APPLICATIONS OF SECURE MULTIPARTY COMPUTATION

Cryptology and Information Security Series

The Cryptology & Information Security Series (CISS) presents the latest research results in the theory and practice, analysis and design, implementation, application and experience of cryptology and information security techniques. It covers all aspects of cryptology and information security for an audience of information security researchers with specialized technical backgrounds.

Coordinating Series Editors: Raphael C.-W. Phan and Jianying Zhou

Volume 13

Recently published in this series

Applications of Secure Multiparty Computation

Edited by

Peeter Laud

Cybernetica AS, Ülikooli 2, 51003 Tartu, Estonia

and

Liina Kamm

Cybernetica AS, Ülikooli 2, 51003 Tartu, Estonia

IOS

Press

Amsterdam • Berlin • Tokyo • Washington, DC

LEGAL NOTICE

The publisher is not responsible for the use which might be made of the following information.

PRINTED IN THE NETHERLANDS

Preface

 We generate and gather a lot of data about ourselves and others, both willingly and unwillingly. Much of that data is considered confidential by someone, and its collection, storage and use are regulated by laws, contracts, societal norms or personal care. The restrictions on the use of data can limit the benefits we can obtain from its analysis, even if all the parties involved agree that it would be beneficial to learn the result of a certain computation applied to confidential data. Examples include the statistical analysis of personally identifiable information (PII) of a large number of people, where the results of a well-designed analysis can no longer be linked to a single person, or a small group of persons, but can be used as a basis for sound policy decisions. Examples also include the monitoring of critical infrastructure and reacting to changes in its status, where different parts of the infrastructure are owned by different entities afraid that their trade secrets might leak. Another example considers negotiations among several parties planning a cooperative activity, where the parties are reluctant to bring all the relevant facts to the negotiation table in fear of leaking their trade secrets and harming their position, but the lack of these facts may lead to a sub-optimal contract.

 In all these examples, the parties would like to execute a specific program on their confidential data, and learn the outcome of the result, while making sure that nothing else about the data is leaked. There is cryptographic technology that is capable of providing the parties with exactly this functionality and guarantees. In *secure multiparty computation (SMC)*, parties provide their inputs to a cryptographic protocol that is used to compute a pre-agreed function in such a manner that anything that a party or a sufficiently small coalition of parties sees during the protocol could be deduced from the party's or the coalition's inputs and outputs. SMC has been a subject of academic studies since the 1980s, when the first constructions were proposed and it was shown that any Boolean circuit could be securely evaluated with only a polynomial overhead. As it is possible to express any computation as a Boolean circuit (again, with only a polynomial overhead), it can also be performed securely. Since then, several different SMC techniques have appeared with different communication patterns, different numbers of parties they are able to support, different coalitions of adversarial parties they are able to tolerate, and different adversarial actions they can handle.

 While SMC has been the subject of academic studies for more than 30 years, the first attempts to use it for actual computations only took place in the early 2000s. Indeed, though the first constructions of SMC were theoretically efficient, the constants hidden in their complexity measures were too large to render them practicable. The situation has steadily improved since then and by now, even relatively large computational tasks can be solved securely.

 Academic research has concentrated on improving the times of securely executing Boolean circuits, or arithmetic circuits consisting of addition and multiplication gates. This makes a lot of sense from the theoretical point of view, as these operations are sufficient to express any computation with polynomial overhead. It is also advisable to use these operations when expressing certain combinatorial or numerical computation tasks as circuits. For many other kinds of computations, though, the cost of converting them

into a circuit that only consists of additions and multiplications, together with the overheads of SMC, may be too much for practical applications. There have been relatively few research efforts to

- devise efficient SMC protocols for other operations, e.g. comparisons;
- efficiently combine the existing SMC protocols into secure protocols for other primitive operations used in algorithms, e.g. array access.

In our opinion, these are the advances that will encourage the take-up of SMC techniques in many different areas for many different tasks.

This book describes research, mostly performed over the last few years, that has brought forth these kind of advances. The results described show how certain algorithmic steps can be performed in a privacy-preserving manner, and how to combine these steps into larger privacy-preserving applications. We start this book with two chapters about fundamental SMC techniques. In Chapter 1, we describe the different kinds of protocols for SMC, and the ways they represent private data in a manner that prevents a small coalition from learning anything about the individual values. In Chapter 2 we discuss the composability of SMC protocols. For building large privacy-preserving applications, it is very important to ensure that the protocols can be composed freely or almost freely. Underlying the composability results is a very convenient abstraction of SMC — the ideal functionality called the arithmetic black box (ABB) that allows computing parties to perform operations with the data it stores without revealing it to those parties.

In Chapter 3 we study the society's preparedness to use SMC and adapt its processes around it. We discuss where and how SMC techniques could make the fastest inroads. Following the recommendations given in this chapter can decrease the time it takes to achieve tangible outcomes from SMC research.

We continue with an application area determined in Chapter 3 as one of the most likely to rapidly benefit from SMC techniques. In Chapter 4 we demonstrate methods for privacy-preserving statistical analysis. We explain how the functions and distributions often used in these analyses can be computed by SMC protocols. In Chapter 5 we show how we can avoid not only leaks during the computation, but also leaks through the results of statistical analyses — we show how mechanisms for differential privacy can be implemented through SMC.

Another important primitive in building privacy-preserving applications is oblivious data access — reading and writing the contents of arrays according to secret indices. In Chapter 6 we present efficient and composable methods for this building block. Some uses of these methods are presented in Chapter 7 where we show how to construct privacy-preserving protocols for language processing or business process analysis.

We continue with methods for checking the behavior of the participants of SMC protocols. In Chapter 8 we show how game-theoretic methods can encourage parties to input correct data to multiparty computation protocols. Chapters 9 and 10 describe different post-execution methods for verifying the correctness of the behavior of computing parties. The first of these is a general method where the honest majority can determine all deviating parties. The second method is applicable to tasks where, in practice, verifying the result of a computation can be much simpler than computing it. The method is demonstrated on the task of privacy-preserving linear programming.

In Chapter 11 we explore the limits of another method for privacy-preserving computation — transforming a task on the basis of its algebraic properties, publishing the

transformed task and transforming the solution of the published task back into the solution of the original task. We find that, for linear programming, this method is most likely not applicable, as any transformation that preserves the confidentiality of the original task probably needs to solve the task. Still, the method can be applicable for solving systems of linear equations.

We finish the book with Chapter 12 by reviewing existing practical applications of SMC. In this chapter, we do not describe various research prototypes that have been developed by many different groups, but only the applications that have processed real data. We describe how the data was collected and processed, and how the results were made public.

In September 2014, a workshop for disseminating the results of the UaESMC project, as well as other results on applied SMC took place as a satellite event of ESORICS. In addition to talks by the project partners, there were two talks by the representatives of other research groups, for which we are very grateful. Both speakers have also generously contributed to this book as authors, and their chapters fit in well with the overall theme.

<div align="right">

Peeter Laud and Liina Kamm
Tartu, April 2015

</div>

This page intentionally left blank

Contents

This page intentionally left blank

Chapter 1
Basic Constructions of Secure Multiparty Computation

Peeter LAUD [a], Alisa PANKOVA [a], Liina KAMM [a], and Meilof VEENINGEN [b]

[a] *Cybernetica AS, Estonia*
[b] *Eindhoven University of Technology, Netherlands*

Abstract. In this chapter, we formally define multiparty computation tasks and the security of protocols realizing them. We give a broad presentation of the existing constructions of secure multiparty computation (SMC) protocols and explain why they are correct and secure. We discuss the different environmental aspects of SMC protocols and explain the requirements that are necessary and sufficient for their existence.

Introduction

There are several ⬚ graphy textbooks that rigorously cover the basic definitions and constructions of secure multiparty computation, e.g. [1,2,3]. In this introductory chapter, we do not attempt to repeat this rigorous treatment. Instead, we will give the basic security definitions of SMC and present the major ideas behind different SMC protocols, sufficient for understanding the algorithms and protocols in the rest of this book. We refer to other sources for thorough proofs and discussions on these constructions.

1. Definitions

An SMC protocol for a functionality f allows a number of parties to evaluate f on the inputs they have provided, and learn its outputs without learning anything beyond their own inputs and outputs. Almost all SMC techniques expect f to be expressed as a Boolean or an arithmetic circuit, and process it gate by gate. We can specify a multiparty computation task using the following definitions.

Definition 1 *An* arithmetic circuit *over a ring R is a tuple* $C = (G, V_{in}, V_{out}, \lambda)$*, where*

- $G = (V, E)$ *is a directed acyclic graph, where the incoming edges of each vertex have been linearly ordered;*
- $V_{in} \subseteq \{v \in V \mid \overrightarrow{\deg}(v) = 0\}$ *and* $V_{out} \subseteq V$ *denote the input and output vertices of the circuit (here* $\overrightarrow{\deg}(v)$ *denotes the number of incoming edges of the vertex V);*
- λ *assigns to each* $v \in V$ *an operation* $\lambda(v) : R^{\overrightarrow{\deg}(v)} \to R$*.*

A Boolean circuit can be seen as a special case of Def. 1, where $R = \mathbb{Z}_2$. The semantics of a circuit $C = (G, V_{\text{in}}, V_{\text{out}}, \lambda)$ extend mapping $V_{\text{in}} \to R$ to mapping $V \to R$, thereby assigning values to all vertices in V_{out}.

Definition 2 *A* multiparty computation task *for a set of parties* $\mathbf{P} = \{P_1, \ldots, P_n\}$ *is a tuple* $f = (C, T_{\text{in}}, T_{\text{out}})$, *where* $C = (G, V_{\text{in}}, V_{\text{out}}, \lambda)$ *is an arithmetic circuit,* $T_{\text{in}} : V_{\text{in}} \to \mathbf{P}$ *determines which party provides each input, and* $T_{\text{out}} \subseteq V_{\text{out}} \times \mathbf{P}$ *states which outputs are learned by which parties.*

To solve the multiparty computation task, the parties execute some protocol Π, with the party P_i having an interactive (Turing) machine M_i that implements the steps of P_i in this protocol. At first, the machine M_i receives the inputs x_i from P_i, and in the end, returns the outputs to P_i. Here, x_i is a mapping from the set $T_{\text{in}}^{-1}(P_i)$ to R. We let $\mathbf{x} : V_{\text{in}} \to R$ denote the concatenation of all parties' inputs. Let $\mathbf{x}[P_i]$ denote x_i. For a subset of parties $\mathbf{P}' \subseteq \mathbf{P}$, we let $\mathbf{x}[\mathbf{P}']$ denote the tuple of all $\mathbf{x}[P]$ with $P \in \mathbf{P}'$.

In the threat model of SMC, some parties may be corrupted, but the honest parties do not know which ones. The secrecy of honest parties' inputs has to be protected against the coalition of corrupt parties. Also, the honest parties should still obtain correct outputs despite the actions of corrupt parties. We can formalize both of these requirements through the real/ideal-model paradigm. In this paradigm, we specify our desired properties through a protocol that contains an ideal component that makes these properties "obviously hold". In case of SMC, this ideal component $\mathcal{F}_{\text{SMC}}^f$ collects the inputs of all parties, computes the functionality f, and hands the outputs back to the parties. Correctness and secrecy are obvious, because $\mathcal{F}_{\text{SMC}}^f$ indeed computes f, and each party only gets back its own outputs. The execution of this ideal protocol produces certain outputs for the honest parties, as well as for the adversary (modeled as a Turing machine) controlling all the corrupted parties. The output by the adversary may reflect its guesses about the inputs and outputs of honest parties. In the real protocol, the ideal component is not available and the messages exchanged are different, but all honest parties and the adversary again produce some outputs. The protocol is secure if any outputs produced by the real protocol could also have been produced by the ideal protocol — for any real adversary, there is an ideal adversary, so that their outputs look the same in some sense, even when taking into account the outputs by honest parties. We formalize these notions below.

Definition 3 *The* ideal component *for securely computing the multiparty functionality* f *by n parties is an interactive Turing machine* $\mathcal{F}_{\text{SMC}}^f$ *that works as follows:*

- *On input* x_i *from the i-th party, where* x_i *has the correct type (i.e. it maps* $T_{\text{in}}^{-1}(P_i)$ *to the ring R), it stores* x_i *and ignores further inputs from the i-th party.*
- *After receiving* $x_1, \ldots, x_n$, *it computes* $\mathbf{z} = f(\mathbf{x})$, *where* $\mathbf{z}$ *is a mapping from* V_{out} *to R. Let* z_i *be the restriction of* $\mathbf{z}$ *to* $T_{\text{out}}^{-1}(P_i)$.
- *For all i, the machine* $\mathcal{F}_{\text{SMC}}^f$ *sends* z_i *to the i-th party.*

Definition 4 *Let* $\mathbf{P}_c \subset \mathbf{P}$ *be the set of corrupted parties, S the adversarial Turing machine controlling them,* $f = (C, T_{\text{in}}, T_{\text{out}})$ *a multiparty computation task for the set of parties* $\mathbf{P}$, *and* $\mathbf{x}$ *a possible input for this task. The* ideal-model outcome *of computing* f *on* $\mathbf{x}$ *with corrupted parties* $\mathbf{P}_c$ *and adversary S is a probability distribution* $\text{IDEAL}_{f,S}^{\mathbf{P}_c}(\mathbf{x})$ *sampled as follows:*

1. *Send* $\mathbf{x}[\mathbf{P}_c]$ *to S. The adversary S returns* $\mathbf{y}$ — *a mapping from* $T_{in}^{-1}(\mathbf{P}_c)$ *to R, specifying the actual inputs of corrupted parties to* $\mathcal{F}_{SMC}^f$ *computing f.*
2. *Each honest party* P_i *sends* x_i *to* $\mathcal{F}_{SMC}^f$. *Each corrupted party* P_i *sends* y_i *to* $\mathcal{F}_{SMC}^f$. *Each party* P_i *receives* z_i *from* $\mathcal{F}_{SMC}^f$. *Let* $\mathbf{z}_c$ *be the concatenation of the values* z_i *for* $P_i \in \mathbf{P}_c$.
3. *Send* $\mathbf{z}_c$ *to the adversary S. It returns* $(\mathbf{P}_{h'}, r_A)$, *where* $\mathbf{P}_{h'} \subseteq \mathbf{P} \backslash \mathbf{P}_c$ *and* r_A *is a tuple of elements of R.*
4. *For each honest party* $P_i \in \mathbf{P}_{h'}$, *let* r_i *be the tuple of values* $\mathbf{z}(v)$, *where* $(v, P_i) \in T_{out}$. *For each honest party* $P_i \notin \mathbf{P}_{h'}$, *let* $r_i = \perp$.
5. *Output* r_A *and* r_i *for all honest parties* P_i.

We see that in sampling $\text{IDEAL}_{f,S}^{\mathbf{P}_c}(\mathbf{x})$, the adversary indeed learns only the inputs of corrupted parties. But as the adversary controls these parties, it can somewhat affect the outcome of computing f by choosing corrupted parties' inputs itself. This is a power that we obviously have to tolerate because the real-model adversary can do the same (as specified in Def. 5). The adversary receives the outcome of f for corrupted parties and is able to influence which of the honest parties actually receive their outputs. Again, this corresponds to the capability of the real-model adversary to prematurely stop the execution of the protocol, as defined next.

Definition 5 *Let* $\mathbf{P}_c \subset \mathbf{P} = \{P_1, \ldots, P_n\}$ *be the set of corrupted parties, A the adversarial Turing machine controlling them,* $\Pi = (M_1, \ldots, M_n)$ *a multiparty computation protocol for task f (with* M_i *being the machine executed by* P_i), *and* $\mathbf{x}$ *a possible input to the protocol. The* real-model outcome *of executing* Π *on* $\mathbf{x}$ *with corrupted parties* $\mathbf{P}_c$ *and adversary A is a probability distribution* $\text{REAL}_{\Pi,A}^{\mathbf{P}_c}(\mathbf{x})$ *sampled as follows:*

1. *Execute in parallel the machines* $M_i(\mathbf{x}[P_i])$, *computing the messages for parties* $P_i \in \mathbf{P} \backslash \mathbf{P}_c$, *and* $A(\mathbf{x}[\mathbf{P}_c])$, *computing the messages for all parties in* $\mathbf{P}_c$. *During the execution, all messages sent to parties in* $\mathbf{P}_c$ *are routed to A, and A is allowed to send messages on behalf of any* $P \in \mathbf{P}_c$. *Let* r_A *be the output of A and* r_i *the output of* M_i. *Messages sent between honest parties are not learned by A.*
2. *Output* r_A *and* r_i *for all honest parties* P_i.

Having defined the outcomes of ideal and real executions, the security of an SMC protocol is straightforward to define. Typically, security is not provided against any set $\mathbf{P}_c$ of corrupted parties, but only against certain coalitions. It is possible to precisely keep track of tolerated coalitions [4], but in this book, we simplify the presentation and only consider *threshold adversaries* that are allowed to corrupt up to t parties for some $t < n$.

Definition 6 *An n-party protocol* Π *for a functionality f is a* secure multiparty computation protocol tolerating at most t malicious parties *if for all* $\mathbf{P}_c \subseteq \mathbf{P}$ *with* $|\mathbf{P}_c| \leq t$ *and all adversaries A, there is an adversary S, so that for all possible inputs* $\mathbf{x}$ *to f,*

$$\text{REAL}_{\Pi,A}^{\mathbf{P}_c}(\mathbf{x}) \stackrel{d}{=} \text{IDEAL}_{f,S}^{\mathbf{P}_c}(\mathbf{x}) \ . \tag{1}$$

The sign $\stackrel{d}{=}$ in Eq. (1) denotes that the two distributions have to be very close to each other. This closeness can be interpreted in different ways.

- One may require the two distributions to be equal.
- In practice, an equally acceptable requirement is the statistical ε-closeness of the two distributions, where ε is an acceptable failure probability (typically around 2^{-80} or less).
- Alternatively, one may require the two distributions to be merely computationally indistinguishable [5], meaning that no efficient (i.e. probabilistic polynomial-time) algorithm can tell them apart with success probability that is non-negligibly better than $1/2$. In this case, we actually have two families of distributions, indexed by the security parameter η determining the length of cryptographic keys etc. in Π. The running time of Π, A and S (as functions of η) must also be polynomial in η. The success probability of the distinguishing algorithm must be at most $1/2 + \alpha(\eta)$, where α is a *negligible* function (i.e. $\lim_{\eta \to \infty} \eta^c \cdot \alpha(\eta) = 0$ for all c). If cryptographic constructions are part of Π, then this is the natural level of closeness in Eq. (1).

In Def. 6, the adversary is given full control over the parties it controls. In practice, the adversary may be unable to change the execution of these parties, but still be able to observe their internal state and the messages they exchange with other parties. We thus also define security against *semi-honest* parties. We obtain this definition by making the following changes to Def. 4, Def. 5 and Def. 6:

- In step. 1 of Def. 4, the output $\mathbf{y}$ from the adversary S must equal $\mathbf{x}[\mathbf{P}_c]$.
- In step. 3 of Def. 4, the set $\mathbf{P}_{h'}$ must be equal to $\mathbf{P} \backslash \mathbf{P}_c$.
- While participating in the protocol in the first step of Def. 5, the adversary A must use the actual machines M_i (for the parties $P_i \in \mathbf{P}_c$) to compute the messages sent by P_i.
- In Def. 6, modified definitions of ideal- and real-model outcomes must be used.

In literature, malicious parties are also called "active" and semi-honest parties are called "passive". Correspondingly, one speaks about active vs. passive security, and about actively vs. passively secure protocols. These synonyms will also be used interchangably throughout this book.

2. Oblivious Transfer

Oblivious transfer (OT) is a two-party computational task. The inputs from the first party, called the *sender*, are two bit-strings m_0, m_1 of the same length. The input of the second party, called the *receiver* is a bit b. The output of the receiver is m_b, while the sender gets no outputs. Oblivious transfer is used as a sub-protocol in several SMC protocols.

2.1. Basic Construction

The following construction first appeared in [6]. It requires a cyclic group $\mathbb{G}$ where the Diffie-Hellman problem is hard, e.g. the group $\mathbb{Z}_p^*$. Let g be a fixed generator of $\mathbb{G}$. The *computational Diffie-Hellman problem (CDH)* is to construct g^{xy} from g, g^x and g^y for random integers $x, y \in \{0, \ldots, |\mathbb{G}| - 1\}$. The possibly easier *decisional Diffie-Hellman problem (DDH)* is to distinguish tuples (g, g^x, g^y, g^{xy}) from tuples (g, g^x, g^y, g^z), again for random integers x, y, z. A problem is hard if no efficient algorithm can solve it with a non-

negligible success probability (in case of decisional problems, with a success probability that is non-negligibly better than $1/2$). Hence, in the definitions of CDH and DDH, group $\mathbb{G}$ actually depends on the security parameter η.

Let H be a cryptographic hash function mapping elements of $\mathbb{G}$ to bit-strings of the length $|m_0| = |m_1|$. For $b \in \{0, 1\}$ let $\bar{b} = 1 - b$. The following protocol securely realizes oblivious transfer.

1. The sender generates a random $C \leftarrow \mathbb{G}$ and sends it to the receiver.
2. The receiver picks $a \leftarrow \{0, \ldots, |\mathbb{G}| - 1\}$, sets $h_b = g^a$ and $h_{\bar{b}} = C \cdot h_b^{-1}$. It sends h_0, h_1 to the sender.
3. The sender checks that $h_0 h_1 = C$. If not, it aborts the protocol. Otherwise, it generates $r_0, r_1 \xleftarrow{\$} G$ and sends $g^{r_0}, g^{r_1}, c_0 = H(h_0^{r_0}) \oplus m_0$ and $c_1 = H(h_1^{r_1}) \oplus m_1$ to the receiver.
4. The receiver computes $m_b = c_b \oplus H((g^{r_b})^a)$.

We see that in this protocol, the sender basically treats h_0 and h_1 as public keys for ElGamal encryption [7]. It encrypts m_0 with h_0 and m_1 with h_1. The receiver is able to decrypt under one of the keys, but not under the other one. The protocol is information-theoretically secure against the sender (even if it is malicious), because (h_0, h_1) is uniformly distributed among the pairs of elements of $\mathbb{G}$ whose product is C. Security against a semi-honest receiver follows from the hardness of the DDH problem. In general, security against a malicious receiver is difficult to prove. However, it will follow from CDH if the hash function H is assumed to be a random oracle [8], i.e. $H(x)$ is a random bit-string independent of any other $H(x')$ (or several of them).

We see that that the computational complexity of this OT construction is similar to public-key operations. Indeed, as key exchange can be built on OT [9], it is unlikely that it could be implemented with cheaper symmetric-key primitives only [10].

2.2. Random Oblivious Transfer

Random oblivious transfer (ROT) is a variation of OT that we present here for the benefit of Sec. 2.3 on increasing the practicality of OT. It is a randomized two-party task, where neither the sender nor the receiver input anything, the sender obtains two uniformly, independently sampled random messages r_0, r_1 of predetermined length, and the receiver obtains a random bit b and the message r_b.

Clearly, with the help of OT we can build ROT — the sender and the receiver will just run OT with random inputs. We can also use ROT to build OT as follows. Let the sender have two messages m_0 and m_1, and the receiver have the bit b.

1. The sender and the receiver run ROT, with the sender receiving r_0, r_1 and the receiver receiving b' and $r_{b'}$.
2. The receiver sends $c = b \oplus b'$ to the sender.
3. The sender sends $m_0' = m_0 \oplus r_c$ and $m_1' = m_1 \oplus r_{\bar{c}}$ to the receiver.
4. The receiver computes $m_b = m_b' \oplus r_{b'}$.

Indeed, $m_b' \oplus r_{b'} = m_b \oplus r_{b \oplus c} \oplus r_{b'} = m_b$. The protocol is secure for the receiver, because c is independent of b. It is also secure for the sender, because $m_{\bar{b}}$ is masked by $r_{\bar{b'}}$, which the receiver does not have. If the ROT protocol is secure against malicious adversaries, then the resulting OT protocol also has the same security level.

2.3. Extending Oblivious Transfers

Even though public-key encryption is expensive, it is widely used through the hybrid mechanism — to encrypt a long message m, generate a symmetric key k, encrypt m under k, and k (which is much shorter) under the public-key encryption primitive. As we show next, similar constructions exist for OT — a small number of OT instances can be converted into a large number of OT instances with only the help of symmetric-key cryptography.

First, an OT instance for transferring a short message from the sender to the receiver can be converted into an OT instance for large messages by considering these short messages as keys that encrypt real messages. If the sender has two long messages m_0 and m_1, and the receiver has a bit b, then the sender may generate two keys k_0, k_1, send $\mathcal{E}nc(k_0, m_0)$ and $\mathcal{E}nc(k_1, m_1)$ to the receiver, and use OT to transfer k_b to the receiver. Second, m OT instances for messages of the length n, with $m \ll n$, can be converted into n ROT instances for messages of the length m, as we show next [11]. Such an OT extension construction is the main tool to make OT-s practicable in various protocols. The construction where $s[i]$ denotes the i-th bit of the bit-string s, is the following:

1. The receiver randomly generates messages $r_0^1, \ldots, r_0^m, c$ of the length n. It defines $r_1^i = r_0^i \oplus c$ for all $i \in \{1, \ldots, m\}$.
2. The sender generates a bit-string b of the length m.
3. The receiver and the sender use m instances of OT (with roles reversed) to transfer $q^i = r_{b[i]}^i$ from the receiver to the sender, where $i \in \{1, \ldots, m\}$.
4. For each $j \in \{1, \ldots, n\}$, the sender defines the m-bit string s_0^j as consisting of the bits of $q^1[j], \ldots, q^m[j]$. It also defines $s_1^j = s_0^j \oplus b$.
5. For each $j \in \{1, \ldots, n\}$, the receiver defines the m-bit string s^j as consisting of bits $r_0^1[j], \ldots, r_0^m[j]$.
6. In the j-th instance of ROT, the output to the sender is $H(j, s_0^j), H(j, s_1^j)$, and the output to the receiver is $c[j], H(j, s^j)$.

Here, H is a cryptographic hash function from pairs of integers and m-bit strings to m-bit strings. Indeed, one may not simply return s_0^j, s_1^j to the sender and s^j to the receiver, because they satisfy $s_0^j \oplus s_1^j = s_0^{j'} \oplus s_1^{j'}$ for all j, j'. The hash function H is used to break this correlation between different pairs (s_0^j, s_1^j).

The functionality of the construction is easy to verify and its security can be proved if H is modeled as a random oracle. However, the full power of the random oracle is not needed to break the correlations. The authors of the construction introduce the notion of *correlation-robust hash functions* [11] and show that this is sufficient for security.

If the underlying OT protocol is secure against malicious adversaries, then the presented ROT construction is also secure against a malicious sender. But it is only secure against a semi-honest receiver, because of the need to maintain the relationship $r_0^i \oplus r_1^i = c$ for all i. If $r_0^i \oplus r_1^i$ can take different values for different i-s, then the receiver may be able to learn both of the sender's messages. Security against a malicious receiver can be obtained through the following *cut-and-choose* technique [12]. Here, σ is a statistical security parameter, which affects the complexity of the construction and the sucess probability of a cheating receiver.

1. Run σ copies of the previous construction.

2. Let the sender randomly choose $\sigma/2$ of these copies. In these, the receiver reveals to the sender all the messages it has generated. If they are not consistent, the sender aborts. Otherwise, the messages in these $\sigma/2$ copies are discarded and the runs of the other $\sigma/2$ copies are combined as described in the steps below.
3. The receiver randomly picks a bit-string c' of the length n. The bits of c' are the choice bits of the receiver in n instances of ROT.
4. For each $j \in \{1, \ldots, n\}$ and for each of the $\sigma/2$ runs still in use, the receiver tells the sender whether the bits $c'[j]$ and $c[j]$ (in this copy) were the same. If they were not, then the sender swaps s_0^j and s_1^j in this copy.
5. In each instance of ROT, the two messages output to the sender and the message output to the receiver are exclusive ORs of the same messages in each of the $\sigma/2$ copies still in use.

We can again verify that the construction is functional. Its security is based on the use of exclusive OR in combining the non-discarded runs: if in at least one of them, the receiver cannot know both messages to the sender, then it cannot know them in the combined execution either. The probability that the check in step 2 is passed, but no honestly generated runs remain afterwards, is at most $2^{-\sigma/2}$.

3. The GMW Protocol

Goldreich et al. [13] proposed one of the first SMC protocols. Let the multiparty computation task f for n parties be given as a Boolean circuit, where the possible operations are exclusive OR, conjunction, and passing constants. The protocol evaluates the circuit gate by gate, representing the value computed at each gate in a privacy-preserving manner and invoking subprotocols to construct the representation of the result of a gate from the representations of its inputs. In this protocol, the representation $[\![b]\!]$ of a Boolean value $b \in \mathbb{Z}_2$ consists of n Boolean values $[\![b]\!]_1, \ldots, [\![b]\!]_n$ satisfying $[\![b]\!]_1 \oplus \cdots \oplus [\![b]\!]_n = b$. The component $[\![b]\!]_i$ is known to party P_i. The protocol works as follows:

Inputs If party P_i provides an input x for the input vertex v, it will randomly generate $b_1, \ldots, b_{n-1} \xleftarrow{\$} \mathbb{Z}_2$ and define $b_n = b_1 \oplus \cdots \oplus b_{n-1} \oplus x$. It sends b_j to party P_j, which will use that value as $[\![x]\!]_j$.

Constants A constant c computed by a nullary gate is represented as $[\![c]\!] = (c, 0, 0, \ldots, 0)$.

Addition If the result x of gate v is computed as $x = y_1 \oplus y_2$ for some y_1 and y_2 computed in gates v_1 and v_2, and the representations $[\![y_1]\!]$ and $[\![y_2]\!]$ have already been computed, then each party P_i defines $[\![x]\!]_i = [\![y_1]\!]_i \oplus [\![y_2]\!]_i$.

Multiplication If the result x of some gate is computed as $x = y_1 \wedge y_2$, and $[\![y_1]\!]$ and $[\![y_2]\!]$ are already available, then the representation $[\![x]\!]$ is computed as follows. We have $x = \bigoplus_{i=1}^n \bigoplus_{j=1}^n [\![y_1]\!]_i \wedge [\![y_2]\!]_j$. For each $i, j, k \in \{1, \ldots, n\}$, party P_k will learn a value $c_{ijk} \in \mathbb{Z}_2$, such that $\bigoplus_{k=1}^n c_{ijk} = [\![y_1]\!]_i \wedge [\![y_2]\!]_j$. These values are computed as follows:

- If $i = j$, then $c_{iii} = [\![y_1]\!]_i \wedge [\![y_2]\!]_i$ and $c_{iik} = 0$ for $k \neq i$.
- If $i \neq j$ then $c_{iji} \in \mathbb{Z}_2$ is chosen randomly by P_i. Party P_i defines the bits $d_0 = c_{iji}$ and $d_1 = [\![y_1]\!]_i \oplus c_{iji}$. Parties P_i and P_j use oblivious transfer to send $d_{[\![y_2]\!]_i}$ to P_j; this value is taken to be c_{ijj}. If $k \notin \{i, j\}$, then $c_{ijk} = 0$.

Afterwards, each party P_k defines $[\![x]\!]_k = \bigoplus_{i=1}^{n} \bigoplus_{j=1}^{n} c_{ijk}$.

Outputs If party P_i is expected to learn the value x computed in some gate v, and $[\![x]\!]$ has already been computed, then each party P_j sends $[\![x]\!]_j$ to P_i. Party P_i will output $x = [\![x]\!]_1 \oplus \cdots \oplus [\![x]\!]_n$.

It is not difficult to verify that the protocol correctly computes f. The protocol is secure against a passive adversary that controls an arbitrary number (i.e. up to $n-1$) of parties, if the protocol used for oblivious transfer is secure against passive adversaries. Indeed, as long as the adversary does not know all the components of the representation $[\![x]\!]$, and if each component of this representation is distributed uniformly, then the adversary has no idea about the actual value of x. Apart from the sharing of inputs, the only place in the protocol where an adversarially controlled P_j may receive a message from an honest P_i is during oblivious transfer, where P_j learns c_{ijj}. This value is masked by a freshly generated c_{iji}. Hence the view of the adversary can be simulated by generating random bits for all messages that adversarially controlled parties receive.

The protocol is actually more general than presented above. In addition to exclusive OR and conjunction, all other binary Boolean operations can be handled in a manner similar to the multiplication protocol. Indeed, in this protocol, party P_i defines the bits $d_b = c_{iji} \oplus [\![y_1]\!]_i \wedge b$ for $b \in \{0,1\}$. Party P_j receives the bit that corresponds to $b = [\![y_2]\!]_j$. Instead of conjunction, any other operation can be used to compute d_b.

The protocol is not secure against malicious adversaries, because there are no checks to ensure that the parties are behaving according to the protocol. A generic way to achieve security is to use zero-knowledge proofs [5] to show that the protocol is being followed [1, Chapter 7.4]. Due to the high cost of these proofs, they are not used in practice.

4. Secure Multiparty Computation Based on Garbled Circuits

Garbled circuits [14] present a different approach to two-party SMC. Let $f = (C, T_{\text{in}}, T_{\text{out}})$ be a two-party computation task where $C = (G, V_{\text{in}}, V_{\text{out}}, \lambda)$ is a Boolean circuit and $G = (V, E)$. Without a loss of generality, assume that the vertices in V_{out} have no successors. Also assume that only party P_2 gets outputs (i.e. $T_{\text{out}} = V_{\text{out}} \times \{P_2\}$). We discuss later, how a (private) output to P_1 can be provided. In its most basic form, a *garbling* of C is the result of the following steps:

1. For each $v \in V$, generate two keys k_v^0 and k_v^1 (for a chosen symmetric-key encryption scheme).
2. Let $w \in V \backslash V_{\text{in}}$. Let $u, v \in V$ be the two predecessors of w (in this order). Let $\otimes$ be the operation $\lambda(w)$ of w. Let g_w denote the following *garbling* of w: a random permutation of the four ciphertexts $\mathcal{E}nc(k_u^0, \mathcal{E}nc(k_v^0, k_w^{0 \otimes 0}))$, $\mathcal{E}nc(k_u^0, \mathcal{E}nc(k_v^1, k_w^{0 \otimes 1}))$, $\mathcal{E}nc(k_u^1, \mathcal{E}nc(k_v^0, k_w^{1 \otimes 0}))$, and $\mathcal{E}nc(k_u^1, \mathcal{E}nc(k_v^1, k_w^{1 \otimes 1}))$.
3. Let $v \in V_{\text{out}}$. The *output garbling* g_v^{out} of v is a random permutation of the two ciphertexts $\mathcal{E}nc(k_v^0, 0)$ and $\mathcal{E}nc(k_v^1, 1)$.
4. Output keys k_v^0 and k_v^1 for $v \in V_{\text{in}}$, and all garblings and output garblings constructed in the previous steps.

To compute f in a privacy-preserving manner, party P_1 garbles circuit C and sends all garblings and output garblings of vertices to P_2. For each $v \in V_{\text{in}}$, where $T_{\text{in}}(v) = P_1$,

party P_1 also sends the key k_v^b to P_2, corresponding to the input b of P_1 to vertex v. For each $v \in V_{\text{in}}$, where $T_{\text{in}} = P_2$, parties P_1 and P_2 use oblivious transfer to transmit k_v^b to P_2, corresponding to the input b of P_2 to vertex v. Party P_2 will then *evaluate* the garbled circuit — going through the vertices of C in topological order, it attempts to decrypt the ciphertexts in the garbling g_v of each vertex v, using the keys it learned while processing the ancestors of v. Assuming that P_2 recognizes when decryption fails, it finds that it cannot decrypt two out of four ciphertexts, can remove one layer of encryption for one ciphertext, and can remove both layers of encryption for one ciphertext. Hence P_2 learns one key while processing g_v. This key is equal to k_v^b for the bit b that would have been computed in vertex v if f had been executed in the clear, but P_2 does not know whether this key is k_v^0 or k_v^1. Similarly, for output garblings g_v^{out}, party P_2 attempts to decrypt the two ciphertexts using the key it learned at v. One of the decryptions is successful and results in a bit that P_2 takes as the result from gate v.

This protocol provides security against semi-honest adversaries (that have corrupted one party). It is quite clearly secure for P_2 if the used OT protocol is secure, as P_2 only interacts with P_1 through that protocol. Security for P_1 follows from the security properties of the encryption scheme, from the inability of P_2 to obtain both keys of any gate, and from its inability to find out whether the key it has corresponds to bit 0 or 1 [15].

