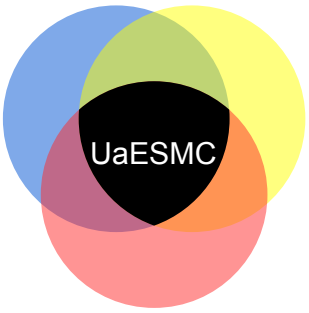

Project N°: **FP7-284731**

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 D2.2.3
Advances in SMC techniques

Due date of deliverable: 31st July 2015

Actual submission date: 31st July 2015

Start date of the project: **1st February 2012** Duration: **42 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 SMC techniques

This document summarizes deliverable D2.2.3 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 last period 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 give protocols for solving systems of linear equations in privacy-preserving manner. Our protocols constitute a nice example of adapting existing algorithms for running on private (secret-shared) data.
- On the other hand, we show that certain transformation-based techniques cannot in general give us privacy-preserving protocols for solving linear equation systems. The situation here is more complex than for linear programming, though, and for certain sets of possible systems of equations, some transformation-based methods may have acceptable privacy guarantees and good performance.
- We provide a novel protocol transformation that turns any passively secure multiparty computation protocol with honest majority to a protocol where any misbehaviour is detected after the execution. Our transformation is significantly more efficient than the one we've proposed in the previous reporting period. It is also more widely applicable, including protocols over rings, not fields. Hence it can be applied to the protocols of Sharemind.
- We show how the security property for SMC protocols can be decomposed into several ones, one of which is privacy. We show that privacy is also composable and aiming for this property only can give us more efficient SMC protocols. We also show that security can be recovered by composing a private protocol with a simple secure protocol.
- We give protocols for obliviously reading many elements of a private vector in parallel, and for obliviously writing many values into a private vector in parallel. Our protocols are more efficient than the existing implementations of Oblivious RAM (ORAM) on top of SMC. Our protocols allow any efficient algorithm for the Parallel Random Access Machine (PRAM) to be transformed to a privacy-preserving computation, as long as the control flow of that algorithm does not depend on private data.

List of Authors

Liina Kamm (CYB)
Peeter Laud (CYB)
Alisa Pankova (CYB)
Pille Pullonen (CYB)

Contents

Chapter 1

Introduction

This report gives an overview of the advances in secure multiparty computation techniques made during the last period 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 of UaESMC [3]. Even more, they have been motivated by the desire to have a comprehensive set of techniques available for the UaESMC framework. We find that our progress during the last period has gone a long way towards removing the existing gaps in the applicability of SMC to particular kinds of problems.

In the last reporting period, two of our research lines have converged on a particular problem, namely the privacy-preserving solution of systems of linear equations. This problem has to be solved for certain questions in privacy-preserving statistics, e.g. for linear regression. On the other hand, after showing the likely non-existence of transformation-based methods for privacy-preserving linear programming, we turned our attention to this seemingly more basic problem, which would also be useful in certain privacy-preserving algorithms for LP. We report both the possibility and the impossibility results in Chapter 2 of this report. We also note that both privacy-preserving statistics and optimization have been selected as example problems at the beginning of UaESMC [3].

We have studied the relationships between different properties of SMC protocols, and how to efficiently turn a protocol satisfying a weaker property into one that satisfies a stronger one. We have shown that the usual definition of security against passive adversaries is not the most primitive one, and a more basic property — privacy — is the one that the designers of SMC protocols should aim towards. We discuss these details in Chapter 3. Moving forward from protocols secure against passive adversaries, we have shown how to transform them in a way that any misbehaviours by adversarially controlled parties will be detected after the end of the protocol. We have proposed something similar during the previous reporting period, but the transformation we describe in Chapter 4 is both more efficient and more widely applicable. In particular, it applies directly to the three-party protocol set of Sharemind, secure against one passively corrupted party.

At the end of the previous reporting period, we identified a number of gaps [2] in the state of the art of SMC protocols, that may negatively influence our ability to translate efficient algorithms into SMC protocols. One of the gaps was on accessing data according to addresses that were themselves private. The existing methods were based on implementing the protocols for Oblivious RAM on top of secure multiparty computation. These methods had a number of problems, including the conceptual complexity, computational complexity and parallelizability. In Chapter 5 we report on a set of protocols that we have developed for oblivious data accesses that significantly improve in all these aspects. The protocols are simpler, their overheads are smaller, and they are naturally expected to run many instances in parallel, which goes along well with the secret-sharing based SMC protocol sets employed by Sharemind and also other SMC frameworks. The method is applicable for many different computational problems, the examples of which can be seen also in deliverable D4.2.2 [18].

Chapter 2

Privacy-preserving Solving of Systems of Linear Equations

A system of linear equations over a field $\mathbb{F}$ with k equations and k unknowns is specified by a $k \times k$ matrix $\mathbf{A}$ and a (column) vector $\vec{b}$ of length k, with entries from $\mathbb{F}$. To solve the system means to find a vector $\vec{x} \in \mathbb{F}^k$, such that $\mathbf{A} \cdot \vec{x} = \vec{b}$. In non-degenerate case, there is exactly one $\vec{x}$ satisfying this equation.

If a SMC protocol set is based on secret sharing over some finite field $\mathbb{F}$, then it is generally easy to solve systems of linear equations over this $\mathbb{F}$ [13]. Often we are interested in solving the systems of equations over the field $\mathbb{R}$ instead. In this chapter we present certain methods for doing it, as well as discuss approaches that are likely not going to work.

In existing SMC frameworks, e.g. SHAREMIND, the private values, belonging to $\mathbb{R}$, can be expressed (with finite precision) either as fixed- or floating-point numbers. There exist protocols for performing arithmetic operations with values expressed like this, as well as for invoking elementary functions on them. Our protocols for solving systems of linear equations call these lower-level protocols as subroutines.

In the following, the representation of private values is denoted by $[\![\cdot]\!]$. For any v, the write-up $[\![v]\!]$ denotes that v has been shared among the computing parties in the SMC framework.

2.1 Adaptations of Numeric Methods to SMC

In [6, Sec. 6] (attached to the deliverable) we show how the methods of Gaussian elimination and LU decomposition can be adapted to run on top of a SMC framework. In the following, we give an overview of the adaptions. The goal of the SMC protocols based on these algorithms is to find $[\![\vec{x}]\!]$, given $[\![\mathbf{A}]\!]$ and $[\![\vec{b}]\!]$.

Both methods make use of pivoting that should take place around an element with the largest absolute value, in order to minimize numerical instabilities. Alg. 1 describes the function **maxLoc** that finds the pivot element and its location from a given vector of private values. To avoid leaking information about equal values in a vector, the indices are first permuted to ensure cryptographic privacy. This means that the indices are traversed in random order during each execution. On line 4, the current maximum element is compared with the element that is being viewed. On lines 5 and 6 the value and its location are determined obliviously. Namely, if the new element is larger, it will be considered the new maximum element and its location will be recorded based on the same comparison result. In case of several maximum elements, it returns the location of one of these elements.

Of the two algorithms for solving a system of linear equations, let us first look more closely at Gaussian elimination with back substitution. Alg. 2 gives the privacy-preserving version of the Gaussian elimination algorithm.

At the start of the algorithm (line 2), the rows of the input matrix $[\![\mathbf{A}]\!]$ are shuffled along with the elements of the dependent variable vector that have been copied to $[\![\vec{x}]\!]$, retaining the relations. On lines 5-11, the pivot element is located from the remaining matrix rows and then the rows are interchanged so that the one with the pivot element becomes the current row. Note that as all the matrix indices are public,

Algorithm 1: maxLoc: Finding the first maximum element and its location in a vector in a privacy-preserving setting.

Data: A vector $[\![\vec{a}]\!]$ of length n.
Result: The maximum element $[\![b]\!]$ and its location $[\![l]\!]$ in the vector.

1 Let $\pi(j)$ be a permutation of indices $j \in \{1, \ldots, n\}$
2 $[\![b]\!] \leftarrow [\![\vec{a}_{\pi(1)}]\!]$ and $[\![l]\!] \leftarrow \pi(1)$
3 **for** $i = \pi(2)$ **to** $\pi(n)$ **do**
4 $\quad [\![c]\!] \leftarrow (|[\![\vec{a}_{\pi(i)}]\!]| > |[\![b]\!]|)$
5 $\quad [\![b]\!] \leftarrow [\![b]\!] - [\![c]\!] \cdot [\![b]\!] + [\![c]\!] \cdot [\![\vec{a}_{\pi(i)}]\!]$
6 $\quad [\![l]\!] \leftarrow [\![l]\!] - [\![c]\!] \cdot [\![l]\!] + [\![c]\!] \cdot \pi(i)$
7 **return** $([\![b]\!], [\![l]\!])$

the conditionals work in the public setting. As we need to use the pivot element as the divisor, we need to check whether it is 0. However, we do not want to reveal information about the location of this value, so on line 12, we privately make a note of whether any of the pivot elements are 0, and on line 26, we finish the algorithm early if we are dealing with a singular matrix. In Sharemind, division by 0 will not be reported during privacy-preserving computations as this will reveal the divisor immediately.

On lines 15 - 18, elements on the pivot line are reduced. Similarly, on lines 19 - 24, elements below the pivot line are reduced. Finally, on lines 27 - 29, back substitution is performed to get the values of the coefficients.

The main difference between the original and the privacy-preserving Gaussian elimination algorithm is actually in the **maxLoc** function. In the original version, elements are compared, one by one, to the largest element so far and at the end of the subroutine, the greatest element and its location are found. Our algorithm does the same, except that it uses oblivious choice instead of straightforward if-clauses. This way, we are able to keep the largest element secret and only reveal its location at the end without finding out other relationships between elements in the vector during the execution of this algorithm.

Let us now look at LU decomposition. In the ordinary setting, this method is faster than the Gaussian elimination method. LU decomposition uses matrix decomposition to achieve this speed-up. If we can decompose the input matrix into a lower and upper triangular matrix $\mathbf{L}$ and $\mathbf{U}$, respectively, so that $\mathbf{L} \cdot \mathbf{U} = \mathbf{A}$, we can use forward substitution and back substitution on these matrices, similarly to the process we used in the Gaussian elimination method. Alg. 3 gives the privacy-preserving version of LU decomposition. Note that the elements on the diagonal of the lower triangular matrix $\mathbf{L}$ are equal to 1. Knowing this, $\mathbf{L}$ and $\mathbf{U}$ can be returned as one matrix so that the diagonal and the elements above it belong to the upper triangular matrix $\mathbf{U}$ and the elements below the diagonal belong to the lower triangular matrix $\mathbf{L}$ without losing any information.

