input contains many equal elements. Its increased running time on such inputs also leaks the number of equal elements.

Oblivious sorting networks are a great choice when we can precompute or cache the network structure. In that case, the algorithm provides perfect privacy with a reasonable performance.

Our novel oblivious radix sorting algorithm leaks less information than constructions based on shuffling and declassified comparison results. As input sizes grow, its performance comes closer to that of quicksort on random data, because it does not need to use the relatively expensive comparison operations.

Algorithm	Secure operation complexity
Quicksort (average)	$\mathsf{Shuffle}(n) + \mathcal{O}(\log n)\mathsf{Comp}(\mathcal{O}(n)) + \mathcal{O}(\log n)\mathsf{Declassify}(\mathcal{O}(n))$
Naive comparison sort	$\mathsf{Shuffle}(n) + \mathsf{Comp}(n(n-1)/2) + \mathsf{Declassify}(n(n-1)/2)$
Sorting network	$\sum_{i=1}^{m} \mathsf{Comp}(\ell_i) + \mathsf{Mult}(4\ell_i)$
Radix sort	$k \cdot (\mathsf{ShareConv}(n) + \mathsf{Mult}(n) + \mathsf{Shuffle}(n, 2) + \mathsf{Declassify}(n))$

Table 2: Secure operation complexity of oblivious sorting algorithms. n is the number of elements to sort and k is number of digits, where applicable. For sorting networks, m is the number of stages in the network and ℓ_i is the number of CompEx operations on the i-th stage.

8. REFERENCES

[1] D. Bogdanov. *Sharemind: programmable secure computations with practical applications*. PhD thesis, University of Tartu, 2013.

[2] D. Bogdanov, P. Laud, and J. Randmets. Domain-Polymorphic Programming of Privacy-Preserving Applications. Cryptology ePrint Archive, Report 2013/371, 2013. `http://eprint.iacr.org/`.

[3] D. Bogdanov, M. Niitsoo, T. Toft, and J. Willemson. High-performance secure multi-party computation for data mining applications. *International Journal of Information Security*, 11(6):403–418, 2012.

[4] M. Burkhart, M. Strasser, D. Many, and X. Dimitropoulos. SEPIA: privacy-preserving aggregation of multi-domain network events and statistics. In *Proc. of USENIX conference on Security*, pages 15–15. USENIX Association, 2010.

[5] R. Canetti. Universally composable security: A new paradigm for cryptographic protocols. In *Proc. of FOCS'01*, pages 136–145. IEEE Computer Society, 2001.

[6] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to Algorithms*, chapter 8.2 Counting Sort, pages 168–170. MIT Press and McGraw-Hill, 2nd edition, 2001.

[7] I. Damgård, V. Pastro, N. P. Smart, and S. Zakarias. Multiparty computation from somewhat homomorphic encryption. In *Proc. of CRYPTO'12*, volume 7417 of *LNCS*, pages 643–662. Springer, 2012.

[8] C. Dwork. Differential privacy. In *Proceedings of the 33rd International Colloquium on Automata, Languages and Programming. ICALP'06*, volume 4052 of *LNCS*, pages 1–12. Springer, 2006.

[9] J. Edmonds. *How to Think about Algorithms*, chapter 5.2 Counting Sort (a Stable Sort), page 72âĂŞ75. Cambridge University Press, 2008.

[10] A. V. Evfimievski, R. Srikant, R. Agrawal, and J. Gehrke. Privacy preserving mining of association rules. In *Proc. of KDD '02*, pages 217–228, 2002.

[11] C. Gentry. Fully homomorphic encryption using ideal lattices. In *Proc. of STOC'09*, pages 169–178. ACM, 2009.

[12] B. Goethals, S. Laur, H. Lipmaa, and T. Mielikäinen. On private scalar product computation for privacy-preserving data mining. In *Proc. of ICISC '04*, volume 3506 of *LNCS*, pages 104–120. Springer, 2005.

[13] K. Hamada, R. Kikuchi, D. Ikarashi, K. Chida, and K. Takahashi. Practically Efficient Multi-party Sorting Protocols from Comparison Sort Algorithms. In *Proc. of ICISC'12*, volume 7839 of *LNCS*, pages 202–216. Springer, 2013.

[14] H. Hollerith. US395781 (A) - ART OF COMPILING STATISTICS. European Patent Office, 1889. `http://worldwide.espacenet.com/publicationDetails/biblio?CC=US&NR=395781`.