The functionality f can be modified and the protocol slightly extended to provide output to P_1 as well. If $f^{\#}$ is the functionality we want to compute, giving $f_1^{\#}(x_1, x_2)$ to P_1 and $f_2^{\#}(x_1, x_2)$ to P_2 (where x_i is the input from P_i), then we let f provide the output $(f_1^{\#}(x_1, x_2) \oplus r, f_2^{\#}(x_1, x_2))$ to P_2 on inputs (x_1, r) from P_1 and x_2 from P_2. When invoking the secure two-party protocol for f, party P_1 lets r be a random bit-string. After executing the protocol, P_2 sends $f_1^{\#}(x_1, x_2) \oplus r$ back to P_1 who unmasks it.

There are a number of optimizations that reduce the cryptographic load of the garbled circuit construction. Instead of using an encryption scheme secure against chosen-plaintext attacks (as our security arguments in previous paragraphs tacitly assumed), the construction of garbling can be modified so that a block cipher, or even just a pseudorandom generator is used [16]. In case of a block cipher, it is possible to do all encryptions with a single key [17], meaning that if any standard cipher (e.g. AES) is used, the key expansion must be done only once. In addition, it is possible to construct the garblings so that the evaluator knows which ciphertext out of the four possible ones it has to decrypt.

More substantial optimizations can significantly reduce the effort of garbling and evaluating the circuit. When using the *free-XOR* technique [18], the garbler first selects a random and private bit-string R of the same length as the keys. It will then select the keys k_v^0 and k_v^1 for each non-XOR gate v so that $k_v^0 \oplus k_v^1 = R$. For each XOR-gate w with inputs u and v, it defines $k_w^0 = k_u^0 \oplus k_v^0$ and $k_w^1 = k_w^0 \oplus R$. In this way, $k_u^b \oplus k_v^c = k_w^{b \oplus c}$ for all $b, c \in \{0, 1\}$. The non-XOR gates are garbled as usual. No effort has to be made to garble the XOR-gates. Similarly, the evaluator only has to compute a single exclusive OR of bit-strings in order to evaluate a garbled XOR-gate. When using the free-XOR technique, one attempts to minimize not the size of the entire circuit, but the number of the non-XOR gates in it.

If we use block ciphers in garbling the circuit, then we can choose the keys k_w^0, k_w^1 for some gate w so that one of the ciphertexts in the garbling of w is a constant ciphertext, e.g. **0**. If u and v are the input gates of w and $\otimes$ is the operation of w, then we can randomly choose $b_u, b_v \in \{0, 1\}$ and define $k_w^{b_u \otimes b_v} = \mathcal{D}ec(k_v^{b_v}, \mathcal{D}ec(k_u^{b_u}, \mathbf{0}))$. The ciphertext **0** does not have to be sent from the garbler to the evaluator, reducing the

communication complexity by 25% [19]. This technique is compatible with the free-XOR technique described above. A different technique allows eliminating two out of four elements of the garbling of an AND-gate, still keeping the compatibility with free XORs [20].

It is possible to make garbled circuits secure against malicious adversaries. Again, cut-and-choose techniques can be used. To make sure that P_1 actually garbles the circuit that both parties have agreed to evaluate, it is going to garble the same circuit not just once, but σ times for a statistical security parameter σ. Party P_2 selects $\sigma/2$ out of them, and P_1 hands over all randomness that was used to produce these garblings of the circuit for f. After checking the validity of the opened garbled circuits, party P_2 executes the other $\sigma/2$ of the garbled circuits and takes the majority of their results as the final result. It is important to combine the results from different circuits using majority, instead of failing if P_2 receives several different results from the $\sigma/2$ circuits it is executing, as the fact whether or not P_2 has failed can give one bit of information about the inputs of P_2 to a malicious P_1.

Using the cut-and-choose technique introduces further complications relating to the preparation of inputs. Namely, P_2 has to make sure that the keys it receives are valid (and that complaints about invalidity do not leak information to P_1), correspond to its inputs, and to the same inputs of P_1 in all garbled circuits. We refer to [21] for details.

If P_2 is malicious, then the output of P_1 obviously cannot be simply masked with a random r as P_2 could modify it afterwards. Instead, the original functionality $f^\#$ is modified to compute an authenticated encryption [22] of the output of P_1, using a key that is part of the input of P_1 to the circuit. This encryption is learned by P_2 and sent back to P_1 who verifies its integrity and decrypts it.

5. Secure Multiparty Computation Based on Shamir's Secret Sharing

A *secret sharing* scheme allows a value to be shared among n parties so that certain coalitions of them can recover it from their shares, and certain other, smaller coalitions obtain no information about that value from their shares. Most frequently, there is a threshold t, so that all the coalitions with a size of at least t can find the value, and no coalition smaller than t gets any information. We will explore this case. Secret sharing is relevant for SMC because a number of operations can be performed on secret-shared values without leaking any further information about the values themselves to small coalitions.

Shamir's secret sharing scheme [23] is based on polynomial interpolation. Let $\mathbb{F}$ be a field with at least $n+1$ elements. Let $c_1, \ldots, c_n$ be mutually different, non-zero elements of $\mathbb{F}$. If a *dealer* wishes to share a value v among the parties $P_1, \ldots, P_n$, it will randomly generate a polynomial f of a degree of $t-1$ at most, satisfying $f(0) = v$, and send $s_i = f(c_i)$ to party P_i. The polynomial is generated by randomly generating $a_1, \ldots, a_{t-1} \xleftarrow{\$} \mathbb{F}$ and defining $f(x) = v + a_1 x + a_2 x^2 + \cdots + a_{t-1} x^{t-1}$.

Polynomials over fields can be interpolated: for any t points (i.e. argument-value pairs), there is exactly one polynomial of a degree of $t-1$ at most that passes through these points. For a fixed set of arguments, the coefficients of this polynomial can be computed as linear combinations of the values of the polynomial. In particular, using the notation of the previous paragraph, for each $\mathbf{I} = \{i_1, \ldots, i_t\}$ of the size t, there are coefficients $\lambda_{i_1}^{\mathbf{I}}, \ldots, \lambda_{i_t}^{\mathbf{I}}$, so that $v = \sum_{j \in \mathbf{I}} \lambda_j^{\mathbf{I}} s_j$. Using this equality, any t parties can recover

the secret. On the other hand, fewer than t parties obtain no information at all about the secret. If $\mathbf{I}' \subseteq \{1, \ldots, n\}$ and $|\mathbf{I}'| < t$, then for any v' there is a polynomial of a degree of $t - 1$ at most that passes through the points $\{(i, s_i)\}_{i \in \mathbf{I}'}$ and $(0, v')$. Moreover, for each $v' \in \mathbb{F}$, the number of such polynomials is the same.

It is possible to perform computations with shared values. If $s_1^1, \ldots, s_n^1$ are shares for v_1 (using polynomial f_1), and $s_1^2, \ldots, s_n^2$ are shares for v_2 (using polynomial f_2), then $s_1^1 + s_1^2, \ldots, s_n^1 + s_n^2$ are shares for value $v_1 + v_2$ (using polynomial $f_1 + f_2$). Indeed, adding points corresponds to adding polynomials. If the degrees of f_1 and f_2 are at most $t - 1$, then the degree of $f_1 + f_2$ is also at most $t - 1$. Similarly, the value cv for a public $c \in \mathbb{F}$ is represented by the shares $cs_1, \ldots, cs_n$. Hence, linear combinations of shared values can be computed by computing the same linear combinations of shares.

The multiplication of shared values is also possible through a protocol between $P_1, \ldots, P_n$. The following protocol [24] requires $2t - 1 \leq n$. Using the same notation as in the previous paragraph, let $f = f_1 \cdot f_2$. The degree of f is at most $2t - 2$. Party P_i computes $f(c_i) = s_i^1 \cdot s_i^2$ and shares it among all n parties using a polynomial of a degree of $t - 1$ at most. If all parties do it, they have available the shared values of $f(c_1), \ldots, f(c_n)$. As $n \geq 2t - 1$, the value $vv' = f(0)$ is a linear combination of these shared values. Each party computes this linear combination on its shares, resulting in a sharing of vv'.

The described protocols are constituents of an SMC protocol among n parties, secure against a passive adversary corrupting at most $t - 1$ parties, where $2t - 1 \leq n$. In other words, the number of corrupted parties must be smaller than $n/2$. We can securely evaluate arithmetic circuits (in which the operations are constants, additions, multiplications with a constant, and multiplications) over a finite field $\mathbb{F}$. Similarly to Sec. 3, the circuit is evaluated gate-by-gate and the result $v \in \mathbb{F}$ of a gate is represented as $[\![v]\!] = ([\![v]\!]_1, \ldots, [\![v]\!]_n)$, so that $[\![v]\!]_1, \ldots, [\![v]\!]_n$ are a Shamir's secret sharing of v secure against $t - 1$ parties. The protocol works as follows:

Inputs A party P_i providing an input v to some vertex will secret-share v among all n parties (including itself).

Constants The constant $c \in \mathbb{F}$ is represented by shares $(c, c, \ldots, c)$. Indeed, these are the values of the constant polynomial $f(x) = c$ (of degree 0, i.e. at most $t - 1$) at the points $c_1, \ldots, c_n$.

Arithmetic operations The representation $[\![v]\!]$ of a value v computed in a gate for addition, multiplication with a constant, or multiplication is computed from the representations of the values in the predecessors of that gate, using the protocols described in this section.

Outputs If the party P_j is expected to learn a value v computed in some gate, then each party P_i sends $[\![v]\!]_i$ to P_j. The party P_j uses interpolation to compute v.

The protocol is secure against passive adversaries. A simulator is easy to construct after noting that a coalition of parties of the size of $t - 1$ or less only ever sees random values of $\mathbb{F}$ as incoming messages. But a malicious party P_i can cheat in the following places:

- When sharing a value, it can pick a polynomial of degree t or more.
- In the multiplication protocol, it can incorrectly compute $s_i^1 \cdot s_i^2$, where s_i^1 and s_i^2 are its shares of the values that are multiplied together.

- When sending its share to another party (which is supposed to learn the value in some gate), it can send a wrong share.

The first and last issue are solved by using *verifiable secret sharing (VSS)* [25,26,27]. If the VSS scheme has the necessary homomorphic properties, then it can be used to solve the second issue as well. Next, we will describe Pedersen's VSS scheme [25], which is conceptually simple, although not the most efficient. It can be used to make the described SMC protocol secure against malicious adversaries.

Let a *broadcast channel* be available. All parties receive the same values over this channel, even if they were broadcast by a malicious party. A broadcast channel can be built with the help of digital signatures. Let $\mathbb{F} = \mathbb{Z}_p$ for some prime number p and let $\mathbb{G}$ be a group where the discrete logarithm problem is hard, and $|\mathbb{G}| = p$. Let $g, h \in \mathbb{G}$, so that $C = \log_g h$ would be unknown to all parties. Such g and h are easy to generate: each party P_i broadcasts a random $g_i \in \mathbb{G}$ and $h_i \in \mathbb{G}$; and g and h are defined as the products of those.

When using Pedersen's VSS, a dealer computes the sharing $[\![v]\!]$ of a value $v \in \mathbb{Z}_p$ among the parties $P_1, \dots, P_n$ in the following manner:

- Randomly generate $a_1, \dots, a_{t-1}, a'_0, \dots, a'_{t-1} \overset{\$}{\leftarrow} \mathbb{Z}_p$. Let $a_0 = v$. Define $f(x) = v + a_1 x + \cdots + a_{t-1} x^{t-1}$ and $f'(x) = a'_0 + a'_1 x + \cdots + a'_{t-1} x^{t-1}$.
- Broadcast $y_j = g^{a_j} h^{a'_j}$ for all $j \in \{0, \dots, t-1\}$.
- Send $(f(c_i), f'(c_i))$ to party P_i.

The party P_i, having received (s_i, s'_i) can verify its share with respect to the published values. The following equation must hold:

$$g^{s_i} h^{s'_i} = \prod_{j=0}^{t-1} y_j^{c_i^j} \ . \tag{2}$$

Indeed, if the share (s_i, s'_i) corresponds to the broadcast values, then taking the logarithm (to the base of g) of the sides of this equation gives us $s_i + C s'_i = f(c_i) + C f'(c_i)$. If the dealer were able to produce a different set of broadcast values that would still pass verification, it could use them to recover C. On the other hand, the broadcast values do not reveal anything further about the value v. In fact, each y_j is a random element of $\mathbb{G}$ as it has been masked with $h^{a'_j}$.

We now show how the issues defined above can be solved with Pedersen's VSS.

- *Ensure that the polynomial degree is at most $t-1$.* By construction, the logarithm of $\prod_{j=0}^{t-1} y_j^{c_i^j}$ is equal to some polynomial g of degree $t-1$ evaluated on c_i. However, as we have shown above, the same logarithm equals $f(c_i) + C f'(c_i)$. We need to show that the degree of f is at most $t-1$ in this case. If $\deg(f(c_i) + C f'(c_i)) = d_1 \leq t-1$, but $\deg(f(c_i)) = d_2 > t-1$, then the last $d_2 - d_1$ coefficients of the polynomials f and $C f'$ should be negations of each other. The coefficients of f and f' are known to the sender and, hence, it may compute $C = -a_i/a'_i$ for some $i \in \{d_1, \dots, d_2\}$.
- *Multiplication protocol.* In the particular multiplication protocol presented above, each party computes $s = s^1 \cdot s^2$ and then shares it amongst all the n parties. The party that shares the product $s = s^1 \cdot s^2$ can commit the shares s_i of s exactly in

the same way as the input is committed, publishing $(g^{s_i}, h^{s'_i})$. Assuming that the shares of s^1 and s^2 have already been committed in the same way, the party has to prove that $s = s^1 \cdot s^2$.

A linear combination can be computed locally by any party. The commitment for $\alpha_1 s_1 + \ldots + \alpha_m s_m$ is $(g^{\alpha_1 s_1 + \ldots + \alpha_m s_m}, h^{\alpha_1 s'_1 + \ldots + \alpha_m s'_m}) = ((g^{s_1})^{\alpha_1} \cdot \ldots \cdot (g^{s_m})^{\alpha_m}, (h^{s'_1})^{\alpha_1} \cdot \ldots \cdot (h^{s'_m})^{\alpha_m})$, where $(g^{s_i}, h^{s'_i})$ have already been committed. Since s is a linear combination of s_i, the parties are able to compute $(g^s, h^{s'})$. Similarly, they compute $(g^{s^1}, h^{s'^1})$ and $(g^{s^2}, h^{s'^2})$. Now there are different ways of verifying $s = s^1 \cdot s^2$.

∗ Verifying a multiplication is easy if the homomorphic scheme allows verifying products (there is an operation $\odot$ so that $E(x) \odot E(y) = E(x \cdot y)$ for a commitment function E). Such schemes exist, but they are not very efficent in practice (see Sec. 6 for details).

∗ If the group $\mathbb{G}$ supports a *bilinear pairing* (a function $e : \mathbb{G} \times \mathbb{G} \mapsto \mathbb{G}$ so that $e(g^a, g^b) = e(g, g)^{ab}$ for all $g \in \mathbb{G}$, $a, b \in \mathbb{Z}$), then the product $s = s^1 \cdot s^2$ can be verified as $e(g^{s^1}, g^{s^2}) = e(g, g)^{s^1 s^2}$.

∗ *Multiplication triples* can be used to reduce all the multiplications to linear combinations. This is described in more detail in Sec. 7.

∗ The most general method for verifying multiplications is to use any zero-knowledge proofs of knowing $\bar{s}^2$, so that $g^{\bar{s}^2} = g^{s^2}$ and $(g^{s^1})^{\bar{s}^2} = g^{s^1 s^2}$. Here, $g^{\bar{s}^2} = g^{s^2}$ proves that $\bar{s}^2 = s^2$ as a discrete logarithm is unique in the group $\mathbb{G}$.

- *Verifying a share sent by another party.* Each secret input s is committed as $(g^s, h^{s'})$, where s' is the leading coefficient of the polynomial f'. Each intermediate value is either a linear combination or a product of the previous values. These operations can be verified as shown in the previous clause.

6. Secure Multiparty Computation Based on Threshold Homomorphic Encryption

Threshold homomorphic encryption is another method that allows revealing a secret value to any coalition of at least t parties, while giving no information about that value to any coalition smaller than t.

In a (t, n)-threshold homomorphic cryptosystem, anybody can perform the following operations using the public key:

- Encrypt a plaintext.
- Add (denoted $\oplus$) two ciphertexts to obtain a (uniquely determined) encryption of the sum of the corresponding plaintexts.
- Multiply (denoted $\otimes$) a ciphertext by a constant to obtain a (uniquely determined) encryption of the product of the plaintext with the constant.

Decryption is only possible if at least t out of the n decryption keys are known. An example of such a cryptosystem is the threshold variant of the Paillier cryptosystem from [28]; see [29] for details.

Computation on homomorphic encryptions can be performed by the n parties holding the decryption keys for the cryptosystem. Additions and multiplications by a constant can be performed locally using the homomorphic operators $\oplus$ and $\otimes$. Multiplications of

encryptions X of x and Y of y can be performed using an interactive protocol between the n parties due to [30]. In this protocol, each party i chooses a random value d_i, and broadcasts encryptions D_i of d_i and $E_i = Y \otimes (-d_i)$ of $y \cdot (-d_i)$. The parties then compute $X \oplus D_1 \oplus \cdots \oplus D_n$, and threshold decrypt it to learn $s = x + d_1 + \ldots + d_n$. This allows them to compute the encryption $Z = Y \otimes (x + d_1 + \cdots + d_n)$ of $y \cdot (x + d_1 + \cdots + d_n)$ and, hence, also an encryption of $x \cdot y$ as $Z \oplus \bigoplus_{i=1}^{n} E_i$.

While computations on homomorphic encryptions are much slower than computations on secret shares, the advantage is that it is easy to make them secure against an active adversary. Namely, in the above multiplication protocol, the parties can use zero-knowledge proofs to prove that each E_i indeed contains the product $y \cdot (-d_i)$ and that their share of the threshold decryption of $X \oplus D_1 \cdots \oplus D_n$ was correct. To perform these proofs, [29] uses a multiparty variant of Σ-protocols. Recall that a Σ-protocol for a binary relation R is a three-move protocol in which a potentially malicious prover convinces an honest verifier that he knows a witness for a certain statement. First, the prover sends an announcement to the verifier. The verifier responds with a uniformly random challenge. Finally, the prover sends its response, which the verifier verifies. In order to let all n parties prove statements to each other simultaneously, they jointly generate a single challenge to which they all respond. Namely, each party broadcasts a commitment to its announcement, the parties jointly generate a challenge, and, finally, the parties broadcast their response to this challenge, along with an opening of their commitment.

Combining these techniques, we get an SMC protocol among n parties, which is secure against an active adversary corrupting at most $t - 1$ parties. The protocol securely evaluates arithmetic circuits over the plaintext ring $\mathbb{Z}_N$ of the threshold homomorphic cryptosystem, e.g. N is an RSA modulus in the case of Paillier encryption. The protocol also achieves robustness in the sense that, if at least t parties are honest, then all parties are guaranteed to learn the computation result. In particular, if $t = 1$, then the protocol guarantees privacy and correctness for all honest parties (but no robustness). If $t = \lceil (n+1)/2 \rceil$, then the protocol guarantees privacy, correctness, and robustness if fewer than t parties are corrupted. The full protocol works in the manner described below.

Inputs Party P_i providing an input v broadcasts a homomorphic encryption of v. Each party proves knowledge of the corresponding plaintext. This prevents parties from adaptively choosing their inputs based on the inputs of others.

Constants The constant $c \in \mathbb{Z}_N$ is represented by encryption $C = \mathsf{Enc}(c)$ of c with fixed randomness.

Addition If the result x of a gate is computed as $x = y_1 + y_2$ and encryptions Y_1 and Y_2 have already been computed, then each party defines $X = Y_1 \oplus Y_2$.

Multiplication Multiplication is performed using the interactive n-party protocol described above, i.e. the parties exchange encryptions D_i, E_i, perform an interactive multiplication proof, exchange threshold decryptions, and perform an interactive decryption proof. Finally, they compute the product encryption from the decrypted value and the encryptions E_i.

Outputs If party P_j is expected to learn a value v computed in some gate as encryption V, then it broadcasts an encryption D of a random value d and proves knowledge of the plaintext of D. Then, all parties provide threshold decryptions of $Z = V \oplus D$ and prove correctness of the decryption z. Finally, party P_j computes $v = z - d$.

The homomorphic encryption scheme defined above can be applied *locally* (i.e. without any interaction) to various tasks for which computing the sum and the scalar product of plaintexts is sufficient. In general, it is not applicable to an arbitrary computation. *Fully homomorphic encryption* [31] includes an operation denoted $\odot$ on ciphertexts (in addition to $\oplus$ and $\otimes$ mentioned before) that allows anybody to multiply two ciphertexts to obtain a (uniquely determined) encryption of the product of the corresponding plaintexts. Supporting addition and multiplication is sufficient to perform an arbitrary computation on plaintexts. However, fully homomorphic encryption schemes are much less efficient, and hence, a simpler homomorphic scheme is preferable for a more constrained task that does not require the support of arbitrary computation (such as computing linear combinations). Alternatively, some special multiplication protocols like the one presented in this section can be used.

7. Secure Multiparty Computation Based on Somewhat Homomorphic Encryption

In this section we describe the protocol SPDZ that was initially proposed in [32]. The main idea of this protocol is to introduce an expensive preprocessing phase that allows making the online phase cheap.

The SPDZ protocol makes use of the following techniques:

- *Message authentication codes (MAC)* prevent the shares from being affected by a malicious adversary.
- *Beaver triples* reduce multiplication to linear combinations.
- *Somewhat homomorphic encryption* is used in the preprocessing phase to compute MACs and Beaver triple shares.

Somewhat homomorphic encryption [33] satisfies the properties of homomorphic encryption (defined in Sec. 6). The difference is that it supports only a finite number of sequential multiplications. A group that supports a bilinear pairing (see Sec. 6) is an example of somewhat homomorphic encryption with at most one sequential multiplication. SPDZ uses somewhat homomorphic encryption in the preprocessing phase which can be seen as many smaller arithmetic circuits that are evaluated in parallel, and where the number of sequential multiplications is not large.

Message authentication codes make it difficult for the adversary to modify the messages by introducing special correctness checks that are generated for the inputs and propagated by the operations. SPDZ uses two kinds of additive sharings that allow message checking. The generation of these shares is described in more detail in [32].

1. The first sharing is

$$\langle a \rangle = (\delta, (a_1, \ldots, a_n), (\gamma(a)_1, \ldots, \gamma(a)_n)) \ ,$$

where $a = a_1 + \cdots + a_n$ and $\gamma(a)_1 + \cdots + \gamma(a)_n = \alpha(a + \delta)$. The party P_i holds the pair $(a_i, \gamma(a)_i)$, and δ is public. The interpretation is that $\gamma(a) = \gamma(a)_1 + \cdots + \gamma(a)_n$ is the MAC authenticating the message a under the global key α.
We have $\langle a \rangle + \langle b \rangle = \langle a + b \rangle$ for secret values a and b where $+$ is pointwise addition: $\langle a+b \rangle = (\delta_a + \delta_b, (a_1 + b_1, \ldots, a_n + b_n), (\gamma(a)_1 + \gamma(b)_1, \ldots, \gamma(a)_n +$

$\gamma(b)_n)$. If c is a constant, $c + \langle a \rangle = (\delta - c, (a_1 + c, a_2, \ldots, a_n), (\gamma(a)_1, \ldots, \gamma(a)_n))$. Hence, any linear combination can be computed locally, directly on the shares.

2. The second sharing is

$$[\![\alpha]\!] = ((\alpha_1, \ldots, \alpha_n), (\beta_1, \ldots, \beta_n), (\gamma(\alpha)_1^i, \ldots, \gamma(\alpha)_n^i)_{i \in [n]}) \ ,$$

where $\alpha = \alpha_1 + \cdots + \alpha_n$ and $\gamma(\alpha)_1^i + \cdots + \gamma(\alpha)_n^i = \alpha \beta_i$. The party P_i holds the values $\alpha_i, \beta_i, \gamma(\alpha)_i^1, \ldots, \gamma(\alpha)_i^n$. The idea is that $\gamma(\alpha)_1^i + \cdots + \gamma(\alpha)_n^i$ is the MAC authenticating α under the private key β_i of P_i. To open $[\![\alpha]\!]$, each P_j sends to each P_i its share α_j of α and its share $\gamma(\alpha)_j^i$ of the MAC on α made with the private key of P_i. P_i checks that $\sum_{j \in [n]} \gamma(\alpha)_j^i = \alpha \beta_i$. To open the value to only one party P_i (so-called *partial opening*), the other parties simply send their shares only to P_i, who does the checking. Only shares of α and $\alpha \beta_i$ are needed for that.

Beaver triples [34] are triples of values (a, b, c) over the corresponding finite field $\mathbb{F}$ where the computation takes place, generated by randomly picking $a, b \xleftarrow{\$} \mathbb{F}$ and computing $c = a \cdot b$. Precomputing such triples can be used to linearize multiplications. This may improve the efficiency of the online protocol phase significantly, as computing a linear combination can be done locally. In order to compute $\langle x \rangle \cdot \langle y \rangle$, if a triple $(\langle a \rangle, \langle b \rangle, \langle c \rangle)$ has been precomputed and shared already, we may first compute and publish $x' := \langle x \rangle - \langle a \rangle$ and $y' := \langle y \rangle - \langle b \rangle$, and then compute a linear combination $\langle x * y \rangle = (x' + \langle a \rangle)(y' + \langle b \rangle) = x'y' + y'\langle a \rangle + x'\langle b \rangle + \langle a \rangle\langle b \rangle = x'y' + y'\langle a \rangle + x'\langle b \rangle + \langle c \rangle$. Here, the publication of x' and y' leaks no information about x and y. In this way, interactive multiplication is substituted with an opening of two values. In SPDZ, a sufficient number of such triples is generated in the preprocessing phase. The triples are random and depend neither on the inputs nor on the function that is being computed. The checks performed during preprocessing ensure that all triples used during the online phase are valid.

In the specification of the protocol, it is assumed for simplicity that a broadcast channel is available, that each party has only one input, and only one public output value has to be computed. The number of input and output values can be generalized to an arbitrary number without affecting the overall complexity (as shown in [32]). The protocol works as follows:

Initialization The parties invoke the preprocessing to get:

- The shared secret key $[\![\alpha]\!]$.
- A sufficient number of Beaver triples $(\langle a \rangle, \langle b \rangle, \langle c \rangle)$.
- A sufficient number of pairs of random values $\langle r \rangle$, $[\![r]\!]$.
- A sufficient number of single random values $[\![t]\!]$, $[\![e]\!]$.

The generation of all these values is presented in more detail in [32]. It is based on a somewhat homomorphic encryption scheme.

Inputs If the party P_i provides an input x_i, a pre-shared pair $\langle r \rangle$, $[\![r]\!]$ is taken and the following happens:

1. $[\![r]\!]$ is opened to P_i.
2. P_i broadcasts $x_i' = x_i - r$.
3. The parties compute $\langle x_i \rangle = \langle r \rangle + x_i'$.

Addition In order to add $\langle x \rangle$ and $\langle y \rangle$, locally compute $\langle x+y \rangle = \langle x \rangle + \langle y \rangle$.

Multiplication To multiply $\langle x \rangle$ and $\langle y \rangle$ the parties do the following:

1. Take the two triples $(\langle a \rangle, \langle b \rangle, \langle c \rangle)$, $(\langle f \rangle, \langle g \rangle, \langle h \rangle)$ from the set of the available ones and check that indeed, $a \cdot b = c$. This can be done as follows:

   - Open a random value $[\![t]\!]$.
   - Partially open $a' = t\langle a \rangle - \langle f \rangle$ and $b' = \langle b \rangle - \langle g \rangle$.
   - Evaluate $t\langle c \rangle - \langle h \rangle - b'\langle f \rangle - a'\langle g \rangle - a'b'$ and partially open the result.
   - If the result is not zero the protocol aborts, otherwise go on with $(\langle a \rangle, \langle b \rangle, \langle c \rangle)$.

   The idea is that as t is random, it is difficult for the adversary to generate malicious shares so that the result would be 0. This check can be done as part of the preprocessing for all triples in parallel, and hence, only one random value t is sufficient.

2. Partially open $x' = \langle x \rangle - \langle a \rangle$ and $y' = \langle y \rangle - \langle b \rangle$. Compute $\langle z \rangle = x'y' + x'\langle b \rangle + y'\langle a \rangle + \langle c \rangle$.

Outputs The output stage starts when the parties already have $\langle y \rangle$ for the output value y, but this value has not been opened yet. Before the opening, it should be checked that all parties have behaved honestly.

- Let $a_1, \ldots, a_T$ be all values publicly opened so far, where

$$\langle a_j \rangle = (\delta_j, (a_{j1}, \ldots, a_{jn}), (\gamma(a_j)_1, \ldots, \gamma(a_j)_n)) \ .$$

  A new random value $[\![e]\!]$ is opened, and parties set $e_i = e^i$ for $i \in [T]$. All parties compute $a = \sum_{j \in [T]} e_j a_j$.
- Each P_i commits to $\gamma_i = \sum_{j \in [T]} e_j \gamma(a_j)_i$. For the output value $\langle y \rangle$, P_i also commits to the shares $(y_i, \gamma(y)_i)$ in the corresponding MAC.
- $[\![\alpha]\!]$ is opened.
- Each P_i opens the commitment γ_i, and all parties check that $\alpha(a + \sum_{j \in [T]} e_j \delta_j) = \sum_{i \in [n]} \gamma_i$. If the check is not passed, the protocol aborts. Otherwise, the parties conclude that all the messages a_j are correct.
- To get the output value y, the commitments to $(y_i, \gamma(y)_i)$ are opened. Now y is defined as $y := \sum_{i \in [n]} y_i$, and each player checks that $\alpha(y + \delta) = \sum_{i \in [n]} \gamma(y)_i$. If the check is passed, then y is the output.

This verifies that all the intermediate values a_j, and also y, have indeed all been computed correctly.

8. Universally Composable Secure Multiparty Computation

The security definitions in Sec. 1 consider the case where only a single instance of the SMC protocol is running. This is rarely the case in reality where we want to run several instances, possibly concurrently, and possibly together with other protocols. *Universal composability (UC)* [35] has emerged as the way to define the security of protocols that is preserved under concurrent compositions. In UC definitions of the security of protocols, the ideal and real executions contain one more entity, which is called the *environment* that models the other parts of the system in addition to the protocol.

The ideal-model and real-model executions in Def. 4 and Def. 5 are modified to incorporate the environment $\mathcal{Z}$ that depends on the functionality offered by the protocol. The modifications are the following:

- The parties $P_1, \ldots, P_n$ receive their inputs $x_1, \ldots, x_n$ from $\mathcal{Z}$. They also hand back their outputs to $\mathcal{Z}$.
- During the execution, $\mathcal{Z}$ can freely communicate with the adversary (S or A). This models the adversary's influence on other parts of the system.
- At some moment, the environment $\mathcal{Z}$ stops and outputs a bit. This bit is the result of the execution.

We thus obtain the probability distributions $\text{IDEAL}_{f,S}^{\mathbf{P}_c}(\mathcal{Z})$ and $\text{REAL}_{\Pi,A}^{\mathbf{P}_c}(\mathcal{Z})$, both over the set $\{0,1\}$. Intuitively, the bit output by $\mathcal{Z}$ represents its guess whether it participated in an ideal-model or a real-model execution. We can now define the UC security of SMC protocols, following the general pattern of UC definitions where a real system Π is at least as secure as an ideal system $\mathcal{F}$ if anything that can happen to the environment when interacting with the real system can also happen when interacting with the ideal system.

Definition 7 *An n-party protocol Π for a functionality f is at least as secure as $\mathcal{F}_{\mathsf{SMC}}^{f}$ against attackers corrupting at most t parties, if for all $\mathbf{P}_c \subseteq \mathbf{P}$ with $|\mathbf{P}_c| \leq t$ and all adversaries A, there exists an adversary S, so that for all possible environments $\mathcal{Z}$,*

$$\text{REAL}_{\Pi,A}^{\mathbf{P}_c}(\mathcal{Z}) \stackrel{d}{=} \text{IDEAL}_{f,S}^{\mathbf{P}_c}(\mathcal{Z}) \ . \tag{3}$$
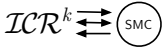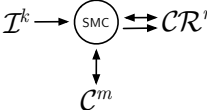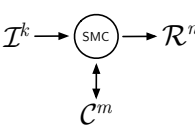
It is known that if several secure protocols (or several instances of the same secure protocol) are run concurrently, the adversary may be able to attack the resulting system as the parties may confuse messages from different sessions [36, Chapter 2]. In practice, this confusion is eliminated by using suitable session identifiers on messages. This in turn requires the parties to somehow agree on these identifiers.

9. Party Roles and Deployment Models of Secure Multiparty Computation

The descriptions of SMC protocols given in previous sections assume that all parties of the SMC protocol provide some inputs to it (possibly trivial ones), participate actively in the computation, and receive some outputs (possibly trivial ones). In practice, the number of involved parties may be large. This happens in particular when there are many parties that provide inputs to the computation. In this case, some of the protocols described before become inefficient, and other cannot be used at all.

To implement large-scale SMC applications with many parties, we break the symmetry among them. We consider three different party roles that define which parties can see what and who is in charge of certain operations. These three roles include *input parties*, who secret-share or encrypt the data they provide to the computation. Each of the protocols described in this chapter has, or can be amended with an input phase that can be executed by anyone without knowledge of any secrets set up to execute the SMC protocol. The input parties send the secret-shared or encrypted values to the *computing parties*, who carry out the SMC protocols on the hidden values. The number of computing parties is kept small in order to efficiently execute the SMC protocol. The computing

Table 1. SMC deployment models and examples of applications

Basic deployment model	Examples of applications
$\mathcal{ICR}^k \rightleftarrows \text{SMC}$	**The classic millionaires' problem [14]** *Parties:* Two, Alice and Bob (both $\mathcal{ICR}$) *Overview:* The millionaires Alice and Bob use SMC to determine who is richer **Joint genome studies [38]** *Parties:* Any number of biobanks (all $\mathcal{ICR}$) *Overview:* The biobanks use SMC to create a joint genome database and study a larger population
$\mathcal{IC}^k \rightleftarrows \text{SMC} \longrightarrow \mathcal{R}^m$	**Studies on linked databases [37]** *Parties:* The Ministry of Education, the Tax Board, the Population Register (all $\mathcal{IC}$) and the Statistical Office ($\mathcal{R}$) *Overview:* Databases from several government agencies are linked to perform statistical analyses and tests
$\mathcal{IR}^k \rightleftarrows \text{SMC} \longleftrightarrow \mathcal{C}^m$	**Outsourcing computation to the cloud [31]** *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 his/her data
$\mathcal{ICR}^k \rightleftarrows \text{SMC} \rightleftarrows \mathcal{IR}^m$	**Collaborative network anomaly detection [39]** *Parties:* Network administrators (all $\mathcal{IR}$), a subset of whom are running computing servers (all $\mathcal{ICR}$) *Overview:* A group of network administrators use SMC to find anomalies in their traffic
$\mathcal{I}^k \longrightarrow \text{SMC} \rightleftarrows \mathcal{CR}^n$ $\updownarrow$ $\mathcal{C}^m$	**The sugar beet auction [40]** *Parties:* Sugar beet growers (all $\mathcal{I}$), Danisco and DKS (both $\mathcal{CR}$) and the SIMAP project ($\mathcal{C}$) *Overview:* Sugar beet growers and their main customer use SMC to agree on a price for purchase contracts
$\mathcal{I}^k \longrightarrow \text{SMC} \longrightarrow \mathcal{R}^n$ $\updownarrow$ $\mathcal{C}^m$	**The Taulbee survey [41]** *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 [42]** *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 on its members

parties send the encrypted or secret-shared results to the *result parties*, who combine the received values in order to see the results. A party can have one or several of these roles. Table 1 from [37] describes several practical prototype applications within the described party role paradigm.

10. Classes of Properties of Secure Multiparty Computation Protocols

In previous sections, we have given examples of different sharing schemes and SMC protocols. We have seen that the protocols can be classified into those secure against a passive adversary and those secure against an active adversary. There are even more protocol classifications, some of which we present in this section. Even more protocol properties can be found in [43].

Trusted setup. Some protocols require pre-sharing of certain information before the start of an execution. This information is independent from the actual protocol inputs on which the functionality is computed. Here we consider pre-sharing that is done in a trusted setup.

A common reference string (CRS) [44] is a polynomial-length string that comes from a certain pre-specified distribution. All the involved parties must have access to the same string. Introducing a CRS makes it possible to remove some interaction from the protocol. For example, the random values that must be generated by one party and sent to another can be pre-shared before the execution starts.

A protocol may use a public key infrastructure (PKI), where a public and a secret key are issued to each party. The PKI can be used for various purposes such as signatures, commitments, and ensuring that only the intended receiver gets the message. Its advantage compared to a CRS is that it can be reused (unless it is used for certain tasks such as bit commitments, where the secret key is revealed), while in general, a CRS cannot be reused and a new instance has to be generated for each protocol run.