Similarly to Alg. 2, first the pivot element is found using the **maxLoc** function. After the elements are exchanged, the row permutations are saved for use in the algorithm in order to solve the set of linear equations. As a result, the decomposition matrix and the permutations are returned. The permutations are public information but they reveal nothing about the original dataset because the rows have been shuffled before they are input into the decomposition algorithm similarly to what was done in Alg. 2.

Alg. 4 shows how to solve a set of linear equations using LU decomposition. The matrix rows are shuffled as in Alg. 2 and the LU decomposition matrix is composed using the **LUDecomp** function. As an additional result we receive the permutation that was done for pivoting purposes during the decomposition phase. Next, on row 5, elements of vector $[\![\vec{b}]\!]$ containing the dependent variable are permuted to be in concurrence with the permutations performed during the decomposition phase.

On lines 6 - 7, forward substitution is performed using the values from the lower triangular matrix. Finally, on lines 8 - 9, back substitution is performed using the values from the upper triangular matrix.

Algorithm 2: Privacy-preserving Gaussian elimination with back substitution for a matrix equation $[\![\mathbf{A}]\!][\![\vec{x}]\!] = [\![\vec{b}]\!]$.

Data: a $k \times k$ matrix $[\![\mathbf{A}]\!]$, a vector $[\![\vec{b}]\!]$ of k values.
Result: Vector $[\![\vec{x}]\!]$ of coefficients.

Let $[\![\vec{x}]\!]$ be a copy of $[\![\vec{b}]\!]$
Obliviously shuffle $[\![\mathbf{A}]\!], [\![\vec{x}]\!]$ retaining the dependencies
Let $[\![c]\!] \leftarrow$ **false** privately store the failure flag during execution
for $i = 1$ **to** $k - 1$ **do**
    Let $[\![\vec{m}]\!]$ be a subvector of $[\![\mathbf{A}_{u,v}]\!]$ such that $u \in \{i, \ldots, k\}, v = i$
    $([\![t]\!], [\![irow]\!]) \leftarrow$ **maxLoc**$([\![\vec{m}]\!])$
    $irow \leftarrow \mathsf{declassify}([\![irow]\!]) + i$
    if $irow \neq i$ **then**
        for $j = 1$ **to** k **do**
            Exchange elements $[\![\mathbf{A}_{irow,j}]\!]$ and $[\![\mathbf{A}_{i,j}]\!]$
        Exchange element $[\![\vec{x}_{irow}]\!]$ and $[\![\vec{x}_i]\!]$
    $[\![c]\!] \leftarrow [\![c]\!] \vee ([\![\mathbf{A}_{i,i}]\!] = 0)$
    $[\![pivinv]\!] \leftarrow [\![\mathbf{A}_{i,i}]\!]^{-1}$
    $[\![\mathbf{A}_{i,i}]\!] \leftarrow 1$
    foreach $j \in \{1, \ldots, k\}$ **do**
        if $j \neq i$ **then**
            $[\![\mathbf{A}_{i,j}]\!] \leftarrow [\![\mathbf{A}_{i,j}]\!] \cdot [\![pivinv]\!]$
        $[\![\vec{x}_i]\!] \leftarrow [\![\vec{x}_i]\!] \cdot [\![pivinv]\!]$
    for $m = i + 1$ **to** k **do**
        for $j = 1$ **to** k **do**
            if $j \neq i$ **then**
                $[\![\mathbf{A}_{m,j}]\!] \leftarrow [\![\mathbf{A}_{m,j}]\!] - [\![\mathbf{A}_{i,j}]\!] \cdot [\![\mathbf{A}_{m,i}]\!]$
        $[\![\vec{x}_m]\!] \leftarrow [\![\vec{x}_m]\!] - [\![\vec{x}_i]\!] \cdot [\![\mathbf{A}_{m,i}]\!]$
        $[\![\mathbf{A}_{m,i}]\!] \leftarrow 0$
if *declassify*$([\![c]\!])$ **then**
    return "Singular matrix"
$[\![\vec{x}_k]\!] \leftarrow \frac{[\![\vec{x}_k]\!]}{[\![\mathbf{A}_{k,k}]\!]}$
for $i = k - 1$ **downto** 1 **do**
    $[\![\vec{x}_i]\!] \leftarrow [\![\vec{x}_i]\!] - \sum_{j=i+2}^{k} [\![\mathbf{A}_{i,j}]\!] \cdot [\![\vec{x}_j]\!]$
return $[\![\vec{x}]\!]$

Algorithm 3: LUDecomp: Privacy-preserving LU decomposition for a symmetric matrix $[\![\mathbf{B}]\!]$.

Data: a $k \times k$ matrix $[\![\mathbf{B}]\!]$.
Result: The LU decomposition matrix $[\![\mathbf{B}]\!]$ and $\vec{q}$ containing the row permutations.

Let $[\![c]\!] \leftarrow 0$ be a Boolean value
for $i = 1$ **to** k **do**
  Let $[\![\vec{m}]\!]$ be a subvector of $[\![\mathbf{B}_{u,v}]\!]$ such that $u \in \{i, \ldots, k\}, v = i$
  $([\![t]\!], [\![irow]\!]) \leftarrow \mathbf{maxLoc}([\![\vec{m}]\!])$
  $irow \leftarrow \mathsf{declassify}([\![irow]\!])$
  if $irow \neq i$ **then**
    for $j = 1$ **to** k **do**
      Exchange elements $[\![\mathbf{B}_{irow,j}]\!]$ and $[\![\mathbf{B}_{i,j}]\!]$
  $[\![c]\!] \leftarrow [\![c]\!] \vee ([\![\mathbf{B}_{i,i}]\!] = 0)$
  $\vec{q_i} \leftarrow irow$
  $[\![ipiv]\!] \leftarrow [\![\mathbf{B}_{i,i}]\!]^{-1}$
  for $m = i + 1$ **to** k **do**
    $[\![\mathbf{B}_{m,i}]\!] \leftarrow [\![\mathbf{B}_{m,i}]\!] \cdot [\![ipiv]\!]$
    for $j = i + 1$ **to** k **do**
      $[\![\mathbf{B}_{m,j}]\!] \leftarrow [\![\mathbf{B}_{m,j}]\!] - [\![\mathbf{B}_{m,i}]\!] \cdot [\![\mathbf{B}_{i,j}]\!]$
if *declassify*$([\![c]\!])$ **then**
  return "Singular matrix"
return $([\![\mathbf{B}]\!], \vec{q})$

Algorithm 4: Solving the linear regression task $[\![\vec{y}]\!] \approx [\![\mathbf{X}]\!][\![\vec{\beta}]\!]$ using the LU decomposition matrix in a privacy-preserving setting.

Data: An $n \times k$ data matrix $[\![\mathbf{X}]\!]$ and an n element response vector $[\![\vec{y}]\!]$.
Result: A vector $[\![\vec{b}]\!]$ of coefficients.

Compute correlation matrix $[\![\mathbf{A}]\!] \leftarrow [\![\mathbf{X}]\!]^T [\![\mathbf{X}]\!]$
Compute new target vector $[\![\vec{b}]\!] \leftarrow [\![\mathbf{X}]\!]^T [\![\vec{y}]\!]$
Shuffle $[\![\mathbf{A}]\!], [\![\vec{b}]\!]$ retaining the dependencies
$([\![\mathbf{B}]\!], \vec{q}) \leftarrow \mathbf{LUDecomp}([\![\mathbf{A}]\!])$
Rearrange $[\![\vec{b}]\!]$ based on permutation $\vec{q}$
for $i = 2$ **to** k **do**
  $[\![\vec{b_i}]\!] \leftarrow [\![\vec{b_i}]\!] - \sum_{j=1}^{i} [\![\mathbf{B}_{i,j}]\!] \cdot [\![\vec{b_j}]\!]$
for $i = k$ **downto** 1 **do**
  $[\![\vec{b_i}]\!] \leftarrow \left([\![\vec{b_i}]\!] - \sum_{j=i+1}^{k} [\![\mathbf{B}_{i,j}]\!] \cdot [\![\vec{b_j}]\!] \right) \cdot [\![\mathbf{B}_{i,i}]\!]^{-1}$
return $[\![\vec{b}]\!]$

2.2 Transformation-based Solution Methods

In deliverables D2.2.1 [4, Chap. 3.1] and D2.2.2 [5, Chap. 3] we have investigated linear programming (LP). We have found that, at least using current approaches, achieving a good standard definition that guarantees security is impossible for transformation-based outsourcing of LP tasks. In this chapter, we take a look at linear equation systems. Hiding them should be conceptually simpler, and one can no longer perform the geometric attacks that were proposed against LP in [5, Chap. 3]. Instead of hiding the entire feasible region, we constrain our task to hiding just a single point, thus considering a linear equation system $\mathbf{A}\vec{x} = \vec{b}$ for a $n \times n$ full-rank matrix $\mathbf{A}$. Nevertheless, it turns out that even in such simplified settings finding a good transformation for linear equations over real numbers is challenging. We studied the class of affine transformations for matrices over real numbers, found out which forms are possible at all, and stated some properties that the transformation and the initial matrices must satisfy in order to make the initial matrices perfectly (or statistically) indistinguishable after applying the transformation. Our work provides both possibility and impossibility results.

2.2.1 Problem Statement

Our particular task is to solve a full-rank $n \times n$ system $\mathbf{A}\vec{x} = \vec{b}$. Since the most complex part of finding $\vec{x} = \mathbf{A}^{-1}\vec{b}$ is computing $\mathbf{A}^{-1}$ over $\mathbb{R}$, and the consequent multiplication by $\vec{b}$ is relatively easy, the problem can be reduced to outsourcing of a square matrix inverse computation.