[15] K. V. Jónsson, G. Kreitz, and M. Uddin. Secure Multi-Party Sorting and Applications. Cryptology ePrint Archive, Report 2011/122, 2011. `http://eprint.iacr.org/`.

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

[17] S. Laur, J. Willemson, and B. Zhang. Round-Efficient Oblivious Database Manipulation. In *Proc. of ISC'11*, pages 262–277, 2011.

[18] A. Machanavajjhala, D. Kifer, J. Gehrke, and M. Venkitasubramaniam. L-diversity: Privacy beyond k-anonymity. *ACM Trans. Knowl. Discov. Data*, 1(1):3, 2007.

[19] A. Narayanan and V. Shmatikov. Robust de-anonymization of large sparse datasets. In *Proc. of IEEE S&P '08*, pages 111–125, 2008.

[20] P. Paillier. Public-key cryptosystems based on composite degree residuosity classes. In *Proc. of EUROCRYPT'99*, volume 1592 of *LNCS*, pages 223–238. Springer, 1999.

[21] A. Shamir. How to share a secret. *Communications of the ACM*, 22:612–613, November 1979.

[22] L. Sweeney. k-anonymity: a model for protecting privacy. *Int. J. Uncertain. Fuzziness Knowl.-Based Syst.*, 10(5):557–570, 2002.

[23] G. Wang, T. Luo, M. T. Goodrich, W. Du, and Z. Zhu. Bureaucratic protocols for secure two-party sorting, selection, and permuting. In *Proc. of ASIACCS'10*, pages 226–237. ACM, 2010.

[24] B. Zhang. Generic Constant-Round Oblivious Sorting Algorithm for MPC. In *Provable Security*, volume 6980 of *LNCS*, pages 240–256. Springer, 2011.

APPENDIX

A. COMPARE-AND-EXCHANGE ON VECTORS

To sort matrix data using sorting networks, we need a CompEx function that works on vectors like the one described in Equation (2). A suitable algorithm is provided in Algorithm 5.

$$\mathsf{CompEx}(\mathcal{A}, \mathcal{B}, k) = \begin{cases} (\mathcal{B}, \mathcal{A}), & \text{if } \mathcal{A}_k > \mathcal{B}_k \\ (\mathcal{A}, \mathcal{B}), & \text{otherwise.} \end{cases} \quad (2)$$

Algorithm 5: Algorithm for obliviously comparing and exchanging two rows in a matrix.

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

B. SORTING NETWORK GENERATION BENCHMARKS

We benchmarked sorting network generation in the `libsortnetwork` library to choose the most suitable network generation algorithm for our implementation. Our goal was to create a network with the minimal number of rounds in a minimal time and, preferably, have a low number of CompEx functions.

We evaluated Batcher's bitonic mergesort network, Batcher's odd-even mergesort network and Parberry's pairwise sorting network. For each kind of network, we used the library to generate networks of various sizes, measured the time and counted the number of CompEx functions needed to evaluate it.

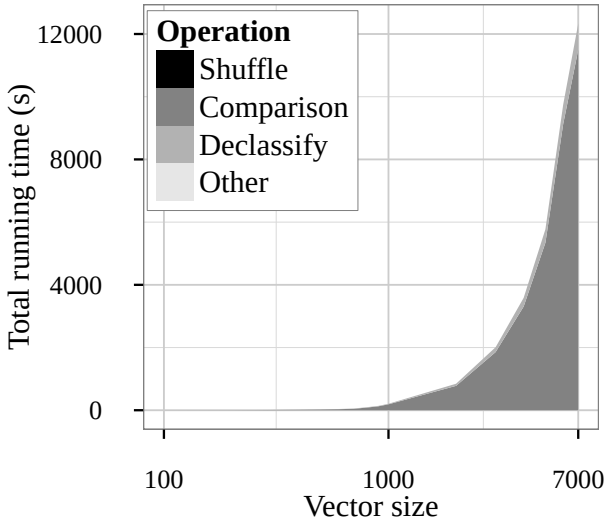
We found that the number of comparators is very similar for all algorithms and the same in many cases. Figure 5 shows a comparison of the running times and CompEx function call counts. We found that the library provides a bitonic mergesort network in the shortest time and while it has slightly more CompEx gates, the round count is the same and, therefore, the number of vector operations will be the same.