If there is no trusted setup, it is still possible to achieve the same properties that the trusted setup gives (for example, include a key exchange subprotocol), at the expense of an online protocol execution phase.

Existence of a broadcast channel. A broadcast channel allows a party to send the same message to all other parties in such a way that each receiver knows that each other (honest) party has received exactly the same message.

If there is no explicit broadcast channel, it can still be modeled in some settings. For example, if at least $2n/3 + 1$ of the n parties are honest, then a broadcast can be implemented as follows. If P_i wants to broadcast m, it sends (init, i, m) to all other parties. If a party P_j receives (inti, i, m) from P_i, it sends (echo, i, m) to all parties (including itself). If a party P_j receives (echo, i, m) from at least $n/3 + 1$ different parties, then it sends (echo, i, m) to all parties too. If a party P_j receives (echo, i, m) from at least $2n/3 + 1$ different parties, then it accepts that P_i has broadcast m. It can be shown that if at least one party accepts m, then all the other honest parties do as well.

Assumption level. The security of protocols can be based on the intractability of certain computational tasks. Some protocols use quite specific assumptions such as factoring or finding the minimal distance of vectors generated by a matrix over a finite field. In some cases, the intractability has not even been formally reduced to well-known open problems. Even if no efficient algorithm for solving these tasks is known right now, it may still be solved in the future. Some complex tasks (e.g. factoring) can be solved in polynomial time using quantum computation.

Instead of assuming the hardness of a particular computational task, the security may be based on a more general assumption such as the existence of trapdoor functions. For a trapdoor function f, given an input x, it is easy to compute $f(x)$, but it is difficult to

compute x from $f(x)$ unless a special *trapdoor* is known, which may depend on f itself, but not on x. A weaker assumption is the existence of one-way functions that do not require the existence of a trapdoor. When implementing a protocol, a specific one-way function f can be chosen. If it turns out that this particular f is not one-way, the protocol will not be immediately broken, as some other f can be used instead. In this case, the particular implementation becomes insecure, but not the whole protocol.

It is not known if one-way functions exist. There are no computational problems whose hardness can be steadily proven, so in the best-case scenario no computational assumptions are used. The next level is *statistical* security, where the data may leak only with negligible probability. If the leakage probability is 0, then we have *perfect* security.

Maliciousness. In previous sections, we considered protocols secure against passive and active adversaries. We describe two intermediate levels between passive and active adversaries.

A *fail-stop* adversary [45] follows the protocol similarly to the passive adversary, except for the possibility of aborting. This means that the adversary has the power to interrupt the protocol execution, but nothing more compared to the passive one.

A *covert* adversary [46] estimates the probability of being caught. It deviates from the protocol as long as this probability is sufficiently low.

Adversary mobility. A *static* adversary chooses a set of corrupted parties before the protocol starts. After that, the set of corrupted parties stays immutable.

An *adaptive* adversary adds parties to the malicious set during the execution of the protocol, until the threshold is reached. The choice of the next corrupted party depends on the state of the other parties corrupted so far.

A *mobile* adversary can not only add new parties to the malicious set during the execution of the protocol, but also remove them, so that some other party can be corrupted instead.

Corrupted parties. In the simplest case, a single adversary that corrupts a set of parties.

In the case of *mixed adversary security*, different sets of parties can be corrupted by different adversaries. For example, it may happen that one of them is passive, and the other active.

In the case of *hybrid security*, the protocol may tolerate different sets of corrupted parties with different capabilities. For example, one set of malicious parties is computationally bounded, while the other is not.

Fairness. If a protocol has the *agreement* property, then if at least one *honest* party receives its output, then all the other honest parties do as well. If a protocol has the *fairness* property, then if *any* party receives its output, then all the honest parties do as well.

Composability. If a protocol is secure in the *stand-alone* model, then it is secure only if it is executed once, and there are no other protocols running. For example, if one protocol uses PKI for commitments, the secret key is published when the commitment is opened, and the keys cannot be reused. Hence, the protocol can be run only once.

If the protocol is *sequentially* composable, then it is secure regardless of any other instance of the same protocol running before and after it. However, there may still be problems if some other protocol is running in parallel. For example, a party P_1 may instantiate two protocol executions with P_2 and P_3, pretending to be P_3 for P_2. If P_2

requires proof that it indeed communicates with P_3, and sends a corresponding challenge to which only P_3 can respond, then P_1 may deliver this challenge to P_3 in a parallel protocol session in which it is the turn of P_1 to send the challenge.

A protocol that supports *parallel* composition is secure even if several instances are executed in parallel, regardless of the timings that the adversary inserts between the rounds of all the protocol runs. However, it can still be insecure in the presence of some other protocols. For example, a protocol that uses PKI for message transmission can be secure in parallel composition, but executing another protocol that uses the same PKI for commitments will break it.

A *universally* composable [35] protocol is secure, regardless of the environment. More details can be found in Sec. 8.

The presented protocol properties can be seen as dimensional axes that define some space in which the protocols can be compared to each other. These axes are not orthogonal: if we want to improve one property then the worsening of some other property may be unavoidable. Below are some possibility and impossibility results of combining these properties that show how different axes depend on each other. More results can be found in [43].

10.1. Impossibility Results

If there are no cryptographic assumptions (assuming the existence of private channels), the following is required:

- If statistical or perfect security is obtained, then either broadcast or private channels must be assumed [47].
- For unconditional security (negligible failure probability) against t maliciously corrupted parties, $n/3 \leq t < n/2$, a broadcast channel is required [47].
- There can be no unconditionally secure protocol against an adversary controlling the majority of the parties. Two-party computation is impossible without cryptographic assumptions [47].
- No protocol can have perfect security against more than $n/3$ maliciously corrupted adversaries [47].
- Let t_a, t_p, and t_f be the number of actively corrupt, passively corrupt, and fail-corrupt parties, respectively. Perfect security is possible if and only if $3t_a + 2t_p + t_f < n$, regardless of the existence of a broadcast channel (a party that is passively corrupt and fail-corrupt at the same time is counted twice) [48].
- Unconditional security without a broadcast channel is possible if and only if $2t_a + 2t_p + t_f < n$ and $3t_a + t_f < n$ [48].
- Unconditional security, with a broadcast channel, is possible if and only if $2t_a + 2t_p + t_f < n$ [48].

For cryptographic security against $n/3$ or more maliciously corrupt players, either a trusted key setup or a broadcast channel is required [13].

Fail-stop adversaries can be tolerated only if there are fewer than $n/2$ corrupt parties, no matter what other assumptions we use [49].

No protocol with security against malicious adversaries can tolerate more than $n/2$ corrupted parties without losing the complete fairness property [50].

There is no protocol with UC security against more than $n/2$ corrupt parties without setup assumptions [51].

10.2. Possibility Results

Passive adversaries can be handled if fewer than $n/2$ participants are corrupt [47]. Active adversaries can be handled, if fewer than $n/2$ participants are corrupt and we are willing to tolerate an exponentially small chance of failure or leakage [50].

In the case of a mixed adversary, there are protocols that tolerate fewer than $n/3$ actively corrupt parties and further passively corrupt parties, so that the total number of corrupt parties is fewer than $n/2$ [48].

Using cryptographic assumptions, we can tolerate any number of active adversaries (although no protection against failures is guaranteed). As an example, we may take any protocol that uses threshold homomorphic encryption (see Sec. 6), taking an *n*-out-of-*n* threshold encryption.

References

[1] Oded Goldreich. *Foundations of Cryptography: Volume 2, Basic Applications*. Cambridge University Press, New York, NY, USA, 2004.

[2] Ronald Cramer, Ivan Damgård, and Jesper Buus Nielsen. Secure multiparty computation and secret sharing: An information theoretic approach (book draft), May 2013. http://www.cs.au.dk/~jbn/mpc-book.pdf.

[3] Manoj Prabhakaran and Amit Sahai, editors. *Secure Multiparty Computation*. Number 10 in Cryptology and Information Security Series. IOS Press, January 2013.

[4] Martin Hirt and Ueli M. Maurer. Player simulation and general adversary structures in perfect multiparty computation. *J. Cryptology*, 13(1):31–60, 2000.

[5] Oded Goldreich. *Foundations of Cryptography: Basic Tools*, volume 1. Cambridge University Press, New York, NY, USA, 2000.

[6] Mihir Bellare and Silvio Micali. Non-interactive oblivious transfer and spplications. In Gilles Brassard, editor, *Advances in Cryptology - CRYPTO '89, 9th Annual International Cryptology Conference, Santa Barbara, California, USA, August 20-24, 1989, Proceedings*, volume 435 of *Lecture Notes in Computer Science*, pages 547–557. Springer, 1989.

[7] Taher El Gamal. A public key cryptosystem and a signature scheme based on discrete logarithms. *IEEE Transactions on Information Theory*, 31(4):469–472, 1985.

[8] Mihir Bellare and Phillip Rogaway. Random oracles are practical: A paradigm for designing efficient protocols. In Dorothy E. Denning, Raymond Pyle, Ravi Ganesan, Ravi S. Sandhu, and Victoria Ashby, editors, *CCS '93, Proceedings of the 1st ACM Conference on Computer and Communications Security, Fairfax, Virginia, USA, November 3-5, 1993.*, pages 62–73. ACM, 1993.

[9] Joe Kilian. Founding Crytpography on Oblivious Transfer. In *Proceedings of the Twentieth Annual ACM Symposium on Theory of Computing*, STOC '88, pages 20–31, New York, NY, USA, 1988. ACM.

[10] Russell Impagliazzo and Steven Rudich. Limits on the provable consequences of one-way permutations. In Johnson [52], pages 44–61.

[11] Yuval Ishai, Joe Kilian, Kobbi Nissim, and Erez Petrank. Extending oblivious transfers efficiently. In Boneh [53], pages 145–161.

[12] David Chaum, Claude Crépeau, and Ivan Damgård. Multiparty Unconditionally Secure Protocols (Extended Abstract). In Simon [54], pages 11–19.

[13] 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.

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

[15] Yehuda Lindell and Benny Pinkas. A Proof of Security of Yao's Protocol for Two-Party Computation. *J. Cryptology*, 22(2):161–188, 2009.

[16] Yehuda Lindell, Benny Pinkas, and Nigel P. Smart. Implementing two-party computation efficiently with security against malicious adversaries. In Rafail Ostrovsky, Roberto De Prisco, and Ivan Visconti, editors, *Security and Cryptography for Networks, 6th International Conference, SCN 2008, Amalfi, Italy,*

September 10-12, 2008. Proceedings, volume 5229 of *Lecture Notes in Computer Science*, pages 2–20. Springer, 2008.

[17]   Mihir Bellare, Viet Tung Hoang, Sriram Keelveedhi, and Phillip Rogaway. Efficient garbling from a fixed-key blockcipher. In *2013 IEEE Symposium on Security and Privacy, SP 2013, Berkeley, CA, USA, May 19-22, 2013*, pages 478–492. IEEE Computer Society, 2013.

[18]   Vladimir Kolesnikov and Thomas Schneider. Improved garbled circuit: Free XOR gates and applications. In Luca Aceto, Ivan Damgård, Leslie Ann Goldberg, Magnús M. Halldórsson, Anna Ingólfsdóttir, and Igor Walukiewicz, editors, *Automata, Languages and Programming, 35th International Colloquium, ICALP 2008, Reykjavik, Iceland, July 7-11, 2008, Proceedings, Part II - Track B: Logic, Semantics, and Theory of Programming & Track C: Security and Cryptography Foundations*, volume 5126 of *Lecture Notes in Computer Science*, pages 486–498. Springer, 2008.

[19]   Benny Pinkas, Thomas Schneider, Nigel P. Smart, and Stephen C. Williams. Secure two-party computation is practical. In Mitsuru Matsui, editor, *Advances in Cryptology - ASIACRYPT 2009, 15th International Conference on the Theory and Application of Cryptology and Information Security, Tokyo, Japan, December 6-10, 2009. Proceedings*, volume 5912 of *Lecture Notes in Computer Science*, pages 250–267. Springer, 2009.

[20]   Samee Zahur, Mike Rosulek, and David Evans. Two halves make a whole: Reducing data transfer in garbled circuits using half gates. Cryptology ePrint Archive, Report 2014/756, 2014. http://eprint.iacr.org/.

[21]   Yehuda Lindell and Benny Pinkas. Secure two-party computation via cut-and-choose oblivious transfer. In Yuval Ishai, editor, *Theory of Cryptography - 8th Theory of Cryptography Conference, TCC 2011, Providence, RI, USA, March 28-30, 2011. Proceedings*, volume 6597 of *Lecture Notes in Computer Science*, pages 329–346. Springer, 2011.

[22]   Charanjit S. Jutla. Encryption modes with almost free message integrity. *J. Cryptology*, 21(4):547–578, 2008.

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

[24]   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.

[25]   Torben P. Pedersen. Non-interactive and information-theoretic secure verifiable secret sharing. In Feigenbaum [55], pages 129–140.

[26]   Ranjit Kumaresan, Arpita Patra, and C. Pandu Rangan. The round complexity of verifiable secret sharing: The statistical case. In Masayuki Abe, editor, *ASIACRYPT*, volume 6477 of *Lecture Notes in Computer Science*, pages 431–447. Springer, 2010.

[27]   Michael Backes, Aniket Kate, and Arpita Patra. Computational verifiable secret sharing revisited. In Dong Hoon Lee and Xiaoyun Wang, editors, *ASIACRYPT*, volume 7073 of *Lecture Notes in Computer Science*, pages 590–609. Springer, 2011.

[28]   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.

[29]   Ronald Cramer, Ivan Damgård, and Jesper Buus Nielsen. Multiparty computation from threshold homomorphic encryption. In Birgit Pfitzmann, editor, *EUROCRYPT*, volume 2045 of *Lecture Notes in Computer Science*, pages 280–299. Springer, 2001.

[30]   Ivan Damgård and Jesper Buus Nielsen. Universally composable efficient multiparty computation from threshold homomorphic encryption. In Boneh [53], pages 247–264.

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

[32]   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.

[33]   Craig Gentry. *A fully homomorphic encryption scheme*. PhD thesis, Stanford University, 2009. crypto.stanford.edu/craig.

[34]   Donald Beaver. Efficient multiparty protocols using circuit randomization. In Feigenbaum [55], pages 420–432.

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

[36]   Yehuda Lindell. *Composition of Secure Multi-Party Protocols, A Comprehensive Study*, volume 2815 of

Lecture Notes in Computer Science. Springer, 2003.

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

[38] 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.

[39] 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.

[40] 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 *Financial Cryptography*, pages 325–343, 2009.

[41] Joan Feigenbaum, Benny Pinkas, Raphael Ryger, and Felipe Saint-Jean. Secure computation of surveys. In *EU Workshop on Secure Multiparty Protocols*, 2004.

[42] Dan Bogdanov, Riivo Talviste, and Jan Willemson. Deploying secure multi-party computation for financial data analysis - (short paper). In Angelos D. Keromytis, editor, *Financial Cryptography*, volume 7397 of *Lecture Notes in Computer Science*, pages 57–64. Springer, 2012.

[43] Jason Perry, Debayan Gupta, Joan Feigenbaum, and Rebecca N. Wright. Systematizing secure computation for research and decision support. In Michel Abdalla and Roberto De Prisco, editors, *Security and Cryptography for Networks - 9th International Conference, SCN 2014, Amalfi, Italy, September 3-5, 2014. Proceedings*, volume 8642 of *Lecture Notes in Computer Science*, pages 380–397. Springer, 2014.

[44] Manuel Blum, Paul Feldman, and Silvio Micali. Non-interactive zero-knowledge and its applications. In *Proceedings of the Twentieth Annual ACM Symposium on Theory of Computing*, STOC '88, pages 103–112, New York, NY, USA, 1988. ACM.

[45] Zvi Galil, Stuart Haber, and Moti Yung. Cryptographic computation: Secure fault-tolerant protocols and the public-key model (extended abstract). In Carl Pomerance, editor, *Advances in Cryptology - CRYPTO '87*, volume 293 of *Lecture Notes in Computer Science*, pages 135–155. Springer Berlin Heidelberg, 1988.

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

[47] Michael Ben-Or, Shafi Goldwasser, and Avi Wigderson. Completeness theorems for non-cryptographic fault-tolerant distributed computation (extended abstract). In Simon [54], pages 1–10.

[48] Matthias Fitzi, Martin Hirt, and Ueli Maurer. Trading correctness for privacy in unconditional multiparty computation. In Hugo Krawczyk, editor, *Advances in Cryptology — CRYPTO '98*, volume 1462 of *Lecture Notes in Computer Science*, pages 121–136. Springer-Verlag, August 1998. Corrected proceedings version.

[49] R Cleve. Limits on the security of coin flips when half the processors are faulty. In *Proceedings of the Eighteenth Annual ACM Symposium on Theory of Computing*, STOC '86, pages 364–369, New York, NY, USA, 1986. ACM.

[50] Tal Rabin and Michael Ben-Or. Verifiable secret sharing and multiparty protocols with honest majority (extended abstract). In Johnson [52], pages 73–85.

[51] Ran Canetti, Eyal Kushilevitz, and Yehuda Lindell. On the limitations of universally composable two-party computation without set-up assumptions. In Eli Biham, editor, *Advances in Cryptology – EUROCRYPT 2003*, volume 2656 of *Lecture Notes in Computer Science*, pages 68–86. Springer Berlin Heidelberg, 2003.

[52] David S. Johnson, editor. *Proceedings of the 21st Annual ACM Symposium on Theory of Computing, May 14-17, 1989, Seattle, Washigton, USA*. ACM, 1989.

[53] Dan Boneh, editor. *Advances in Cryptology - CRYPTO 2003, 23rd Annual International Cryptology Conference, Santa Barbara, California, USA, August 17-21, 2003, Proceedings*, volume 2729 of *Lecture Notes in Computer Science*. Springer, 2003.

[54] Janos Simon, editor. *Proceedings of the 20th Annual ACM Symposium on Theory of Computing, May 2-4, 1988, Chicago, Illinois, USA*. ACM, 1988.

[55] Joan Feigenbaum, editor. *Advances in Cryptology - CRYPTO '91, 11th Annual International Cryptology Conference, Santa Barbara, California, USA, August 11-15, 1991, Proceedings*, volume 576 of *Lecture Notes in Computer Science*. Springer, 1992.

Chapter 2
Stateful Abstractions of Secure Multiparty Computation

Peeter LAUD [a]

[a] *Cybernetica AS, Estonia*

Abstract. In this chapter we present the Arithmetic Black Box (ABB) functionality. It is an ideal functionality that preserves the privacy of the data it stores and allows computations to be performed on the data stored, as well as retrieve certain pieces of data. We show that it is a very convenient abstraction of secure multiparty computation (SMC), which enables easy extensions and the construction of large privacy-preserving applications on top of it. In this chapter, we give a detailed definition of the ABB functionality and present different ways in which it can be implemented securely. We explain what extending an ABB means and how it is integrated into larger applications.

1. Stateful Secure Multiparty Computation

The secure multiparty computation (SMC) functionalities explored previously have been "one-shot": the parties provide input to the protocol, the protocol is executed, and the parties receive the output. Quite often, this does not correspond to real use cases — the input data may be received over time, a number of various computations may be executed on the same input data at different times, and the results of earlier computations may affect what is computed later and regarding which inputs. As an example, several statistical queries may be made against a private database, with the exact parameters of these queries being private to individual parties as well. At the same time, the database may undergo modifications and the parties may post new records, with each party providing a few fields of the record.

We can think of an extra, trusted party that maintains the state of this database [1, Sec. 7.7.1.3]. In this case, the reactive computation by n parties may be reduced to "ordinary" SMC by $n + 1$ parties. In order to securely evaluate (with security against up to t parties) a stateful functionality $(y_1, \ldots, y_n) = f(x_1, \ldots, x_n)$ that additionally reads and writes some persistent state, there needs to be an SMC protocol $\hat{\Pi}$ for the $(n + 1)$-party functionality $(y_1, \ldots, y_n, s') = \hat{f}(x_1, \ldots, x_n, s)$ that tolerates up to t corrupted parties among the first n.

In practice, the state maintained by the trusted party is somehow shared among the n parties, and the actions of the trusted party during the execution of the protocol $\hat{\Pi}$ are emulated by these parties as well, possibly using SMC subprotocols. Thus, the whole functionality is in the form $((y_1, s'_1), \ldots, (y_n, s'_n)) = f((x_1, s_1), \ldots, (x_n, s_n))$, where $s_1, \ldots, s_n$

are the shares of the database s, and $s'_1, \ldots, s'_n$ are the shares of the updated database s'. The sharings must provide privacy against t corrupted parties. If we require security against malicious parties, then the sharings also have to be verifiable and the functionality f must make sure that $s_1, \ldots, s_n$ have not been tampered with.

In this way, reactive n-party SMC is reduced to ordinary n-party SMC. The reduction is less than satisfactory, though, as the details of securely maintaining the state are visible in the definition of the ideal functionality. In the following, we define a more abstract ideal functionality for reactive SMC, show how to implement it securely, and explain how universally composable implementations of this functionality make the building of larger privacy-preserving applications particularly convenient. Our treatment of this topic is significantly inspired by [2].

2. Specifying the Arithmetic Black Box

The arithmetic black box is an ideal functionality $\mathcal{F}_{\mathsf{ABB}}$ that stores the values handed over to it, performs operations with the values stored according to the users' instructions (thereby adding new values to the store), and gives certain stored values back to the users, if a sufficiently large number of parties requests them. The functionality is reactive — after a command from the users it updates its store, potentially gives an answer, and is then ready for the next command.

The functionality $\mathcal{F}_{\mathsf{ABB}}$ is parametrized by the number of computing parties n, and by the allowed sets $\mathbf{P}_c \subseteq \mathbf{P} = \{P_1, \ldots, P_n\}$ of corrupted parties. To model the case where SMC is offered as a service, $\mathcal{F}_{\mathsf{ABB}}$ is also parametrized by the sets $\mathbf{P}_{\mathrm{in}}, \mathbf{P}_{\mathrm{comp}}, \mathbf{P}_{\mathrm{out}} \subseteq \mathbf{P}$ of *input*, *computing* and *output parties*. Here, the actual SMC operations are intended to be carried out by the parties in $\mathbf{P}_{\mathrm{comp}}$, while the data on which the computations are performed is provided by the parties in $\mathbf{P}_{\mathrm{in}}$, and the results are learned by the parties in $\mathbf{P}_{\mathrm{out}}$. We require $\mathbf{P}_{\mathrm{in}} \cup \mathbf{P}_{\mathrm{comp}} \cup \mathbf{P}_{\mathrm{out}} = \mathbf{P}$, but these sets do not have to be disjoint.

The functionality $\mathcal{F}_{\mathsf{ABB}}$ runs in two phases: the initialization phase and the computation phase. The initialization phase starts with the adversary specifying the set $\mathbf{P}_c$ of corrupted parties. If $\mathbf{P}_c$ is not among the allowed sets of corrupted parties, $\mathcal{F}_{\mathsf{ABB}}$ itself becomes corrupted — it will forward all the messages it receives to the adversary and behave according to the adversary's instructions. During the rest of the execution, $\mathcal{F}_{\mathsf{ABB}}$ no longer directly communicates with the corrupted parties. Instead, it will send to the adversary all messages meant for a corrupted party, and receive from the adversary the commands that the corrupted party would have provided.

The initialization phase continues by honest computing parties, as well as the adversary, who sends to $\mathcal{F}_{\mathsf{ABB}}$ the description of the ring R in which the computations will take place. The ring R is not specified as a parameter of $\mathcal{F}_{\mathsf{ABB}}$ because it may depend on the initialization of the computing parties. For example, it may be the ring $\mathbb{Z}_n$ for the modulus n used by the homomorphic encryption scheme set up by the parties. The ring must be agreed on also by the adversary S, as the adversary may need information about it (e.g. the public key) to perform a successful simulation. If the honest computing parties provide the descriptions of different rings R, then $\mathcal{F}_{\mathsf{ABB}}$ again becomes corrupted. If the adversary's R is different from the ring described by honest computing parties then $\mathcal{F}_{\mathsf{ABB}}$ shuts down.

The computation phase is the main phase of $\mathcal{F}_{\mathsf{ABB}}$. Internally, $\mathcal{F}_{\mathsf{ABB}}$ maintains a partial function *val* from *handles* to elements of the ring R. A handle is just a (bit-)string

stating the name used to refer to a particular value. In the specification, we may need to construct new handles from existing ones. For this purpose, we let $(\cdot, \cdot)$ be an injective function from pairs of bit-strings to handles.

At the start of the computation phase, *val* is undefined everywhere. The computation phase proceeds in *rounds*. At the beginning of each round, $\mathcal{F}_{\text{ABB}}$ receives zero or more commands from each of the parties. As these commands are executed, the definition of *val* is extended. All honest computing parties must give exactly the same commands to $\mathcal{F}_{\text{ABB}}$ (except for the store-command, as explained below). If this constraint is violated, the functionality $\mathcal{F}_{\text{ABB}}$ sends *val* to the adversary and becomes corrupted. Each command specifies zero or more handles that it *uses* and zero or more handles that it *defines*. The functionality $\mathcal{F}_{\text{ABB}}$ considers the commands received from honest computing parties; if $\mathcal{F}_{\text{ABB}}$ has not become corrupted then these will definitely coincide with respect to the used and defined handles (even the store-commands). The functionality $\mathcal{F}_{\text{ABB}}$ makes an *execution plan* for the commands from honest computing parties, ensuring that the used handles of each command are already in the range of *val* or defined by earlier commands. It is an error to

- define the same handle several times, define a handle already in the domain of *val*;
- use a handle that has not been defined, or have circular dependencies in the received commands (so that it is impossible to sort them topologically).

In case of an error, $\mathcal{F}_{\text{ABB}}$ sends *val* to the adversary and becomes corrupt.

Next, the functionality $\mathcal{F}_{\text{ABB}}$ sends the sanitized execution plan to the adversary. In the sanitized plan, each command is replaced by its sanitized version that omits sensitive values. We state the sanitized form of each command below. The adversary either approves or rejects the plan. In case of rejection, (reject) is sent by $\mathcal{F}_{\text{ABB}}$ to all the parties and the round ends.

In case of approval, the commands from the execution plan are executed. A command may succeed or fail. The success status of each command is sent back to the honest computing parties, and to the adversary. If a command fails, any handle that it was supposed to define remains undefined, and the subsequent commands using this handle will fail as well. The following commands are possible, and are executed as follows:

- On input $(\text{input}, (x, i))$ from all honest computing parties, and (input, x, v) from an input party P_i (if it was corrupted, then this input was actually given by the adversary), define $val((x, i)) = v$. The sanitized version of this command is $(\text{input}, (x, i), \text{true})$. If there was no input from P_i for the handle (x, i), then the command fails. The sanitized version of this command is $(\text{input}, (x, i), \text{false})$. In both cases, this command uses no handles, and defines the handle (x, i).
- On input $(\text{compute}, y_1, \ldots, y_l, \otimes, x_1, \ldots, x_k)$ from all honest computing parties, where $\otimes$ is a k-ary operation (which returns an l-tuple of values) that is among the operations that $\mathcal{F}_{\text{ABB}}$ can execute: compute $(w_1, \ldots, w_l) = \otimes(val(x_1), \ldots, val(x_k))$ and define $val(y_1) = w_1, \ldots, val(y_l) = w_l$. This command is equal to its sanitized version. It uses the handles $x_1, \ldots, x_k$ and defines the handles $y_1, \ldots, y_l$. Note that $\otimes$ may be probabilistic, and it may also probabilistically fail.
- On input (store, x, v) from an honest computing party P_i, and input $(\text{store}, x, ?, i)$ from all other honest computing parties: let $val(x) = v$. The sanitized version of

this command is $(\text{store}, x, ?, i)$. This command uses no handles and defines the handle x. This command cannot fail.

- On input $(\text{store}, x, ?, i)$ from all honest computing parties, where P_i is a corrupted computing party: the functionality $\mathcal{F}_{\text{ABB}}$ sends this command to the adversary and expects back a value $v \in R$, or nothing. If v is received, the functionality defines $val(x) = v$. If nothing is received, then the command fails. This command uses no handles and defines the handle x. It is equal to its sanitized version.

- On input $(\text{declassify}, x, i)$ from all honest computing parties, the functionality looks up $v = val(x)$ and sends (x, v) to the party P_i. This command uses the handle x and defines no handles. It is equal to its sanitized version.

A few points are worthy of discussion. First, it is immediately obvious that the adversary cannot see the actual values in the range of val, except for those that it itself provides through the input- or store-commands, or learns through the declassify command. The adversary can see the sequence of the commands executed. We cannot truly keep a real adversary (acting against the actual protocol Π) from learning this through traffic analysis. Hence, we claim that the specification of $\mathcal{F}_{\text{ABB}}$ corresponds to our intuition of a reactive SMC functionality.

Second, the operations $\otimes$ can be quite diverse. We always assume that a certain set of operations is available; these can be used to define others. We assume the presence of addition and multiplication operations. Also, for each $c \in R$, we assume that we have unary operations for constant addition: $v \mapsto c + v$, and constant multiplication: $v \mapsto cv$. For each $c \in R$, we also assume that there exists a nullary operation that returns c.

Clearly, it is reasonable to allow the operations to be probabilistic — there are many cases where we want to generate a random element of R. There can also be operations that return several values, e.g. the generation of a random invertible $v \in R^*$ together with its inverse. This operation also serves as an example of an operation that can probabilistically fail, at least in case of certain implementations of $\mathcal{F}_{\text{ABB}}$.

We could actually be even more general — instead of a single ring with certain operations, we could have a many-sorted algebraic structure with operations of mixed sorts. We are going to see an example of this in Chapter 6, where certain handles point to elements of the ring R, while others point to elements of the symmetric group S_k for some fixed k. There is an operation of permuting the values, taking an element $\sigma \in S_k$ and k ring elements $x_1, \ldots, x_k$, and returning k elements $y_1, \ldots, y_k$ satisfying $y_i = x_{\sigma(i)}$. In case of many sorts, we assume that the handles are typed — looking at a handle we can tell the sort of the value to which it points.

Third, we have stated that in certain cases, $\mathcal{F}_{\text{ABB}}$ becomes corrupt, i.e. it hands its internal state over to the adversary and follows its orders in the future. This can only happen if the user of the functionality $\mathcal{F}_{\text{ABB}}$ does something wrong. Normally, all honest computing parties should execute the same sequence of operations, which does not contain any programming errors.

3. Security of ABB Implementations

The functionality $\mathcal{F}_{\text{ABB}}$ provides SMC capability to a number of users. As we do not know who these users are and what else they are executing, it only makes sense to define the security of $\mathcal{F}_{\text{ABB}}$ in a universally composable (UC) way. In UC definitions, an extra

entity — the environment $\mathcal{Z}$ — models the activities taking place apart from the execution of the functionality or protocol we are interested in. In the following, we define how $\mathcal{Z}$ is executed together with an n-party ideal functionality $\mathcal{F}$ or a real protocol Π influenced by the adversary.

Ideal execution. In the beginning, the adversary S may *corrupt* certain parties among $P_1, \ldots, P_n$ by notifying these parties and $\mathcal{F}$. The environment $\mathcal{Z}$, the adversary S and the ideal functionality $\mathcal{F}$ will then execute. During the execution, $\mathcal{Z}$ does not directly communicate with $\mathcal{F}$. Instead, it makes queries to the parties $P_1, \ldots, P_n$ and expects answers from them. An uncorrupted party P_i forwards the queries it gets from $\mathcal{Z}$ to $\mathcal{F}$ and relays the answers from $\mathcal{F}$ back to $\mathcal{Z}$. A corrupted party no longer communicates with any other component. The adversary S may freely communicate with both $\mathcal{Z}$ and $\mathcal{F}$ (according to the interface provided by the latter). At some point, $\mathcal{Z}$ stops and outputs a bit. Let $\mathbf{p}_{\mathcal{F}\|S\|\mathcal{Z}}$ denote the probability that this bit is 1.

Real execution. The protocol Π is realized by the interactive (Turing) machines $M_1, \ldots, M_n$, executed by the parties $P_1, \ldots, P_n$. At the beginning of the execution, the adversary A may again corrupt certain parties among $P_1, \ldots, P_n$ by notifying them. Afterwards, $\mathcal{Z}$ and A run in parallel with the machines M_i for uncorrupted P_i. The uncorrupted parties P_i relay the queries and answers between $\mathcal{Z}$ and M_i. A corrupted party P_i ignores any further input. All messages meant for corrupted parties are sent to A instead. Also, A can masquerade any corrupt party to any M_i. At some point, $\mathcal{Z}$ again stops and outputs a bit. Let $\mathbf{p}_{\Pi\|A\|\mathcal{Z}}$ denote the probability that this bit is 1.

The bit output by $\mathcal{Z}$ should be interpreted as its guess whether or not it is a part of the real execution. The protocol Π is a secure implementation if anything that can happen to $\mathcal{Z}$ in the real execution can also happen in the ideal execution [3]. Formally,

Definition 1 *A protocol Π is (perfectly) at least as secure as* the ideal functionality $\mathcal{F}$, if *for all adversaries A there exists an adversary S, such that for all environments $\mathcal{Z}$, we have $\mathbf{p}_{\mathcal{F}\|S\|\mathcal{Z}} = \mathbf{p}_{\Pi\|A\|\mathcal{Z}}$.*

Here the equality between the two probabilities is a very strong requirement and obliges the protocol Π to use only information-theoretically secure methods. To model the security of protocols using cryptography, we relax the equality of the probabilities to statistical closeness, or to the negligibility of their difference. See [4, Sec. 1.3] for more details.

A protocol Π is a secure implementation of $\mathcal{F}_{\mathrm{ABB}}$ if it satisfies the previous definition. This definition actually corresponds to an *active* adversary corrupting the parties — it assumes full control over the parties it has corrupted, receiving the messages meant for them and computing the replies. In the context of SMC, we are also interested in *passive* adversaries that may see all inputs and the internal state of corrupted parties, but do not interfere with the actual execution. Hence we define that

- an adversary A in the real execution is *passive* if it computes the messages a corrupted party P_i sends out in the same way as the machine M_i that is part of the definition of the protocol Π;

• an adversary S in the ideal execution with $\mathcal{F}_{\mathsf{ABB}}$ is *passive* if it never rejects the execution plans from $\mathcal{F}_{\mathsf{ABB}}$, and always responds with a value $v \in R$ when $\mathcal{F}_{\mathsf{ABB}}$ forwards the command $(\mathsf{store}, x, ?, i)$ from honest parties to it.

We say that Π is a passively secure implementation of $\mathcal{F}_{\mathsf{ABB}}$, if Def. 1 holds for all passive adversaries A and, moreover, the adversary S can be chosen from among the passive adversaries.

Very often, the security of the implementation Π of some ideal functionality $\mathcal{F}$ is argued as follows. For each A, we have to provide an adversary S so that the environment $\mathcal{Z}$ could not distinguish $\Pi\|A$ from $\mathcal{F}\|S$. The adversary S is constructed as a composition $Sim\|A$, where the *simulator Sim* does not depend on A or $\mathcal{Z}$. The task of the simulator is to translate the messages between $\mathcal{F}$ and A so that A would think it was communicating with the components of Π. In effect, this makes Π indistinguishable from $\mathcal{F}\|Sim$.

4. Example Implementations

Several SMC protocols for evaluating Boolean and arithmetic circuits, outlined in Chapter 1, can actually be viewed as secure implementations of $\mathcal{F}_{\mathsf{ABB}}$, as they do not require the circuit to be fixed before the start of the protocol. We will review some of them.

4.1. Implementations Based on Shamir's Secret Sharing

Consider the functionality $\mathcal{F}_{\mathsf{ABB}}$ for n parties that all fulfill the roles of input, computing and output parties. Let the number of tolerated corruptions be $t < n/2$. Let the initialization phase always pick the field $\mathbb{F}$ as the ring R in which the computations take place, with $|\mathbb{F}| \geq n+1$. The functionality $\mathcal{F}_{\mathsf{ABB}}$ can be implemented by a protocol Π, specified by interactive Turing machines $M_1, \ldots, M_n$, with the machine M_i executed by the party P_i. Let $c_1, \ldots, c_n \in \mathbb{F}$ be n different non-zero elements of $\mathbb{F}$.

The initialization phase of Π is trivial. During the computation phase, each machine M_i (for an uncorrupted P_i) maintains a partial function val_i from the handles to the elements of $\mathbb{F}$ (initially empty). For a handle x, the intended relationship between the point $val(x)$ in the state of $\mathcal{F}_{\mathsf{ABB}}$ and the points $val_i(x)$ in the states of the machines M_i is the obvious one — there is a polynomial f_x of degree at most t, such that $f_x(0) = val(x)$ and $f_x(c_i) = val_i(x)$ for each i where P_i is uncorrupted.