Outsourcing matrix inverse over a finite field $\mathbb{F}$ is easy and can be done as follows. Matrix multiplication is much easier than matrix inverse. Given an $n \times n$ invertible matrix $\mathbf{A}$ over $\mathbb{F}$, the client generates a random invertible matrix $\mathbf{R} \xleftarrow{\$} \mathbb{F}^{n\times n}$. The product $\mathbf{RA}$ may now be published, and $(\mathbf{RA})^{-1}$ computed offline by any powerful solver. Knowing $\mathbf{R}$, the result $\mathbf{A}^{-1}$ can be obtained from $(\mathbf{RA})^{-1} = \mathbf{A}^{-1}\mathbf{R}^{-1}$ as $\mathbf{A}^{-1}\mathbf{R}^{-1} \cdot \mathbf{R}$. Since invertible $n \times n$ matrices over $\mathbb{F}$ form a multiplicative group $GL(n, \mathbb{F})$, the product $\mathbf{RA}$ is distributed uniformly in $GL(n, \mathbb{F})$ for any $\mathbf{A} \in GL(n, \mathbb{F})$, so $\mathbf{RA}$ leaks no information about $\mathbf{A}$. There do exist algorithms for efficient sampling of (almost) uniformly distributed matrices over $\mathbb{F}$. This fact entices to try out something similar for matrices over real numbers.

2.2.2 Security Definition

We use the same secure transformation definition that we used for linear programming in [4, Chap. 3.1] and [5, Chap. 3]. Let $\mathfrak{T} \subseteq \{0,1\}^*$ denote 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 defined as a pair of functions $\mathcal{F} : \mathfrak{T} \times \{0,1\}^* \to \mathfrak{T} \times \{0,1\}^*$ and $\mathcal{G} : \mathfrak{S} \times \{0,1\}^* \to \mathfrak{S}$.

A *correct problem transformation* is a pair $(\mathcal{F}, \mathcal{G})$ that satisfies

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

where r is the randomness used by the transformation. For simplicity, let $\mathcal{F}(T)$ denote a randomized function that first samples r and then runs $\mathcal{F}(T; r)$. We assume that both $\mathcal{F}$ and $\mathcal{G}$ are polynomial-time with respect to the task description size.

Some information may be intentionally leaked by the transformation, due to being impractical to hide. This is captured by the notion of *side information function* $\Phi : \mathfrak{T} \to \{0,1\}^*$. When transforming a task T, we do not try to hide the information in $\Phi(T)$. For example, if T is not transformed at all, then $\Phi(T) = T$. The function Φ is assumed to be polynomial-time.

Definition 2.2.1. *A transformation $(\mathcal{F}, \mathcal{G})$ for $\mathfrak{T}$ is (computationally)* Φ-private *if the advantage of any probabilistic polynomial-time adversary $\mathcal{B} = (\mathcal{B}_1, \mathcal{B}_2)$ is a negligible function of η in the following experiment*

$\mathbf{Exp}_{\mathcal{B}}^{\mathfrak{T},\Phi}$:

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

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

In the following, we work with real numbers, hence it may be unclear what it means for a probabilistic polynomial-time algorithm to process these numbers. Nevertheless, we can define *perfect* and *statistical* Φ-privacy, by letting $\mathcal{B}_1$ and $\mathcal{B}_2$ be any functions and, in case of perfect privacy, require the advantage of $\mathcal{B}$ to be 0. For a fixed η, we say that the transformation provides at least σ bits of security, if the advantage of $\mathcal{B}$ is at most $2^{-\sigma}$. This level of security is achieved iff $SD(T_0', T_1') \leq 2^{-\sigma}$ for all $T_0, T_1 \in \mathfrak{T}$ with $\Phi(T_0) = \Phi(T_1)$ and $|T_0| = |T_1| = \eta$, where T_0' and T_1' are the first components of $\mathcal{F}(T_0)$ and $\mathcal{F}(T_1)$ respectively. Typically, we want the security level to be at least η^c for some constant $c > 0$.

In our case, the set of tasks $\mathfrak{T}$ corresponds to a set $\mathcal{A}$ of $n \times n$ invertible matrices. The set $\mathcal{A}$ depends on the application in which the matrix inversion problem appears; it is quite likely that $\mathcal{A}$ is not the whole $GL(n, \mathbb{R})$. It may be possible that for some applications involving the inversion of matrices, privacy-preserving outsourcing via transformation is possible, while other applications must use other means of inverting matrices.

2.2.3 Our Results

In our work, we have studied transformations of the form $\mathcal{F}(\mathbf{A}) = \mathbf{PAQ} + \mathbf{R}$ (*affine* transformations) for $\mathbf{A} \in \mathbb{R}^{n \times n}$, $\mathbf{P} \in \mathbb{R}^{m \times n}$, $\mathbf{Q} \in \mathbb{R}^{n \times m}$, $\mathbf{R} \in \mathbb{R}^{m \times m}$. For this, we achieved both positive and negative results.

First, we have shown that, since the computation of $\mathbf{A}^{-1} = \mathcal{G}(\mathbf{B}^{-1})$ should be efficient (and hence $\mathbf{B}$ should be easier to compute than $\mathbf{A}^{-1}$), the most general type of affine transformation is of the form $\mathbf{A} \mapsto \mathbf{P} \begin{pmatrix} \mathbf{A} & \mathbf{0} \\ \mathbf{0} & \mathbf{I} \end{pmatrix} \mathbf{Q}$. Since matrix determinant is multiplicative, $\det \left(\mathbf{P} \begin{pmatrix} \mathbf{A} & \mathbf{0} \\ \mathbf{0} & \mathbf{I} \end{pmatrix} \mathbf{Q} \right) = \det \mathbf{P} \cdot \det \mathbf{A} \cdot \det \mathbf{Q}$, and we can immediately see that only matrices with the same determinant absolute values can be perfectly indistinguishable. If we want to achieve statistical indistinguishability for arbitrary $\mathbf{A}_1$ and $\mathbf{A}_2$, the entries of $\mathbf{P}$ and $\mathbf{Q}$ grow in $\Omega(|\det \mathbf{A}_1 / \det \mathbf{A}_2|^{2^\eta})$ for a security parameter η. This is similar to hiding a real number multiplying it by another random real number (the case of (1×1) matrix), where $a \in \mathbb{R}$ and $-a \in \mathbb{R}$ can be indeed made perfectly indistinguishable multiplying them by a random number r coming from any symmetric distribution. However, for matrices of higher dimensions, there exist matrices that have the same determinants, but are still quite distinguishable for any distribution of random matrices $\mathbf{P}$ and $\mathbf{Q}$.

If we try to limit hiding to just multiplication by $\mathbf{P}$ with bounded operator ℓ_2 norms (in this case, sampling of $\mathbf{P}$ would be easier to define), which works well in matrices over finite fields, being able to achieve perfect secrecy implies that we should be able to sample uniformly from a certain group $\mathcal{G}$ that depends on the set of matrices $\mathcal{A}$ that we want to hide. According to known facts about the properties of groups, if we can sample from a group uniformly, this group is conjugate to an orthogonal group $O(n, \mathbb{R})$ (matrices U such that $U^T = U^{-1}$), and hence the inverse of any $\mathbf{G} \in \mathcal{G}$ can be obtained just by transposing $\mathbf{G}$, without needing any outsourcing. The set of matrices $\mathcal{A}$ is related to $\mathcal{G}$ in such a way that inverting the elements of $\mathcal{G}$ make it easy to invert the elements of $\mathcal{A}$. In other words, if we know the distribution of $\mathbf{P}$ that would hide $\mathbf{A}$ well, we could invert $\mathbf{A}$ more easily by other means, without needing outsourcing at all.

If we use a more general transformation $\mathcal{F}(\mathbf{A}) = \mathbf{PAQ}$ with bounded ℓ_2 operator norms for hiding, then we should be able to sample uniformly from $\mathcal{A}$. This point is the gap between the possibility and impossibility results: although we have found examples of $\mathcal{A}$ that $\mathbf{PAQ}$ can hide, but $\mathbf{PA}$ cannot, it still does not mean that there does not exist an easier algorithm for inverting the elements of such $\mathcal{A}$.

If we are using matrices $\mathbf{P}$ and $\mathbf{Q}$ with unbounded ℓ_2 operator norms, then we can still achieve statistical security. Some working constructions are given in Sec. 2.2.4.

Although we finally have found some possibility results, they all unfortunately still leak the determinant, unless the entries of $\mathbf{P}$ and $\mathbf{Q}$ grow in $\Omega(c^{2^\eta})$ where c is the maximal ratio absolute value of different determinants. This is not what we would like to get in practice.

A summary of our findings is depicted in Fig. 2.1. The technical details can be found in [27], which is also included in Appendix B.

2.2.4 Working Constructions

Our general impossibility results hold for matrices with bounded ℓ_2-norm, which immediately discard perfect hiding, but still leave hope for statistical hiding. In this section, we present some possibility results.

2.2.4.1 Hiding triangular matrices by $\mathbf{B} = \mathbf{PA}$

We have shown in [27] that, using $\mathbf{B} = \mathbf{PA}$, we cannot perfectly hide matrices with eigenvalues $\lambda \neq \pm 1$. Moreover, it extends to statistical hiding: although theoretically it is possible, the sizes of entries would grow too large. Nevertheless, we propose some hiding for matrices with eigenvalues $\lambda = \pm 1$.

Let $\mathbf{A}$ be a lower triangular matrix with $a_{ii} = \pm 1$ (this is exactly the case when we have $\lambda = \pm 1$ for each eigenvalue λ of $\mathbf{A}$). Let $\mathbf{P} = (\vec{p}_1 | \dots | \vec{p}_n)$. Due to triangularity of $\mathbf{A}$, We can write $\mathbf{PA} = (a_{11}\vec{p}_1 + \dots + a_{n1}\vec{p}_n | \dots | a_{n-1,n-1}\vec{p}_{n-1} + a_{n,n-1}\vec{p}_n, a_{nn}\vec{p}_n)$. Since $a_{nn} = 1$, the last vector of $\mathbf{PA}$ does not depend on $\mathbf{A}$.