C. PERFORMANCE ANALYSIS OF MATRIX SORTING

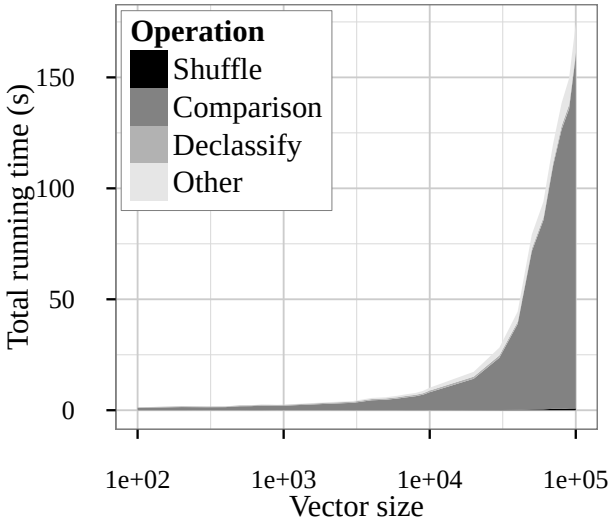
We benchmarked matrix sorting on $n \times 10$-element matrices by sorting them based on the first column. Figure 6 shows that even though there are ten times more data, the running time is not increased tenfold. This is explained by the vectorization of the matrix sorting implementations. There is one significant change, as sorting networks no longer benefit as heavily from caching the network structure, the oblivious exchanges for the full columns take up quite some time.

While sorting ten times as much data does not necessarily take that much time, it still takes more resources, as can be seen from Figures 7 and 8. Otherwise, the relations between different implementations remain the same.

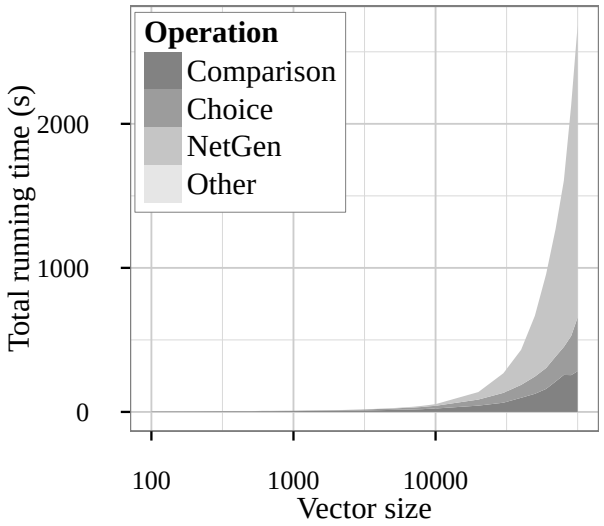
For sorting networks, we also implemented sorting by two and three different columns. This involves more comparisons and logic operations to combine the comparison results. According to the results in Figure 9, the number of columns by which to sort does not make a significant difference in the running time.



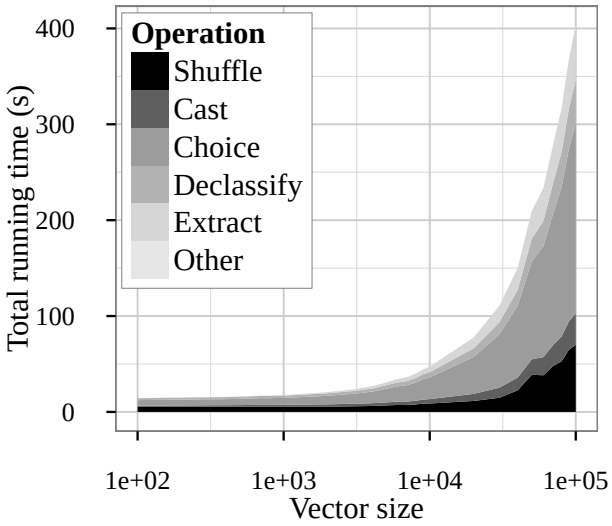
(a) Naive sorting



(b) Quicksort (average)



(c) Sorting networks



(d) Radix sort

Figure 1: Running time breakdown for implemented sorting algorithms.