An implementation secure against passive adversaries functions similarly to Sec. 5 of Chapter 1. For a party P_i to input a value v under handle (x, i), or to store a value v under such a handle, it randomly generates a polynomial f of degree at most $t-1$ satisfying $f(0) = v$, updates $val_i((x, i)) = f(c_i)$ and sends $f(c_j)$ to party P_j for all $j \neq i$. The party P_j receives $f(c_j)$ and stores it as $val_j(x)$. For additions, multiplications with a constant, and adding a constant, the parties perform the same operation with their shares. To multiply two values pointed to by handles x and y, the machines $M_1, \ldots, M_n$ execute the multiplication protocol for shared values [5] as described in Chapter 1. For declassifying the value pointed to by handle x to the party P_i, all other M_j send $val_j(x)$ to M_i, which recombines them to the stored value.

Due to the properties of polynomials over finite fields, the view (i.e. the received messages) of any coalition of at most t parties consists of only uniformly randomly distributed elements of $\mathbb{F}$. This allows for a straightforward construction of the simulator

Sim — it uses random elements of $\mathbb{F}$ to model any messages sent from an honest party to a corrupted one, except when simulating the execution of the declassification command. In this case, *Sim* learns the declassified value from $\mathcal{F}_{\mathsf{ABB}}$ and can pick the shares of the honest parties so that they would recombine to the correct value. An important part of the proof shows that the intended relationship between the mappings *val* and val_i indeed holds throughout the execution.

VIFF [6], SEPIA [7] and PICCO [8] are among the existing implementations of $\mathcal{F}_{\mathsf{ABB}}$, based on Shamir's secret sharing. These frameworks offer an API to the application programmer that is similar to the ABB specification in Sec. 2.

If security against malicious parties is desired, then beside $val_i(x)$, the state of M_i also contains the necessary commitments for each known handle x. In particular, if Pedersen's verifiable secret sharing (VSS) [9] is used, as described in Chapter 1, the commitments to the coefficients of the polynomial f used to share $val(x)$ are stored by all M_i alongside $val_i(x)$. These commitments are distributed using a broadcast functionality $\mathcal{F}_{\mathsf{BC}}$, ensuring that all parties have the same commitments. In turn, the functionality $\mathcal{F}_{\mathsf{BC}}$ is implemented using either cryptographic mechanisms (signatures) or information-theoretically secure protocols (usable if $t < n/3$).

4.2. Sharemind

The SHAREMIND SMC platform [10,11] provides an implementation of $\mathcal{F}_{\mathsf{ABB}}$ for a number of parties that include exactly three computing parties, and is secure against the adversary passively corrupting one computing party (the number of corruptions of non-computing parties is unlimited). The domain of computations in SHAREMIND is many-sorted: it provides arithmetic and relational operations in rings $\mathbb{Z}_{2^n}$ of n-bit integers for all $n \in \mathbb{N}$, as well as conversions between integers of different bit-lengths. This domain is fixed, and thus the initialization phase of SHAREMIND is trivial, too. During computation, each of the computing parties P_1, P_2, P_3 maintains a partial mapping val_i from handles to $\bigcup_{n \in \mathbb{N}} \mathbb{Z}_{2^n}$; the intended relationship between the point $val(x)$ in the state of $\mathcal{F}_{\mathsf{ABB}}$, and the points $val_i(x)$ in the states of the computing parties is $val(x) = val_1(x) + val_2(x) + val_3(x)$ in the ring $\mathbb{Z}_{2^n}$, where n is the intended bit-length of the value of x.

In the following description we adopt the notation used in many works on ABB implementations, where $val_i(x)$ is denoted by $[\![x]\!]_i$ and the tuple of all $[\![x]\!]_i$ by $[\![x]\!]$. The basic computational protocols for SHAREMIND are the following:

- To input a value v of length n under handle x, the inputting party generates three random values $v_1, v_2, v_3 \in \mathbb{Z}_{2^n}$ subject to $v_1 + v_2 + v_3 = v$, and sends v_i to the computing party P_i, which defines $[\![x]\!]_i = v_i$.
- To declassify a value to which handle x points, each computing party P_i sends $[\![x]\!]_i$ to the party supposed to learn $val(x)$, and this party adds them together.
- To add two values of the same bit-length, or to multiply a value with a constant c, each computing party performs this operation on its own share(s).
- The multiplication protocol of SHAREMIND is given in Alg. 2, with the Reshare subprotocol given in Alg. 1. Both algorithms are discussed more thoroughly in [11]. Here the indices of parties are given *modulo* 3.

Similarly to Shamir's secret-sharing-based implementations, the view of any single party in the ABB implementation of SHAREMIND consists of independent, uniformly ran-

---

Algorithm 1: Resharing protocol $\llbracket w \rrbracket \leftarrow \mathsf{Reshare}(\llbracket u \rrbracket)$ in SHAREMIND

Data: Value $\llbracket u \rrbracket \in \mathbb{Z}_{2^n}$
Result: Value $\llbracket w \rrbracket$ such that $w = u$ and the components of $\llbracket w \rrbracket$ are independent of everything else

Party P_i generates $r_i \xleftarrow{\$} \mathbb{Z}_{2^n}$, sends it to party P_{i+1}
Party P_i computes $\llbracket w \rrbracket_i \leftarrow \llbracket u \rrbracket_i + r_i - r_{i-1}$

Algorithm 2: Multiplication protocol in the ABB of SHAREMIND

Data: Values $\llbracket u \rrbracket$ and $\llbracket v \rrbracket$
Result: Value $\llbracket w \rrbracket$, such that $w = uv$

1   $\llbracket u' \rrbracket \leftarrow \mathsf{Reshare}(\llbracket u \rrbracket)$
2   $\llbracket v' \rrbracket \leftarrow \mathsf{Reshare}(\llbracket v \rrbracket)$
3   Party P_i sends $\llbracket u' \rrbracket_i$ and $\llbracket v' \rrbracket_i$ to party P_{i+1}
4   Party P_i computes $\llbracket w' \rrbracket_i \leftarrow \llbracket u' \rrbracket_i \cdot \llbracket v' \rrbracket_i + \llbracket u' \rrbracket_i \cdot \llbracket v' \rrbracket_{i-1} + \llbracket u' \rrbracket_{i-1} \cdot \llbracket v' \rrbracket_i$
5   $\llbracket w \rrbracket \leftarrow \mathsf{Reshare}(\llbracket w' \rrbracket)$

---

domly distributed elements of $\mathbb{Z}_{2^n}$. Thus, the simulator demonstrating the security of the implementation works in the same way — all messages to the corrupt party (except during the declassification command) are simulated as random elements of $\mathbb{Z}_{2^n}$. Here n may be different for different messages; it is determined by the commands actually executed.

As an example of the simplicity of protocols in the ABB implementation of SHARE-MIND, we also present the protocol for checking the equality of two values stored in the ABB. Note that $\llbracket x \rrbracket = \llbracket y \rrbracket$ is equivalent to $(\llbracket x \rrbracket - \llbracket y \rrbracket) = 0$, and the subtraction of two stored values can be computed without communication. This idea is used to check whether a stored value is equal to zero in Alg. 3 taken from [11].

Algorithm 3: Equality protocol in the ABB of SHAREMIND

Data: Value $\llbracket u \rrbracket \in \mathbb{Z}_{2^n}$
Result: Value $\llbracket b \rrbracket \in \mathbb{Z}_2$, where $b = 1$ iff $u = 0$

1   Party P_1 generates $r \xleftarrow{\$} \mathbb{Z}_{2^n}$, sends r to P_2 and $\llbracket u \rrbracket_1 - r$ to P_3
2   Party P_1 takes $(\llbracket b_1 \rrbracket_1, \ldots, \llbracket b_n \rrbracket_1) = (1, \ldots, 1) \in \mathbb{Z}_2^n$
3   Party P_2 defines $\llbracket b_i \rrbracket_2$ as the i-th bit of $\llbracket u \rrbracket_2 + r$ (for all $i \in \{1, \ldots, n\}$)
4   Party P_3 defines $\llbracket b_i \rrbracket_3$ as the i-th bit of $r - \llbracket u \rrbracket_1 - \llbracket u \rrbracket_3$ (for all $i \in \{1, \ldots, n\}$)
5   Let $\llbracket b \rrbracket = \bigwedge_{i=1}^{n} \llbracket b_i \rrbracket$

In Alg. 3, party P_1 first shares u among parties P_2 and P_3 only. This does not violate privacy, as only one of the parties can be corrupted. Let $\llbracket u' \rrbracket$ be the following shared value:

$$(\llbracket u' \rrbracket_1, \llbracket u' \rrbracket_2, \llbracket u' \rrbracket_3) = (0, \llbracket u \rrbracket_2 + r, \llbracket u \rrbracket_3 + \llbracket u \rrbracket_1 - r) \ .$$

Hence $u' = u$. As $\llbracket u' \rrbracket_1 = 0$, the value u' is zero if $\llbracket u' \rrbracket_2 = -\llbracket u' \rrbracket_3$. On lines 2–5 of Alg. 3, the computing parties check this equality. The bit $\llbracket b_i \rrbracket = \llbracket b_i \rrbracket_1 + \llbracket b_i \rrbracket_2 + \llbracket b_i \rrbracket_3$ is equal to 1

if the i-th bits of $[\![u']\!]_2$ and $-[\![u']\!]_3$ are equal. The result $[\![b]\!]$ is computed as the conjunction of the bits $[\![b_i]\!]$ using $n-1$ invocations of the multiplication protocol (Alg. 2) for 1-bit values. These invocations can be parallelized so that there are only $\lceil \log n \rceil$ rounds of multiplication.

Several more specific protocols have been constructed and implemented for the additively shared values, as used by SHAREMIND [11,12,13].

4.3. Implementations Using Homomorphic Encryption

While the existing implementations of ABB are based on the previously described representations due to performance reasons, other possible constructions have been proposed. In [14], where the notion of ABB was originally proposed, the implementation was based on *threshold homomorphic encryption* — on public-key encryption that additionally has the following properties:

- The secret key sk has been secret-shared among the parties $P_1, \ldots, P_n$; let sk_i denote the share owned by P_i. To decrypt a ciphertext c and make the plaintext known to some party P', the party P_i (for $i \in \{1, \ldots, n\}$) computes $m_i = \mathcal{D}ec(sk_i, c)$ and sends it to P'. The party P' combines the *plaintext shares* $m_1, \ldots, m_n$ and learns the plaintext. Potentially, only $t < n$ shares are needed to recover the plaintext, and it may also be possible to detect incorrect shares.
- The ciphertexts c_1 and c_2 corresponding to the plaintexts m_1 and m_2 can be combined to a ciphertext $c_1 \oplus c_2$ corresponding to the plaintext $m_1 + m_2$. Here m_1 and m_2 are elements of a certain ring R and the addition takes place in this ring. Similarly, given a value $v \in R$, a ciphertext c corresponding to the plaintext $m \in R$ can be transformed into a ciphertext $v \odot c$ corresponding to $v \cdot m$. These transformations can be done without access to the secret key.

The homomorphic encryption scheme by Paillier [15] has been used in ABB implementations most often. It is based on algebraic structures similar to RSA and the methods for adding threshold decryption to RSA [16] can be adapted to it.

For this ABB implementation Π, the initialization phase is no longer trivial — here the machines $M_1, \ldots, M_n$ realizing Π have to agree on a common public key pk and learn the shares $sk_1, \ldots, sk_n$ of the secret key sk. For Paillier's encryption scheme, the key generation is the same as for RSA and the distributed mechanisms for that [17] can be used. The initialization phase also defines the ring R as $\mathbb{Z}_N$, where N is the modulus of the agreed RSA key.

During the execution, the machines M_i maintain a partial mapping val_i from handles to ciphertexts. The correspondence between the mapping val maintained by $\mathcal{F}_{\text{ABB}}$ and the mappings val_i maintained by the machines M_i is simply $val(x) = \mathcal{D}ec(sk, val_i(x))$ for all handles x that have been defined. In particular, all mappings $val_1, \ldots, val_n$ are equal to each other.

Obviously, a value is input to the computation by simply encrypting it, and declassified by the output party learning all plaintext shares. The addition of values and the multiplication of a value with a public constant are done locally by each computing party using the homomorphic properties of the encryption. A protocol for the multiplication of two values stored in the ABB is given in Chapter 1.

If the adversary corrupts fewer than t parties necessary for correct decryption, then the ciphertexts do not give it any information on the actual values. Thus, the simulator

Sim can send arbitrary ciphertexts to the adversary on behalf of the corrupted parties. Whenever a corrupted party learns a declassified value, the simulator is able to fix the rest of the plaintext shares so that all the shares recombine to the value made public by $\mathcal{F}_{\mathsf{ABB}}$.

Homomorphic encryption (without the threshold decryption property) can also be used to give an ABB implementation for two parties, where the values are additively shared by the parties. In the initialization phase, party P_1 generates a keypair (pk, sk) for Paillier encryption and sends pk to P_2. The computations will take place in the ring $R = \mathbb{Z}_N$, where N is the modulus of pk. Each value v is stored as (v_1, v_2), where P_i knows v_i and $v_1 + v_2 = v$. It is clear how the inputs and declassifications as well as additions and multiplications with a public value are implemented. To compute the product of $v = v_1 + v_2$ and $v' = v'_1 + v'_2$, the parties proceed as follows. Party P_i is able to compute $w_i = v_i \cdot v'_i$. They additionally need to compute random r_1, r_2, s_1, s_2 so that $r_1 + r_2 = v_1 \cdot v'_2$ and $s_1 + s_2 = v_2 \cdot v'_1$. Party P_i can then take $w_i + r_i + s_i$ as its share of $v \cdot v'$.

The following protocol computes an additive sharing $r_1 + r_2$ of ab, where P_1 knows a and P_2 knows b. Party P_1 sends $c = \mathcal{E}nc(pk, a)$ to P_2. Party P_2 generates a random $r_2 \in \mathbb{Z}_N$, computes $c' = b \odot c \oplus \mathcal{E}nc(pk, -r_2)$ and sends it back to P_1. Party P_1 defines $r_1 = \mathcal{D}ec(sk, c')$. The protocol is secure against a semi-honest adversary.

Such an implementation of the ABB is provided by the TASTY framework [18]. It has also been used as part of the implementation of numerous two-party protocols, often implicitly.

4.4. Implementations for Boolean Circuits

The previous implementations of ABB can be adapted to run Boolean circuits, as the computations in the field $\mathbb{Z}_2$ can be emulated in larger rings. Indeed, if we encode false as 0 and true as 1, then $b_1 \wedge b_2$ is equal to $b_1 \cdot b_2$ and $b_1 \oplus b_2$ is equal to $b_1 + b_2 - 2b_1b_2$ in any ring (where 2 means $1 + 1$).

Still, the protocols presented in Chapter 1 for the private evaluation of Boolean circuits can also be adapted to ABB implementation, secure against the passive adversaries (at least). The Goldwasser-Micali-Widgerson protocol in Chapter 1 has already been presented as a protocol Π_{GMW} for an ABB storing Booleans. We can think of Π_{GMW} working in the $\mathcal{F}_{\mathsf{OT}}$-hybrid model, where $\mathcal{F}_{\mathsf{OT}}$ is the ideal functionality for oblivious transfer. A simulator *Sim* for demonstrating that Π_{GMW} is at least as secure as $\mathcal{F}_{\mathsf{ABB}}$ converts the commands in the execution plan received from $\mathcal{F}_{\mathsf{ABB}}$ into the notifications that $\mathcal{F}_{\mathsf{OT}}$ sends to the adversary. To generate these notifications, the simulator does not need to know the actual values used by the computation. The resulting protocol is secure against the same kinds of adversaries as the implementation used for $\mathcal{F}_{\mathsf{OT}}$. Universally composable implementations for oblivious transfer have been proposed in [19].

Even garbled circuits [20] (see Chapter 1) can be turned into an ABB implementation for two parties secure against a passive adversary, again in the $\mathcal{F}_{\mathsf{OT}}$-hybrid model. In this implementation, the machines M_1 of party P_1 (the garbler) and M_2 of party P_2 (the evaluator) maintain the following state. Party P_1 has two partial mappings val_1^{true} and val_1^{false} from handles to encryption keys. Party P_2 has one partial mapping val_2 from handles to encryption keys. Before and after each command, val_1^{true}, val_1^{false} and val_2 are defined for the same handles. The relationship between these mappings and the mapping val maintained by $\mathcal{F}_{\mathsf{ABB}}$ is the following: $val_2(x) = val_1^{val(x)}(x)$ for all handles x in the

domain of these mappings. Whenever M_1 and M_2 receive a command to evaluate a gate on some of the values they have stored and store the result under the handle x, the machine M_1 generates two new keys and stores them in $val_1^{\text{true}}(x)$ and $val_1^{\text{false}}(x)$, garbles this gate and sends it to M_2, which proceeds to evaluate it and learn the key corresponding to $val(x)$. Whenever the garbling party P_1 inputs a new Boolean value, the machine M_1 generates two new keys and sends one of them to M_2. Whenever the evaluating party P_2 inputs a new Boolean value, M_1 generates two new keys and transmits one of them to M_2 with the help of $\mathcal{F}_{\text{OT}}$. Whenever the value to which handle x points is declassified for party P_2, the machine M_1 encrypts b under $val_1^b(x)$ (for $b \in \{\text{true}, \text{false}\}$) and sends both ciphertexts to M_2, which succeeds in decrypting one of them. If this value had been declassified for party P_1 instead, the machine M_2 would have simply sent $val_2(x)$ to M_1.

In this ABB implementation, the view of any single party consists of random encryption keys and ciphertexts. It is possible to simulate either the view of P_1 or the view of P_2. Obviously, these simulation strategies are quite different due to the asymmetric nature of the protocol. We will not provide the details of the construction of the simulator *Sim* here.

Such *streaming garbled circuits* have been investigated in [21], even though not in the context of ABB implementations. The methods for securing them against malicious adversaries have also been investigated [22].

5. Extending the ABB

The definition of UC security (Def. 1) is the appropriate security notion for protocols running in an open world. The security definition also has a property — composability — that is very useful in the context of ABB-based SMC. It allows us to use the ABB to construct large privacy-preserving applications with precisely defined privacy guarantees, and to establish these guarantees by arguing in terms of the properties of $\mathcal{F}_{\text{ABB}}$ only, not its implementation. The composability also allows us to "add new operations" to an ABB: to implement an ABB with a richer set of operations on top of an ABB that only provides more basic operations.

In order to use the ideal components in the security arguments of larger protocols, we have to define the execution of real protocols for ideal components. A *hybrid protocol* $\Pi^{\mathcal{F}}$ for n parties consists of interactive Turing machines $M_1, \ldots, M_n$ implementing the functionality of the protocol for each party, as well as an ideal component $\mathcal{F}$ (also for n parties). The execution of $\Pi^{\mathcal{F}}$ together with the parties $P_1, \ldots, P_n$, adversary A and the environment $\mathcal{Z}$ proceeds in the same way as the real execution we defined in Sec. 3. Additionally, the machines $M_1, \ldots, M_n$ as well as the adversary may make queries to $\mathcal{F}$ according to the interface it provides, and receive replies from it. When the adversary corrupts a number of parties at the beginning of the execution, it also sends the same corruption query to $\mathcal{F}$.

If a protocol $\Pi^{\mathcal{G}}$ (for an ideal functionality $\mathcal{G}$) is at least as secure as $\mathcal{F}$, then we say that Π securely implements $\mathcal{F}$ *in the $\mathcal{G}$-hybrid model*. The following composition theorem [3] holds.

Theorem 1 *If Π securely implements $\mathcal{F}$ in the $\mathcal{G}$-hybrid model and Ξ securely implements $\mathcal{G}$, then Π^{Ξ} (protocol Π, where the calls to $\mathcal{F}$ have been replaced with the executions of the Turing machines realizing Ξ) securely implements $\mathcal{F}$.*

The composition theorem is used several times in later chapters of this book (often implicitly) when arguing the security of a privacy-preserving task that is implemented on top of an ABB. We also use it to add new operations to (the implementation of) an ABB and argue the security of these additions. Let us demonstrate the technique by adding some useful operations to a basic ABB that computes with the elements of a finite field $\mathbb{F}$ [23]. We define the following ideal functionalities:

- $\mathcal{F}^1_{\text{ABB}}$ is an arithmetic black box as defined in Sec. 2, with a certain number of parties and their roles, tolerating certain corruptions, and further specified as follows:

  * In the initialization phase, only a finite field $\mathbb{F}$ may be used as the ring R;
  * The available operations to be used in the compute-command are binary addition, binary multiplication, addition of constant c (for all $c \in \mathbb{F}$), multiplication with a constant c (for all $c \in \mathbb{F}$) and the generation of constant c (for all $c \in \mathbb{F}$).

- $\mathcal{F}^2_{\text{ABB}}$ is an ABB identical to $\mathcal{F}^1_{\text{ABB}}$, except that the set of available operations also contains a nullary operation that returns a uniformly randomly chosen element of $\mathbb{F}$. Note that this element will remain secret, i.e. it can be accessed only through a handle.

- $\mathcal{F}^3_{\text{ABB}}$ is an ABB identical to $\mathcal{F}^2_{\text{ABB}}$, except that the set of available operations also contains a nullary operation that returns a pair of values (v, v^{-1}), where v is uniformly randomly sampled from $\mathbb{F}^*$.

- $\mathcal{F}^4_{\text{ABB}}$ is an ABB identical to $\mathcal{F}^3_{\text{ABB}}$, except that the set of available operations also contains the operation prefmult that takes a list of arguments $(v_1, \ldots, v_k)$ and returns $(w_1, \ldots, w_k)$, where $w_j = \prod_{i=1}^{j} v_i$. All v_i must be elements of $\mathbb{F}^*$. If a certain handle used points to 0, then $\mathcal{F}^4_{\text{ABB}}$ sends its entire internal state λ to the adversary and becomes corrupt.

In [23], further operations are added to the ABB, eventually arriving at equality and inequality comparisons between values stored in the ABB.

We will now show how to securely implement each $\mathcal{F}^i_{\text{ABB}}$ in the $\mathcal{F}^{i-1}_{\text{ABB}}$-hybrid model. The protocols we specify work in rounds, similarly to $\mathcal{F}_{\text{ABB}}$. One round for the implementation of $\mathcal{F}^i_{\text{ABB}}$ may contain several rounds for the component $\mathcal{F}^{i-1}_{\text{ABB}}$. The progression of rounds of $\mathcal{F}^{i-1}_{\text{ABB}}$ is controlled by honest computing parties.

5.1. Implementing $\mathcal{F}^2_{\text{ABB}}$

The protocol Π_2 securely implementing $\mathcal{F}^2_{\text{ABB}}$ is specified by the interactive Turing machines $M_1, \ldots, M_n$ working for the parties $P_1, \ldots, P_n$. In each round, the machine M_i receives a number of commands from P_i, unless it is corrupted. The machine M_i makes a two-part execution plan. The first part contains all requests for the creation of random elements, while the second part contains all other commands received in the current round. For a computing party P_i, the machine M_i will then handle the random element creation requests as follows:

- On input (compute, x, random), generate a random $v \in \mathbb{F}$ and send the command (store, $(1, (x, i)), v$) to $\mathcal{F}^1_{\text{ABB}}$. Also send the commands (store, $(1, (x, j)), ?, j$) to $\mathcal{F}^1_{\text{ABB}}$ for all computing parties P_j different from P_i and finish the round of $\mathcal{F}^1_{\text{ABB}}$. Let $\{c_1, \ldots, c_l\}$ be the set of indices of computing parties that actually stored a

value in $\mathcal{F}_{\mathsf{ABB}}^1$ during the finished round; this set becomes known to all honest computing parties. In the next round, send the commands

$$(\mathsf{compute}, (1, (x, n+1)), \text{``+''}, (1, (x, c_1)), (1, (x, c_2)))$$

$$(\mathsf{compute}, (1, (x, n+2)), \text{``+''}, (1, (x, n+1)), (1, (x, c_3)))$$

$$\dots\dots\dots$$

$$(\mathsf{compute}, (0, x), \text{``+''}, (1, (x, n+l-2)), (1, (x, c_l)))$$

to $\mathcal{F}_{\mathsf{ABB}}^1$, in order to add together all the values pointed to by handles $(1, (x, c_i))$ and let the handle $(0, x)$ point to the sum. Finish the round of $\mathcal{F}_{\mathsf{ABB}}^1$.

If P_i is not a computing party, then M_i simply ignores the requests to create random elements. As a second step of implementing a round of $\mathcal{F}_{\mathsf{ABB}}^2$, the machines M_i forward all other commands to $\mathcal{F}_{\mathsf{ABB}}^1$ and relay the answers back to P_i. While doing this, they translate each handle x to $(0, x)$.

We see that for each command to create a handle x in $\mathcal{F}_{\mathsf{ABB}}^2$ pointing to some value, a handle $(0, x)$ is created in $\mathcal{F}_{\mathsf{ABB}}^1$ during the run of the protocol Π_2, pointing to the same value. Next, we describe a simulator *Sim* in order to argue the security of the implementation of $\mathcal{F}_{\mathsf{ABB}}^2$.

The simulator *Sim* receives the execution plans for each round of $\mathcal{F}_{\mathsf{ABB}}^2$, thereby learning the commands submitted for execution by honest parties. The simulator translates the execution plan of each round to the execution plans of the rounds of $\mathcal{F}_{\mathsf{ABB}}^1$. This consists of separating the commands to generate random elements, translating them into sequences of commands similarly to the machines M_i, and translating the handles. The simulator *Sim* asks the adversary for the approval of the execution plans for all rounds of $\mathcal{F}_{\mathsf{ABB}}^1$. If rejected, it rejects the execution plan of $\mathcal{F}_{\mathsf{ABB}}^2$. The simulator also relays the adversary's answers to different commands back to $\mathcal{F}_{\mathsf{ABB}}^2$.

We can show that if the machines $M_1, \dots, M_n$ and $\mathcal{F}_{\mathsf{ABB}}^1$ of the protocol Π_2 start a round in a state equivalent to the state of the composition of $\mathcal{F}_{\mathsf{ABB}}^2$ and *Sim*, then they also finish in equivalent states. Out of all the machines, only $\mathcal{F}_{\mathsf{ABB}}^1$ and $\mathcal{F}_{\mathsf{ABB}}^2$ have persistent states. The states of these two machines are equivalent if for each handle x pointing to a value v in $\mathcal{F}_{\mathsf{ABB}}^2$ there is a handle $(0, x)$ in $\mathcal{F}_{\mathsf{ABB}}^1$ pointing to the same value, and vice versa. There may be more handles in $\mathcal{F}_{\mathsf{ABB}}^1$ that are not in the form $(0, x)$. These handles do not affect subsequent computations because the machines $M_1, \dots, M_n$ do not access them. As the equivalence of states is preserved, we have a form of *probabilistic bisimulation* [24] between Π_2 and $\mathcal{F}_{\mathsf{ABB}}^2 \| Sim$.

The previous argument shows that Π_2 is perfectly at least as secure as $\mathcal{F}_{\mathsf{ABB}}^2$ in the $\mathcal{F}_{\mathsf{ABB}}^1$-hybrid model. We note that if A is a passive adversary, then so is $Sim \| A$. Hence Π_2 is also a passively secure implementation of $\mathcal{F}_{\mathsf{ABB}}^2$ in the $\mathcal{F}_{\mathsf{ABB}}^1$-hybrid model.

5.2. Implementing $\mathcal{F}_{\mathsf{ABB}}^3$

The protocol Π_3 securely implementing $\mathcal{F}_{\mathsf{ABB}}^3$ in the $\mathcal{F}_{\mathsf{ABB}}^2$-hybrid model is again specified by the interactive Turing machines $M_1, \dots, M_n$ working for the parties $P_1, \dots, P_n$. In principle, these machines also work quite similarly to the previous case, extracting novel commands (to create pairs of random non-zero elements and their inverses) from a set of

all the commands received during the current round, processing these commands separately, and forwarding all other commands simply to $\mathcal{F}_{ABB}^2$, all the while translating the handles. This time, however, the machines M_i maintain a persistent state, the structure of which is very simple — the machines count the number of rounds they have been running.

As stated, when a machine M_i receives the commands for the current round c, it filters out the commands $(\mathsf{compute}, x, x', \mathsf{randinv})$ requesting the generation of a random non-zero element of $\mathbb{F}$ and its inverse, and letting the handles x and x' point to them. Let N be the number of these commands in the c-th round. The machine M_i computes a number N', so that out of N' pairs of uniformly randomly generated elements of $\mathbb{F}$, the probability of getting at least N pairs with non-zero components is at least q_c. Here q_c is a round-specific success probability, which is a part of the description of M_i. Later on, we will explain how the probabilities q_c are chosen.

The requests to generate N_c pairs of random invertible elements together with their inverses, pointed to by handles $x_1, x_1', \ldots, x_N, x_N'$, are handled by the machines M_i as follows. They query $\mathcal{F}_{ABB}^2$ for $2N'$ random elements of $\mathbb{F}$, letting the handles $y_{1,c}, y_{1,c}', \ldots, y_{N',c}, y_{N',c}'$ point to them. These handles are in a form that cannot occur in commands received from $P_1, \ldots, P_n$. If necessary, the handles may be translated analogously to the implementation of $\mathcal{F}_{ABB}^2$ above. The machines M_i then ask $\mathcal{F}_{ABB}^2$ to compute $z_{j,c} = y_{j,c} \cdot y_{j,c}'$ for each $j \in \{1, \ldots, N'\}$ and declassify the results, obtaining the values $v_{1,c}, \ldots, v_{N',c}$. According to the choice of N', at least N of those values are non-zero with the probability q_c. Without a loss of generality, let $v_{1,c}, \ldots, v_{N,c}$ be non-zero. The machine M_i asks $\mathcal{F}_{ABB}^2$ to compute $x_j = 1 \cdot y_{j,c}$ and $x_j' = v_{j,c}^{-1} \cdot y_{j,c}'$, for $j \in \{1, \ldots, N\}$. Clearly, the values pointed to by x_j and x_j' are inverses of each other.

With the probability at most $1 - q_c$, there are fewer than N non-zero values among $v_{1,c}, \ldots, v_{N',c}$. In this case M_i shuts down.

After handling the requests to generate random invertible elements and their inverses, the machine M_i handles all other commands similarly to the implementation of $\mathcal{F}_{ABB}^2$ by the protocol Π_2.

We want the implementation Π_3 to be statistically at least as secure as $\mathcal{F}_{ABB}^3$, for some statistical security parameter σ. We thus choose the probabilities q_c so that $\sum_{c=1}^{\infty}(1 - q_c) \leq 2^{-\sigma}$.

To demonstrate the security of Π_3, we again have to present a simulator *Sim* and a probabilistic bisimulation between Π_3 and $\mathcal{F}_{ABB}^3 \| Sim$. Again, *Sim* converts the execution plans of $\mathcal{F}_{ABB}^3$ that have been sent to the adversary into execution plans of $\mathcal{F}_{ABB}^2$. In particular, the commands $(\mathsf{compute}, x, x', \mathsf{randinv})$ are translated into commands to generate random elements of $\mathbb{F}$, to multiply them, and to the revealings of random values, generated by multiplying two uniformly randomly generated elements of $\mathbb{F}$. We can now argue that if the conditional on the revealed value $w = vv'$ is non-zero for uniformly randomly generated v and v', the value v is independent from w. Hence the simulator *Sim* does not need any information about the random invertible values generated by $\mathcal{F}_{ABB}^3$ in order to construct a valid simulation.

The simulation fails only if the machines M_i in the protocol Π_3 shut down. As we have argued before, this can only happen with the probability at most $2^{-\sigma}$.

5.3. Implementing $\mathcal{F}_{\mathsf{ABB}}^4$

The machines $M_1, \ldots, M_n$ of the protocol Π_4 securely implementing $\mathcal{F}_{\mathsf{ABB}}^4$ in the $\mathcal{F}_{\mathsf{ABB}}^3$-hybrid model work similarly to the machines of Π_3 and Π_2. The novel commands $(\mathsf{compute}, y_1, \ldots, y_k, \mathsf{prefmult}, x_1, \ldots, x_k)$ are handled by the machines M_i of the computing parties P_i as follows.

The queries $(\mathsf{compute}, z_0, z_0', \mathsf{randinv}), \ldots, (\mathsf{compute}, z_k, z_k', \mathsf{randinv})$ are made by the machines M_i to $\mathcal{F}_{\mathsf{ABB}}^3$. They will then issue multiplication commands to $\mathcal{F}_{\mathsf{ABB}}^3$ in order to obtain handles $t_1, \ldots, t_k$, computed as $t_j = z_{j-1}' \cdot x_j \cdot z_j$. Machines M_i then ask the handles $t_1, \ldots, t_k$ to be declassified, obtaining the values $v_1, \ldots, v_k \in \mathbb{F}$. Finally, machines M_i ask $\mathcal{F}_{\mathsf{ABB}}^3$ to compute $y_j = (v_1 \cdots v_j) \cdot z_0 \cdot z_j'$. Clearly, $y_j = x_1 \cdots x_j$.

The simulator *Sim* handles the prefix multiplications as follows. It translates the command $(\mathsf{compute}, y_1, \ldots, y_k, \mathsf{prefmult}, x_1, \ldots, x_k)$ into the commands described in the previous section. The values $v_1, \ldots, v_k$ are simulated as uniformly randomly chosen non-zero elements of $\mathbb{F}$. This is a valid simulation as long as the values to which $x_1, \ldots, x_k$ point are non-zero — the multiplication with values to which $z_0, \ldots, z_k$ point serves to completely randomize $v_1, \ldots, v_k$. If some of the values that $x_1, \ldots, x_k$ point to are zero then this simulation is no longer valid. But in this case, we have specified that $\mathcal{F}_{\mathsf{ABB}}^4$ becomes corrupt, handing its entire internal state over to *Sim*, which now has sufficient information to perform a valid simulation. Hence Π_4 is perfectly at least as secure as $\mathcal{F}_{\mathsf{ABB}}^4$ in the $\mathcal{F}_{\mathsf{ABB}}^3$-hybrid model.

6. Programming the ABB

The arithmetic black box is a convenient abstraction of SMC and highly useful in the realization of large privacy-preserving applications. These applications perform certain operations with public values and use the ABB for all privacy-sensitive values. The application can hence be specified as a program in a language that allows the programmer to specify whether a value is public or private. Such languages have been proposed in [25], their type systems being inspired by the type systems for secure information flow [26].

In several of the following chapters, we are going to use a similar, though less rigorously specified (imperative) language for presenting privacy-preserving protocols for complex tasks, making use of the ideal functionality $\mathcal{F}_{\mathsf{ABB}}$. In our write-up, x represents a variable with public value. In the implementation of the protocol in $\mathcal{F}_{\mathsf{ABB}}$-hybrid model, realized by machines $M_1, \ldots, M_n$ for the parties $P_1, \ldots, P_n$ of the protocol, as well as the component $\mathcal{F}_{\mathsf{ABB}}$, the value of such variable x is stored at each of the computing machines M_i. On the other hand, the write-up $[\![y]\!]$ denotes a variable with a private value. In the implementation, the computing machines M_i store a handle for such a variable, and this handle is associated with a value inside $\mathcal{F}_{\mathsf{ABB}}$. Only public values can be used to control the flow of the program.

One can build expressions containing the variables to be executed by the machines M_i. The actual execution depends on whether the variables are public or private. For example, the write-up $x \cdot y$ denotes that each machine M_i multiplies the values of the variables x and y. On the other hand, $x \cdot [\![y]\!]$ means that the machines M_i ask $\mathcal{F}_{\mathsf{ABB}}$ to multiply the value pointed to by $[\![y]\!]$ with the public value of x, and store it inside $\mathcal{F}_{\mathsf{ABB}}$ under a new handle, which is also provided by the machines M_i according to some public

mechanism. The write-up $[\![x]\!] \cdot [\![y]\!]$ means that the machines M_i ask $\mathcal{F}_{\mathsf{ABB}}$ to multiply the values pointed to by $[\![x]\!]$ and $[\![y]\!]$ and store the result under a new handle. In general, the actual commands sent to $\mathcal{F}_{\mathsf{ABB}}$ depend on the types of the operands.

The programs may contain commands to declassify some $[\![x]\!]$. In this case, all computing machines M_i ask $\mathcal{F}_{\mathsf{ABB}}$ to declassify the handle $[\![x]\!]$ to all parties. Also, the program may require one of the computing parties to store some value in $\mathcal{F}_{\mathsf{ABB}}$. This is translated into the store-commands of the ABB.