We see that, starting from an arbitrary $\vec{p}_n$, we can generate $\vec{p}_i$ in such a way that it hides $\vec{p}_{i+1}$. Namely, if the entries of $\vec{p}_n$ are bounded by 1 (which is the least possible bound), let the entries of $\vec{p}_{n-1}$ be uniformly distributed between 0 and $c \cdot 2^\eta$, where $c \geq |a_{n,n-1}|$. In this way, $a_{n-1,n-1}\vec{p}_{n-1} + a_{n,n-1}\vec{p}_n$ statistically hides $\vec{p}_n$ with at least η bits of security. On the next step, to hide the $(n-2)$-th column with at least η bits of security, we have to sample the entries of $\vec{p}_{n-2}$ uniformly from between 0 and $c(2^\eta + 2^{2\eta})$, where c is an upper bound for both $|a_{n,n-2}|$ and $|a_{n-1,n-2}|$. Proceeding to $\vec{p}_1$, we get that its entries should be sampled uniformly from between 0 and $c(2^\eta + 2^{2\eta} + \dots + 2^{(n-1)\eta})$, where c is an upper bound for the absolute values of all entries in the first column of A.

The preceding discussion shows that in order to statistically hide matrices from a set $\mathcal{A}$ satisfying the following:

- a matrix $\mathbf{A} \in \mathcal{A}$ is lower triangular with the main diagonal containing only the values ± 1;
- the absolute values of the entries of the matrices in $\mathcal{A}$ are bounded by c;

Since lower triangular matrices can be inverted much more easily than general matrices, it does not make any practical sense to outsource it like that. Nevertheless, we can use this hiding as a building block for more general matrices, as we show in the next section.

2.2.4.2 Hiding based on QR-decomposition

Let $\mathcal{A}$ be an arbitrary set of $n \times n$ matrices with bounded entries, and determinant ± 1. Any matrix can be represented as $\mathbf{A} = \mathbf{Q}_A\mathbf{R}_A$, where $\mathbf{Q}_A \in O(n, \mathbb{R})$ and $\mathbf{R}_A$ is an upper triangular matrix with positive entries on the diagonal (such a decomposition is unique). Note that $|\mathbf{Q}_A| = 1$ in any case. We put the following further restrictions on $\mathcal{A}$:

- there exists $c \in \mathbb{R}$ that upper bounds the absolute values of the entries of $\mathbf{R}_A$ for any $\mathbf{A} \in \mathcal{A}$;
- the entries on the main diagonal of $\mathbf{R}_A$ are ± 1.

Taking $\mathcal{D}_P$ from Sec. 2.2.4.1, we may now take $\mathcal{D}_{(P,Q)} = \{(\mathbf{U}, \mathbf{P}^T) \mid \mathbf{U} \xleftarrow{\$} O(n, \mathbb{R}), \mathbf{P} \leftarrow \mathcal{D}_P\}$. The distribution $\mathcal{D}_{(P,Q)}$ statistically hides $\mathcal{A}$, because for any $\mathbf{A} \in \mathcal{A}$, the matrix $\mathbf{UQ}_A$ is a uniformly distributed orthogonal matrix, and the product $\mathbf{R}_A\mathbf{P}^T$ statistically hides $\mathbf{R}_A$.

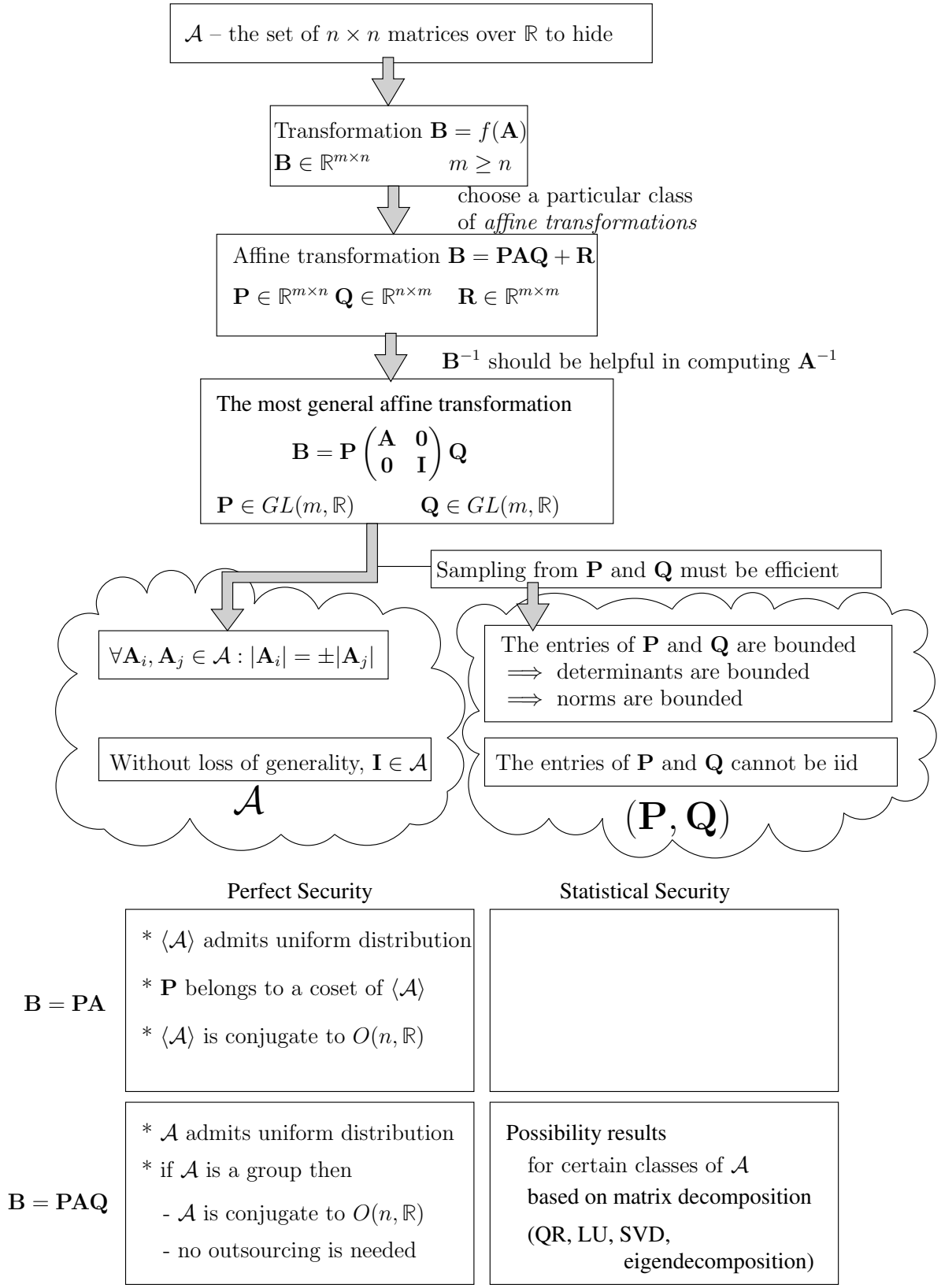

Figure 2.1: Possibility and impossibility results for outsourcing the inversion of real matrices in a set $\mathcal{A}$ with help of affine transformations

There exist algorithms that allow to sample uniformly from $O(n, \mathbb{R})$. Even if the secure implementation of these sampling algorithms is inefficient, the proposed hiding method can be useful at least in SMC setting. Indeed, in this setting, a random orthogonal matrix can be generated simply by each computing party locally, generating a random orthogonal matrix, entering it into the computation, and multiplying these matrices using SMC protocols.

Similar hiding constructions are based on other well-known matrix decompositions (LU,SVD, eigendecomposition), see [27] for details. Unfortunately, they are not generalizable to an arbitrary set $\mathcal{A}$, even assuming $\forall \mathbf{A} \in \mathcal{A} : \det \mathbf{A} = \pm 1$, and setting constraints on eigenvalues or singular values is also necessary.

2.3 Conclusion

We see that the most general methods that we have for solving linear equation systems are based on Gaussian elimination or on some decomposition-based method. The applicability of transformation-based methods is subject to further study. We note that there are also generic transformation-based methods. These involve scaling up the coefficients of the equations, such that they become integers. These can be embedded in a sufficiently large finite field, and the resulting system can be solved. The field must be large enough to contain the square of the determinant of any submatrix of the system of equations. Hence the field will quickly grow very large and it is unclear if any performance benefit can be obtained from such an approach.

Chapter 3

From Private to Secure SMC Primitives

Secure multi-party computation protocols for practical functionalities are built from small protocols that correspond to common arithmetic or boolean operations. Hence, to claim that the resulting functionalities are secure, we require security and composability of the component protocols. This chapter explores the properties that the core protocols have to provide in order to be suitable as efficient building blocks for large functionalities with passive security. Traditional composability notions are too restrictive for secret sharing based secure computation and using them directly yields protocols with suboptimal behaviour or results in settings where it is hard to manage a large set of protocols. We propose a new input privacy notion that is sufficient for protocols that output secret shared elements and allows for more efficient protocols. The idea of this chapter is to give an intuitive introduction to the necessary building blocks and choices, full formalization of the introduced ideas can be found in [7].

3.1 Security definitions

Security of secure multi-party computation protocols is defined by specifying an ideal functionality. The ideal functionality or a trusted third party is a secure environment where all parties can input their secret inputs and that only gives out the desired result. This corresponds to the best case scenario where each party only sees its own inputs and the final output of the computation. In case any other special behaviour is desired, the ideal functionality can be tweaked to accommodate for this as well. For a real protocol, the ideal functionality is a specification of the desired properties for both correctness and security. A protocol is secure if no adversary can efficiently distinguish whether it is interacting with other parties in a real protocol or with a simulator and the ideal functionality. The simulator is necessary because the parties in the real protocol exchange messages that are not present in the ideal functionality setting and the simulator has to create these messages based on the information available in the ideal world setting.

A secure ideal functionality of a secure multi-party protocol that operates on secret shares is a machine that gets the shared values as an input, restores the input values and computes the output in plain. In the end it shares the output and gives the resulting shares back. The important part in such specification is that there is no connection between the value of party's input shares and the output share.

3.2 Composability

A secure protocol is said to be composable if it retains its security properties when executed in arbitrary combination with other computations. The most commonly used formalism of protocol composition is universal composability (UC) [8] which is very broad and allows to encompass different cryptographic functionalities. However, our work relies on the reactive simulatability (RSIM) formalism [31] that is less expressive than UC in general, but allows to easily describe SMC functionalities. Notably, all of the control over network scheduling is given to the adversary, but otherwise the channels are secure and authenticated. As usual, security is defined by comparing the ideal world specification to the actual real world protocol.