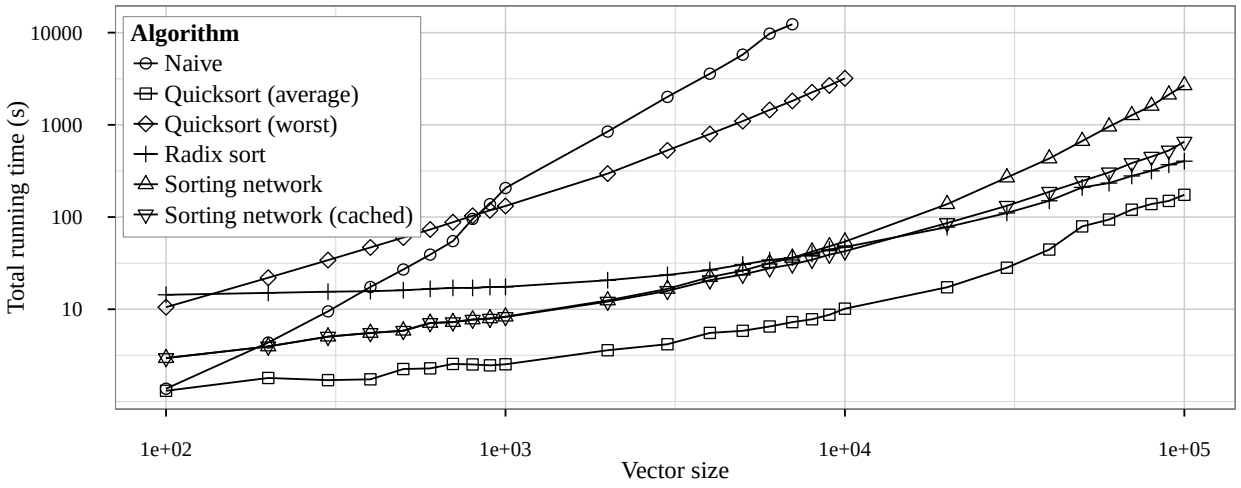


Figure 2: Comparison of the running time of oblivious sorting algorithms.

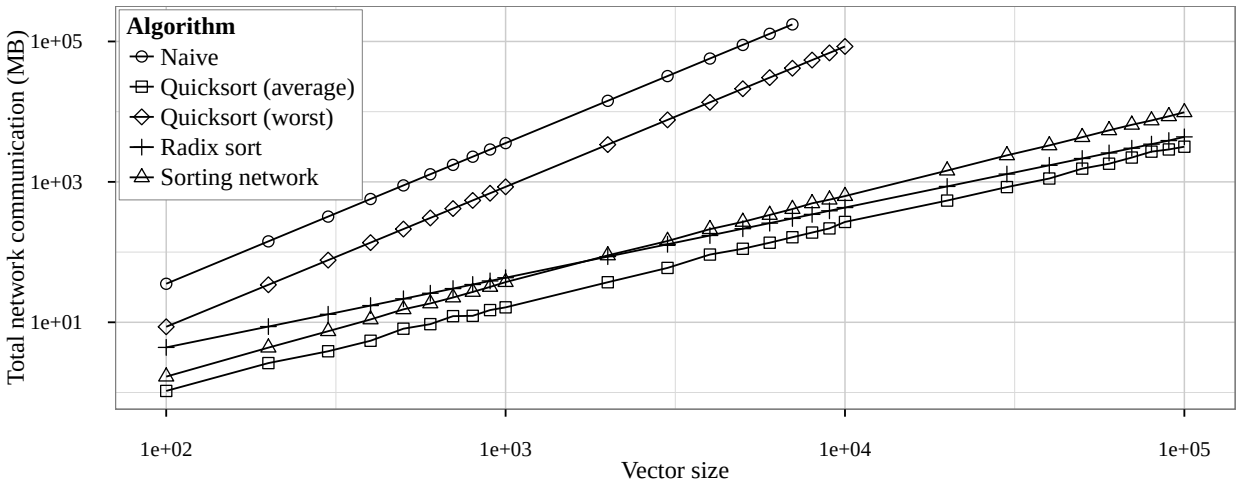


Figure 3: Comparison of the network usage of oblivious sorting algorithms.