If there are no declassification or storing commands in the program, then the security of the protocol is immediate, even in the presence of malicious adversaries corrupting any number of parties. In other words, the protocol Π realized by $M_1, \dots, M_n$ and $\mathcal{F}_{\mathsf{ABB}}$ is at least as secure as some ideal functionality $\mathcal{F}_{\mathsf{App}}$ for the application that we intended to implement. Indeed, in this case the simulator *Sim* converts the invocations of the commands to $\mathcal{F}_{\mathsf{App}}$ into commands to $\mathcal{F}_{\mathsf{ABB}}$ and forwards them to the adversary. This conversion is made according to the programs that realize the commands of $\mathcal{F}_{\mathsf{App}}$. If there are declassification commands, then we must also argue how *Sim* simulates the declassified values. If there are commands to store values, and we aim for security against malicious adversaries, then we must show that the subsequent computation can verify that the stored values were correct. Note that the use of a public value known by all computing parties does not require a store-command, as $\mathcal{F}_{\mathsf{ABB}}$ accepts commands where some of the operands are constant.

We see that specifying the protocol Π for the functionality $\mathcal{F}_{\mathsf{App}}$ in this manner gives us very strong security guarantees. These guarantees only hold in the $\mathcal{F}_{\mathsf{ABB}}$-hybrid model, though. Thus, in practice, the implementation of $\mathcal{F}_{\mathsf{App}}$ tolerates the same kinds of adversaries as the implementation of $\mathcal{F}_{\mathsf{ABB}}$ we are going to use.

References

[1] Oded Goldreich. *Foundations of Cryptography: Volume 2, Basic Applications*. Cambridge University Press, New York, NY, USA, 2004.

[2] Tomas Toft. *Primitives and Applications for Multi-party Computation*. PhD thesis, University of Aarhus, Denmark, BRICS, Department of Computer Science, 2007.

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

[4] Oded Goldreich. *Foundations of Cryptography: Basic Tools*, volume 1. Cambridge University Press, New York, NY, USA, 2000.

[5] 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.

[6] 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.

[7] 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.

[8] Yihua Zhang, Aaron Steele, and Marina Blanton. PICCO: a general-purpose compiler for private distributed computation. In Ahmad-Reza Sadeghi, Virgil D. Gligor, and Moti Yung, editors, *2013 ACM SIGSAC Conference on Computer and Communications Security, CCS'13, Berlin, Germany, November 4-8, 2013*, pages 813–826. ACM, 2013.

[9] Torben P. Pedersen. Non-interactive and information-theoretic secure verifiable secret sharing. In Joan Feigenbaum, editor, *CRYPTO*, volume 576 of *Lecture Notes in Computer Science*, pages 129–140. Springer, 1991.

[10] 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.

[11] 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.

[12] Liina Kamm and Jan Willemson. Secure floating point arithmetic and private satellite collision analysis. *International Journal of Information Security*, pages 1–18, 2014.

[13] Toomas Krips and Jan Willemson. Hybrid model of fixed and floating point numbers in secure multiparty computations. In Sherman S. M. Chow, Jan Camenisch, Lucas Chi Kwong Hui, and Siu-Ming Yiu, editors, *Information Security - 17th International Conference, ISC 2014, Hong Kong, China, October 12-14, 2014. Proceedings*, volume 8783 of *Lecture Notes in Computer Science*, pages 179–197. Springer, 2014.

[14] Ronald Cramer, Ivan Damgård, and Jesper Buus Nielsen. Multiparty computation from threshold homomorphic encryption. In Birgit Pfitzmann, editor, *EUROCRYPT*, volume 2045 of *Lecture Notes in Computer Science*, pages 280–299. Springer, 2001.

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

[16] Victor Shoup. Practical threshold signatures. In Bart Preneel, editor, *Advances in Cryptology - EUROCRYPT 2000, International Conference on the Theory and Application of Cryptographic Techniques, Bruges, Belgium, May 14-18, 2000, Proceeding*, volume 1807 of *Lecture Notes in Computer Science*, pages 207–220. Springer, 2000.

[17] Dan Boneh and Matthew K. Franklin. Efficient generation of shared RSA keys. *J. ACM*, 48(4):702–722, 2001.

[18] 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.

[19] Chris Peikert, Vinod Vaikuntanathan, and Brent Waters. A framework for efficient and composable oblivious transfer. In David Wagner, editor, *Advances in Cryptology - CRYPTO 2008, 28th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 17-21, 2008. Proceedings*, volume 5157 of *Lecture Notes in Computer Science*, pages 554–571. Springer, 2008.

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

[21] Lior Malka. Vmcrypt: modular software architecture for scalable secure computation. In Yan Chen, George Danezis, and Vitaly Shmatikov, editors, *Proceedings of the 18th ACM Conference on Computer and Communications Security, CCS 2011, Chicago, Illinois, USA, October 17-21, 2011*, pages 715–724. ACM, 2011.

[22] Seny Kamara, Payman Mohassel, and Ben Riva. Salus: a system for server-aided secure function evaluation. 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 797–808. ACM, 2012.

[23] 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.

[24] Rob J. van Glabbeek, Scott A. Smolka, and Bernhard Steffen. Reactive, generative and stratified models of probabilistic processes. *Inf. Comput.*, 121(1):59–80, 1995.

[25] Dan Bogdanov, Peeter Laud, and Jaak Randmets. Domain-polymorphic programming of privacy-preserving applications. In Alejandro Russo and Omer Tripp, editors, *Proceedings of the Ninth Workshop on Programming Languages and Analysis for Security, PLAS@ECOOP 2014, Uppsala, Sweden, July 29, 2014*, page 53. ACM, 2014.

[26] Andrew C. Myers. Jflow: Practical mostly-static information flow control. In Andrew W. Appel and Alex Aiken, editors, *POPL '99, Proceedings of the 26th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, San Antonio, TX, USA, January 20-22, 1999*, pages 228–241. ACM, 1999.

Chapter 3
Social Need for Secure Multiparty Computation

Laur KANGER [a] and Pille PRUULMANN-VENGERFELDT [a]

[a] *University of Tartu, Estonia*

Abstract. The aim of this chapter is to introduce and discuss the potential socio-technical barriers that might be hindering the adoption of Secure Multiparty Computation (SMC) techniques. By investigating the conditions of adoption of technology under development, we are able to find different solutions that support the technology adoption process. We set out to interview the potential future users of SMC technologies to investigate the most likely adopters and to find the best directions for future developments of SMC. SMC is compared with the existing practices of data handling to demonstrate its advantages as well as disadvantages. In the current development phase, the focus of SMC advances needs to be on the usefulness and on finding an appropriate niche for the technology.

1. Introduction

A popular view of the relation between technological possibilities and social needs is one of two partially overlapping circles. On the one hand, not all technological opportunities are deemed socially desirable and thus realizable [Perez, 2010, p. 186]. On the other hand, not all needs can be fulfilled with existing technologies. The overlap between those two sections is considered to be the ideal space for new technologies.

The weakness of this view is that it tends to treat both technological opportunities and social needs as largely fixed things 'out there'. While this works well enough for established technologies — after all, the need for faster information processing has driven the evolution of computers for decades — it is much more problematic for emerging technologies such as secure multiparty computation (SMC). For in this case certainty in both dimensions is lacking. First, from the developers side the full application potential of the technology is unknown: the space of the things that are technologically possible has simply not been explored much. Second, there are no clearly articulated user preferences that could guide technological development. Third, the users know very little about the technology to be able to form any concrete use cases or understanding of the potential of the technology.

In these conditions there is a curious stalemate. On the one hand, technology developers are unsure of what exactly to develop given that not only the wishes of the prospective users but the identity of those potential users is unknown. On the other hand, the prospective users are unable to articulate their preferences because they are insufficiently informed about new technological capabilities or because there may be no prototypes

to try out in the first place. The question then becomes about breaking this vicious circle. In other words, how can we facilitate technology adoption in the conditions of high uncertainty both in technological and market-related issues?

This chapter provides an overview of how we tackled this issue in the case of SMC. We begin with a brief description of our research methodology. SMC is then compared with the existing practices of data handling to demonstrate its advantages as well as disadvantages. We then proceed to identify the main barriers to the adoption of SMC identified over the course of our research. The presentation of findings is situated in a wider theoretical context illustrating that in many ways the challenges met by SMC are ones characteristic of emerging technologies in general. The final section summarizes the practical implications of the findings.

2. Research Methodology

Data on potential users of SMC was collected in two rounds. The first round consisted of 25 semi-structured interviews with people from various countries and various fields of expertise. Interviewees were introduced to the capability models which included short definitions of terms connected to SMC and 12 illustrated examples of use cases worked out by the team. Before the interview, the users were asked to read the capability models and to draw attention to cases with which they can connect their field of expertise. Overall there were three aims: 1) to detect the potential application contexts of SMC (prospective users and uses); 2) to identify the preconditions for SMC adoption; 3) to identify potential barriers to SMC adoption.

Results from the first round provided input for the second round of interviews with five more experts. As many interviewees had expressed interest in the statistical applications of SMC techniques, a demo was prepared for assessment. The interviewees were also asked to assess the viability of a number of barriers compiled on the basis of the first round of interviews and the results of the FIRE project (focused on the factors affecting the adoption of privacy and security related technologies, see Kanger 2013 for more details). Furthermore, the research team had prepared a few more use case scenarios. Thus, the second round of interviews had four goals: 1) to test the typology of barriers constructed by the researchers; 2) to gather user feedback on the statistics demo; 3) to probe the feasibility of new use case scenarios; 4) to map additional use contexts along with the identification of preconditions of and barriers to adoption.

3. SMC and Existing Practices of Data Handling

It goes without saying that in order for the technology to be accepted, it has to improve on existing practices in certain aspects. If the new technology were better than the existing solutions in every respect the adoption would be a relatively straightforward matter. However, more often than not the adoption of new technologies disrupts existing practices, old habits and organizational routines — meaning that from the viewpoint of a potential adopter the new technology may be characterized by certain disadvantages as well. This often makes the adoption process a more contested and complicated issue than it may seem at first [Tushman and Anderson, 1986]. Through our interviews, we

established a list of existing data handling and data analysis practices by comparing them to the possibilities offered by SMC (see Table 1). We summarized the existing practices through keywords and attempted to identify their pros and cons from the interviews. In the comparative table we also added an assessment of the potential impact of SMC on existing practices, which was conducted by the team developing the technology.

Table 1.: Comparison of the pros and cons of existing practices with an evaluation of the potential of SMC to influence these practices.

EXISTING PRACTICES	PROS	CONS	POTENTIAL OF SMC TO INFLUENCE EXISTING PRACTICES
Data analysis			
Anonymization prior to giving data to a researcher	The secret is kept through professional ethical standards	• Very dependent on individual ethical integrity • Can involve extra work and be time-consuming • Individuals may still be identified through auxiliary data	+ No prior anonymization needed − Introduces computational overhead
Anonymization by research team	The secret is kept through professional ethical standards	• Very dependent on individual ethical integrity	+ No prior anonymization needed − Introduces computational overhead
Data is filtered as well as anonymized	Very unique cases are excluded from the analysis	• Can limit the type of analysis • Can exclude too many cases in an attempt to avoid back-tracing the individual	All the previous + No filtration needed
Keeping the data and the analyst separate	The secret is kept through professional ethical standards	• Very dependent on individual ethical integrity • Can involve extra work and be time-consuming • Can be costly in terms of resources • Needs authorized personnel able to devote enough time	+ The responsiveness of SMC is higher than that provided by authorized personnel

EXISTING PRACTICES	PROS	CONS	POTENTIAL OF SMC TO INFLUENCE EXISTING PRACTICES
Data operations are performed by a third party	The secret is kept through professional ethical standards	• Very dependent on individual ethical integrity • Can involve extra work and be time-consuming • Can be costly in terms of the resources required • Needs authorized personnel able to devote enough time	+ The responsiveness of SMC is higher than that provided by authorized personnel
Limiting research problem	Access to data dictates the scope of potential research	• When access to data dictates the scope and potential of the research, the research might be limited, potentially limiting some breakthrough discoveries or data-based policies that may require a different approach to data	+ All data can be made available − Individual values cannot be accessed (raw data cannot be seen)
Data sharing			
A secure pathway to data, full disclosure	While access to data is kept secure, the secret is kept through professional ethical standards	• Very dependent on individual ethical integrity • Can involve extra work for those authorizing access • Can be costly in terms of resources (providing special working space or maintaining secure channels)	+ Not dependent on individual ethical integrity to such a large extent − Individual values cannot be accessed (raw data cannot be seen)
Data sharing can be made obligatory by law	In legislative terms, everyone has to share their data whatever the privacy concerns	• Can reduce willingness to cooperate • The system can be slow or inefficient	+ As individual values cannot be seen, there is no incentive to withhold information

Below is a short discussion of each strategy in relation to the possibilities of SMC.

Anonymization.   This can be done prior to giving data over to the researcher or by the research team itself. In case of anonymization the contracts entered into to keep the privacy of the participants and the researchers' code of ethics have been the key to privacy preservation. A member of the research team is assigned the task to anonymize the data. The other option is that anonymous data is given to the researcher. This option often limits the kind of research that can be done, as in many cases the data is heavily filtered before it is handed over to the researchers. This means that sometimes many cases are eliminated from the analysis that could have been retained with SMC. The strength of this kind of approach is that the researcher has full access to the data given to them and can thus see the data and correct any errors (or delete erroneous cases). The critical part is heavy reliance on the trustworthiness of individual researchers.

Keeping the data and the analyst separate.   This practice has meant that only authorized personnel can conduct operations on the data. Thus, privacy is protected by contracts and research ethics, but the number of people who have access to the data is even more limited. If the researchers have limited access to the database, they can compose queries either based on prior knowledge or on anonymized sample records from the database. Queries are then sent for processing and only the results are presented. This is in some sense very similar to SMC, but instead of the SMC algorithms, the statistician manually processes all such queries. This can be very time-consuming. In addition, sending queries, waiting for results and improving queries can be a frustrating experience. This process can be very expensive as well, since the authorized handlers of the data are limited in number. This limits the researcher's possibility to check the quality of the data, especially in order to find data entry mistakes as finding errors in data can be challenging with any kind of data and access possibilities. The system is heavily reliant on the trustworthiness of the individual researcher.

Third-party solution.   This is a version of the above, where a third party is trusted with the data and operations. Often such a third party can also collect data on behalf of the contractors and carry out the whole research process as a service. This can be time-consuming and costly, and the third party needs to be trustworthy and uninterested in leakages.

Limiting the research problem.   This is a version of privacy preservation where research questions are formulated to match the real-life data access, not necessarily the needs of research or policy-making. As the potential benefit, the researcher focuses on what can be done and what is accessible and is not distracted by all the potential options within the data. The drawback is that the researcher might miss out on the interesting potential of the collected data.

Only the pathway to the data is secure.   For instance, a VPN is used to access data or a designated computer is used for computations. In this case, data handling is not secure and trust is placed in the ethical integrity of the researcher and signed contracts. This requires a lot of effort from the data handler, privacy is lost, and trust in researchers and data handling personnel must be great.

Sharing data is made obligatory by law.   For instance, it is compulsory for competing private enterprises to share logistical resource related information with the state in case of

military-critical food resources in different locations. The benefit of this solution is that it is a long-standing tradition. The drawback can be the inefficiency of such a system and the reluctance of different parties to share full information. Often, severe delays will need to be built into such a system to make the data less sensitive, thus making the info less operative and also less relevant. An example of such a case is the collection of financial data by the statistical office, which can take six months or longer.

In the comparative table we also added the assessment of the potential impact of SMC on existing practices, which was conducted by the team developing the technology. To conclude the discussion of existing data handling practices, the main drawbacks can be seen in not enabling access to individual records and introducing a computational overhead. Perhaps more implicitly, however, the table also draws attention to the compatibility of SMC with firmly established existing practices, e.g. making data sharing mandatory by law. A look at the preconditions of and barriers to the adoption of SMC applications allows getting more insight into which organizations potentially have greater incentive to experiment with SMC and which ones can be excluded from consideration.

4. Baseline Conditions for and Barriers to Adoption

We discuss the conditions that have to be met in order for the emerging technology to be adopted. Table 2 presents a list of factors singled out by the interviewees. As we can see, the determinants of technology adoption roughly fall into five analytical categories: information, need, task-technology fit, resources and organizational fit. In the following, theories of technology adoption are used to categorize each factor.

4.1. Information

Having a sufficient overview of the nature and possibilities of SMC forms a major baseline for adoption. The interviewees singled out two aspects as being particularly important. First, a clear understanding of how the use of technology translates into increased profits/savings in monetary terms, that is, a clear business case. Second, there has to be a convincing usage precedent. In principle, this would enable the potential adopters to avoid (or reduce) the cost of first-hand trial and error learning and the high risk of being the first adopters.

The interviewees also stressed the potential downsides of insufficient information. Naturally, if a business case seems to be inadequate or if it is not clearly communicated, the users see no reason to adopt the technology. However, the mere lack of use cases may also hinder adoption as the majority of potential users could adopt a collective wait-and-see attitude, thereby delaying further diffusion. According to one interviewee, confusion about the underlying working principle may also promote irrational fears about data leakage despite the fact that SMC is designed to avoid it. Concerning the specific example of the defense sector, it was also argued that developers from the private sector are often unaware of its actual needs and therefore unable to propose sufficiently attractive applications of new technologies.

We need to establish an alignment between perceived needs and proposed technological solutions. One way is to turn attention to the technical visionaries of each organization that is, people with technical and organizational know-how (including the standard

Table 2. Baseline conditions for and barriers to the adoption of SMC.

BASELINE CONDITION	IF THE BASELINE CONDITION FAILS
1. Being sufficiently informed	1. Insufficient information/communication
Perceived business case	Lack of business case
Presence of other users (possibility of learning from experience)	"Wait-and-see"
	Insufficient knowledge
	The working principle is not understood
	Fear of data leaks despite everything
	Insufficient knowledge about the client's actual needs
2. The presence of perceived need	2. A functional substitute already exists
Sufficient motivation to consider adoption	Research ethics
Decreasing the complexity of work processes	A trusted third party
Negotiations, organizing	Centralized collection of data guaranteed by law
Speeding up work processes	The perceived costs of inaction do not seem high enough
Monetary costs	
3. The presence of a task-technology fit	3. Technology fits ill for the task
The data must be sensitive enough	The demo does not offer enough attractive features
The need for data processing must be regular enough	May be unfeasible to use for large data
	Coordination costs become very high if there is the need to agree on more than 100 definitions
4. Sufficient resources	4. Lack of resources
The enterprise must be big enough to afford such data processing	Connectivity problems between various databases
There has to be a need to have enough time and resources to experiment with the technology in order to imagine possible uses in the first place	Not enough information on the process costs of enterprises
	Intelligence data may be difficult to formalize
5. Minimal need for change: a good organizational fit	5. Avoidance of uncertainty
Low entry barrier, the need for a standardized application	Lack of willingness to change existing work routines
	Desire to see the initial data
	Centralization of databases
	Desire to avoid the complexity and uncertainty involved in the adoption of new technology
Sufficient usability	Uncertainty regarding adoption costs
Compatibility with existing data analysis software	Implementation may demand too much organizational effort
	Implementation may be too expensive
	Low usability (high learning barrier)
Management support	

procedures of data collection, handling and analysis), interest in new technologies and a vision of how to apply new technologies to existing problems. These visionaries may be viewed as intermediaries between the target organization and the developer. Moreover, if SMC techniques spark their interest, they may start to act as intra-organizational promoters of adoption. People with such characteristics, however, are scarce and usually quite busy. Our team experienced this drawback during the second round of interviews: although one interviewee agreed to help to establish contact with a few specialists in the defense and security sector, eventually none of them replied the request for an interview.

The importance of learning processes between various stakeholders is well recognized by a framework called Strategic Niche Management (SNM) [Kemp et al., 1998, Schot and Geels, 2007, 2008]. SNM conceptualizes the evolution of nascent technologies as a self-reinforcing cycle of three activities: 1) articulation of visions and expectations about the potential of new technologies; 2) social learning, drawing general lessons from the experiments with specific technologies and disseminating them; 3) building networks of actors [Raven and Geels, 2010]. The results of a recent simulation [Lopolito et al., 2013] suggest that information campaigns play a crucial role in vision articulation, which, in turn, promotes further knowledge diffusion and resource mobilization.

SNM is a quasi-evolutionary approach [Schot and Geels, 2007] stressing that through the process of niche-building, a mutual alignment between a new technology and its environment is achieved along three dimensions: the technology itself, user preferences and regulatory structures. This draws attention to the active role of niche-builders: *"Innovators do not succeed ... because their innovations fortuitously fit into predefined sets of niche constraints; rather, they construct their own niches"* [Schot and Geels, 2007, p. 606]. Thus, not only the developers and prospective users but also the researchers themselves become the instruments of niche-building. A few examples: during the interviews the respondents were asked to imagine different application contexts for SMC, the project workshops brought together actors developing SMC techniques who did not know each other, and the discussions that followed identified promising applications to be explored. By enhancing network-building, articulation of expectations, and social learning, events such as these gradually contribute to niche-building and an emerging community of SMC stakeholders.

4.2. Need

A perceived need constitutes the second baseline for adoption. The motivation to consider adoption obviously plays a major role. More specifically, however, the interviewees also stressed that the adoption of SMC should translate into smaller coordination efforts, smaller monetary costs or a smoother work process – but preferably into the combination of all three. Consider the issue of input data quality: while in the case of centralized databases one party can clean all the data, the situation is different for SMC where access to and comparison of separate data inputs is excluded by definition. Therefore, the means to ensure the quality of input data need to be found. This, in turn, implies reaching certain agreements on how the data is to be cleaned by each party. Whether such coordination costs outweigh the ones of centralized data cleaning may not be clear from the outset.

Often the problems SMC can solve in principle may not seem that pressing at all and hence, the costs of inaction are not that apparent. As expressed by an interviewee (I4): *"One can always say that we did not need it before, but we do not need it now"*.

Hence, new possibilities offered by SMC are not considered and the existing organizational practices of problem-solving continue to be used.

It may also be that for a variety of actors, the existing functional substitutes work well enough. These substitutes can be as simple as research ethics and mutual trust. Often a trusted third party is used. In certain domains, the incumbent third party is so well established that potential rethinking of current practices is outright rejected. For example, an enterprise involved in market research declined the interview on the grounds that its data collection and analysis was already centralized and thus, there was no need for SMC. As indicated above, in some cases the situation may be further institutionalized by legal means – for example, interviewees from Statistics Estonia pointed out that they had legal rights to obtain data from various sources. In fact, SMC was even argued to be potentially harmful for the organization as – if the technology was more widely known and applied – this would give various parties a good pretext for not giving out the source data.

This example illustrates the point about niche-building, as discussed above: it is not only technology but its environment that needs to be actively shaped for the diffusion of technology to take off. In many cases new technologies do not have a good fit with the existing selection environment. Data collection by national statistical offices provides a good example. In principle it is possible to imagine a legal framework in which a lot of data processing can take place without the data being collected by one centralized actor. But as long as this practice continues to be institutionalized, incumbent actors see no need for change. There are no incentives to adopt new technologies that are perceived as effectively undermining the existing organizational skill base and competences.

Recent work has suggested that the breakthrough of emerging technologies is not only dependent on niche-internal factors. It is also shaped by existing large-scale socio-technical regimes (e.g. automobility, digital computing) and wider macro-level events such as financial crises or governmental reforms. A multi-level perspective on socio-technical transitions [Geels, 2005, Geels and Schot, 2010] suggests that instead of replacing these regimes, new niche technologies quite often begin with establishing symbiotic links with them. For example, steam engines were first employed to assist sailing ships in case there was little wind. It follows that the initial success of SMC techniques might be located within the framework of existing solutions, i.e. tailored to solving some specific problems in existing systems. But it also means seeking out actors that are not performing too well in the existing regimes and who would thus be more motivated to try alternative solutions.

While perceived usefulness has been repeatedly shown as the most crucial determinant of adoption [Venkatesh et al., 2003, 2012], the technology life cycle theory [Anderson and Tushman, 1990, Tushman and Rosenkopf, 1992, Tushman and Murmann, 2002] suggests that this issue may be particularly accentuated for emerging technologies. The technology life cycle theory proposes that until the emergence of a dominant design (such as IBM PC for microcomputers or Windows for operation systems), enterprises experiment with a variety of different designs and architectures. As the technology itself and user preferences are yet to stabilize, the basic functionality and potential applications remain unclear. In these conditions the very usefulness of the technology is contested and usability concerns, while present to some extent, play only a minor role. This conclusion was largely supported by our second round of interviews during which it appeared that the interviewees were much more interested in discussing the potential

applications of SMC rather than assessing the statistics demo. According to the technology life cycle theory, usability concerns start to take a more prominent role only after the appearance of a dominant design. This becomes especially relevant when the technology enters the phase of incremental change, as in these conditions, explicit and stabilized user preferences come to drive the development process.

4.3. Task-Technology Fit

An important aspect of technology adoption also concerns assessing whether the technology in its current form is suitable for the task at hand and subsequently, whether it is the technology or the task itself that requires further adjustments. In this regard the interviewees argued that the data must be sensitive enough and the need for such data processing regular in order to consider the adoption of SMC. It was also pointed out that in some instances SMC may fail to deliver on its promises. For example, one interviewee from Statistics Estonia was worried that, in the case of big data, SMC could be too slow and costly. A specialist from the domain of business process optimization, on the other hand, referred to high coordination costs between parties when the number of definitions to be agreed upon (e.g. the exact meaning of terms such as delivery date) exceeds more than a hundred items.

In some cases the technology only needs minor adjustments. This was the case for the statistics demo where the development team did not see the addition of new functionalities as particularly difficult. The cases of large data or business process optimization may present somewhat greater challenges, however. And in some cases other ways of problem-solving may be more effective than SMC. Once again, in the context of great technological and market uncertainty, these questions cannot be answered easily. This requires ongoing dialogue with prospective users as well as continuous technological experimentation.

4.4. Resources

Even when the prospective user is sufficiently informed, has a clearly perceived need for SMC and has defined the task in a manner for which the technology would be highly suitable, no adoption may still follow because of inadequate resources. For one, as the costs of applying SMC techniques are rather unclear at the moment, it may turn out that only large enterprises are positioned well enough to consider adoption. One interviewee argued that the public sector would need some time and resources to experiment with the technology in order to even start imagining novel use opportunities. Failing that, SMC could merely remain "an interesting and fun toy", the usefulness of which is acknowledged in rhetoric but not in practice.

But the cost of use is not the only possible barrier to adoption. Important obstacles may arise from the quality of input data itself. For example, intelligence data is often presented in a format that is difficult to quantify. Related to business process optimization, only mature industries (e.g. automobile, electronics) have a sufficient amount of information about various business process costs (e.g. the cost of buying from different suppliers), which other sectors may simply lack. In other words, in a number of cases the adoption of SMC seems to set excessively high requirements to the quality of input data.

The need for sufficient time and other resources for experimentation can be explained by the amount of cognitive work involved. The potential adoption of a new technology involves individual and organizational learning along a number of dimensions:

1. Familiarizing oneself with the working principles of SMC (understanding what it is capable of doing in general).
2. Locating some potential contexts of use for SMC, some real-life problems to be solved (connecting the abstract working principle to possible specific applications, connecting local problems to abstracted use cases).
3. Assessing the feasibility of using SMC for the identified problem, filtering out some possibilities (i.e. assessing the task-technology fit).
4. Assessing the resources necessary to employ SMC, including the absorptive capacity for technological change of the organization in question.
5. Trying out SMC pilots in practice, obtaining first-hand implementation experience, considering the need for further adjustments (possibly innovating during diffusion – see Fleck (1994) for the concepts of 'innofusion' and 'learning by trying').

The importance of social learning is well acknowledged in SNM literature [Kemp et al., 1998, Schot and Geels, 2008]. Compared to mature and well-established technologies characterized by crystallized practices, routines and institutional support, emerging technologies are better thought of as "hopeful monstrosities" [Mokyr, 1990] exhibiting low actual but high expected future performance. For example, one could think of the first cars compared to horses or the first mobile phones compared to landline phones. SNM proposes to solve this problem by creating niches to protect emergent technologies from direct market competition until the technologies in question have matured enough. Niche-building is expensive and time-consuming, however. For this reason, the public sector or public-private sector cooperation often provides the best starting point for technological experimentation. This is also reflected in our results. For example, one interviewee speculated that SMC could be used for assessing the frequency and diffusion of critical incidents, e.g. cyber-attacks on banks. Defense resource planning and pooling of intelligence data were also mentioned as possible domains of application.

Niche development, however, is also characterized by many fundamental dilemmas (see Schot and Geels 2008, p. 549, for an overview), one of which concerns network-building. Namely, the success of networking depends on being able to include actors with many resources and a high level of competence. However, such actors may often have vested interests in existing technologies and, as such, may neglect or even actively block the development of new ones. Outsider organizations are often better at working with new radical innovations; however, they are often low on both resources and competence. As there is neither dominant design nor standardized solutions for SMC, it means that its adoption cost is still relatively unclear. Thus, the development of SMC may become hindered by a situation where motivated developers lack resources, while resource-rich actors lack motivation to adopt and/or develop SMC techniques. It can be seen that considerations such as this may decisively narrow the potential range of application contexts for SMC, making the issue of identifying the promising contexts of technological experimentation most crucial indeed.

4.5. Organizational Fit

Finally, even a well-informed organization with ample resources and a perceived need for SMC may still abstain from adopting the technology because of poor organizational fit. Every organization has a set of routines, formal and informal work procedures, ways of using technology, rules and regulations, which new technologies are likely to disrupt. This can happen in many ways, e.g. the new technology needs to be installed, new skills and competencies have to be developed, new types of services and support may be needed, new supplier-user relations need to be established and so on. The extent to which new technologies can be seamlessly integrated with the existing organizational practices refers to the degree of organizational fit.

One way to achieve better organizational fit is to keep the entry barrier low. As indicated by one interviewee, this would mean a certain standardization of SMC applications so the prospective user would not have to spend too much time thinking about issues related to hardware, software and data preparation. Another argued that if SMC is used for statistics applications, it should have similar functionalities and roughly the same look and feel as the existing data analysis software. In different ways these examples point to the importance of uncertainty avoidance. It may be that the disruption of existing organizational practices contributes to technology adoption: as observed by one interviewee, in the military sector technological changes are often preceded by changes in management.

An evolutionary theory of economic change offers an explanation of and context to these observations. According to the evolutionary theory of economic change [Nelson and Winter, 1982], organizations try to maintain their routines. If problems occur, the search for solutions starts around initial organizational and technological competences. If the problem is not solved, the scope of the search gradually widens. This, in turn, means that for incumbent organizations that are doing well, emerging technologies should aim to offer a high degree of organizational fit. This might be less the case for organizations that have been incentivized to widen their scope of search for possible technological solutions.

As previously indicated, standardization is one traditional way in which adoption costs have been decreased. However, the technology cycle theory postulates that dominant designs do not emerge in regulated or low volume markets [Tushman and Murmann, 2002]. Therefore, from the collective learning perspective, it becomes crucial to find out whether it is possible to develop standardized configurations of SMC applications.

4.6. Practical Implications

The question posed in this chapter is how to deal with the problem of technological and market uncertainty typical of emerging technologies. The interview results placed in a theoretical context suggested a number of answers, which are summarized below.

- Which organizations not to target?
  * Considering the low level of maturity of SMC solutions at the moment, less attention should be paid to targeting incumbent organizations characterized by one or more of the following conditions:
    * a lack of particular internal or external pressure for change,

* ∗ the perception of SMC as potentially undermining the existing organizational competencies, resources and practices,
  * ∗ a need for the substantial rethinking of an existing business model and/or working practices in order to employ SMC.
- Which organizations to target?
  * ∗ Pilot applications of SMC should prioritize the public sector or cooperation between the private and public sector as the locus of experimentation until the technology has matured enough to enter the market.
  * ∗ A recent management change may be a facilitating organizational factor for attracting interest in SMC pilot applications as the change in leadership may also act as an incentive for altering existing organizational routines.
  * ∗ In addition to focusing on limited and well-definable problems, business process optimization solutions should target large enterprises in mature industries.
  * ∗ If an incumbent organization is targeted, then the degree of organizational fit should be relatively high. On the other hand, actors on the margins of the existing regime are prone to experimenting more and in such cases the level of organizational fit may be lower.
- How to approach the target organizations?
  * ∗ Simple means for explaining and demonstrating the basic working principles of SMC should be found to keep the initial learning barrier low.
  * ∗ More attention should be paid to the profitability calculation of SMC applications in order to reduce uncertainties related to adoption.
  * ∗ The communication of real-life applications as well as their results warrants special attention, as these are likely to signal to prospective users the potential benefits of adoption.
  * ∗ Attention should be paid to mapping the changes in the work process introduced by the adoption of SMC across various application contexts. The substitutions of certain types of activities for others, the efficiency gained in the overall work process, and possible trade-offs should be clearly outlined for the prospective users to reduce uncertainty about the changes in the work process required by the adoption of SMC applications.
  * ∗ Direct contacts should be established with the technical visionaries of target organizations. The latter may be viewed as entry points for further contacts.
- Which technological developments should be kept in focus?
  * ∗ The degree of tolerated organizational fit depends on the type of organization: well-performing regime incumbents are likely to be more sensitive to potential changes in organizational routines than those operating on the margins of the existing regime or the ones that are experiencing performance problems.
  * ∗ When it comes to applying SMC in the private sector, the first signaling applications should focus on smaller, well-definable problems in order to avoid the issue of accumulating coordination costs.
  * ∗ Future efforts should be devoted to probing the feasibility of developing standardized configurations of SMC applications, which, in turn, would enable keeping the entry barrier low, thus facilitating further adoption.

∗ In general, considering the level of maturity of SMC solutions, ensuring the usefulness of SMC should take precedence over usability concerns.

We believe that when the issues summarized above are duly considered, SMC will have great potential as a usable and efficient measure to ensure the security and privacy of data. The progress and success of real-life applications discussed in Chapter 12 is one good way of moving away from the high instability of SMC. The presentation and visibility of success cases can be seen as crucial for the realization of SMC's potential.

References

Carlota Perez. Technological revolutions and techno-economic paradigms. *Cambridge Journal of Economics*, 34(1):185–202, 2010.

Laur Kanger. D6.1 – Addressing societal concerns on legal and privacy issues in ICT-related projects. Public deliverable, The FIRE Project (call FP7-ICT-2011-8 of the ICT Work Program 2011/12). Available online at: `http://www.trustworthyictonfire.com/outcomes/public-deliverables?download=4:d6-1`, 2013.

Michael L Tushman and Philip Anderson. Technological discontinuities and organizational environments. *Administrative science quarterly*, pages 439–465, 1986.

René Kemp, Johan Schot, and Remco Hoogma. Regime shifts to sustainability through processes of niche formation: the approach of strategic niche management. *Technology Analysis & Strategic Management*, 10(2):175–198, 1998.

Johan Schot and Frank W Geels. Niches in evolutionary theories of technical change. *Journal of Evolutionary Economics*, 17(5):605–622, 2007.

Johan Schot and Frank W Geels. Strategic niche management and sustainable innovation journeys: theory, findings, research agenda, and policy. *Technology Analysis & Strategic Management*, 20(5):537–554, 2008.

RPJM Raven and FW Geels. Socio-cognitive evolution in niche development: comparative analysis of biogas development in denmark and the netherlands (1973–2004). *Technovation*, 30(2):87–99, 2010.

A Lopolito, Piergiuseppe Morone, and R Taylor. Emerging innovation niches: An agent based model. *Research Policy*, 42(6):1225–1238, 2013.

Frank W Geels. *Technological transitions and system innovations: a co-evolutionary and socio-technical analysis*. Edward Elgar Publishing, 2005.

F. W. Geels and J. Schot. The dynamics of socio-technical transitions: A socio-technical perspective. *Transitions to Sustainable Development: New Directions in the Study of Long Term Transformative Change*, pages 11–101, 2010.

Viswanath Venkatesh, Michael G Morris, Gordon B Davis, and Fred D Davis. User acceptance of information technology: Toward a unified view. *MIS quarterly*, pages 425–478, 2003.

Viswanath Venkatesh, James YL Thong, and Xin Xu. Consumer acceptance and use of information technology: extending the unified theory of acceptance and use of technology. *MIS quarterly*, 36(1):157–178, 2012.

Philip Anderson and Michael L Tushman. Technological discontinuities and dominant designs: A cyclical model of technological change. *Administrative science quarterly*, pages 604–633, 1990.

Michael L Tushman and Lori Rosenkopf. Organizational Determinants of Technological Change: Toward a Sociology of Technological Evolution. *Research in organizational behavior*, 14:312–347, 1992.

M Tushman and Johann Peter Murmann. Dominant designs, technology cycles, and organizational outcomes. *Managing in the modular age: architectures, networks, and organizations*, 2002.

Joel Mokyr. *The lever of riches: Technological creativity and economic progress*. Oxford University Press, 1990.

Richard R Nelson and Sidney G Winter. *An evolutionary theory of economic change*. Cambridge, MA & London: The Belknap Press of Harvard University Press, 1982.