A common approach of a protocol designer would be to make all core protocols secure under composition and then combine them arbitrarily for larger functionalities. However, in the following we define an input privacy property that is weaker than security but is sufficient for most core protocols in secure computation and allows for more efficient protocols. Our approach on one hand seemingly contradicts the common arithmetic black box (ABB) [11] approach of formalizing security of SMC but can also be seen as an extra layer to prove that a set of proposed real world protocols implement the ABB without the need to always consider all protocols together. Similarly to the ABB model, we also obtain a setting where an ideal functionality of a composed protocol is a single unit. As a benefit, input private protocols can be more efficient than their secure counterparts and providing proofs of input privacy is often easier.

The core of our result is the formalization of ordered composition where the ordering is defined with respect to data dependency and not necessarily the time of execution. Hence, our definition is complementary to the existing composability definitions that proposes additional restrictions but can be used in combination with previous definitions. We aim for universally composable protocols with ordered composition.

Definition 3.2.1. *(Ordered and fully ordered composition) Two protocols P_1 and P_2 are in ordered composition $P_1 \to P_2$ if there is no data-flow from P_2 to P_1. The ordered composition is fully ordered if all the outputs of P_1 are inputs to P_2 and not used elsewhere.*

This formalism is necessary to enforce the requirement that only intermediate values of the composed protocol are allowed to be less secure than required by the traditional UC security definition. Especially, we need to establish a special role for the last protocol that provides the outputs of the computation and therefore has to enforce more guarantees. As the main result, we show that the fully ordered composition of an input private and a secure protocol is secure in the passive model, provided that all outputs of the composed protocol are produced by the secure protocol. In addition, it is important to note that a suitable ordering of the protocols in arithmetic or boolean circuit can always be found based on the topological order as such circuits are acyclic directed graphs.

3.3 Input Privacy

A protocol is input private if all that a party learns could be deduced from its inputs. The respective ideally private functionality is such that only allows party to see its input. Essentially, the difference between proving the two properties is that in case of security the simulation uses inputs and outputs of the corrupted parties whereas in case of privacy we only use the inputs. Hence, it captures the idea that adversary sees nothing other than its own inputs, with a special focus on keeping the inputs of other parties hidden.

Intuitively, input privacy property guarantees that the secret inputs of honest parties remain classified throughout the computation. Particularly, this restricts the notion to apply only to protocols that do not provide a public output that depends on the inputs. However, in case of secret sharing, each party has a share as output and this share on its own does not give any information about the value of the output nor the inputs. Hence, we can consider input privacy as a desired property for protocols that output shared elements. Security is still required in case any output is made public. In general, secure functionality may not be input private, nonetheless, any secure secret sharing based protocol that outputs shared elements is also input private thanks to the security of the sharing scheme.

Input privacy is a weaker notion than security with respect to integrity and confidentiality of the outputs. Especially, an adversary may be able to control the outputs or just learn them as long as this does not reveal the inputs. However, the main vulnerability is that using private outputs without care in the following protocols might leak information about the inputs. Nonetheless, we show that a combination with other input private protocols gives an input private composition and finishing this combination with a secure protocol results in a secure composed protocol.

3.4 Security of the Composed Protocols

A key aspect to meaningfully combine secure and input private protocols is to ensure the correctness of the input private part of the composition as correctness is required from secure composition. However, it can be relaxed when secure finalizing protocols do not use the outputs of the private part. On the other hand, secure final protocol has to be restricted not to reveal too much information about its input shares. These requirements are formalized in notions called *output predictability* and *joint output predictability*.

Informally, a composition is said to be output predictable if there exists a predictor that based on the inputs of the composition can produce equivalent outputs. However, these have to be indistinguishable from the outputs of the real composition in a setting where the adversary observes the run of the composition but not the outputs of the predictor. Moreover, a composition of the real input private protocol and secure ideal functionality and the composition of both ideal functionalities are required to have the same predictor for jointly predictable outcome. Joint predictability is required in order to ensure that the private protocol is correct with respect to its ideal specification. The most commonly used composition contains any correct deterministic private functionality finished by a secure resharing protocol and such composition is always jointly output predictable. In general, for deterministic functionalities, correctness and output predictability imply joint output predictability and therefore this does not introduce significant difficulties to the proof.

The informal statements of the two main theorems that ensure the security of the proposed proof framework are as follows.

Theorem 3.4.1. *A fully ordered composition of input private and secure protocol with jointly predictable outcome where all outcomes result from secure protocol is secure.*

Theorem 3.4.2. *An ordered composition of input private protocols is input private.*

These results ensure that it is allowed to combine private protocols and that secure protocols are only required in the end of the computation. Commonly security is needed for resharing just before publishing or storing the outcome. The efficiency gain of this approach is significant for protocols with many intermediate steps that would otherwise need resharing after many components. In addition, composition of private protocols into a large private protocol is no harder to achieve than secure composition.

3.5 Conclusion

This chapter suggested that it is possible to use input private protocols instead of secure protocols in SMC. Furthermore, combining them together results in an input private protocol meaning that input privacy is a composable property. However, most importantly, input privacy as an intermediate goal does not give any penalties as computation can be finished by a secure protocol to turn it into a composable secure protocol that can be used in arbitrary combinations with other protocols. Current work does not consider active security which adds more complexity to formalizing and proving output predictability.

The work described in this chapter appears in all details in paper [7] and is also included as Appendix F of this document.

Chapter 4

Verifiable SMC through Preprocessing

In this chapter we consider verifiable SMC. Instead of using a protocol secure against active adversary, one may use a more efficient protocol secure against passive adversary, and at the end of the computation check which parties have followed the protocol and which have not. Such verification makes sense if the majority of parties are honest, and this approach may be more efficient than actively secure SMC protocols.

In [5, Chap.5], we have reported on the verification mechanism based on linear probabilistically checkable proofs [25]. We have now found that preprocessing can significantly speed up and conceptually simplify the verification. Our new results are presented in this chapter. The full description of our work can be found in [26], which is also included in Appendix C.

Similarly to [5, Chap.5], in this chapter we present a transformation that allows to convert a protocol secure against a passive adversary to a protocol secure against a covert adversary (one that acts as active as far as the probability of being caught is non-negligible).

4.1 Initial Settings

There are n parties, at most $t < n/2$ of whom are corrupted. The protocol has r rounds, where the ℓ-th round computations of the party P_i, the results of which are sent to the party P_j, are given by an arithmetic circuit C_{ij}^{ℓ} over rings $\mathbb{Z}_{2^{n_1}}, \ldots, \mathbb{Z}_{2^{n_K}}$ ($n_i < n_j$ for $i < j$). We define the following gates for such a circuit:

- The operations + (addition) and $*$ (multiplication) over rings $\mathbb{Z}_{2^{n_1}}, \ldots, \mathbb{Z}_{2^{n_K}}$.
- The operations trunc and zext are between rings. Let $x \in \mathbb{Z}_{2^{n_x}}$, $y \in \mathbb{Z}_{2^{n_y}}$, $n_x < n_y$.
  - $x = \mathsf{trunc}(y)$ computes $x = y \mod 2^{n_x}$, going from a larger ring to a smaller ring.
  - $y = \mathsf{zext}(x)$ takes x and uses exactly the same value $y = x$ in $\mathbb{Z}_{2^{n_y}}$. This can be treated as taking the n_x bits of x and extending them with zero bits to n_y bits.
- The operation bits from an arbitrary ring $\mathbb{Z}_{2^n}$ to $(\mathbb{Z}_2)^n$ performs a bit decomposition. Although bit decomposition can be performed by other means, we introduce a separate operation, as it is reasonable to implement a faster verification for it.

More explicit gate types can be added to the circuit, but the ones presented here are sufficient for any computation, and adding verification for more gate types is quite straightforward.

4.2 Verifying Basic Gate Operations

Formally, the gate operations of Sec. 4.1 are verified as follows.

1. The bit decomposition operation $(z_0, \ldots, z_{n-1}) := \mathsf{bits}(z)$:
   check $z = z_0 + z_1 \cdot 2 + \cdots + z_{n-1} \cdot 2^{n-1}$;
   check $\forall j : z_j \in \{0, 1\}$.

2. The transition from $\mathbb{Z}_{2^m}$ to a smaller ring $\mathbb{Z}_{2^n}$: $z := \mathsf{trunc}(x)$:
   check $z = x \mod 2^n$ (where mod can be computed locally since 2^n divides 2^m).

3. The transition from $\mathbb{Z}_{2^n}$ to a larger ring $\mathbb{Z}_{2^m}$: $z := \mathsf{zext}(x)$:
   compute locally the shares of $y := \mathsf{trunc}(z)$ from z;
   check $x = y$;
   check $z = z_0 + z_1 \cdot 2 + \ldots + z_{n-1} \cdot 2^{n-1}$;
   check $\forall j : z_j \in \{0, 1\}$.

In this way, all the ring-specific operations have been reduced to computing linear combinations, mod operations, and equality checks. After all the inputs and the outputs of the circuit are shared amongst the verifiers (the way in which it is done is a part of our protocol), the linear combinations and mod operations can be computed locally on shares. The equalities may be verified afterwards succinctly in parallel. However, the initial circuit may still contain some multiplication gates that cannot be computed locally and hence would provide a large computational overhead for the verifiers. We would like to get rid of all the multiplications, and we do it using *Beaver triples* (preshared triples of random ring elements r_x, r_y, r_{xy} such that $r_x * r_y = r_{xy}$). For example, if we want to multiply $x * y$, and a triple (r_x, r_y, r_{xy}) is already precomputed and preshared, we may first compute and publish $x' := x - r_x$ and $y' := y - r_y$ (x' and y' leak no information about x and y), and then compute the linear combination $x * y = (x' + r_x) * (y' + r_y) = x'y' + r_x y' + x' r_y + r_x r_y = x'y' + r_x y' + x' r_y + r_{xy}$.

For each multiplication gate of C_{ij}^{ℓ}, a Beaver triple is generated in the corresponding ring. The triple is known by the prover, and is also shared amongst the verifiers. The verifiers may use these shared triples to make the multiplication local. The ordinary use of Beaver triples [12, 9, 1] requires declassifying two ring elements x' and y' for each multiplication, which in our case be done in one round once for the entire computation since all these values are known by the prover already before the verification starts.