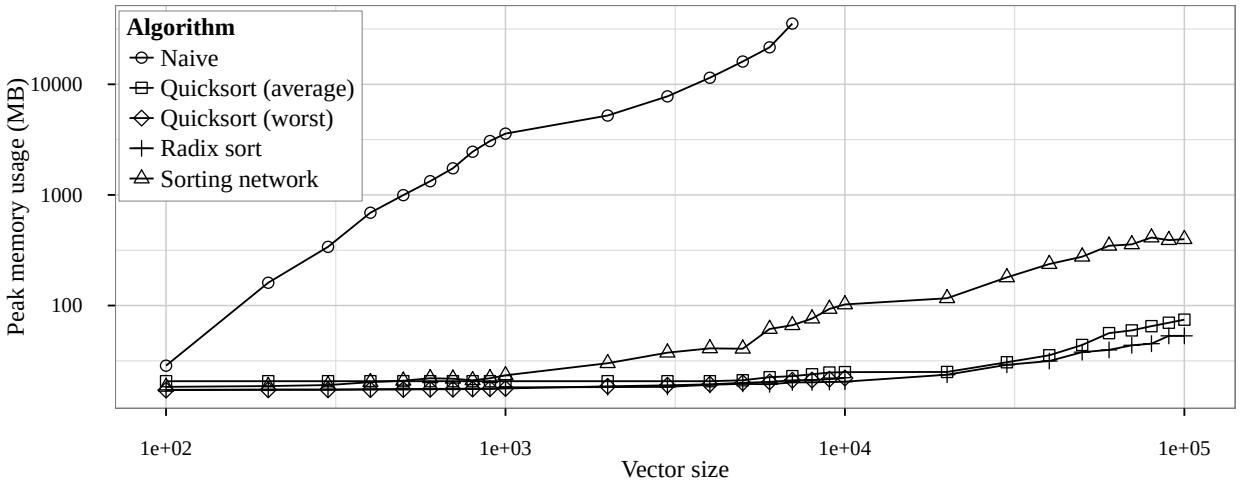


Figure 4: Comparison of the memory usage of oblivious sorting algorithms.

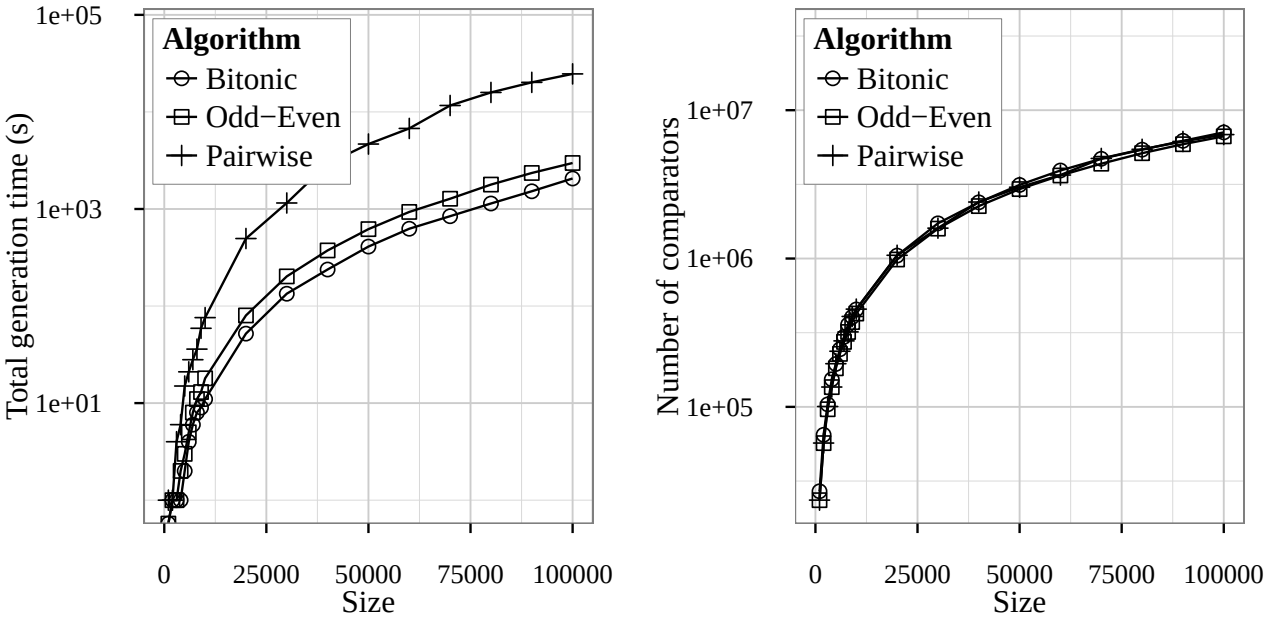


Figure 5: Benchmarking results for sorting network generation.