Chapter 4
Statistical Analysis Methods Using Secure Multiparty Computation

Liina KAMM [a], Dan BOGDANOV [a], Alisa PANKOVA [a] and Riivo TALVISTE [a]

[a] *Cybernetica AS, Estonia*

Abstract. This chapter gives an overview of privacy-preserving versions of the analysis methods and algorithms that are most commonly used in statistical analysis. We discuss methods for data collection and sharing, and describe privacy-preserving database joins and sorting. From simple statistical measures, we discuss the count and sum of elements, quantiles, the five-number summary, frequency tables, mean, variance, covariance and standard deviation. We look into outlier detection and explore privacy-preserving versions of several different statistical tests, such as Student's t-test, Wilcoxon rank-sum and signed-rank tests and the χ^2-test. We discuss how to evaluate the significance of the test statistic in the privacy-preserving environment. We give several options for linear regression and conclude the chapter with a privacy-preserving method for data classification.

Keywords. Privacy-preserving statistical analysis, statistical testing, secure multiparty computation

Introduction

Privacy issues are especially topical with regard to personal data processing. Be it income, health, behavior or location data, all sensitive information must be handled with special care. Pseudonymization techniques are one way of dealing with the privacy issue, but other times data are pre-aggregate or noise is added by the data controllers before analysis. This chapter explores the possibility of performing privacy-preserving statistical analyses in a secure multiparty setting.

Privacy can be considered from two orthogonal aspects. First, how much side information is leaked in addition to the desired results. Second, what the desired results leak about individual data records. The former can be addressed using cryptographic methods, and the latter is described by output privacy.

In this chapter, we only look at cryptographic privacy. We consider a statistical study to be cryptographically private if the results reveal only the values that are desired as outputs. Our algorithms return values in a privacy-preserving format. Which of these values can be opened and, as a result, given to the analyst must be determined by the study plan. This requires a privacy risk assessment to evaluate the leakage and publication is decided on a case-by-case basis. This chapter does not offer an automatic solution to determining the values that can be made public.

Even though we only concentrate on cryptographic privacy, let us briefly discuss the meaning of output privacy. We consider a study output private if the results reveal nothing important about the input values [1]. Determining what is important and what is not is a difficult matter. Methods such as k-anonymity [2], l-diversity [3], t-closeness [4] and differential privacy [5] offer various levels of output privacy. Whereas differential privacy has been designed to provide output privacy directly, the other three contribute indirectly by adding noise to the inputs.

The algorithms described in this chapter can be implemented for any secure multiparty computation system, provided that the system has the following capabilities: classifying and declassifying values, integer, Boolean and floating-point arithmetic, comparison, shuffling, division and square root.

1. Data Collection and Sharing

When data is gathered in the secret shared database, metadata must also be added. While information like data types, attribute count and the amount of data is not hidden in some implementations of secret shared databases, it can be obfuscated to some extent. However, to use aggregation, we need to know the data types, as the corresponding protocols can depend on the data type. In the following, we assume that information about the data types and the descriptions of the ontologies, classifiers or domains is available to the analyst. This does not leak information about the inputs.

Sometimes single values are missing from a gathered dataset. One of the first challenges of privacy-preserving statistical analysis is to deal with missing data. In the standard setting, these values are either left empty or marked as not available (NA). While typical statistical analysis tools are built to manage the NA values efficiently, it is more problematic in the case of privacy-preserving tools as these values can only be distinguished from real values at the cost of privacy. Instead, we consider two options that do not leak this information. First, we can use a default value as the missing value (e.g. minimum integer or zero) and secret share it among parties similarly to other values. Due to the nature of secret sharing, this value will be indistinguishable from other values. However, before running any algorithms, we need to make a private comparison with this value and take the value into account so that it would not interfere with our algorithms. As an example, consider computing the mean value when the default values have not been filtered out. The problem is that the overhead of a comparison at the beginning of each algorithm execution is considerable and, therefore, we would like to avoid it.

The other solution is to put the weight on storage instead. We can use an extra attribute for each attribute to hold the information about missing values. This introduces a storage overhead of one shared bit of extra data per entry. Depending on the implementation of the database, the latter uses extra storage space of $n \cdot k \cdot x$ bits, where n is the number of entries, k is the number of attributes, and x is the smallest data unit that can be stored in the database.

In consideration of further computations, it is more useful to encode this extra information about data vector $[\![\mathbf{a}]\!]$ as a Boolean mask vector of available values $[\![a(\mathbf{a})]\!]$, where the element is 1 if the corresponding value in vector $[\![\mathbf{a}]\!]$ is available, and 0 otherwise.

For selection and filtering, our aim is similar: to hide which elements from $[\![\mathbf{a}]\!]$ correspond to the given filter value. Hence, we give the result of the selection as a pair of

vectors ($[\![\mathbf{a}]\!], [\![\mathbf{m}]\!]$), where $[\![\mathbf{a}]\!]$ is the original vector and $[\![\mathbf{m}]\!]$ is a mask vector with the value 1, where the element of $[\![\mathbf{a}]\!]$ corresponded to the filter value, and 0 otherwise. If the value vector $[\![\mathbf{a}]\!]$ already has filters associated with it or is connected to an availability vector, conjunction or disjunction is applied to the mask vectors. This ensures that the availability information is accounted for in later computations.

In most statistical algorithms, the provided filter can be used automatically during calculations. However, in some cases, it is necessary to cut the data vector $[\![\mathbf{a}]\!]$—keep a subset vector that contains only filtered data. The **cut** function first obliviously shuffles the value and mask vectors, retaining the correspondence of the elements. Next, the mask vector is declassified and the values for which the mask vector holds a zero are removed from the value vector $[\![\mathbf{a}]\!]$. The obtained vector containing only the elements that corresponded to a filter is returned to the user. This leaks the number of records that corresponded to a given filter. While we generally want to keep this information secret, it is sometimes necessary to reveal it due to study requirements. For instance, if a statistical analyst wanted to perform statistical tests on a subset of data, it would be important for him or her to know whether there were enough records left in the subset for the results to be meaningful.

2. Oblivious Database Join with Unique Keys

Let us have two secret-shared tables $[\![\mathbf{T_1}]\!]$ and $[\![\mathbf{T_2}]\!]$ with m_1 and m_2 rows and n_1 and n_2 columns, respectively. In the following, we will obliviously compute $J([\![\mathbf{T_1}]\!], [\![\mathbf{T_2}]\!], p)$ where p denotes the *key columns*. A naive oblivious database join operation on these two tables would first generate the full Cartesian product of the two tables (or their key columns) and then apply oblivious comparison to all possible key pairs. Rows with non-matching keys are then removed and the resulting table is obliviously shuffled to hide which rows stayed in the joined table. This solution is secure but requires $\Theta(m_1 m_2)$ comparison operations.

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

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

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

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

Data: Two secret shared tables $[\![\mathbf{T_1}]\!]$ and $[\![\mathbf{T_2}]\!]$ with key columns $[\![\mathbf{k_1}]\!]$ and $[\![\mathbf{k_2}]\!]$.

Result: Joined table $[\![\mathbf{T}^*]\!]$.

Parties obliviously shuffle each database table $[\![\mathbf{T_i}]\!]$ resulting in a new shuffled table $[\![\mathbf{T_i^*}]\!]$ with a key column $[\![\mathbf{k_i^*}]\!]$.

Parties choose a pseudorandom permutation π_s by generating random shares of the shared key $[\![s]\!]$.

Parties obliviously evaluate π_s on all shared key columns $[\![\mathbf{k_i^*}]\!]$.

Parties publish all values $\pi_s([\![\mathbf{k_{i\ j}^*}]\!])$ and use a standard database join to merge the tables based on columns $\pi_s([\![\mathbf{k_i^*}]\!])$. Let $[\![\mathbf{T}^*]\!]$ be the resulting table.

If there are some non-unique keys in some key column $\pi_s([\![\mathbf{k_i^*}]\!])$, parties should perform an additional oblivious shuffle on the secret-shared table $[\![\mathbf{T}^*]\!]$.

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

By combining the secure oblivious AES evaluation and the oblivious shuffle from [6], we get an efficient instantiation of the Alg. 1. For all database sizes, the resulting protocol does $\Theta(m_1 + m_2)$ share-computing operations and $\Theta(m_1 \log m_1 + m_2 \log m_2)$ public computation operations.

More details about privacy-preserving database join and oblivious database join can be found in [7].

3. Sorting

3.1. Principles of Privacy-Preserving Sorting

Privacy-preserving sorting is both a useful tool in statistical processing and an important primitive in other data analysis algorithms. We require that sorting be *oblivious* of the data. This means that the sorting algorithm must rearrange the input data into the desired order without being able to learn the values or even their relations.

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

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

Algorithm 2: Algorithm for sorting an array of integers.

    Data: Input array $\mathcal{D} \in \mathbb{Z}_n^k$ and a sorting network $\mathcal{N} \in \mathbb{L}^m$.
    Result: Sorted output array $\mathcal{D}' \in \mathbb{Z}_n^k$.
1 **foreach** $\mathbb{L}_i \in \mathcal{N}$ **do**
2   **foreach** $(x, y) \in \mathbb{L}_i$ **do**
3 $(\mathcal{D}_x, \mathcal{D}_y) \leftarrow \mathsf{CompEx}(\mathcal{D}_x, \mathcal{D}_y)$

A sorting network is basically an arrangement of CompEx functions so that if the comparators of all the layers of the sorting network are applied on the input data array, the output data array will be sorted according to the desired condition. For a more detailed explanation of sorting networks with examples, see [8]. For a more thorough comparison of privacy-preserving sorting algorithms and their benchmark results, see [9].

3.2. Implementing Sorting Networks Obliviously

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

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

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

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

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

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

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

Implementing a generator for sorting networks is a well-researched area. As a starting point, please see the classical work of Knuth [8].

While data-independent oblivious sorting can easily be implemented using sorting networks, oblivious Hoare's selection [10] is more complex, because the partitioning sub-procedure needs to publish random comparison results. This can be solved by running a shuffling procedure before the selection. As the elements of the resulting vector are in random order, the declassification of some comparison results leaks no information about the input values. As Hoare's selection algorithm has linear asymptotic complexity, while common sorting networks consist of $\Theta(n \log^2 n)$ comparison gates, selection is potentially faster[1] if we deal with large datasets. Practical results, however, indicate that in the privacy-preserving setting, using sorting networks is faster.

3.3. Sorting More Complex Data Structures

We often need to sort data that is in a format other than an array. For example, we may want to sort a table of values according to a certain key column. In this case, we need to redefine the CompEx operation to work on the full data structure.

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

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

$$\mathsf{CompEx}(\mathcal{A}, \mathcal{B}, c) = \begin{cases} (\mathcal{B}, \mathcal{A}), & \text{if } \mathcal{A}_c > \mathcal{B}_c \\ (\mathcal{A}, \mathcal{B}), & \text{otherwise.} \end{cases} \tag{1}$$

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

This algorithm has the following assumptions for oblivious implementation:

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

---

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

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

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

3.4. Optimization Using Parallel Operations

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

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

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

The same approach also works to parallelize sorting on matrices in Alg. 3. The CompEx function can again be performed as a single parallel comparison. The flipping of the comparison results, the multiplications and the additions can all be performed in parallel, allowing this oblivious function to be efficiently implemented as well.

4. Descriptive Statistics and Simple Statistical Measures

As discussed in Chapter 3, many statistical analysts emphasize the need to see the data before analyzing it in order to gain an understanding of the cohort and to find out which tests would be interesting to conduct. With privacy-preserving computations and

datasets, the aim is to keep the data secret so that no one would have access to individual values. Therefore, we cannot give an analyst direct access to individual data values.

However, when dealing with large datasets, the analyst can have trouble grasping the data by looking at the individual values anyway. They will perform simple measurements on the data and draw plots and graphs to get an overview of the characteristics. For this reason we claim that, given access to common aggregate values, there is no need to see the individual values to determine the nature of the data. Hence, we provide secure implementations of descriptive statistics and simple statistical measures.

4.1. Count and Sum

Based on the data representation described in Sec. 1, it follows that the count of values in a private vector $[\![\mathbf{a}]\!]$ is the sum of elements in the corresponding availability vector $[\![\mathrm{a}(\mathbf{a})]\!]$.

The sum of a vector of values is computed by first multiplying the elements with the corresponding values in the availability vector and then adding the results together.

4.2. Five-Number Summary and Frequency Tables

The five-number summary is a descriptive statistic that includes the minimum, lower quartile, median, upper quartile and maximum of a vector. This statistic can be quite revealing and can leak information as we are outputting individual values instead of an aggregated value. On the other hand, it is a good tool for drawing attention to outliers.

The five numbers that make up the summary statistics are quantiles and can be computed using the corresponding formula. Different formulae for this purpose can be viewed in [11]. Let p be the percentile we want to find and let $\mathbf{a}$ be a vector of values sorted in ascending order. Then we denote the chosen formula for computing a quantile by $\mathbf{Q}(p, \mathbf{a})$.

As an example, consider $\mathbf{Q_7}$ from [11]. This is the default quantile computation algorithm in the statistical analysis environment R:

$$\mathbf{Q}_7(p, \mathbf{a}) = (1 - \gamma) \cdot \mathbf{a}_j + \gamma \cdot \mathbf{a}_{j+1} \quad,$$

where $j = \lfloor (n-1)p \rfloor + 1$, n is the size of vector $\mathbf{a}$, 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. The privacy-preserving version of this algorithm is the same, only the elements of vector $[\![\mathbf{a}]\!]$ are in secret-shared format.

Alg. 4 from [12] describes how to find the five-number summary of a vector $[\![\mathbf{a}]\!]$ using the chosen quantile formula $\mathbf{Q}$. The algorithm uses the **cut** function to retain only the values that corresponded to a filter. As mentioned before, function **cut** leaks the count n of such elements. There is a possibility of keeping n secret using oblivious choice. However, this significantly slows down the algorithm. The oblivious method is discussed in [12].

The five-number summary can be graphically represented as a box-and-whiskers plot that gives a good overview of the data. If the data is separated into different classes (defined by mask vectors), the box plots based on the summaries calculated from these classes can be used to gain a visual idea of what the data distribution looks like in each

---

Algorithm 4: Five-number summary of a vector.

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

1 $[\![\mathbf{x}]\!] \leftarrow \mathbf{cut}([\![\mathbf{a}]\!], [\![\mathbf{m}]\!])$
2 $[\![\mathbf{b}]\!] \leftarrow \mathbf{sort}([\![\mathbf{x}]\!])$
3 $[\![min]\!] \leftarrow [\![\mathbf{b}_1]\!]$
4 $[\![max]\!] \leftarrow [\![\mathbf{b}_n]\!]$
5 $[\![lq]\!] \leftarrow \mathbf{Q}(0.25, [\![\mathbf{b}]\!])$
6 $[\![me]\!] \leftarrow \mathbf{Q}(0.5, [\![\mathbf{b}]\!])$
7 $[\![uq]\!] \leftarrow \mathbf{Q}(0.75, [\![\mathbf{b}]\!])$
8 **return** $([\![min]\!], [\![lq]\!], [\![me]\!], [\![uq]\!], [\![max]\!])$

Algorithm 5: Algorithm for finding the frequency table of a data vector.

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

1 $[\![\mathbf{x}]\!] \leftarrow \mathbf{cut}([\![\mathbf{a}]\!], [\![\mathbf{m}]\!])$
2 $n \leftarrow \mathsf{declassify}(\mathsf{Sum}([\![\mathbf{m}]\!]))$
3 $k \leftarrow \lceil \log_2(n) + 1 \rceil$
4 $[\![min]\!] \leftarrow \mathbf{min}([\![\mathbf{x}]\!]), [\![max]\!] \leftarrow \mathbf{max}([\![\mathbf{x}]\!])$
5 Compute breaks according to $[\![min]\!], [\![max]\!]$ and k, assign result to $[\![\mathbf{b}]\!]$
6 $[\![\mathbf{c}_1]\!] = (\mathbf{count}([\![\mathbf{x}_i]\!]) \mid [\![\mathbf{x}_i]\!] \leq [\![\mathbf{b}_2]\!], i = 1, \ldots, n)$
7 $[\![\mathbf{c}_j]\!] = (\mathbf{count}([\![\mathbf{x}_i]\!]) \mid [\![\mathbf{b}_j]\!] < [\![\mathbf{x}_i]\!] \leq [\![\mathbf{b}_{j+1}]\!], i = 1, \ldots, n), j = 2, \ldots, k$
8 **return** $([\![\mathbf{b}]\!], [\![\mathbf{c}]\!])$

---

class. This can be used, for example, in order to get an overview before carrying out Student's t-test.

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 visualized as a histogram. The algorithm returns the number of bins and the number of values in each bin. Alg. 5 computes a frequency table for a vector of values similarly to a public frequency calculation algorithm.

4.3. Mean, Variance, Standard Deviation and Covariance

Let $[\![n]\!]$ be the number of subjects in the cohort and let N be the size of the attribute vector $[\![\mathbf{x}]\!]$. We assume that the vector $[\![\mathbf{x}]\!]$ has been multiplied point-wise with the mask vector $[\![\mathbf{m}]\!]$ so that the elements that do not belong to the cohort are equal to 0. The most common measures for data are the arithmetic mean, variance and standard deviation:

- mean $[\![\bar{x}]\!] = \frac{1}{[\![n]\!]} \sum_{i=1}^{N} [\![\mathbf{x}_i]\!]$,

- variance $\llbracket \sigma^2 \rrbracket = \frac{1}{\llbracket n \rrbracket - 1} \sum_{i=1}^{N} (\llbracket \mathbf{x}_i \rrbracket - \llbracket \overline{x} \rrbracket)^2$, and
- standard deviation $\llbracket \sigma \rrbracket = \sqrt{\llbracket \sigma^2 \rrbracket}$.

These measures require the system to be able to handle private addition, multiplication, division and square root. If the size n of a chosen cohort is public, we can use division with a public divisor instead. This operation is faster and less complex than division with a private divisor. Depending on whether these values can be published, we can also use public square root instead of its private counterpart. As with all private computations, taking the square root of a private value is considerably slower than the public version of this protocol. If protocols for these operations exist, the implementation of these measures is straightforward in the privacy-preserving setting.

Trimmed mean is a version of mean where outliers have been eliminated. It can be computed by first removing a percent of the data from either end of the sorted data vector and then computing the mean. The trimming can be done using quantile computation.

Covariance is the measure of how two attributes change together. Covariance can be estimated using the following formula:

$$\mathbf{cov}(\llbracket \mathbf{x} \rrbracket, \llbracket \mathbf{y} \rrbracket) = \frac{1}{\llbracket n \rrbracket - 1} \left(\sum_{i=1}^{N} \llbracket \mathbf{x}_i \rrbracket \llbracket \mathbf{y}_i \rrbracket - \frac{1}{\llbracket n \rrbracket} \cdot \sum_{i=1}^{N} \llbracket \mathbf{x}_i \rrbracket \sum_{i=1}^{N} \llbracket \mathbf{y}_i \rrbracket \right) \ .$$

5. Outlier Detection

This section describes how to perform outlier detection in an oblivious manner. Once the outliers have been identified, they will not be shown to the data analyst. Hence, the outlier detection algorithm produces a Boolean mask vector indicating which data points are outliers.

There are many different outlier detection methods and choosing a specific one to use is highly dependent on the properties of the dataset. In this section, we will only consider univariate outlier detection mechanisms.

The simplest outlier detection method is to remove the minimal and maximal $n\%$ of the data values. This can be done using the privacy-preserving quantile computation, as 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 vector $\llbracket \mathbf{a} \rrbracket$. 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 $\llbracket \mathbf{a} \rrbracket$ to $\mathbf{Q}(0.05, \llbracket \mathbf{a} \rrbracket)$ and $\mathbf{Q}(0.95, \llbracket \mathbf{a} \rrbracket)$, and then multiplying the resulting mask vectors.

Another robust measure of detecting outliers that does not depend on the specific data scheme is to pick a property that captures the shape of the concrete dataset. Traditionally, sample mean and sample variance are used for this, but these properties are highly influenced by the existence of outliers. Thus, using them to detect outliers may yield suboptimal results. Instead, Hampel [13,14] suggests using median and median absolute deviation (MAD) as the properties to describe a given dataset and detect possible outliers. For a dataset X, its element x is considered an outlier if

$$\mathbf{median} - x > \lambda \cdot \mathrm{MAD}, \tag{2}$$

where

$$\mathsf{MAD} = \mathbf{median}_i(|X_i - \mathbf{median}_j(X_j)|)$$

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

As MPC protocols are often network-bound, we would like to have algorithms that are easily parallelizable so that we could use SIMD style operations. This algorithm is a good candidate as we do not have to compute everything one element at a time. For a given dataset we can compute the median and MAD once and use them as constants in Eq. 2 so classifying a given element becomes an evaluation of inequality with a constant. Hence, it is possible to classify all elements of a dataset at once.

6. Comparison of Two Populations

6.1. Filtering the Data

To compare two populations, we first need to distribute the data into two groups based on some condition. There are two slightly different ways of doing this. First, we can choose the subjects in one group and assume that the rest are in group two (e.g. people with high blood pressure and everyone else). Second, we can put subjects into both groups (e.g. men who are older than 65 and have high blood pressure, and men who are older than 65 but do not have high blood pressure). There is a clear difference between these selection categories and they yield either one or two mask vectors. In the former case, we calculate the second mask vector by flipping all the bits in the existing mask vector. These groups are often called case and control groups. The case group includes subjects with a certain trait and the control group includes those without the trait.

To calculate the means, variances and standard deviations, we first multiply pointwise the attribute vector *a* with the mask vector *m* so that the values that do not belong to the population would not interfere with the calculations. To count the elements, it suffices to sum the values of the mask vector.

6.2. Student's t-test

In the following, let $[\![\mathbf{a}]\!]$ be the vector we are testing and $[\![\mathbf{m_i}]\!]$ be the mask vector for population $i \in \{ca, co\}$ representing the case and control population, respectively. Let $[\![n_i]\!] = \mathsf{Sum}([\![\mathbf{m_i}]\!])$ be the count of subjects in population *i*.

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

$$[\![\sigma^2]\!] = \frac{([\![n_{ca}]\!] - 1)[\![\sigma_{ca}^2]\!] + ([\![n_{co}]\!] - 1)[\![\sigma_{co}^2]\!]}{[\![n_{ca}]\!] + [\![n_{co}]\!] - 2} .$$

The t-test statistic is

Algorithm 6: Wilcoxon rank-sum test.

Data: Value vector $[\![\mathbf{a}]\!]$ and corresponding mask vectors $[\![\mathbf{m_{ca}}]\!]$ and $[\![\mathbf{m_{co}}]\!]$.

Result: Test statistic $[\![w]\!]$.

1 $[\![\mathbf{m}]\!] \leftarrow [\![\mathbf{m_{ca}}]\!] + [\![\mathbf{m_{co}}]\!] - ([\![\mathbf{m_{ca}}]\!] \cdot [\![\mathbf{m_{co}}]\!])$

2 $([\![\mathbf{x}]\!], [\![\mathbf{m'_{ca}}]\!], [\![\mathbf{m'_{co}}]\!]) \leftarrow \mathbf{cut^*}(([\![\mathbf{a}]\!], [\![\mathbf{m_{ca}}]\!], [\![\mathbf{m_{co}}]\!]), [\![\mathbf{m}]\!])$

3 $([\![\mathbf{y}]\!], [\![\mathbf{b_{ca}}]\!], [\![\mathbf{b_{co}}]\!]) \leftarrow \mathbf{sort}([\![\mathbf{x}]\!], [\![\mathbf{m'_{ca}}]\!], [\![\mathbf{m'_{co}}]\!])$

4 $[\![\mathbf{r}]\!] \leftarrow \mathbf{rank}([\![\mathbf{y}]\!])$

5 $[\![\mathbf{r_{ca}}]\!] \leftarrow [\![\mathbf{r}]\!] \cdot [\![\mathbf{b_{ca}}]\!]$ and $[\![\mathbf{r_{co}}]\!] \leftarrow [\![\mathbf{r}]\!] \cdot [\![\mathbf{b_{co}}]\!]$

6 $[\![R_{ca}]\!] \leftarrow \mathbf{sum}([\![\mathbf{r_{ca}}]\!])$ and $[\![R_{co}]\!] \leftarrow \mathbf{sum}([\![\mathbf{r_{co}}]\!])$

7 $[\![n_{ca}]\!] \leftarrow \mathbf{sum}([\![\mathbf{m_{ca}}]\!])$ and $[\![n_{co}]\!] \leftarrow \mathbf{sum}([\![\mathbf{m_{co}}]\!])$

8 $[\![u_{ca}]\!] \leftarrow [\![R_{ca}]\!] - \frac{[\![n_{ca}]\!] \cdot ([\![n_{ca}]\!]+1)}{2}$ and $[\![u_{co}]\!] \leftarrow [\![n_{ca}]\!] \cdot [\![n_{co}]\!] - [\![u_{ca}]\!]$

9 **return** $[\![w]\!] \leftarrow \mathbf{min}([\![u_{ca}]\!], [\![u_{co}]\!])$

$$[\![t]\!] = \frac{[\![\overline{x_{ca}}]\!] - [\![\overline{x_{co}}]\!]}{[\![s]\!]} \sqrt{\frac{[\![n_{ca}]\!][\![n_{co}]\!]}{[\![n_{ca}]\!] + [\![n_{co}]\!]}} \quad .$$

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

$$[\![\sigma_{\overline{x_{ca}}-\overline{x_{co}}}]\!] = \sqrt{\frac{[\![\sigma^2_{ca}]\!]}{[\![n_{ca}]\!]} + \frac{[\![\sigma^2_{co}]\!]}{[\![n_{co}]\!]}} \quad ,$$

where $[\![\sigma^2_i]\!]$ is the unbiased variance of the population $i \in \{ca, co\}$. The t-test statistic is

$$[\![t]\!] = \frac{[\![\overline{x_{ca}}]\!] - [\![\overline{x_{co}}]\!]}{[\![\sigma_{\overline{x_{ca}}-\overline{x_{co}}}]\!]} \quad .$$

If the null hypothesis is supported, the t-test statistic follows Student's t-distribution with $([\![n_{ca}]\!] + [\![n_{co}]\!] - 2)$ degrees of freedom.

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

6.3. 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, for paired vectors, the corresponding Wilcoxon signed-rank test. The Wilcoxon rank-sum test [15] 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 generally follows the standard algorithm. Alg. 6 gives an overview of 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. 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 (by using element-wise disjunction). The function $\mathbf{cut^*}$ on line 2 differs from its previous usage in that several

---

Algorithm 7: Wilcoxon signed-rank test.

    Data: Paired value vectors $[\![\mathbf{a}]\!]$ and $[\![\mathbf{b}]\!]$ for n subjects, mask vector $[\![\mathbf{m}]\!]$.
    Result: Test statistic $[\![w]\!]$.
  1 $([\![\mathbf{x}]\!], [\![\mathbf{y}]\!]) \leftarrow \mathbf{cut}(([\![\mathbf{a}]\!], [\![\mathbf{b}]\!]), [\![\mathbf{m}]\!])$
  2 $[\![\mathbf{d}]\!] \leftarrow [\![\mathbf{x}]\!] - [\![\mathbf{y}]\!]$
  3 Let $[\![\mathbf{d}']\!]$ be the absolute values and $[\![\mathbf{s}]\!]$ be the signs of elements of $[\![\mathbf{d}]\!]$
  4 $([\![\mathbf{t}]\!], [\![\mathbf{u}]\!]) \leftarrow \mathbf{sort}([\![\mathbf{d}']\!], [\![\mathbf{s}]\!])$
  5 $[\![\mathbf{r}]\!] \leftarrow \mathbf{rank_0}([\![\mathbf{u}]\!])$
  6 **return** $[\![w]\!] \leftarrow \mathbf{sum}([\![\mathbf{u}]\!] \cdot [\![\mathbf{r}]\!])$

---

vectors are cut at once based on the combined filter $[\![\mathbf{m}]\!]$. Function **rank** gives ranks to the elements of the sorted vector. If two or more elements are equal, these elements receive the average rank of all the equal elements. From there on, the ranks are added together and the test statistic $[\![w]\!]$ is computed in the usual manner.

Similarly to Student's paired t-test, the Wilcoxon signed-rank test [16] is a paired difference test. Our version, given in Alg. 7, takes into account Pratt's correction [15] 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.

First, on line 3, both data vectors are cut based on the mask vector similarly to what was done in Alg. 6. The signs are then sorted based on the absolute values $[\![\mathbf{d}']\!]$ (line 4) and the ranking function **rank$_0$** is called. This ranking function differs from the function **rank** because we need to exclude the differences with the value 0. Let the number of 0 values in vector $[\![\mathbf{d}]\!]$ be $[\![k]\!]$. As $[\![\mathbf{d}]\!]$ 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 **rank$_0$** assigns $[\![\mathbf{r}_i]\!] \leftarrow 0$ while $[\![\mathbf{u}_i]\!] = 0$, and works similarly to **rank** on the rest of the vector $[\![\mathbf{u}]\!]$, with the difference that $i \in \{1, \ldots, n - [\![k]\!]\}$.

6.4. The χ^2-Test

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 analyze the frequencies of certain values in the dataset. The corresponding statistical test is known as the χ^2-test.

The standard χ^2-test statistic is computed as

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

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 $d_i = f_{2i}$, and then the frequencies can be presented as the contingency table given in Table 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

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

Table 1. Contingency table for the standard χ^2-test

	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

Algorithm 8: χ^2-test.

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

Result: The test statistic $[\![t]\!]$.

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 $[\![\mathbf{E}_{i,j}]\!] = \frac{[\![r_i]\!] \cdot [\![p_j]\!]}{[\![n]\!]}$

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

5 **return** $[\![t]\!]$

The privacy-preserving version uses the same formula but works on secret-shared data.

Alg. 8 shows how to compute the χ^2-test statistic based on a given contingency table. 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 optimized version of this algorithm that works where the number of options in our test is 2. Then the test statistic can be simplified and written as

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

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

6.5. Evaluating the Significance of the Statistic

In addition to implementing the privacy-preserving version of the statistical algorithm, we may make a second branching in our solution depending on whether or not the sizes of the two cohorts are public.

We introduce three new paradigms for statistical testing in the privacy-preserving setting. Fig. 1 is based on the work in [12]. In [17], we expand this by adding Option 3 to the privacy-preserving setting.

The testing process in the usual setting starts with computing the test statistic and the p-value and comparing them to the significance threshold set by the analyst. In the privacy-preserving setting, there are different concerns for data privacy that require the testing process to be modified to some extent. Let us look at the three options described
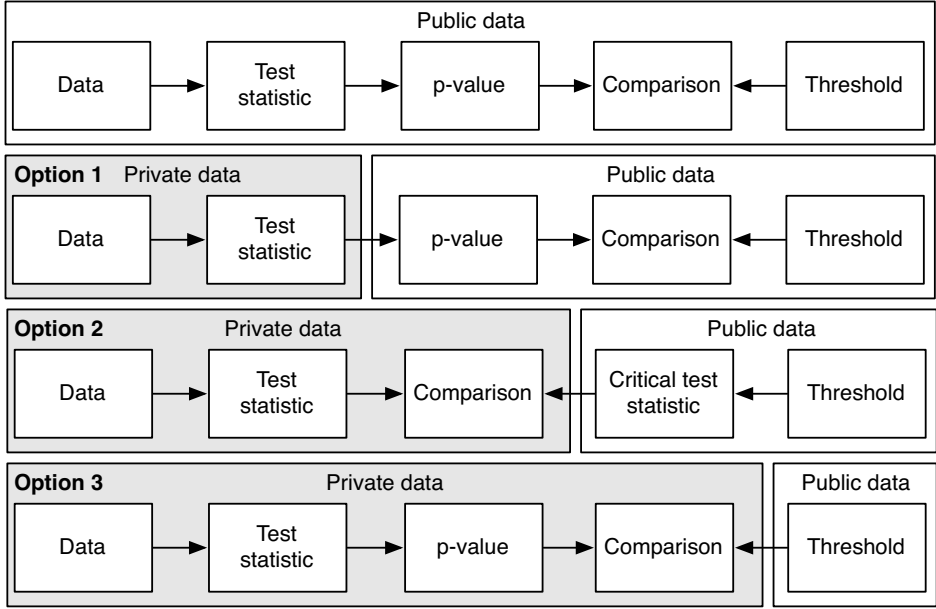
Figure 1. Different statistical testing paradigms in the public and private setting

on Fig. 1 that can be used in the privacy-preserving setting. Note that all these options offer a time-privacy trade-off.

Option 1 can be used when the test statistic and sample sizes can be declassified. The p-value can be computed in public, making statistical testing significantly faster. Option 2 offers better privacy by revealing only the sample sizes but not the computed test statistic. Using the sample sizes and the chosen significance threshold, the analyst can look up or calculate the critical test statistic based on the test being performed. The critical statistic will then be compared to the test statistic received during the testing process. This option is important, for example, in genome-wide association studies, where revealing the p-values can already leak whether or not a donor's sample is in the chosen dataset [18].

Option 3 offers the best privacy guarantee, as only the comparison result is revealed to the analyst. However, this reveals very little information and might not always be acceptable to statistical analysts who are used to seeing p-values as well. In addition, this option is the slowest as the p-value has to be computed in the privacy-preserving setting. The third option is also the closest to the standard setting with the exception that all operations are done in a privacy-preserving manner.

7. Linear Regression

Linear regression can be used to find out how strongly certain variables influence other variables in a dataset. This section is based on the work done in [12] that also contains security proofs of the algorithms.

Let us assume that we have k independent variable vectors of n elements, i.e. $\mathbf{X} = (\mathbf{X}_{j,i})$, where $i \in \{0, \ldots, k\}$ and $j \in \{1, \ldots, n\}$. The vector $\mathbf{X}_{j,0} = (1)$ is an added variable

for the constant term. Let $\mathbf{y} = (\mathbf{y}_j)$ be the vector of dependent variables. We want to estimate the unknown coefficients $\beta = (\beta_i)$ such that

$$\mathbf{y}_j = \beta_k \mathbf{X}_{j,k} + \ldots + \beta_1 \mathbf{X}_{j,1} + \beta_0 \mathbf{X}_{j,0} + \varepsilon_j$$

where the vector of errors $\varepsilon = (\varepsilon_j)$ is assumed to be white Gaussian noise. This list of equations can be compactly written as $\varepsilon = \mathbf{X}\beta - \mathbf{y}$. Most methods for linear regression try to minimize the square of residuals:

$$\|\varepsilon\|^2 = \|\mathbf{y} - \mathbf{X}\beta\|^2 \ . \tag{3}$$

This can be done directly or by first converting the minimization task into its equivalent characterization in terms of linear equations:

$$\mathbf{X}^T \mathbf{X} \beta = \mathbf{X}^T \mathbf{y} \ . \tag{4}$$

For simple linear regression, the aim of the analysis is to find $\hat{\alpha}$ and $\hat{\beta}$ that fit the approximation $\mathbf{y}_i \approx \beta_0 + \beta_1 \mathbf{x}_i$ best for all data points. The corresponding linear equation (4) can be solved directly:

$$\hat{\beta}_1 = \frac{\mathbf{cov}(\mathbf{x}, \mathbf{y})}{\sigma^2(\mathbf{x})}$$

$$\hat{\beta}_0 = \bar{\mathbf{y}} - \hat{\beta}_1 \cdot \bar{\mathbf{x}} \ .$$

As discussed, we can compute covariance and variance in the privacy-preserving setting. Hence, we can estimate the values of $\hat{\beta}_0$ and $\hat{\beta}_1$ directly.

We will look at different methods for solving the system (4) with more than one explanatory variable. First, for $k < 5$, we invert the matrix by computing determinants. For the general case, we give algorithms for the Gaussian elimination method [19] and LU decomposition [19]. We also describe the conjugate gradient method [20] that directly minimizes the square of residuals (3).

In all these algorithms, we assume that the data matrix has been multiplied already with its transpose: $\mathbf{A} = \mathbf{X}^T \mathbf{X}$ and the dependent variable has been multiplied with the transpose of the data matrix: $\mathbf{b} = \mathbf{X}^T \mathbf{y}$.

In the privacy-preserving setting, matrix inversion using determinants is straightforward, and using this method to solve a system of linear equations only requires the use of multiplication, addition and division. For the more general methods of solving systems of linear equations, we first give an algorithm that finds the first maximum element in a vector and also returns its location in the vector, used for finding the pivot element. While the Gaussian and LU decomposition algorithms can be used without pivoting, it is not advisable as the algorithms are numerically unstable in the presence of any roundoff errors [19].

Alg. 9 describes the function **maxLoc** that finds the pivot element and its location from a given vector. 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

---

Algorithm 9: maxLoc: Finding the first maximum element and its location in a vector in a privacy-preserving setting.