4.3 Assumptions

Our verification uses the following assumptions:

- Secure point-to-point channels between each pair of parties.
- Broadcast channels between subsets of parties.
- Functionality that allows to prove to third parties which messages one received during the protocol, and to further transfer such revealed messages. We use the same solution as in [5, Chap.5].
- Functionality that generates shared randomness and Beaver triples in the preprocessing phase (a possible implementation of this functionality that uses honest majority assumption is given in [26]).

4.4 Protocol Outline

The protocol Π_{vmpc} implementing $\mathcal{F}_{vmpc}$ consists of n machines $M_1, \ldots, M_n$ doing the work of parties $P_1, \ldots, P_n$, and the underlying transmission and randomness generation functionalities. The internal state of each M_i contains a bit-vector mlc_i of length n where M_i marks which other parties are acting maliciously. Some t of n parties are assigned to be *verifiers*, and the set of such parties is denoted by $\mathcal{V}$. The goal of the prover is to prove its honestness to each subset of $(t+1)$ verifiers (the number of proofs is thus exponential in the number of parties). Due to the honest majority assumption, at least one of these subsets consists only of honest provers. The proof is accepted iff all the verifier subsets accept the proof.

The protocol Π_{vmpc} runs in six phases: preprocessing, initialization, execution, message commitment, verification, and accusation. For simplicity, we describe each phase as it is defined for one proof (for one fixed set $\mathcal{V}$). These phases are exactly the same for all the subsets of $t' := t + 1$ verifiers, sharing no unique randomness, and hence they can be treated as independent proofs.

Preprocessing. This is a completely offline preprocessing phase that can be performed before any inputs are known. For each multiplication gate of C_{ij}^{ℓ}, a Beaver triple is constructed and shared amongst the verifier set. The randomness vectors $\boldsymbol{r}_i$ used in the initial protocol is also shared amongst the verifiers.

Initialization. In the initialization phase, the inputs $\boldsymbol{x}_i$ are committed. The consistency of input shares is not verified, since using inconsistent shares would only make the prover's own life more difficult.

Execution. The parties run the original protocol as before, just using underlying transmission functionality. If any party starts misbehaving straightforwardly, then the transmission functionality forces the parties to go directly to the accusation phase.

Message commitment. All the n parties finally commit their sent messages $\boldsymbol{c}_{ij}^{\ell}$ for each round $\ell \in [r']$ by sharing them to $\boldsymbol{c}_{ij}^{\ell k}$ and sending these shares to the other parties. Let $\boldsymbol{v}_{ij}^{\ell} = (\boldsymbol{x}_i \| \boldsymbol{r}_i \| \boldsymbol{c}_{1i}^{1} \| \cdots \| \boldsymbol{c}_{ni}^{\ell-1} \| \boldsymbol{c}_{ij}^{\ell})$ be the vector of inputs and outputs to the circuit C_{ij}^{ℓ} that M_i uses to compute the ℓ-th message to M_j. After this phase, M_i has shared $\boldsymbol{v}_{ij}^{\ell}$ among the t' verifier parties. Let $\boldsymbol{v}_{ij}^{\ell k}$ be the share of $\boldsymbol{v}_{ij}^{\ell}$ given to machine M_k.

The proving party now also publishes all the public Beaver triple communication values that are needed to eliminate the multiplications and make the verification one-round. The triples that are not related to communication are not a part of message commitment, and they are published separately.

Verification phase. The proving party publishes all the remaining public Beaver triple values, and also a certain number of bits $\boldsymbol{z}_{ij}^{\ell k}$ for each zext, trunc, and bits operation. Each verifier M_k computes the prover's circuit on its local shares $\boldsymbol{v}_{ij}^{\ell k}$ and $\boldsymbol{z}_{ij}^{\ell k}$. Due to preshared Beaver triples, the computation of addition and multiplication gates is local, and, hence, communication between the verifiers is not needed.

The correctness of operations $(z_1, \ldots, z_{n_e}) := \mathsf{bits}(z)$, $z = \mathsf{zext}(x)$, and $z = \mathsf{trunc}(x)$ is verified as shown in Sec. 4.2. The condition $\forall j : \ z_j \in \{0, 1\}$ can be verified as $z_j^2 = z_j$ (which in turn requires a Beaver triple for computing z_j^2), but there is also another more efficient method for this (see [26] for details). Now, only checks of the form $x - y = 0$ and $z_0 + z_1 \cdot 2 + \ldots + z_{n_e-1} \cdot 2^{n_e-1} - z = 0$ are left. Such checks are just linear combinations of the shared values. Since the parties cannot verify locally if the shared value is 0, they postpone these checks to the last round.

For each Beaver triple, the prover has published $x' = (x - r_x)$ and $y' = (y - r_y)$, but the verifiers have no reason to trust him. The checks $x - x' - r_x = 0$ and $y - y' - r_y = 0$ are also postponed.

Finally, the verifiers come up with the shares $\bar{c}_{ij}^{\ell k}$ of the values $\bar{c}_{ij}^{\ell}$ that should be the outputs of the circuits. The verifiers have to check $c_{ij}^{\ell} - \bar{c}_{ij}^{\ell} = 0$.

In general, the verifies have a set the linear combinations $A_1\boldsymbol{x}_1 = 0, \ldots, A_K\boldsymbol{x}_K = 0$, where $A_i\boldsymbol{x}_i = 0$ has to be checked in $\mathbb{Z}_{2^{n_i}}$. The shares of $\boldsymbol{d}_i^k := A_i\boldsymbol{x}_i^k$ are computed locally. If the prover is honest, then the vectors $\boldsymbol{d}_i^k$ are just shares of a zero vector and, hence, can be revealed without leaking any information.

If the total number of parties is 3, then there is a single verifying set of 2 verifiers. They want to check if $\mathbf{0} = \boldsymbol{d}_i = \boldsymbol{d}_i^1 + \boldsymbol{d}_i^2$, which is equivalent to checking whether $\boldsymbol{d}_i^1 = -\boldsymbol{d}_i^2$. For this, take a collision-resistant hash function and publish $h_{ij}^{\ell 1} := h((\boldsymbol{d}_1^1 \| \ldots \| \boldsymbol{d}_K^1))$ and $h_{ij}^{\ell 2} := h(-(\boldsymbol{d}_1^2 \| \ldots \| \boldsymbol{d}_K^2))$. Check $h_{ij}^{\ell 1} = h_{ij}^{\ell 2}$.

Accusation. Finally, each party outputs the set of parties that it considers malicious.

The formal UC proof for the real functionality can be found in [26].

4.5 Conclusion

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. Using preprocessing phase, this scheme is more efficient than the one proposed in [5], especially for 3-party computation.

Similarly to [5], 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. This would increase the total number of rounds, but not the total communication.

Chapter 5

Privacy-preserving Parallel Array Access

Many algorithms that we may wish to implement in a privacy-preserving manner make use of arrays, which they access according to non-trivial access patterns — patterns, which are not known during compile-time, but may depend on the data the algorithm operates on. In a privacy-preserving implementation, the index, according to which the array is accessed, may itself need to be private. Accessing an array according to a private index is a complex operation, and a naive implementation has the overhead proportional to the length of the array, as it touches all elements of the array to make the exclusion of certain values of the index impossible through observing, which positions were touched. More complex approaches are based on implementations of Oblivious RAM (ORAM) [16] on top of SMC [17, 14, 29, 15, 10, 21]. These approaches have the overhead $O(\log^c n)$ for an array of length n, with $c \approx 3$ in typical implementations.

In UaESMC, most of our work has been on SMC based on secret sharing, as the achievable raw performance of such SMC protocols is the best known. On the other hand, an efficient application of such type of SMC requires the computed algorithm to be highly parallelizable, as the computation of each operation (except the one, according to which the secret sharing scheme is homomorphic) requires communication among parties and thus the round complexity of the protocol is proportional to the depth of the circuit expressing the algorithm. If the algorithm we want to compute contains a large number of array accesses with private indices, then these accesses have to occur in parallel as well. In this chapter, and in [24] (attached to this deliverable) we show that in this case, the touching of all elements of the array can be amortized among all accesses, leading to potentially much smaller overheads per access.

5.1 Required lower-level protocols

We need the following lower-level operations to be implemented in the SMC framework we're working with, in order to implement our parallel oblivious array read and write protocols. We note that in Sharemind, the implementations of all these operations exist and are highly efficient.

- The homomorphic operation, which must be a commutative group operation over the set of possible private values. This operation must be free and its inverse must also be free. If the private values come from a ring or a field, then this operation is the addition.
- Multiplication of private values. Equality and less-than checks.
- Oblivious shuffles — private permutations of values — together with operations to *apply* and *unapply* them to vectors of values. Given an oblivious shuffle $[\![\sigma]\!]$ for m elements, and a private vector $([\![v_1]\!], \ldots, [\![v_m]\!])$, the apply operation returns a new private vector $([\![v'_1]\!], \ldots, [\![v'_m]\!])$ with $v'_i = v_{\sigma(i)}$. The unapply operation similarly returns a private vector$([\![v'_1]\!], \ldots, [\![v'_m]\!])$, but this time the equalities $v'_{\sigma(i)} = v_i$ hold. It must be also possible to generate a random shuffle (i.e. unknown to computing parties) for a given number of elements, and to compose a private shuffle and a public shuffle (yielding a new private shuffle).