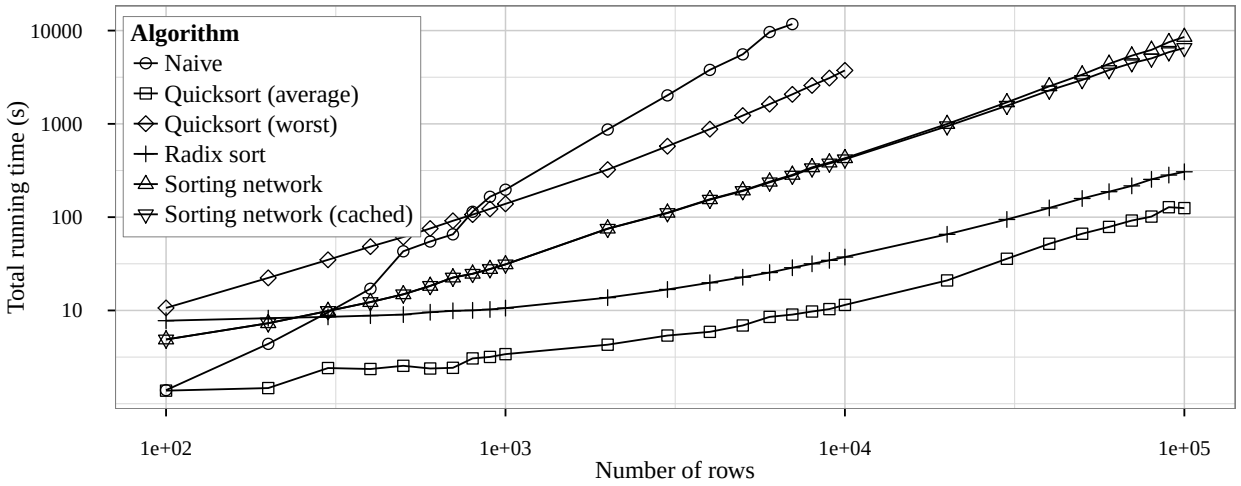


Figure 6: Running time of oblivious sorting algorithms on matrices.

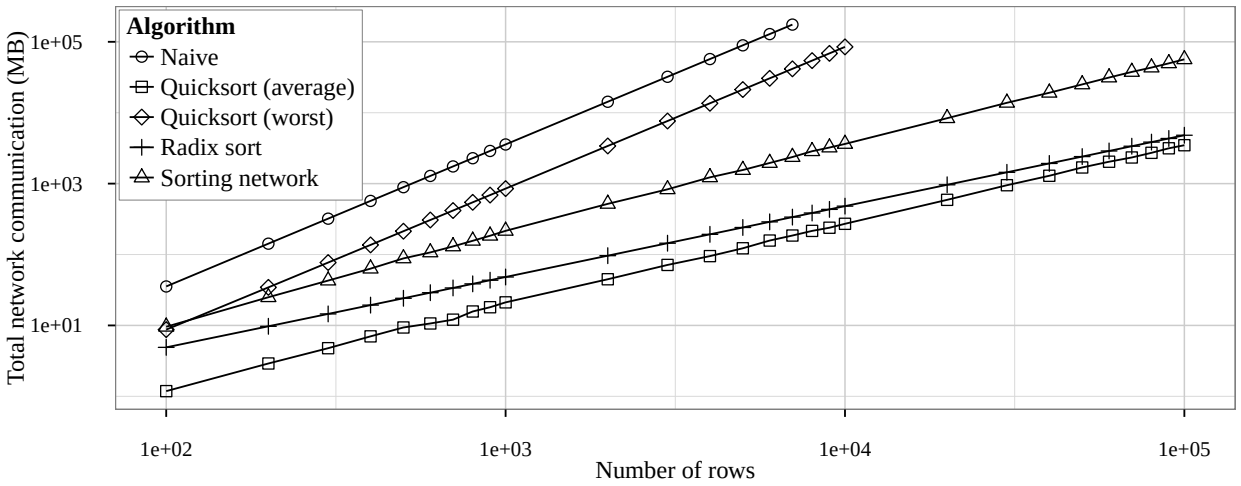


Figure 7: Network usage of oblivious sorting algorithms on matrices.