Data: A vector $[\![\mathbf{a}]\!]$ 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 $[\![b]\!] \leftarrow [\![\mathbf{a}_{\pi(1)}]\!]$ and $[\![l]\!] \leftarrow \pi(1)$
3 for $i = \pi(2)$ **to** $\pi(n)$ **do**
4 $\quad [\![c]\!] \leftarrow (|[\![\mathbf{a}_{\pi(i)}]\!]| > |[\![b]\!]|)$
5 $\quad [\![b]\!] \leftarrow [\![b]\!] - [\![c]\!] \cdot [\![b]\!] + [\![c]\!] \cdot [\![\mathbf{a}_{\pi(i)}]\!]$
6 $\quad [\![l]\!] \leftarrow [\![l]\!] - [\![c]\!] \cdot [\![l]\!] + [\![c]\!] \cdot \pi(i)$
7 return $([\![b]\!], [\![l]\!])$

---

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. 10 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 $[\![\mathbf{x}]\!]$, retaining the relations. On lines 5-8, 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, 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 - 15, elements on the pivot line are reduced. Similarly, on lines 19 - 19, elements below the pivot line are reduced. Finally, on lines 27 - 28, 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

---

Algorithm 10: Privacy-preserving Gaussian elimination with back substitution for a matrix equation $[\![\mathbf{A}]\!][\![\mathbf{x}]\!] = [\![\mathbf{b}]\!]$.

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

1   Let $[\![\mathbf{x}]\!]$ be a copy of $[\![\mathbf{b}]\!]$
2   Obliviously shuffle $[\![\mathbf{A}]\!], [\![\mathbf{x}]\!]$ retaining the dependencies
3   Let $[\![c]\!] \leftarrow$ **false** privately store the failure flag during execution
4   **for** $i = 1$ **to** $k - 1$ **do**
5   Let $[\![\mathbf{m}]\!]$ be a subvector of $[\![\mathbf{A}_{u,v}]\!]$ such that $u \in \{i, \ldots, k\}, v = i$
6      $([\![t]\!], [\![irow]\!]) \leftarrow$ **maxLoc**$([\![\mathbf{m}]\!])$
7      $irow \leftarrow$ declassify$([\![irow]\!]) + i$
8   **if** $irow \neq i$ **then**
9   **for** $j = 1$ **to** k **do**
10   Exchange elements $[\![\mathbf{A}_{irow,j}]\!]$ and $[\![\mathbf{A}_{i,j}]\!]$
11   Exchange element $[\![\mathbf{x}_{irow}]\!]$ and $[\![\mathbf{x}_i]\!]$
12      $[\![c]\!] \leftarrow [\![c]\!] \vee ([\![\mathbf{A}_{i,i}]\!] = 0)$
13      $[\![pivinv]\!] \leftarrow [\![\mathbf{A}_{i,i}]\!]^{-1}$
14      $[\![\mathbf{A}_{i,i}]\!] \leftarrow 1$
15   **foreach** $j \in \{1, \ldots, k\}$ **do**
16   **if** $j \neq i$ **then**
17              $[\![\mathbf{A}_{i,j}]\!] \leftarrow [\![\mathbf{A}_{i,j}]\!] \cdot [\![pivinv]\!]$
18          $[\![\mathbf{x}_i]\!] \leftarrow [\![\mathbf{x}_i]\!] \cdot [\![pivinv]\!]$
19   **for** $m = i + 1$ **to** k **do**
20   **for** $j = 1$ **to** k **do**
21   **if** $j \neq i$ **then**
22                  $[\![\mathbf{A}_{m,j}]\!] \leftarrow [\![\mathbf{A}_{m,j}]\!] - [\![\mathbf{A}_{i,j}]\!] \cdot [\![\mathbf{A}_{m,i}]\!]$
23          $[\![\mathbf{x}_m]\!] \leftarrow [\![\mathbf{x}_m]\!] - [\![\mathbf{x}_i]\!] \cdot [\![\mathbf{A}_{m,i}]\!]$
24          $[\![\mathbf{A}_{m,i}]\!] \leftarrow 0$
25   **if** $declassify([\![c]\!])$ **then**
26   **return** "Singular matrix"
27   $[\![\mathbf{x}_k]\!] \leftarrow \dfrac{[\![\mathbf{x}_k]\!]}{[\![\mathbf{A}_{k,k}]\!]}$
28   **for** $i = k - 1$ **downto** 1 **do**
29      $[\![\mathbf{x}_i]\!] \leftarrow [\![\mathbf{x}_i]\!] - \sum\limits_{j=i+2}^{k} [\![\mathbf{A}_{i,j}]\!] \cdot [\![\mathbf{x}_j]\!]$
30   **return** $[\![\mathbf{x}]\!]$

---

elimination method. Alg. 11 gives the privacy-preserving version of LU decomposition. Note that the elements on the diagonal of the lower triangular matrix **L** are equal to 1. Knowing this, **L** and **U** can be returned as one matrix so that the diagonal and the elements above it belong to the upper triangular matrix **U** and the elements below the diagonal belong to the lower triangular matrix **L** without losing any information.

Algorithm 11: 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 $\mathbf{q}$ containing the row permutations.

1   Let $[\![c]\!] \leftarrow 0$ be a Boolean value

2   **for** $i = 1$ **to** k **do**

3   Let $[\![\mathbf{m}]\!]$ be a subvector of $[\![\mathbf{B}_{u,v}]\!]$ such that $u \in \{i, \dots, k\}, v = i$

4      $([\![t]\!], [\![irow]\!]) \leftarrow \mathbf{maxLoc}([\![\mathbf{m}]\!])$

5      $irow \leftarrow$ declassify$([\![irow]\!])$

6   **if** $irow \neq i$ **then**

7   **for** $j = 1$ **to** k **do**

8   Exchange elements $[\![\mathbf{B}_{irow,j}]\!]$ and $[\![\mathbf{B}_{i,j}]\!]$

9      $[\![c]\!] \leftarrow [\![c]\!] \vee ([\![\mathbf{B}_{i,i}]\!] = 0)$

10     $\mathbf{q}_i \leftarrow irow$

11     $[\![ipiv]\!] \leftarrow [\![\mathbf{B}_{i,i}]\!]^{-1}$

12   **for** $m = i+1$ **to** k **do**

13        $[\![\mathbf{B}_{m,i}]\!] \leftarrow [\![\mathbf{B}_{m,i}]\!] \cdot [\![ipiv]\!]$

14   **for** $j = i+1$ **to** k **do**

15           $[\![\mathbf{B}_{m,j}]\!] \leftarrow [\![\mathbf{B}_{m,j}]\!] - [\![\mathbf{B}_{m,i}]\!] \cdot [\![\mathbf{B}_{i,j}]\!]$

16   **if** *declassify*$([\![c]\!])$ **then**

17   **return** "Singular matrix"

18   **return** $([\![\mathbf{B}]\!], \mathbf{q})$

Similarly to Alg. 10, 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. 10.

Alg. 12 shows how to solve a set of linear equations using LU decomposition. The matrix rows are shuffled as in Alg. 10 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 $[\![\mathbf{b}]\!]$ containing the dependent variable are permuted to be in concurrence with the permutations performed during the decomposition phase.

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

In addition to the methods based on the formulation (4), we look at the conjugate gradient method that tries to minimize the quadratic residuals directly. For a quadratic programming task, the conjugate gradient algorithm is guaranteed to converge in k steps, where k is the number of columns in the matrix $[\![\mathbf{A}]\!]$, provided that all computations are done without errors [21].

Algorithm 12: Solving the linear regression task $[\![\mathbf{y}]\!] \approx [\![\mathbf{X}]\!][\![\beta]\!]$ 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 $[\![\mathbf{y}]\!]$.
Result: A vector $[\![\mathbf{b}]\!]$ of coefficients.

1 Compute correlation matrix $[\![\mathbf{A}]\!] \leftarrow [\![\mathbf{X}]\!]^T [\![\mathbf{X}]\!]$
2 Compute new target vector $[\![\mathbf{b}]\!] \leftarrow [\![\mathbf{X}]\!]^T [\![\mathbf{y}]\!]$
3 Shuffle $[\![\mathbf{A}]\!], [\![\mathbf{b}]\!]$ retaining the dependencies
4 $([\![\mathbf{B}]\!], \mathbf{q}) \leftarrow \mathbf{LUDecomp}([\![\mathbf{A}]\!])$
5 Rearrange $[\![\mathbf{b}]\!]$ based on permutation $\mathbf{q}$
6 **for** $i = 2$ **to** k **do**

7 $\quad \left[\!\left[\mathbf{b}_i\right]\!\right] \leftarrow \left[\!\left[\mathbf{b}_i\right]\!\right] - \sum\limits_{j=1}^{i} \left[\!\left[\mathbf{B}_{i,j}\right]\!\right] \cdot \left[\!\left[\mathbf{b}_j\right]\!\right]$

8 **for** $i = k$ **downto** 1 **do**

9 $\quad \left[\!\left[\mathbf{b}_i\right]\!\right] \leftarrow \left(\left[\!\left[\mathbf{b}_i\right]\!\right] - \sum\limits_{j=i+1}^{k} \left[\!\left[\mathbf{B}_{i,j}\right]\!\right] \cdot \left[\!\left[\mathbf{b}_j\right]\!\right] \right) \cdot \left[\!\left[\mathbf{B}_{i,i}\right]\!\right]^{-1}$

10 **return** $[\![\mathbf{b}]\!]$

As our matrix is symmetric and positive semi-definite, we can use the simplest version of this method with a fixed number of iterations that depends on the number of variables k. Considering that the initial convergence of the conjugate gradient method is rapid during a small number of iterations [21] and that privacy-preserving floating point operations are approximately as imprecise as operations with the float datatype in the normal setting, a fixed number of iterations can be used. Alg. 13 shows how to solve the described quadratic programming task using the conjugate gradient method. As Alg. 13 is a straight line program, it is secure by default.

8. Data Classification Using the Kernel Perceptron Algorithm

Among the protocols for secure classification we look at the kernel perceptron algorithm. A cryptographically secure version of the algorithm has been studied in [22], where different cryptographic techniques are used for different steps of the algorithm.

Without a loss of generality, assume that our goal is to split the data into two classes. If more classes are needed, the classified data can be classified for a second time. Let $\mathbf{x_1}, \ldots, \mathbf{x_n}$ be the data parameter vectors that represent the row vectors of a data matrix $\mathbf{X}$. Let $y_1, \ldots, y_n$ be the corresponding classes of $\mathbf{x_1}, \ldots, \mathbf{x_n}$. The idea of classification is the following: given a training set $S = \{(\mathbf{x_1}, y_1), \ldots, (\mathbf{x_n}, y_n)\}$, find a function g so that $g(\mathbf{x_i}) = y_i$ for as many i as possible.

One of the easiest methods is to separate two classes with a straight line. If the data is linearly separable, then we may take $g(\mathbf{x}) = \langle \mathbf{w}, \mathbf{x} \rangle$ (where $\langle \mathbf{w}, \mathbf{x} \rangle = \sum_{i=1}^{|\mathbf{w}|=|\mathbf{x}|} w_i \cdot x_i$ is a scalar product of vectors $\mathbf{w}$ and $\mathbf{x}$) for a suitable vector $\mathbf{w}$. The question is how do we find a suitable $\mathbf{w}$. This is based on finding the best solution for the equation $\mathbf{X} \cdot \mathbf{w} = \mathbf{y}$ (that matches as many y coordinates as possible). The algorithm is described in more detail in [23, Chapter 2].

Algorithm 13: Privacy-preserving conjugate gradient method.

> **Data**: a $k \times k$ matrix $[\![\mathbf{A}]\!] = [\![\mathbf{X}]\!]^T [\![\mathbf{X}]\!]$, a vector $[\![\mathbf{b}]\!] = [\![\mathbf{X}]\!]^T [\![\mathbf{y}]\!]$ of k values for the dependent variable, number of iterations z.
> **Result**: A vector $[\![\mathbf{x}]\!]$ of coefficients.

1 Let $[\![\mathbf{x}]\!]$ be a vector of k values 0

2 $[\![\mathbf{x}]\!] \leftarrow [\![\mathbf{x}]\!]^T$

3 $[\![\mathbf{r}]\!], [\![\mathbf{p}]\!] \leftarrow [\![\mathbf{b}]\!]$

4 **repeat**

5 $[\![\alpha]\!] \leftarrow \dfrac{[\![\mathbf{r}]\!]^T [\![\mathbf{r}]\!]}{[\![\mathbf{p}]\!]^T [\![\mathbf{A}]\!] [\![\mathbf{p}]\!]}$

6 $[\![\mathbf{x}]\!] \leftarrow [\![\mathbf{x}]\!] + [\![\alpha]\!] [\![\mathbf{p}]\!]$

7 $[\![\mathbf{s}]\!] \leftarrow [\![\mathbf{r}]\!] - [\![\alpha]\!] [\![\mathbf{A}]\!] [\![\mathbf{p}]\!]$

8 $[\![\beta]\!] \leftarrow \dfrac{[\![\mathbf{s}]\!]^T [\![\mathbf{s}]\!]}{[\![\mathbf{r}]\!]^T [\![\mathbf{r}]\!]}$

9 $[\![\mathbf{p}]\!] \leftarrow [\![\mathbf{s}]\!] + [\![\beta]\!] [\![\mathbf{p}]\!]$

10 $[\![\mathbf{r}]\!] \leftarrow [\![\mathbf{s}]\!]$

11 $z \leftarrow z - 1$

12 **until** $z = 0$;

13 **return** $[\![\mathbf{x}]\!]^T$

As not every data vector is linearly separable, it has to be transformed in such a way that it becomes linearly separable. We may add more coordinates, but higher-dimensional data is more difficult to handle. Let us assume that we have a transformation φ that maps the given data vectors to some higher-dimensional linearly separable data vectors. The idea is to train the vector based on the values of some function $k(\mathbf{x_i}, \mathbf{x_j})$ that is computed for all possible data vectors. One example of function k is $k(\mathbf{x_i}, \mathbf{x_j}) = \langle \mathbf{x_i}, \mathbf{x_j} \rangle^2$, which is equal to $\langle \varphi(\mathbf{x_i}), \varphi(\mathbf{x_j}) \rangle$ where φ maps $\mathbf{x_i}$ to a higher-dimensional vector that contains all the possible squares of the initial coordinates of $\mathbf{x_i}$. In this case we can directly compute $k(\mathbf{x_i}, \mathbf{x_j})$ without applying φ to the data vectors. This kind of function k is called a *kernel function*. There are more examples of suitable kernel functions in [23].

For simplicity purposes, the kernel values can be represented by a matrix $\mathbf{K} = (k_{ij}) = (k(\mathbf{x_i}, \mathbf{x_j}))$. In order to use the kernel for classification, we look at the kernel-based classification algorithm called Kernel Perceptron (Algorithm 1 in [22]). We assume that the class vector y consists of values 1 and -1. We may think of looking at the sign of $\langle \mathbf{w}, \varphi(\mathbf{x}) \rangle$ in order to predict the class of $\mathbf{x}$.

As the goal is to implement the privacy-preserving version of Alg. 14, it is again more efficient to parallelize as many computations as possible.

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

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

Algorithm 14: Kernel Perceptron.

Data: A kernel matrix $\mathbf{K}$ and class labels $\mathbf{y} \in \{-1, 1\}^n$.
Result: A weight vector $\alpha \in \mathbb{Z}^n$.
1 $\alpha = 0$;
2 **repeat**
3 $\quad$ **for** $i=1$ **to** n **do**
4 $\quad\quad$ **if** $y_i \cdot \sum_{j=1}^n k_{ij} \cdot \alpha_j \leq 0$ **then**
5 $\quad\quad\quad$ $\alpha_i \leftarrow \alpha_i + y_i$;
6 **until** α *is no longer updated*;

---

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

- **Updating the classifier vector.** The iterations of the for-cycle are expensive. It is more efficient to compute the entire cycle in parallel, but the problem is that each step is actually dependent on the previous steps. Hence, we can only use parallelization by parts. For example, a training vector of length 50 can be computed in 10 sequential blocks of length 5. Increasing the number of blocks (reducing the parallelization) yields better results, but is slower.

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

References

[1] Dan Bogdanov. *Sharemind: programmable secure computations with practical applications*. PhD thesis, University of Tartu, 2013.

[2] Latanya Sweeney. k-anonymity: a model for protecting privacy. *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems*, 10(5):557–570, 2002.

[3] Ashwin Machanavajjhala, Daniel Kifer, Johannes Gehrke, and Muthuramakrishnan Venkitasubramaniam. *L*-diversity: Privacy beyond *k*-anonymity. *IEEE Transactions on Knowledge Discovery from Data*, 1(1), 2007.

[4] Ninghui Li, Tiancheng Li, and S. Venkatasubramanian. Closeness: A New Privacy Measure for Data Publishing. *IEEE Transactions on Knowledge and Data Engineering*, 22(7):943–956, July 2010.

[5] Cynthia 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.

[6] Sven Laur, Jan Willemson, and Bingsheng Zhang. Round-Efficient Oblivious Database Manipulation. In *Proceedings of ISC 2011*, pages 262–277, 2011.

[7] Sven Laur, Riivo Talviste, and Jan Willemson. From Oblivious AES to Efficient and Secure Database Join in the Multiparty Setting. In *Proceedings of the 11th International Conference on Applied Cryptography and Network Security, ACNS'13*, volume 7954 of *LNCS*, pages 84–101. Springer, 2013.

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

[9] Dan Bogdanov, Sven Laur, and Riivo Talviste. A Practical Analysis of Oblivious Sorting Algorithms for Secure Multi-party Computation. In *Proceedings of the 19th Nordic Conference on Secure IT Systems, NordSec 2014*, volume 8788 of *LNCS*, pages 59–74. Springer, 2014.

[10] C. A. R. Hoare. Algorithm 65: Find. *Commun. ACM*, 4(7):321–322, July 1961.

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

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

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

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

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

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

[17] Liina Kamm. *Privacy-preserving statistical analysis using secure multi-party computation*. PhD thesis, University of Tartu, 2015.

[18] Nils Homer, Szabolcs Szelinger, Margot Redman, David Duggan, Waibhav Tembe, Jill Muehling, John V Pearson, Dietrich A Stephan, Stanley F Nelson, and David W Craig. Resolving individuals contributing trace amounts of DNA to highly complex mixtures using high-density SNP genotyping microarrays. *PLoS Genet*, 4(8):e1000167, 2008.

[19] William H. Press, Saul A. Teukolsky, William T. Vetterling, and Brian P. Flannery. *Numerical Recipes 3rd Edition: The Art of Scientific Computing*. Cambridge University Press, New York, NY, USA, 3 edition, 2007.

[20] Magnus R. Hestenes and Eduard Stiefel. Methods of Conjugate Gradients for Solving Linear Systems. *Journal of Research of the National Bureau of Standards*, 49(6):409–436, December 1952.

[21] Owe Axelsson. Iteration number for the conjugate gradient method. *Mathematics and Computers in Simulation*, 61(3–6):421 – 435, 2003. MODELLING 2001 - Second IMACS Conference on Mathematical Modelling and Computational Methods in Mechanics, Physics, Biomechanics and Geodynamics.

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

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

Chapter 5
Achieving Optimal Utility for Distributed Differential Privacy Using Secure Multiparty Computation

Fabienne EIGNER [a], Aniket KATE [a], Matteo MAFFEI [a], Francesca PAMPALONI [b],
and Ivan PRYVALOV [a]

[a] *CISPA, Saarland University, Germany*
[b] *General Electric, Italy*

Abstract. Computing aggregate statistics about user data is of vital importance for a variety of services and systems, but this practice seriously undermines the privacy of users. Recent research efforts have focused on the development of systems for aggregating and computing statistics about user data in a distributed and privacy-preserving manner. Differential privacy has played a pivotal role in this line of research: the fundamental idea is to perturb the result of the statistics before release, which suffices to hide the contribution of individual users.

In this paper, we survey existing approaches to privacy-preserving data aggregation and compare them with respect to system assumptions, privacy guarantees, utility, employed cryptographic methods, and supported sanitization mechanisms. Furthermore, we explore the usage of secure multiparty computations (SMC) for the development of a general framework for privacy-preserving data aggregation. The feasibility of such an approach is a long-held belief, which we support by providing the first efficient cryptographic realization. In particular, we present PrivaDA, a new and general design framework for distributed differential privacy that leverages recent advances in SMC on fixed and floating point numbers. PrivaDA supports a variety of perturbation mechanisms, e.g. the Laplace, discrete Laplace, and exponential mechanisms. We demonstrate the efficiency of PrivaDA with a performance evaluation and its usefulness by exploring two potential application scenarios, namely, web analytics and lecture evaluations.

Keywords. Differential privacy, secure multiparty computation

Introduction

Statistics about user data play a significant role in the digital society: they are used daily for improving services, analyzing trends, performing marketing studies, conducting research, and so on. For instance, website owners rely on third-party analytics services to learn statistical information (e.g. gender, age, nationality) about their visitors; electricity suppliers introduced smart meters in order to constantly monitor users' electricity consumption, which allows them to compute prices based on energy usage trends, to opti-

mize energy distribution, and so on; service providers often ask their users to evaluate the quality of their services with the goal of publishing the aggregate results.

The acquisition and processing of sensitive information poses serious concerns about the privacy of users. The first problem is how and where is the user data aggregated: companies find it convenient to collect and process user data directly, but this gives them access to a wealth of sensitive information. For instance, web analytics rely on user tracking, which allows aggregators to reconstruct a detailed and precise profile of each individual. The second problem is how to publish aggregate data or statistics in a privacy-preserving manner. For example, researchers demonstrated how precise information about the habits of citizens can be reconstructed from the electricity consumption information collected by smart meters [1] and how the identity and state of health of individuals can be derived from genome-wide association studies [2].

Privacy-Preserving Statistics

The research community has long struggled to understand what privacy means in the context of statistics and, consequently, to devise effective privacy protection techniques.

Differential privacy.   Differential privacy (DP) [3] is a popular framework for defining and enforcing privacy for statistics on sensitive data. The fundamental idea is that a query on a database is differentially private if the contribution of an individual in the database can only marginally influence the query result. More precisely, the contribution of each single entry to the query result is bounded by a small constant factor, even if all remaining entries are known. A deterministic query can be made differentially private by perturbing the result with a certain amount of noise. The amount of noise depends on the query itself, and a variety of perturbation algorithms [4, 5] have been proposed for different queries and datatypes (e.g. numerical and non-numerical data, buckets, histograms, graphs).

Distributed differential privacy (DDP).   While the original definition of DP focused on a *centralized setting*, in which a database is queried by a curious entity, subsequent work has extended the definition to a *distributed setting* (e.g. [6, 7]), in which mutually distrustful, and potentially compromised, parties collaborate to compute statistics about distributed data. In particular, Dwork et al. [6] were the first to suggest the idea of employing *secure multiparty computation* (SMC) to aggregate and perturb data in a privacy-preserving distributed manner. In general, in a distributed setting, which will be the focus of this paper, the problem to solve is two-fold: (*i*) how to aggregate data and compute statistics without parties learning each other's data and (*ii*) how to perturb the result so as to obtain DP even in the presence of malicious parties that deviate from the protocol.

State-of-the-art.   Several specialized cryptographic protocols have been proposed recently to solve this problem (e.g. [4, 8–16]), and have paved the way for the enforcement of DP in challenging scenarios, such as smart metering [17, 18] and web analytics [11, 19, 20]. The different works can be grouped into *fully distributed* approaches [10, 12, 15, 16] (see [13] for a comparison), in which users themselves perform the sanitization mechanism in a distributed manner and *server-based* approaches [11, 14, 19, 20] that rely on few (non-colluding) *honest but curious (HbC)* parties, e.g. an aggregator and a publisher, to compute the noise. For a more detailed comparison we refer to Sec. 2.

Despite the tremendous progress made in this field, the widespread deployment of DP in modern systems still faces some open challenges.

First, many existing approaches to DDP exploit the divisibility properties of certain noise mechanisms and let each party produce a little amount of noise, the sum of which yields the noise required to achieve DP. This solution is affected by a trade-off between privacy and *utility*, as the amount of noise each user has to add is proportional to the number of tolerated malicious or failing parties: the more malicious parties, the more noise has to be added and therefore, the less accurate the result. Hence, in order to obtain strong privacy guarantees, each party should assume all others to be malicious, but this leads to an intolerable error ($O(N^2)$, where N is the number of users) as we will show in Sec. 2. Relying on a lower honesty threshold, however, not only gives lower privacy guarantees but also leads parties to the paradox of having to agree on how many of them are dishonest.

Second, several schemes suffer from the answer pollution problem: a single party can substantially pollute the aggregate result by adding excessive noise.

Third, many schemes involve a significant computational effort, which makes them impractical in several scenarios, e.g. for aggregating data stored on mobile devices with limited computation power.

Last, existing solutions are tailored to individual datatypes and perturbation mechanisms. Computing different kinds of queries or employing different perturbation mechanisms requires the use of different protocols that rely on different cryptographic schemes, communication patterns, and assumptions. The engineering effort and usability penalty are significant and discourage system administrators from deploying such technologies.

Our Contributions

In this work, we review and compare existing approaches to distributed privacy-preserving data aggregation. Furthermore, we present PrivaDA, the first generic framework for computing differentially private statistics about distributed data. We show how to achieve provable DDP guarantees, while overcoming the previously discussed limitations, by leveraging recently proposed SMC protocols for floating point numbers [9], fixed point numbers [21], and integers [22]. Our construction refines these schemes, originally designed for the honest but curious setting, so as to make them secure even in the malicious setting. Moreover, considering the recent work by Bendlin et al. [23] and Damgård et al. [24], we make these schemes tolerate any number of faults.

The overall privacy-preserving data aggregation computation is organized in two phases: the aggregation phase, in which the clients securely compute the aggregate result, and the perturbation phase, in which this result is perturbed so as to achieve DDP. To improve performance, SMC is actually conducted by computing parties that collect input shares from each client and perform the required computations. For the perturbation phase, the fundamental idea is to let computing parties jointly compute a random seed (i.e. a random variable in $(0, 1)$), which is then used to produce the required noise by encoding.

The distinctive features of our approach are the following.

Generality. PrivaDA supports a variety of perturbation mechanisms, such as noise drawn from the Laplace and the discrete Laplace (symmetric geometric) distribution as well as the exponential mechanism. Consequently, it is well-suited for a variety of application scenarios.

Strong privacy. As long as at least one of the computation parties is honest, malicious parties cannot learn the aggregate result, the seed or the noise (i.e. DDP is achieved). This is a fundamental difference from other approaches (e.g. [11,19,20], where colluding parties can immediately read the individual user's data.

Optimal utility and resistance to pollution attacks. The result is perturbed with the minimal amount of noise required to achieve DDP, irrespective of the expected number of dishonest users and computation parties. Hence, our protocol provides optimal utility and resistance to answer pollution. We also provide mechanisms to tackle the orthogonal problem of ensuring that the protocol only accepts client inputs that are contained in a set of valid answers.

Efficiency. We demonstrated the feasibility of our approach by implementing the system and conducting a performance evaluation. We stress that the client does not have to perform any expensive computation: she just has to provide each computing party with a share of her data and can then go offline, which makes this approach suitable even for mobile devices. Furthermore, PrivaDA supports a large number of clients without any significant performance penalty.

Outline. The paper is organized in the following manner. Sec. 1 gives some necessary background information on DP and on SMC for arithmetic operations, and Sec. 2 provides a survey of related work. Sec. 3 presents our framework and our algorithms for two query sanitization mechanisms, while Sec. 4 provides an instantiation of the differentially private algorithms with efficient SMC. Sec. 5 analyzes the security of these protocols, and Sec. 6 investigates their performance. Finally, Sec. 7 illustrates two use cases for our framework.

Unpublished content. The present work extends and revises a paper published at AC-SAC 2014 [25]. In this extended version we present some modifications of the original SMC that leverages recent work by Bendlin et al. [23] and Damgård et al. [24] based on [26] to improve performance and to eliminate the honest majority assumption. Furthermore, we have added a detailed survey of related works on DDP, and we have included the proofs of the main security results. Moreover, we have revised the application scenarios and added a new use case.

1. Background

In this section we present the concept of differential privacy and the cryptographic building blocks that PrivaDA builds on.

1.1. Differential Privacy

Differential privacy (DP), originally introduced by Dwork [3], has rapidly become one of the fundamental privacy properties for statistical queries. Intuitively, a query is differentially private if it behaves statistically similarly on all databases D, D' differing in one entry, written $D \sim D'$. This means that the presence or absence of each individual database entry does not significantly alter the result of the query. The definition of DP is parameterized by a number ε that measures the strength of the privacy guarantee: the

smaller the parameter ε, the smaller the risk to join the database. We use $\mathcal{D}$ and $\mathcal{R}$ to denote the domain and range of the query respectively.

Definition 1 (Differential Privacy [3]) *A randomized function* $f : \mathcal{D} \to \mathcal{R}$ *is* ε-*differentially private if for all databases* $D, D' \in \mathcal{D}$ *so that* $D \sim D'$ *and every set* $S \subseteq \mathcal{R}$, *it holds that* $Pr[f(D) \in S] \leq e^{\varepsilon} \cdot Pr[f(D') \in S]$.

A deterministic query can be made differentially private by perturbing its results with noise. In the following, we describe three popular perturbation mechanisms. An important insight is that the amount of noise perturbation depends on the query: the more a single entry affects the query result, the stronger the perturbation has to be. This can be expressed using the notion of the *sensitivity* of queries, which measures how much a query amplifies the distance between two inputs.

Definition 2 (Sensitivity [4]) *The sensitivity* Δf *of a query* $f : \mathcal{D} \to \mathcal{R}$ *is defined as* $\Delta f = \max_{\forall D, D' \in \mathcal{D}. D \sim D'} |f(D) - f(D')|$.

Intuitively, queries of low sensitivity map nearby inputs to nearby outputs. For instance, the query *"how many students like the 'Security' lecture?"* has sensitivity 1, as adding or removing one entry affects the result by at most 1.

Laplace noise. The most commonly used sanitization mechanism for queries returning a numerical result is the *Laplace mechanism* [4], i.e. the addition of random noise drawn according to a Laplace distribution $\mathsf{Lap}(\lambda)$ to the correct query result. As shown by Dwork et al. [4], this mechanism provides ε-DP, if the parameter λ is set to $\frac{\Delta f}{\varepsilon}$. The distribution is both parameterized by the sensitivity of the query and the privacy value ε.

Theorem 1 (DP of the Laplace mechanism [4]) *For all queries* $f : \mathcal{D} \to \mathbb{R}$ *the query* $f(x) + \mathsf{Lap}(\frac{\Delta f}{\varepsilon})$ *is* ε-*differentially private.*

Exponential mechanism. There are many scenarios in which queries return non-numerical results (e.g. strings or trees). For instance, consider the query "*what is your favorite lecture?*". For such queries, the addition of noise either leads to nonsensical results or is not well-defined. To address this issue, McSherry and Talwar [5] proposed the so-called *exponential mechanism*. The mechanism considers queries on databases D that are expected to return a query result a of an arbitrary type $\mathcal{R}$. For our purpose we consider the range $\mathcal{R}$ to be finite, e.g. the set of lectures offered by a university. We refer to each $a \in \mathcal{R}$ as a *candidate*. The mechanism assumes the existence of a *utility function* $q : (\mathcal{D} \times \mathcal{R}) \to \mathbb{R}$ that assigns a real valued score to each possible input-output pair (D, a) that measures the quality of the result a with respect to input D. The higher such a score, the better (i.e. more exact) the result. The mechanism $\varepsilon_q^{\varepsilon}(D)$ aims at providing the best possible result $a \in \mathcal{R}$, while enforcing DP.

Definition 3 (Exponential mechanism [5]) *For all* $q : (\mathcal{D} \times \mathcal{R}) \to \mathbb{R}$ *the randomized exponential mechanism* $\varepsilon_q^{\varepsilon}(D)$ *for* $D \in \mathcal{D}$ *is defined as* $\varepsilon_q^{\varepsilon}(D) := $ *return* $a \in \mathcal{R}$ *with probability proportional to* $e^{\varepsilon q(D,a)}$.

This definition captures the entire class of differential privacy mechanisms, as proven by McSherry and Talwar [5], who also give an encoding of Laplace noise addition by

choosing an appropriate utility function q. The authors also show that the mechanism $\varepsilon_q^{\frac{\varepsilon}{2\Delta q}}(D)$ provides ε-DP.

Theorem 2 (DP of the exponential mechanism [5]) *The mechanism* $\varepsilon_q^{\frac{\varepsilon}{2\Delta q}}(D)$ *is* ε-*differentially private.*

1.2. Secure Multiparty Computation

SMC enables a set of parties $\mathcal{P} = \{P_1, P_2, \ldots, P_\beta\}$ to jointly compute a function on their private inputs in a privacy-preserving manner [27]. More formally, every party $P_i \in \mathcal{P}$ holds a secret input value x_i, and $P_1, \ldots, P_\beta$ agree on some function f that takes β inputs. Their goal is to compute and provide $y = f(x_1, \ldots, x_\beta)$ to a recipient while making sure that the following two conditions are satisfied: (*i*) *correctness:* the correct value of y is computed, and (*ii*) *secrecy:* the output y is the only new information that is released to the recipient (see Sec. 5 for a formal definition).

Although the feasibility of SMC in the computational setting as well as in the information theoretic one has been known for more than 25 years, dedicated work to optimize secure realizations of commonly used arithmetic operations has started only in the last few years [9, 21, 22, 28–30]. Nevertheless, most of these realizations perform limited SMC arithmetic operations over input elements belonging only to finite fields [28] or integers [22]. Thus, they are not well-suited or sufficient for DDP mechanisms. In contrast, we build on SMC for algorithms on fixed point numbers [21] and some recent work by Aliasgari et al. [9], who presented SMC arithmetic algorithms over real numbers represented in floating point form. They also propose SMC protocols for elementary functions such as logarithm and exponentiation, and conversion of numbers from the floating point form to the fixed point form or the integer form and vice-versa. Our work starts by observing that their logarithm and exponentiation SMC protocols, combined with the SMC schemes for the basic integer, fixed point, and floating point number operations, pave the way for a practical design of various perturbation mechanisms for DDP in a completely distributed manner. Nevertheless, to be suitable for our design, we have to enhance and implement this large array of protocols to work against a malicious adversary.

We assume that secret sharing and basic SMC operations take place over a field $\mathbb{F}_q$. Let $[x]$ denote that the value $x \in \mathbb{F}_q$ is secret-shared among $P_1, \ldots, P_\beta$ so that participation from all β parties is required to reconstruct x. Note that $[x] + [y]$, $[x] + c$, and $c[x]$ can be computed by each P_i locally using her shares of x and y, while computation of $[x][y]$ is interactive and performed using Beaver's multiplication triple-based technique [26].

Basic SMC protocols.   We use the following SMC protocols [9, 21, 22, 28] for our DDP mechanisms.

1. The protocol $[r] \leftarrow \mathsf{RandInt}(k)$ allows the parties to generate shares $[r]$ of a random k-bit value r (i.e. $r \in [0, 2^k)$) without interactive operations [28].
2. The protocols $[a] \leftarrow \mathsf{IntAdd}([a_1], [a_2])$ and $[a] \leftarrow \mathsf{IntScMul}([a_1], \alpha)$ allow for the addition of two shared integers and the multiplication of a shared integer with a scalar, respectively, returning a shared integer.
3. The protocols $[b] \leftarrow \mathsf{FPAdd}([b_1], [b_2])$ and $[b] \leftarrow \mathsf{FPScMul}([b_1], \alpha)$ allow for the addition of two shared fixed point numbers and the multiplication of a shared fixed point number with a scalar, respectively, returning a shared fixed point number.

Table 1. Complexity of the employed SMC tasks

Type	Protocol	Rounds	Interactive Operations
Rand. Generation	RandInt	0	0
Reconstruction	Rec	1	1
Addition	IntAdd	0	0
	FPAdd	0	0
	FLAdd	$\log \ell + \log \log \ell + 27$	$14\ell + (\log \log \ell)\log \ell + (\ell + 9)\log \ell + 9k + 4\log k + 37$
Multiplication	FLMul	11	$8\ell + 10$
Division	FLDiv	$2\log \ell + 7$	$2\log \ell(\ell + 2) + 3\ell + 8$
Scalar	IntScMul	0	0
Multiplication	FPScMul	0	0
	FLScMul	10	$8\ell + 7$
Comparison	FLLT	6	$4\ell + 5k + 4\log k + 13$
Conversion	Int2FL	$\log \ell + 13$	$\log \ell(2\ell - 3) - 11$
	FP2FL	$\log \ell + 13$	$\log \ell(2\ell - 3) - 10$
	FL2Int	$3\log \log \ell + 53$	$27\ell + 3(\log \log \ell)\log \ell + 18\log \ell + 20k + 19$
	FL2FP	$3\log \log \ell + 53$	$27\ell + 3(\log \log \ell)\log \ell + 18\log \ell + 24k + 17$
Rounding	FLRound	$\log \log \ell + 30$	$15\ell + (\log \log \ell)\log \ell + 15\log \ell + 8k + 10$
Exponentiation	FLExp2	$12\log \ell + \log \log \ell + 27$	$8\ell^2 + 9\ell + \ell \log \ell + (\log \ell)\log \log \ell + 12k + 9$
Logarithm	FLLog2	$13.5\ell + 0.5\log \ell +$ $3\log \ell + 0.5\ell \log \log \ell +$ $3\log \log \ell + 146$	$15\ell^2 + 90.5\ell + 0.5(\log \ell)(\log \log \ell) +$ $3(\log \ell)\log \log \ell + 0.5\ell^2 \log \ell + 11.5\ell \log \ell +$ $4.5\ell k + 28k + 2\ell \log k + 16\log k + 128$

4. The protocols FLMul, FLDiv, and FLAdd can be used to multiply, divide, or add two shared floating point numbers, respectively. The output is a shared floating point number.