Algorithm 5: Reading n values from the private array

Data: A private vector $[\![\boldsymbol{v}]\!]$ of length m
Data: A private vector $[\![\boldsymbol{z}]\!]$ of length n, with $1 \leq z_i \leq m$ for all i
Result: A private vector $[\![\boldsymbol{w}]\!]$ of length n, with $w_i = v_{z_i}$ for all i

1 **foreach** $i \in \{1, \ldots, m\}$ **do** $[\![t_i]\!] \leftarrow i$;
2 **foreach** $i \in \{1, \ldots, n\}$ **do** $[\![t_{m+i}]\!] \leftarrow [\![z_i]\!]$;
3 $[\![\sigma]\!] \leftarrow \mathsf{sortperm}([\![\boldsymbol{t}]\!])$
4 $[\![\boldsymbol{v'}]\!] \leftarrow \mathsf{prefixsum}^{-1}([\![\boldsymbol{v}]\!])$
5 **foreach** $i \in \{1, \ldots, m\}$ **do** $[\![u_i]\!] \leftarrow [\![v'_i]\!]$;
6 **foreach** $i \in \{1, \ldots, n\}$ **do** $[\![u_{m+i}]\!] \leftarrow 0$;
7 $[\![\boldsymbol{u'}]\!] \leftarrow \mathsf{unapply}([\![\sigma]\!]; \mathsf{prefixsum}(\mathsf{apply}([\![\sigma]\!]; [\![\boldsymbol{u}]\!])))$
8 **foreach** $i \in \{1, \ldots, n\}$ **do** $[\![w_i]\!] \leftarrow [\![u'_{m+i}]\!]$;
9 **return** $[\![\boldsymbol{w}]\!]$

- Declassification of private values in the middle of computation, such that their values become known to computing parties.

We note that with less-than checks, oblivious shuffles and declassification, a highly efficient protocol for sorting a private array becomes possible [19], as long as it is known at compile-time that all elements of the array are different. Moreover, the sorting operation returns an oblivious shuffle that represents the permutation necessary to sort the given private array. In fact, in this chapter we make the computation of the sorting permutation the primary operation and denote it by $[\![\sigma]\!] \leftarrow \mathsf{sortperm}([\![\boldsymbol{v_1}]\!], \ldots, [\![\boldsymbol{v_m}]\!])$; the actual sorting can be performed by applying $[\![\sigma]\!]$ to the vector $[\![\boldsymbol{v}]\!]$. We require sorting to be stable; in general this means that we have to add an extra component to the elements of the vector $[\![\boldsymbol{v}]\!]$ which ensures their inequality.

5.2 Protocol for reading

In Alg. 5, we present our protocol for obliviously reading several elements of an array. Given a vector $\boldsymbol{v}$ of length m, we let $\mathsf{prefixsum}(\boldsymbol{v})$ denote a vector $\boldsymbol{w}$, also of length m, where $w_i = \sum_{j=1}^{i} v_j$ for all $i \in \{1, \ldots, m\}$, and where $\sum$ denotes the application of the homomorphic operation named in Sec. 5.1. Computing $\mathsf{prefixsum}([\![\boldsymbol{v}]\!])$ is a free operation. The inverse operation $\mathsf{prefixsum}^{-1}$ is also free: $v_1 = w_1$ and $v_i = w_i - w_{i-1}$ for all $i \in \{2, \ldots, m\}$.

We see that in Alg. 5, the permutation σ orders the indices which we want to read, as well as the indices $1, \ldots, n$ of the "original array" $\boldsymbol{v}$. Due to the stability of the sort, each index of the "original array" ends up before the reading indices equal to it. In $\mathsf{apply}(\sigma, \boldsymbol{u})$, each element v'_i of $\boldsymbol{v'}$, located in the same position as the index i of the "original array" in sorted $\boldsymbol{t}$, is followed by zero or more 0-s. The prefix summing restores the elements of $\boldsymbol{v}$, with the 0-s also replaced with the element that precedes them. Unapplying σ restores the original order of $\boldsymbol{u}$ and we can read out the elements of $\boldsymbol{v}$ from the latter half of $\boldsymbol{u'}$. A small example is presented in Fig. 5.1.

The reading protocol is secure due to the composability of lower-level protocols. Its asymptotic communication complexity is $O((m+n)\log(m+n))$, and round complexity is $O(\log(m+n))$, both dominated by the sorting operation in line 3 of Alg. 5. When using Alg. 5 in larger applications, a significant optimization is possible, if we note that the sorting operation requires the indices $[\![\boldsymbol{z}]\!]$ of the elements we want to read, but does need the actual values in $[\![\boldsymbol{v}]\!]$. If several reads are performed according to the same indices, then the sorting has to be performed only once.

Let $\boldsymbol{v} = (1, 4, 9, 16, 25)$. Let $\boldsymbol{z} = (3, 2, 4, 3)$. The intermediate values are the following.

- $\boldsymbol{t} = (1, 2, 3, 4, 5, 3, 2, 4, 3)$
- σ is the permutation

1	2	3	4	5	6	7	8	9
1	2	7	3	6	9	4	8	5

  meaning that e.g. the 3rd element in the sorted vector is the 7th element in the original vector.
- $\boldsymbol{v}' = (1, 3, 5, 7, 9)$ and $\boldsymbol{u} = (1, 3, 5, 7, 9, 0, 0, 0, 0)$.
- After applying σ to $\boldsymbol{u}$, we obtain the vector $(1, 3, 0, 5, 0, 0, 7, 0, 9)$.
- After prefixsumming, we get the vector $(1, 4, 4, 9, 9, 9, 16, 16, 25)$. Denote it with $\boldsymbol{y}$.
- After applying the inverse of σ, we get $\boldsymbol{u}' = (1, 4, 9, 16, 25, 9, 4, 16, 9)$. Indeed, to find e.g. u'_7, we look for "7" in the lower row of the description of σ. We find "3" in the upper row, meaning that $u'_7 = y_3$.
- Finally, we return the last n elements of $\boldsymbol{u}'$, which are $\boldsymbol{w} = (9, 4, 16, 9)$.

All values are private.

Figure 5.1: Example of private reading according to Alg. 5

5.3 Protocol for writing

The protocol for writing is conceptually (though not complexity-wise) simpler than the reading protocol. It is presented in Alg. 6. When specifying the desired functionality of the protocol, we have to think of resolving the simultaneous writes to the same element of the updated array. For this purpose, each write is accompanied by a *priority*. Of all writes to the same position, only the one with largest priority goes through. In the algorithm, the existing values in the array receive the minimum priority. We remark that there may be applications, where it makes sense to allow the writing requests to also have lower priorities than keeping existing values, and in this case the line 6 in Alg .6 has to be appropriately modified.

In Alg. 6, the indices and priorities are sorted (with higher-priority elements coming first, and existing elements receiving default priorities). The vector $[\![\boldsymbol{b}]\!]$ is used to indicate the highest-priority position for each index: $b_i = 0$ iff the i-th element in the vector $\boldsymbol{j}'$ is the first (hence the highest-priority) value equal to j'_i. All equality checks in line 10 can be done in parallel. Performing the sort in line 11 moves the highest-priority values to the first m positions. The sorting is stable, hence the values correspond to the indices $1, \ldots, m$ in this order. We thus have to apply the shuffles induced by both sorts to the vector of values $\boldsymbol{v}' = \boldsymbol{v} \| \boldsymbol{w}$, and take the first m elements of the result.

Similarly to the reading protocol, the writing protocol is secure and its asymptotic complexity is $O((m+n)\log(m+n))$ in communication and $O(\log(m+n))$ in rounds. Also, it allows the same kind of optimization for applications that write many times to the same indices, according to the same priorities. The expensive sorting operations do not require the actual values in the existing array, nor the values that are written to it. The sorting in line 11 can be performed with $O(m+n)$ communication and $O(1)$ rounds, as shown in [24], but this does not reduce the overall complexity of the protocol.

5.4 Oblivious Extended Permutations

In *private function evaluation* (PFE), the parties of an SMC want to compute a function with private description. PFE can be realized through the private computation of a universal circuit, but such circuits are large compared to circuits they can express [22]. More recently, PFE has been realized by splitting the task of oblivious circuit evaluation into two parts — (1) obliviously evaluating the gates, and (2) hiding the

Algorithm 6: Obliviously writing n values to a private array

Data: Private vectors $[\![\boldsymbol{j}]\!]$, $[\![\boldsymbol{v}]\!]$, $[\![\boldsymbol{p}]\!]$ of length n, where $1 \leq j_i \leq m$ for all i
Data: Private array $[\![\boldsymbol{w}]\!]$ of length m
Result: Updated $\boldsymbol{w}$: values in $\boldsymbol{v}$ written to indices in $\boldsymbol{j}$, if priority in $\boldsymbol{p}$ is the highest for this position

1 **foreach** $i \in \{1, \ldots, n\}$ **do**
2 $\quad [\![j'_i]\!] \leftarrow [\![j_i]\!]$
3 $\quad [\![p'_i]\!] \leftarrow -[\![p_i]\!]$
4 **foreach** $i \in \{1, \ldots, m\}$ **do**
5 $\quad [\![j'_{n+i}]\!] \leftarrow i$
6 $\quad [\![p'_{n+i}]\!] \leftarrow MAX\_VALUE$
7 $[\![\sigma]\!] \leftarrow \mathsf{sortperm}([\![\boldsymbol{j'}]\!], [\![\boldsymbol{p'}]\!])$
8 $[\![\boldsymbol{j''}]\!] \leftarrow \mathsf{apply}([\![\sigma]\!]; [\![\boldsymbol{j'}]\!])$
9 $[\![b_1]\!] \leftarrow 0$
10 **foreach** $i \in \{2, \ldots, N\}$ **do** $[\![b_i]\!] \leftarrow [\![j''_i]\!] \stackrel{?}{=} [\![j''_{i-1}]\!]$;
11 $[\![\tau]\!] \leftarrow \mathsf{sortperm}([\![\boldsymbol{b}]\!])$
12 **foreach** $i \in \{1, \ldots, n\}$ **do** $[\![v'_i]\!] \leftarrow [\![v_i]\!]$;
13 **foreach** $i \in \{1, \ldots, m\}$ **do** $[\![v'_{n+i}]\!] \leftarrow [\![w_i]\!]$;
14 $[\![\boldsymbol{w'}]\!] \leftarrow \mathsf{apply}([\![\tau]\!]; \mathsf{apply}([\![\sigma]\!]; [\![\boldsymbol{v'}]\!]))$
15 **foreach** $i \in \{1, \ldots, m\}$ **do** $[\![w_i]\!] \leftarrow [\![w'_i]\!]$;
16 **return** $[\![\boldsymbol{w}]\!]$

topology of the circuit in a manner that allows the outputs of the gates to be passed to the inputs of next gates [30]. While the implementation of (1) using SMC is straightforward, *oblivious extended permutations* have been introduced for (2) in [30]. An extended permutation from m elements to n elements is a function $\phi : \{1, \ldots, n\} \to \{1, \ldots, m\}$ (note the contravariance). It is intended to be applied to an m-element vector $(v_1, \ldots, v_m)$, producing an n-element vector $(v_{\phi(1)}, \ldots, v_{\phi(n)})$. An OEP is an extended permutation ϕ realized in a manner that preserves the guarantees of SMC regarding the privacy of ϕ.