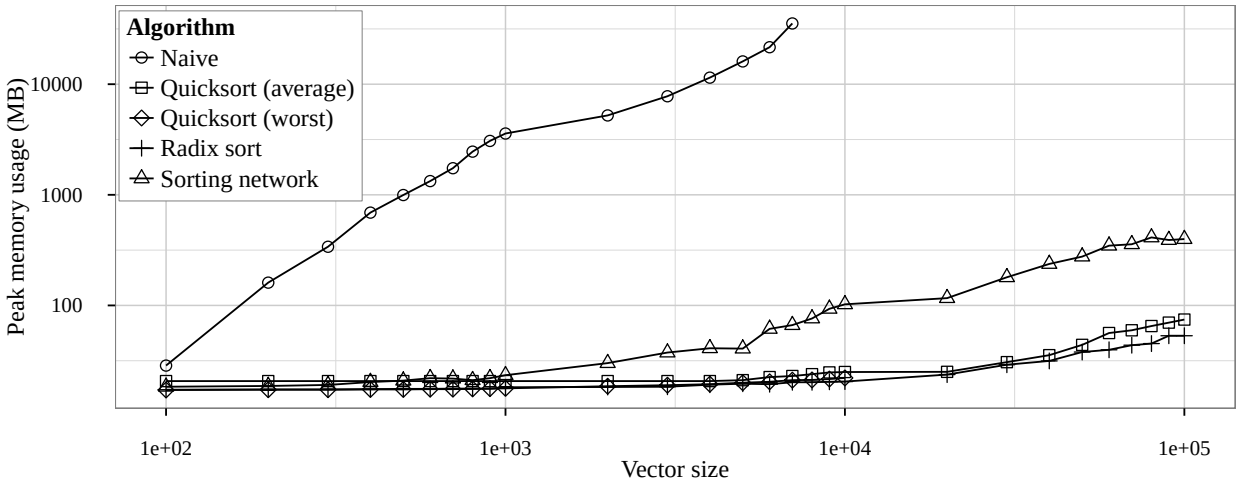


Figure 8: Memory usage of oblivious sorting algorithms on matrices.

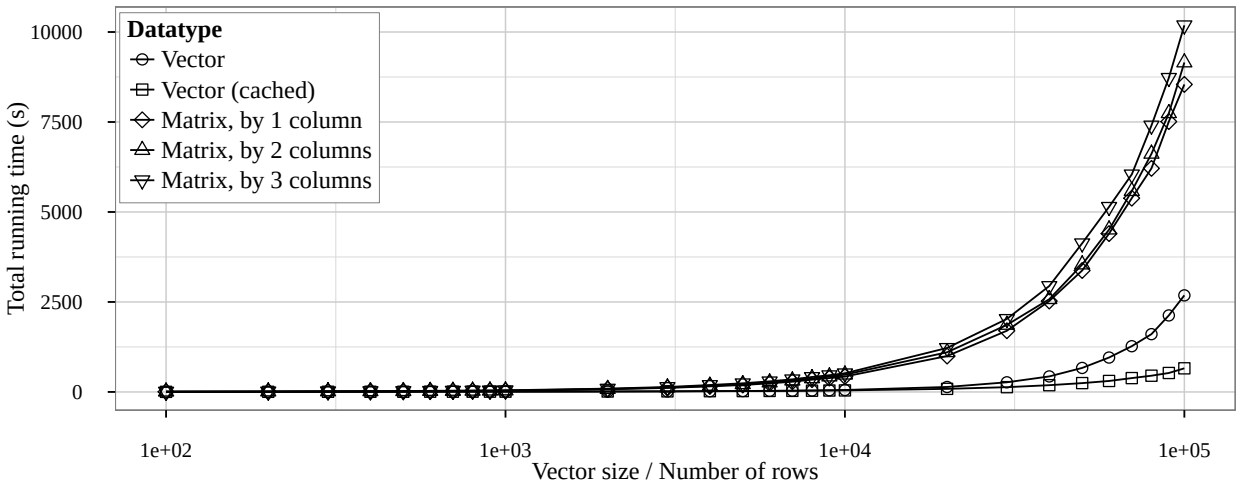


Figure 9: Sorting networks running time on vector and matrix inputs. For matrix inputs, sorting is performed by one, two and three columns.