Our parallel reading algorithm, Alg. 5, essentially constructs an OEP from the vector of indices $[\![\boldsymbol{z}]\!]$ (lines 1–3), and then applies it to the vector $[\![\boldsymbol{v}]\!]$ (the rest of the algorithm). Both the construction and the application are faster, both asymptotically and in practice, than any other previously published protocol. We note that prior to the parallel access protocols described in this chapter, we had proposed a different protocol for OEP [23] (attached to this deliverable), which had worse complexities.

5.5 Discussion

Using our protocols, the cost of n parallel data accesses is $O((m+n)\log(m+n))$, where m is the size of the vector from which we're reading values. Dividing by n, we get that the overhead of one access is $O((1+\frac{m}{n})\log(m+n))$. In practice, the cost overhead depend a lot on our ability to perform many data accesses in parallel. Fortunately, this goal to parallelize coincides with one of the design goals for privacy-preserving applications in general, at least if the representation of private values is based on secret sharing. Parallelization allows to reduce the number of communication rounds necessary for the application, reducing the performance penalty caused by network latency.

Suppose that our application is such that on average, we can access in parallel a fraction of $1/f(m)$ of the memory it uses (where $1 \leq f(m) \leq m$). Hence, we are performing $m/f(m)$ data accesses in parallel, requiring $O(m \log m)$ work in total, or $O(f(m) \log m)$ for one access. For the implementations of Oblivious RAM over SMC, the reported overheads are at least $O(\log^3 m)$. Hence our approach has better asymptotic complexity for applications where we can keep $f(m)$ small.

There exists a sizable body of efficient algorithms for an abstraction of parallel computation, Parallel

Random Access Machines (PRAM). In this model, an arbitrary number of processors are available, executing synchronously and sharing common memory. There exist several subclasses of PRAM, depending on how the read/write conflicts to the same memory location are resolved. A CRCW PRAM allows many processors to read and write the same location at the same time. In priority-CRCW PRAM, concurrent writes to the same location are resolved by (numeric) priorities assigned to the write: only the writing operation with the highest priority gets through [20]. Using our parallel reading and writing protocols, any algorithm for priority-CRCW PRAM can be implemented on an ABB, as long as the control flow of the algorithm does not depend on private data. A goal in designing PRAM algorithms is to make their running time polylogarithmic in the size of the input, while using a polynomial number of processors. There is even a large class of tasks, for which there exist PRAM algorithms with logarithmic running time.

An algorithm with running time t must on each step access on average at least $1/t$ fraction of the memory it uses. A PRAM algorithm that runs in $O(\log m)$ time must access on average at least $\Omega(1/\log m)$ fraction of its memory at each step, i.e. $f(m)$ is $O(\log m)$. When implementing such algorithm on top of SMC using the reading and writing protocols presented in this note, we can say that the overhead of these protocols is at most $O(\log^2 m)$. For algorithms that access a larger fraction of their memory at each step (e.g. the Bellman-Ford algorithm for finding shortest paths in graphs; for which also the optimizations of reading and writing protocols described above apply), the overhead is even smaller.

Bibliography

[1] 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.

[2] Dan Bogdanov, Yiannis Giannakopoulos, Roberto Guanciale, Dilian Gurov, Liina Kamm, Peeter Laud, Alisa Pankova, Pille Pruulmann-Vengerfeldt, Pille Pullonen, Riivo Talviste, Yiannis Tselekounis, and Jan Willemson. Scientific Progress Analysis and Recommendations, January 2014. UaESMC Deliverable 5.2.2.

[3] 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.

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

[5] 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.

[6] Dan Bogdanov, Liina Kamm, Sven Laur, and Ville Sokk. Rmind: a tool for cryptographically secure statistical analysis. *IACR Cryptology ePrint Archive*, 2014:512, 2014.

[7] Dan Bogdanov, Peeter Laud, Sven Laur, and Pille Pullonen. From Input Private to Universally Composable Secure Multi-party Computation Primitives. In Anupam Datta and Cedric Fournet, editors, *IEEE 27th Computer Security Foundations Symposium, CSF 2014, Vienna, Austria, 19-22 July, 2014*, pages 184–198. IEEE, 2014.

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

[9] 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.

[10] 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.

[11] 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.

[12] 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.

[13] Wenliang Du and Mikhail J. Atallah. Privacy-preserving cooperative scientific computations. In *Proceedings of the 14th IEEE Workshop on Computer Security Foundations*, CSFW '01, pages 273–, Washington, DC, USA, 2001. IEEE Computer Society.

[14] Craig Gentry, Kenny A. Goldman, Shai Halevi, Charanjit S. Jutla, Mariana Raykova, and Daniel Wichs. Optimizing oram and using it efficiently for secure computation. In Emiliano De Cristofaro and Matthew Wright, editors, *Privacy Enhancing Technologies*, volume 7981 of *Lecture Notes in Computer Science*, pages 1–18. Springer, 2013.

[15] Craig Gentry, Shai Halevi, Charanjit Jutla, and Mariana Raykova. Private Database Access With HE-over-ORAM Architecture. Cryptology ePrint Archive, Report 2014/345, 2014. `http://eprint.iacr.org/`.

[16] Oded Goldreich and Rafail Ostrovsky. Software Protection and Simulation on Oblivious RAMs. *J. ACM*, 43(3):431–473, 1996.

[17] S. Dov Gordon, Jonathan Katz, Vladimir Kolesnikov, Fernando Krell, Tal Malkin, Mariana Raykova, and Yevgeniy Vahlis. Secure Two-Party Computation in Sublinear (Amortized) Time. In Ting Yu, George Danezis, and Virgil D. Gligor, editors, *the ACM Conference on Computer and Communications Security, CCS'12, Raleigh, NC, USA, October 16-18, 2012*, pages 513–524. ACM, 2012.

[18] Roberto Guanciale, Dilian Gurov, Peeter Laud, Alisa Pankova, Martin Pettai, and Sander Siim. Algorithms for Large-Scale SMC Problems, July 2015. UaESMC Deliverable 4.2.2.

[19] Koki Hamada, Ryo Kikuchi, Dai Ikarashi, Koji Chida, and Katsumi Takahashi. Practically efficient multi-party sorting protocols from comparison sort algorithms. In Taekyoung Kwon, Mun-Kyu Lee, and Daesung Kwon, editors, *ICISC*, volume 7839 of *Lecture Notes in Computer Science*, pages 202–216. Springer, 2012.

[20] Joseph JáJá. *An Introduction to Parallel Algorithms*. Addison-Wesley, 1992.

[21] Marcel Keller and Peter Scholl. Efficient, Oblivious Data Structures for MPC. In Palash Sarkar and Tetsu Iwata, editors, *Advances in Cryptology - ASIACRYPT 2014 - 20th International Conference on the Theory and Application of Cryptology and Information Security, Kaoshiung, Taiwan, R.O.C., December 7-11, 2014, Proceedings, Part II*, volume 8874 of *Lecture Notes in Computer Science*, pages 506–525. Springer, 2014.

[22] Vladimir Kolesnikov and Thomas Schneider. A practical universal circuit construction and secure evaluation of private functions. In Gene Tsudik, editor, *Financial Cryptography*, volume 5143 of *Lecture Notes in Computer Science*, pages 83–97. Springer, 2008.

[23] Peeter Laud. A private lookup protocol with low online complexity for secure multiparty computation. In Siu Ming Yiu and Elaine Shi, editors, *ICICS 2014*, LNCS. Springer, 2014. To appear.

[24] Peeter Laud. Parallel Oblivious Array Access for Secure Multiparty Computation and Privacy-Preserving Minimum Spanning Trees. *Proceedings of Privacy Enhancing Technologies*, 2015(2):188–205, 2015.

[25] Peeter Laud and Alisa Pankova. Verifiable Computation in Multiparty Protocols with Honest Majority. In Sherman S. M. Chow, Joseph K. Liu, Lucas Chi Kwong Hui, and Siu-Ming Yiu, editors, *Provable Security - 8th International Conference, ProvSec 2014, Hong Kong, China, October 9-10, 2014. Proceedings*, volume 8782 of *Lecture Notes in Computer Science*, pages 146–161. Springer, 2014.

[26] Peeter Laud and Alisa Pankova. Preprocessing-based verification of multiparty protocols with honest majority. Cryptology ePrint Archive, Report 2015/674, 2015. `http://eprint.iacr.org/`.

[27] Peeter Laud and Alisa Pankova. Transformation-based outsourcing of linear equation systems over real numbers. Cryptology ePrint Archive, Report 2015/322, 2015. `http://eprint.iacr.org/`.

[28] Peeter Laud and Jan Willemson. Composable oblivious extended permutations. In Frédéric Cuppens, Joaquín García-Alfaro, A. Nur Zincir Heywood, and Philip W. L. Fong, editors, *Foundations and Practice of Security - 7th International Symposium, FPS 2014, Montreal, QC, Canada, November 3-5, 2014. Revised Selected Papers*, volume 8930 of *Lecture Notes in Computer Science*, pages 294–310. Springer, 2014.

[29] Chang Liu, Yan Huang, Elaine Shi, Jonathan Katz, and Michael W. Hicks. Automating efficient ram-model secure computation. In *2014 IEEE Symposium on Security and Privacy, SP 2014, Berkeley, CA, USA, May 18-21, 2014*, pages 623–638. IEEE Computer Society, 2014.

[30] Payman Mohassel and Seyed Saeed Sadeghian. How to Hide Circuits in MPC: an Efficient Framework for Private Function Evaluation. In Thomas Johansson and Phong Q. Nguyen, editors, *EUROCRYPT*, volume 7881 of *Lecture Notes in Computer Science*, pages 557–574. Springer, 2013.

[31] Birgit Pfitzmann and Michael Waidner. A model for asynchronous reactive systems and its application to secure message transmission. In *IEEE Symposium on Security and Privacy*, pages 184–, 2001.