5. The conversion protocols FL2Int (float-to-int), Int2FL (int-to-float), FL2FP (float-to-fixed-point), and FP2FL (fixed-point-to-float) allow us to convert numbers represented as integers, floating point, or fixed point into another one of these representations.

6. The exponentiation protocol FLExp2 takes a shared floating point number $[r]$ as input and returns the shared floating point number corresponding to $[2^r]$.

7. The logarithm computation FLLog2 takes a shared floating point number $[r]$ and either returns the shared floating point number corresponding to $[\log_2 r]$ or an error (for $r \leq 0$).

8. The protocol FLRound takes a shared floating point value $[r]$ as input and operates on two modes (given as an additional argument). If mode $= 0$ then the protocol outputs the floor $[\lfloor r \rfloor]$, otherwise, if mode $= 1$ then the protocol outputs the ceiling $[\lceil r \rceil]$. The output is a shared floating point number.

9. The protocol FLLT allows us to compare two shared floating point numbers and returns $[1]$ if the first operand is less than the second, and $[0]$ otherwise.

In Table 1, we compare the complexities of the SMC protocols described. The complexity of SMC protocols is generally measured in terms of two parameters: interactive operations and rounds. An interactive operation involves every party sending a message to every other party, while round complexity measures the number of sequential invocations of interactive operations. Additional local computations are not included in the complexity.

Intuitively, additions for both integer and fixed point data types are non-interactive and consequently very fast, while for floating point values, the algorithm is quite involved and is also more costly than floating multiplication. Scalar multiplications for integers are free, however, a scalar multiplication for floating point numbers asks for almost the same amount of computation as in the case of FLMul. As expected, the more complex exponentiation and logarithm algorithms are also the most costly. Note that these complexities can be significantly reduced using pre-computation and batched processing [9]. The relative efficiency of these SMC schemes plays a fundamental role in our design.

2. Literature Review

The problem of privacy-preserving data aggregation has recently received increasing attention in the research community. In this section we survey relevant literature, unifying different terminologies employed in different works to allow for a meaningful comparison. The different works can be grouped into *fully distributed* approaches, in which the users themselves utilize the sanitization mechanism in a distributed manner, and *server-based* approaches that rely on few (non-colluding) parties, e.g. an aggregator and a publisher, to compute the noise. Note that the usage of computing parties that jointly generate the noise means that PrivaDA is not fully distributed, but in contrast to existing server-based solutions, it distributes the trust among multiple parties.

2.1. Fully Distributed Setting

Dwork et al. [6] propose a protocol for distributed noise generation: their scheme, however, requires interactions among all users. Some recent works [10, 12, 15, 16] (see also [13] for a comparison) propose fully distributed systems to distributively aggregate time-series data, where the latter are directly perturbed by users and then encrypted in such a way that the aggregator is only able to decrypt their aggregation but nothing else. Rastogi and Nath [15] propose to use the additively homomorphic threshold Paillier cryptosystem to encrypt users' data, and Shamir's secret sharing scheme to combine users' decryption shares. In their system, indeed, decrypting the aggregated result requires an extra interaction round between the aggregator and the users, where the latter are required to be online until the end of the decryption. This provokes both an increased communication cost and a long delay, besides the fact that requiring that all users are simultaneously online is a strong assumption that is inappropriate in several practical settings (e.g. mobile). Furthermore, their scheme supports only sum queries (i.e. no other linear combinations are allowed).

The system proposed by Shi et al. [12, 16] does not need the extra round and is valid for more kinds of queries, even if they are always numerical. By exploiting a system to intersect user groups, [12] compensates for user failures with the help of redundant information, which allows the system to be more resistant and to support dynamic joins and leaves. However, the redundant information, besides degrading utility, increases the communication cost (here around $O(\log N)$ and no longer satisfies the *aggregator obliviousness* (AO) property (according to which the aggregator does not learn anything else but the final query result) as intermediate data are now available to him. Furthermore, that work deals with a slightly weaker definition of privacy, namely, computational (ε, δ)-DP.

Ács and Castelluccia [10] propose another scheme for smart metering systems, which is robust against failures and malicious nodes and is built on a very simple modulo addition-based encryption scheme. This system requires each user to receive, and store, several encryption keys that cancel themselves out when aggregated, so that an explicit decryption is not necessary. Both requiring a user's side storage that is linear in the size of the network, and the need to establish several keys for each user, are not easy to realize in practice. More efficient protocols based on somewhat similar ideas have been proposed in the context of smart meters [17, 18].

2.2. Server-Based Setting

Several works rely on *honest but curious* servers that are additionally assumed not to collude with each other [11,14,19,20] to compute noise perturbation: in case of collusion, not only the noise but also the individual user's data is disclosed. In our SMC scheme PrivADA, colluding parties can neither recover the noise nor directly read the individual user's data.

In their recent work, Li and Cao [31] address the issue of users dynamically joining and leaving the data aggregation protocol, and propose a ring-based interleaved grouping construction that allows for data aggregation with high churn rates. However, their system relies on both a non-collusion assumption between the aggregator and the employed key dealer, and the participation of a threshold of honest users. Since PrivADA uses computing parties, the aggregation is independent of the users after they forward their data to the servers, circumventing the problem of dynamic user joins and leaves.

2.3. Strengths and Weaknesses of Existing Approaches

Table 2 compares some of the most important works about DDP. Here, N denotes the total number of users, and Δ is used to denote the sensitivity of the respective query (see Sec. 1). The function $\beta(\cdot, \cdot)$ is defined as $\beta(x,y) = \frac{\Gamma(x)\Gamma(y)}{\Gamma(x+y)}$, where $\Gamma(x) = \int_0^{+\infty} x^{t-1} e^{-x} dx$; γ is a lower bound on the fraction of honest users that we require to guarantee DP, and α is an upper bound on the number of failures (i.e. data that does not reach the aggregator) the system can accept. For the specifics of how the noise for sanitization is generated for the individual approaches (i.e. the use of Gaussian or Gamma distributions to generate the Laplace noise) we refer to [13]. We note that all papers in the table assume some compromised users (or computation parties), that is, users that follow the protocol correctly but may collude with the aggregator, passing him some information like the noise they have added or their data. Furthermore, the table specifies whether third parties, such as data aggregators (aggr.) or website publishers (publ.), are honest but curious or malicious (i.e. allowed to deviate from the protocol).

Utility. As the table demonstrates, a central drawback of all fully distributed models presented above is the poor utility of the result. This is due to the fact that the amount of noise each user has to add in order to satisfy privacy guarantees depends on other users' behaviors (i.e. the fraction of possibly malicious users and the probability of failure specified by γ, α that are supposed to be known in advance and that must not be exceeded so as to achieve DP). The more users are falsely assumed to be malicious (i.e. small γ, large k) the lower the final accuracy in the worst case. In PrivADA, instead, the noise is generated in a distributed fashion starting from a random seed that is jointly computed

by the computing parties. Differently from the fully distributed models, the final amount of noise obtained is exactly the one required to achieve DP (i.e. the utility is optimal), irrespective of the number of computation parties, the fraction of honest entities, or the probability of failures.

Non-collusion.   Similarly to [11,14,19], PrivaDA relies on a non-collusion assumption, but contrary to those approaches, we distribute the trust not only amongst two, but multiple parties (for which it suffices to assume an honest majority). In [14] an extension to the distributed case is proposed but the authors do not specify a method to distributively generate the noise. We note that we use mutually distrustful computation parties to mitigate the computational effort from the users, but that we could in principle let the users directly execute the perturbation phase if external parties were to be avoided.

Supported queries.   Another drawback, common to all previous models, is the restriction to specific queries and perturbation mechanisms. Most of the models described above, indeed, consider only counting queries, where the function is limited to weighted sums or even only supports sums, and use the Laplace or discrete Laplace mechanism to perturb data. The exponential mechanism, allowing perturbation in case of non-numerical queries, is studied in [32]. They propose a method to securely apply it using SMC. However, the system they propose is valid only for a two-party setting, differently from ours, that instead targets a multiparty scenario. By contrast, PrivaDA does support all three of the above mechanisms, providing a uniform framework to answer different kinds of queries in a differentially private manner.

Table 2. Comparison between the existing DDP schemes

ID	Assumptions	Utility (error)	Cryptoscheme & Perturbation Mechanism	Adversary type	Kind of queries
RN'10 [15]	#hon. users $\geq \gamma N$, bidirectional communication	$O(\frac{\Delta}{\varepsilon}(\frac{k}{\gamma N})^2)$, worst case: $O(N^2)$ $k = $ real #hon. users	Paillier scheme, Lap noise	malicious aggr., no failure	sum-statistics for time-series data (counting queries)
SCRCS'11 [16]	#hon. users $\geq \gamma N$	$O(\frac{\Delta}{\varepsilon\gamma})$, w.c.: $O(\sqrt{N})$	Pollard's Rho, diluted* DLap noise	honest-but-curious aggr.	as above
AC'11 [10]	#failures $\leq \alpha N$, several keys to store for user	$O(\frac{\Delta}{\varepsilon}\beta(\frac{1}{2},\frac{1}{1-\alpha})^{-1})$, w.c.: $O(\beta(\frac{1}{2},N)^{-1})$	modulo-addition scheme, Lap noise	malicious aggr., failures	as above
CSS'12 [12]	#hon. users $\geq \gamma N$	$\tilde{O}(\varepsilon^{-1}(\log N)^{1.5})$, w.c: $\tilde{O}(\sqrt{N}(\log N)^{1.5})$	Pollard's Rho , diluted DLap noise	HbC aggr., failures	as above
CRFG'12 [11]	no collusion aggr.-publ., pre-establ. queries, bidirectional communication	$O(\frac{\sqrt{\log N}}{\varepsilon})$	Goldwasser-Micali scheme, binomial noise	HbC aggr., malicious publ.	SQL-style queries (yes/no answers per buckets)
ACHFG'12 [19]	as above	$O(\frac{\Delta}{\varepsilon})$	Paillier scheme, Lap noise	as above	as above
JK'12 [14]	no collusion aggr.-auth.	$O(\frac{\Delta}{\varepsilon})$	Paillier scheme , Shamir's secret sharing , DLap noise	malicious aggr., HbC auth., failures	linear queries for time-series data
PrivaDA	hon. majority between computation parties	$O(\frac{\Delta}{\varepsilon})$	SMC, Lap, DLap noise, Expon. Mech.	malicious aggr.	multiple kinds of queries

* *Diluted* DLap noise: according to a certain probability p it follows the DLap distribution, otherwise it is set to 0.

Over the course of this paper, we demonstrate that the usage of SMC is ideally suited to minimize the trade-off between utility, strong non-collusion assumptions, and privacy guarantees, inherent in existing systems.

2.4. Alternative Approaches to Privacy

Although in this chapter we focus on the notion of DP, for completeness, we refer to some recent papers that investigate the limitations of this notion [33, 34] and propose alternative definitions of privacy for statistical queries [35–37].

2.5. Alternative Secure Multiparty Computation Schemes for Arithmetic Operations

In addition to the SMC schemes employed in this paper, further SMC building blocks that support integer, floating point, and fixed point functions have recently been proposed by Kamm and Willemson [38] and Krips and Willemson [39].

3. The PrivaDA Framework

In this section we present the PrivaDA framework. We first give a general overview of the setup and then present two mechanisms for achieving DP. A third mechanism, the discrete version of the Laplace mechanism, was presented at ACSAC 2014 [25].

3.1. Setting

We consider a scenario with n users, β computation parties (typically $\beta = 3$, but it can be greater), and an aggregator. Each user P_i has a private input D_i from some domain $\mathcal{D}$. The aggregator would like to compute some aggregate statistics about the users' private inputs, represented by the function $f : \mathcal{D}^n \to \mathcal{R}$. The range $\mathcal{R}$ of f may be a set of numerical values, but not necessarily. The computing parties are responsible for computing and perturbing the aggregate result, which is eventually returned to the data aggregator. The users communicate with the computing parties through secure and authenticated channels. Furthermore, the computing parties are pair-wise connected by secure authenticated channels. The users provide the computing parties with shares of their data and can then go offline. The computing parties engage in an interactive protocol.

Privacy goals. The aggregate result should be differentially private and neither the aggregator, nor the computation parties, nor the users should learn any further information about the individual users' data.

Attacker model. The data aggregator as well as the users may be corrupted and collude. The SMC protocols we adopt for the realization of our approach are based on secret sharing: such SMCs are secure in the malicious setting (i.e. the computing parties may try to deviate from the protocol).
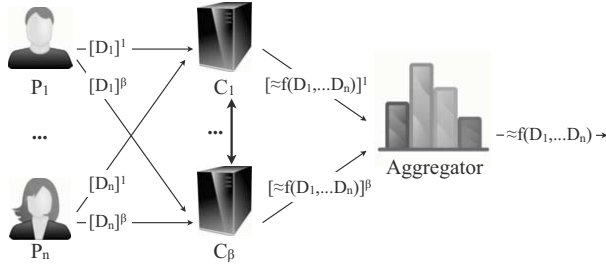
Figure 1. Protocol flow

3.2. Protocol Overview

The protocol proceeds in three steps. First, the users provide the computation parties with shares of their inputs.[1] Secondly, the computing parties run the SMC protocol to compute the aggregate statistics and perturb the result. Finally, each computing party gives the aggregator its share of the result, which is reconstructed by the aggregator. The protocol flow is depicted in Fig. 1. In the following we describe three different algorithms to compute queries sanitized with the Laplace, discrete Laplace, and exponential mechanism, respectively. To ease the presentation, we consider a class of queries for which $f(D_1,\ldots,D_n) = f(D_1) + \ldots + f(D_n)$. Other arithmetic queries can be implemented in a very similar manner using minor modifications of the presented algorithms, as modern SMC schemes provide direct support to a large class of arithmetic operations. The algorithms described below do not rely on specific SMC protocols: we give one possible efficient instantiation in Sec. 6.

3.3. Laplace Mechanism

We now describe an algorithm for the calculation of the Laplace mechanism (LM) for n inputs. We use the following mathematical results [41, 42], which allow us to reduce the problem of drawing a random number according to the Laplace distribution (Lap) to the problem of drawing a uniformly random number between 0 and 1 ($\mathcal{U}_{(0,1]}$) using the exponential distribution (Exp). It holds that the distribution $\mathsf{Lap}(\lambda) = \mathsf{Exp}(\frac{1}{\lambda}) - \mathsf{Exp}(\frac{1}{\lambda})$, where $\mathsf{Exp}(\lambda') = \frac{-\ln \mathcal{U}_{(0,1]}}{\lambda'}$. Thus,

$$\mathsf{Lap}(\lambda) = \lambda\,(\ln \mathcal{U}_{(0,1]}) - \lambda\,(\ln \mathcal{U}_{(0,1]}). \tag{1}$$

In particular, we know that $\lambda = \frac{\Delta f}{\varepsilon}$ guarantees DP. We can, thus, define our algorithm for the addition of Laplace noise on n inputs as shown in Alg. 1. It takes as input (i) n real numbers $d_1,\ldots,d_n$ owned by $P_1,\ldots,P_n$ respectively, which correspond to locally executing the query f on database D_i ($d_i = f(D_i)$) of each P_i, and (ii) the privacy budget parameter λ, set to $\frac{\Delta f}{\varepsilon}$ to guarantee ε-DP. The algorithm returns the real $w = (\sum_{i=1}^{n} d_i) + \mathsf{Lap}(\lambda)$, which is computed by first computing the sum of all d_i and then drawing a random number according to the distribution $\mathsf{Lap}(\lambda)$ (lines 2 - 3) using (1). It concludes

---

[1]Note that our work focuses on guaranteeing the privacy of user data. To solve the orthogonal problem of pollution attacks and prevent malicious users from entering wildly incorrect input shares, we can use ZK range proofs [40] (cf. Sec. 4).

Algorithm 1: Laplace mechanism LM	**Algorithm 2:** Exponential mechanism EM
Data: $d_1, \ldots, d_n$; $\lambda = \frac{\Delta f}{\varepsilon}$ **Result**: $\left(\sum\limits_{i=1}^{n} d_i\right) + \mathsf{Lap}(\lambda)$ 1 $d = \sum\limits_{i=1}^{n} d_i$ 2 $r_x \leftarrow \mathcal{U}_{(0,1]}$; $r_y \leftarrow \mathcal{U}_{(0,1]}$ 3 $r_z = \lambda \left(\ln r_x - \ln r_y\right)$ 4 $w = d + r_z$ 5 **return** w	**Data**: $d_1, \ldots, d_n$; $a_1, \ldots, a_m$; $\lambda = \frac{\varepsilon}{2}$ **Result**: winning a_k 1 $I_0 = 0$ 2 **for** $j = 1$ *to* m **do** 3 $\quad$ $z_j = \sum\limits_{i=1}^{n} d_i(j)$ 4 $\quad$ $\delta_j = e^{\lambda z_j}$ 5 $\quad$ $I_j = \delta_j + I_{j-1}$ 6 $r \leftarrow \mathcal{U}_{(0,1]}$; $r' = r I_m$ 7 $k = \mathsf{binary\_search}(r', \leq, I_0, \ldots, I_m)$ 8 **return** a_k

by adding the sum of the results and the noise together (line 4) and returning the result (line 5).

Privacy of LM. As the LM algorithm implements $\sum_{i=1}^{n} d_i + \lambda \left(\ln \mathcal{U}_{(0,1]}\right) - \lambda \left(\ln \mathcal{U}_{(0,1]}\right) = \sum_{i=1}^{n} d_i + \mathsf{Lap}(\lambda)$, by Thm. 1 it follows that $\mathsf{LM}(d_1, \ldots, d_n, \lambda)$ is ε-differentially private for $\lambda = \frac{\Delta f}{\varepsilon}$, where $d_i = f(D_i)$.

3.4. Exponential Mechanism

Concerning the algorithm to compute the exponential mechanism [5] (EM) for n inputs, our approach is inspired by [32], which is, however, constrained to a 2-party setting.

Inputs and outputs. The algorithm to compute the EM on the join of n databases is presented in Alg. 2. It outputs the candidate $a \in \mathcal{R}$ (where $|\mathcal{R}| = m \in \mathbb{N}$), which is the result of locally executing the desired query f on the databases $D_1, \ldots, D_n$ that are under the control of the participants $P_1, \ldots, P_n$ respectively and sanitizing the joint result using the exponential mechanism. The algorithm takes the following inputs: (*i*) the data sets $d_1, \ldots, d_n$ belonging to the participants $P_1, \ldots, P_n$ respectively, (*ii*) the list of candidates $a_1, \ldots, a_m$, and (*iii*) the privacy parameter λ. Note that in order to guarantee ε-DP, the parameter λ will be set to $\frac{\varepsilon}{2\Delta q}$. For the sake of simplicity, we assume each data set $d_i \in \mathcal{D}$ to be a histogram that is the result of locally executing $f(D_i)$. Each histogram is a sequence of m natural numbers $z_1, \ldots, z_m$ that correspond to the frequency of candidates $a_1, \ldots, a_m \in \mathcal{R}$. For instance, for the query $f :=$ "*What is your favorite lecture?*" the sequence of candidates $a_1, \ldots, a_5$ might be $\mathsf{Algebra, Logic, Security, Cryptography, Java}$ and the individual data set d_2 of student P_2 who prefers the lecture $\mathsf{Security}$ is a histogram of the form $0, 0, 1, 0, 0$. The algorithm outputs the winning candidate a_k drawn according to $\mathcal{E}_q^\varepsilon(d_1, \ldots, d_m)$.

Utility function. Our approach is general and can support any arithmetic utility function. For the sake of presentation, we focus below on the following utility function $q((z_1, \ldots, z_m), a_i) = z_i$ for all histograms $d = (z_1, \ldots, z_m)$ and candidates $a_1, \ldots, a_m$, which returns the frequency z_i of candidate a_i stored in d. For instance, in the above example $q(d_2, \mathsf{Security}) = 1$ and $q(d_2, a_i) = 0$ for all candidates a_i, where $i \in \{1, 2, 4, 5\}$.

Notice that $\Delta q = 1$, so it can be omitted from the algorithm and the privacy parameter λ is thus set to $\frac{\varepsilon}{2}$.

Random variable. Our goal is to compute the exponential mechanism $\varepsilon_q^\varepsilon(D)$ for a discrete range $\mathcal{R}$, where $|\mathcal{R}| = m$. The probability mass [5, 32] for the exponential mechanism is defined as $\Pr\left[\varepsilon_q^\varepsilon(D) = a\right] = \frac{e^{\varepsilon q(D,a)}}{\sum_{j=1}^m e^{\varepsilon q(D,a_j)}}$. As pointed out by Alhadidi et al. [32], drawing a random value according to this distribution corresponds to mapping the above defined probability mass onto the interval $(0,1]$ and drawing a random number r in $(0,1]$ to select the interval of the winning candidate. Formally, $r \leftarrow \mathcal{U}_{(0,1]}$ and $r \in (\sum_{k=1}^{j-1} \Pr\left[\varepsilon_q^\varepsilon(D) = a_k\right], \sum_{k=1}^{j} \Pr\left[\varepsilon_q^\varepsilon(D) = a_k\right]]$ corresponds to $a_j \leftarrow \varepsilon_q^\varepsilon(D)$. For instance, assume $\Pr\left[\varepsilon_q^\varepsilon(D) = a_1\right] = 0.3$ and $\Pr\left[\varepsilon_q^\varepsilon(D) = a_2\right] = 0.7$. We draw a random number r from $(0,1]$ and check whether r is in interval $(0,0.3]$ or in interval $(0.3,1]$. In this example, the drawing of $0.86 \leftarrow \mathcal{U}_{(0,1]}$ corresponds to $a_2 \leftarrow \varepsilon_q^\varepsilon(D)$.

It is easy to see that by multiplying with $S := \sum_{j=1}^m e^{\varepsilon q(D,a_j)}$ the check $r \in (\sum_{k=1}^{j-1} \Pr\left[\varepsilon_q^\varepsilon(D) = a_k\right], \sum_{k=1}^{j} \Pr\left[\varepsilon_q^\varepsilon(D) = a_k\right]]$ is equivalent to $r \cdot S \in (\sum_{k=1}^{j-1} e^{\varepsilon q(D,a_k)}, \sum_{k=1}^{j} e^{\varepsilon q(D,a_k)}]$, as $\Pr\left[\varepsilon_q^\varepsilon(D) = a\right] \cdot S = e^{\varepsilon q(D,a)}$. To optimize complexity, our algorithm will compute the exponential mechanism using the latter version, i.e. by drawing a random number $r \leftarrow \mathcal{U}_{(0,1]}$ and then checking $r \cdot S \in (\sum_{k=1}^{j-1} e^{\varepsilon q(D,a_k)}, \sum_{k=1}^{j} e^{\varepsilon q(D,a_k)}]$ and returning the candidate a_k for which this check succeeds. Thus, our main effort lies in computing the necessary interval borders $(\sum_{k=1}^{j-1} e^{\varepsilon q(D,a_k)}, \sum_{k=1}^{j} e^{\varepsilon q(D,a_k)}]$.

Algorithm. Our algorithm consists of the following steps.[2] First, initialize the interval border I_0 (line 1). Second, compute the joint histogram $d = d_1 + \ldots + d_n$ (line 3) by adding the frequencies for each individual candidate. Third, compute interval borders for candidates (line 4 - 5). Fourth, draw a random value r in $(0,1]$ (line 6) and multiply this value by $I_n = \sum_{j=1}^m e^{\varepsilon q(D,a_j)}$, resulting in the scaled random value r'. Fifth, check in which of the intervals $(I_{j-1}, I_j]$ the random value r' falls (line 7) by using binary search that returns k so that $I_{k-1} < r' \leq I_k$. Finally, return the winning candidate a_k (line 8).

Privacy of EM. The EM algorithm implements the join of n individual histograms, the utility function q as defined above, and the drawing of a random value according to $\varepsilon_q^\lambda(d_1 + \ldots + d_n)$, which is soundly encoded as explained above. Thus, EM$(d_1, \ldots, d_n, a_1, \ldots, a_m, \lambda)$ computes $\varepsilon_q^\lambda(d_1 + \ldots + d_n)$, where q has sensitivity 1 and by Thm. 2 it follows that EM$(d_1, \ldots, d_n, a_1, \ldots, a_m, \lambda)$ is ε-differentially private for $\lambda = \frac{\varepsilon}{2}$, where $d_i = f(D_i)$.

4. Instantiation

In this section, we instantiate the two mechanisms described in the previous section using the recently proposed SMC arithmetic algorithms over integers, and fixed and floating point numbers [9, 21, 22, 28] that we discussed in Sec. 1.

---

[2] Note that depending on the instantiation of SMC, the steps might be slightly modified, or type conversions added, to provide the best efficiency.

Algorithm 3: Distributed Laplace mechanism

Data: Shared fixed point form (γ, f) inputs $[d_1], \ldots, [d_n]$; $\lambda = \frac{\Delta f}{\varepsilon}$

Result: $w = (\sum\limits_{i=1}^{n} d_i) + \mathsf{Lap}(\lambda)$ in fixed point form

1   $[d] = [d_1]$
2   **for** $i = 2$ *to* n **do**
3       $[d] = \mathsf{FPAdd}([d], [d_i])$

4   $[r_x] = \mathsf{RandInt}(\gamma+1); [r_y] = \mathsf{RandInt}(\gamma+1)$
5   $\langle [v_x], [p_x], 0, 0 \rangle = \mathsf{FP2FL}([r_x], \gamma, f = \gamma, \ell, k)$
6   $\langle [v_y], [p_y], 0, 0 \rangle = \mathsf{FP2FL}([r_y], \gamma, f = \gamma, \ell, k)$
7   $\langle [v_{x/y}], [p_{x/y}], 0, 0 \rangle = \mathsf{FLDiv}(\langle [v_x], [p_x], 0, 0 \rangle, \langle [v_y], [p_y], 0, 0 \rangle)$
8   $\langle [v_{ln}], [p_{ln}], [z_{ln}], [s_{ln}] \rangle = \mathsf{FLLog2}(\langle [v_{x/y}], [p_{x/y}], 0, 0 \rangle)$
9   $\langle [v_z], [p_z], [z_z], [s_z] \rangle = \mathsf{FLMul}(\frac{\lambda}{\log_2 e}, \langle [v_{ln}], [p_{ln}], [z_{ln}], [s_{ln}] \rangle)$
10   $[z] = \mathsf{FL2FP}(\langle [v_{z_1}], [p_{z_1}], [z_{z_1}], [s_{z_1}] \rangle, \ell, k, \gamma)$
11   $[w] = \mathsf{FPAdd}([d], [z])$
12   **return** $w = \mathsf{Rec}([w])$

4.1. Secure Multiparty Computation for Distributed Mechanisms

Our protocols to compute the distributed Laplace mechanism and the distributed exponential mechanism are given in Alg. 3, and 4 respectively, and they are explained below.

Number representation. We follow the representation in [9] for integers and for real numbers in fixed point and floating point forms. For floating point form, each real value u is represented as a quadruple (v, p, z, s), where v is an ℓ-bit significand, p is a k-bit exponent, z is a bit which is set to 1 when the value $u = 0$, s is a sign bit, and $u = (1 - 2s) \cdot (1 - z) \cdot v \cdot 2^p$. Here, the most significant bit of v is always set to 1 and thus $v \in [2^{\ell-1}, 2^\ell)$. The k-bit signed exponent p is from the range $\mathbb{Z}_{\langle k \rangle} = (-2^{k-1}, 2^{k+1})$. We use γ to denote the bit-length of values in either integer or fixed point representation, and f to denote the bit-length of the fractional part in fixed point values. Every integer value x belongs to $\mathbb{Z}_{\langle \gamma \rangle} = (-2^{\gamma-1}, 2^{\gamma+1})$, while a fixed point number x is represented as $\bar{x}$ so that $\bar{x} \in \mathbb{Z}_{\langle \gamma \rangle}$ and $x = \bar{x} 2^{-f}$. Finally, it is required that $k > \max(\lceil \log(\ell+f) \rceil, \lceil \log(\gamma) \rceil)$ and $q > \max(2^{2\ell}, 2^\gamma, 2^k)$. For ease of exposition, we assume that $\gamma = 2\ell$ for integers and fixed point numbers, and that $f = \frac{\gamma}{2}$ for fixed point numbers.

Input distribution and output reconstruction. We assume that prior to computation, participants $P_1, \ldots, P_n$ create β shares of their respective integer or fixed point inputs $d_1, \ldots, d_n$ in the (β, β)-sharing form and distribute them amongst the β computing parties $C_1, \ldots C_\beta$, so that each party C_k holds a share of each input value $[d_i]$, for $k \in \{1, \ldots, \beta\}$ and $i \in \{1, \ldots, n\}$.

General overview. For the most part, the instantiation simply unfolds the mathematical operations used in the algorithms presented in Sec. 3 and replaces them by the corresponding SMCs for arithmetic operations that we list in Sec. 1.

Addition for both integers and fixed point numbers is very fast, while for floating point values, the protocol is costly. We thus choose the n shared data inputs $[d_1], \ldots, [d_n]$

to the mechanisms to be fixed point numbers or integers to lower the cost of adding them together to yield the joint unperturbed query result $[d_1] + \ldots + [d_n]$. We compute the noise values in floating point form as the required logarithm and exponentiation operations are only available for distributed floating point arithmetic. We use the conversion operations FP2FL, FL2Int, Int2FL whenever necessary.

Random number generation. As we have seen in the previous section, our algorithms rely heavily on the generation of a random number in the interval $(0, 1]$ drawn according to the uniform distribution $\mathcal{U}_{(0,1]}$. As the SMC suite we consider does not include such a function, we encode it using the existing primitive RandInt for the generation of a random integer. For instance, this is done in steps 4 and 5 of Alg. 3. We first generate a shared $(\gamma + 1)$-bit integer $[r_x]$ using the random number generator RandInt. We then consider this integer to be the fractional part of the fixed point number, whose integer part is 0 (by choosing $f = \gamma$). Afterwards, the fixed point number is converted to floating point form by using the function FP2FL and disregarding the shared sign bit.

Notice that strictly speaking, this generates a random number in $[0, 1)$. We can achieve a transition to the expected interval $(0, 1]$ by slightly modifying the conversion primitive FP2FL so that the shared $[0]$ is replaced by the sharing of $[1]$ in step 3 [9, Sec. 5]. We could avoid the modification of FP2FL and instead transition into the desired interval by subtracting the random number from 1, but this requires an additional costly addition step.

Exponentiation and logarithm. The work by Aliasgari et al. [9] provides SMC protocols for computing exponentiation with base 2 (FLExp2) and logarithm to base 2 (FLLog2). As we often require exponentiation and logarithm to a base $b \neq 2$, we use the following mathematical properties $b^a = 2^{a(\log_2 b)}$ and $\log_b x = \frac{\log_2 x}{\log_2 b}$ to compute exponentiation and logarithm for any base b. For instance, lines 8 - 9 in Alg. 3 and lines 7 - 8 in Alg. 4 use the above equations to compute logarithm and exponentiation to base e respectively.

Distributed Laplace mechanism. The protocol to compute the distributed Laplace mechanism is shown in Alg. 3. Note that the Laplace mechanism can use the simplification $\ln r_x - \ln r_y = \ln \frac{r_x}{r_y}$ and thus reduce the number of necessary logarithm operations FLLog2 as well as the number of follow-up operations.

Distributed exponential mechanism. The protocol to compute the distributed exponential mechanism using SMC is presented in Alg. 4. Each shared input $[d_i]$ consists of an integer array of the size m, representing the histogram of participant P_i. The instantiation follows the steps of the algorithm presented in Alg. 2 by using the insights and techniques we presented in this section. We straightforwardly implement the binary search to find the winning interval/candidate on lines 13 - 17. Note that we need a slightly simplified version of the FLLT protocol that outputs a value $\{0, 1\}$ that does not need to be shared, thus allowing us to output $w_{j_{\min}}$ immediately without reconstruction, which would require additional interactions.

4.2. Mechanisms in the Malicious Setting

In order to achieve DP against malicious computation parties, we need to strengthen the SMC protocols so as to make them resistant to computing parties that deviate from

---

Algorithm 4: Distributed exponential mechanism

Data: $[d_1], \ldots, [d_n]$; the number m of candidates; $\lambda = \frac{\varepsilon}{2}$

Result: m-bit w, s.t. smallest i for which $w(i) = 1$ denotes winning candidate a_i

1   $I_0 = \langle 0, 0, 1, 0 \rangle$

2   **for** $j = 1$ *to* m **do**

3      $[z_j] = 0$

4   **for** $i = 1$ *to* n **do**

5         $[z_j] = \mathsf{IntAdd}([z_j], [d_i(j)])$

6      $\langle [v_{z_j}], [p_{z_j}], [z_{z_j}], [s_{z_j}] \rangle = \mathsf{Int2FL}([z_j], \gamma, \ell)$

7      $\langle [v_{z'_j}], [p_{z'_j}], [z_{z'_j}], [s_{z'_j}] \rangle = \mathsf{FLMul}(\lambda \cdot \log_2 e, \langle [v_{z_j}], [p_{z_j}], [z_{z_j}], [s_{z_j}] \rangle)$

8      $\langle [v_{\delta_j}], [p_{\delta_j}], [z_{\delta_j}], [s_{\delta_j}] \rangle = \mathsf{FLExp2}(\langle [v_{z'_j}], [p_{z'_j}], [z_{z'_j}], [s_{z'_j}] \rangle)$

9      $\langle [v_{I_j}], [p_{I_j}], [z_{I_j}], [s_{I_j}] \rangle =$
      $\mathsf{FLAdd}(\langle [v_{I_{j-1}}], [p_{I_{j-1}}], [z_{I_{j-1}}], [s_{I_{j-1}}] \rangle, \langle [v_{\delta_j}], [p_{\delta_j}], [z_{\delta_j}], [s_{\delta_j}] \rangle)$

10   $[r] = \mathsf{RandInt}(\gamma + 1)$

11   $\langle [v_r], [p_r], 0, 0 \rangle = \mathsf{FP2FL}([r], \gamma, f = \gamma, \ell, k)$

12   $\langle [v'_r], [p'_r], [z'_r], [s'_r] \rangle = \mathsf{FLMul}(\langle [v_r], [p_r], 0, 0 \rangle, \langle [v_{I_m}], [p_{I_m}], [z_{I_m}], [s_{I_m}] \rangle)$

13   $j_{\mathsf{min}} = 1; j_{\mathsf{max}} = m$

14   **while** $j_{\mathsf{min}} < j_{\mathsf{max}}$ **do**

15      $j_{\mathsf{M}} = \lfloor \frac{j_{\mathsf{min}} + j_{\mathsf{max}}}{2} \rfloor$

16   **if** $\mathsf{FLLT}(\langle [v_{I_{j_{\mathsf{M}}}}], [p_{I_{j_{\mathsf{M}}}}], [z_{I_{j_{\mathsf{M}}}}], [s_{I_{j_{\mathsf{M}}}}] \rangle, \langle [v'_r], [p'_r], [z'_r], [s'_r] \rangle)$ **then**

17         $j_{\mathsf{min}} = j_{\mathsf{M}} + 1$ **else** $j_{\mathsf{max}} = j_{\mathsf{M}}$

18   **return** $w_{j_{\mathsf{min}}}$

---

the protocol [40, 43]. Intuitively, to maintain secrecy, one has to enforce two additional properties. First, the protocol-instance observations of honest parties must be consistent with each other, and second, every party must prove that each step of its computation was performed correctly.

Given the real-world impracticality of information-theoretically secure channels and subsequently information-theoretically secure SMC protocols, in the malicious setting we shift to the computational setting. In particular, we employ a computational verifiable secret sharing scheme (VSS) [44] instead of the basic secret sharing scheme to achieve the first secrecy property. For the second secrecy property, we introduce zero-knowledge (ZK) proofs so that a party could prove that a correct secret value is shared among the parties [43], and that shared secret values satisfy some mathematical conditions (e.g. they are in a pre-defined range) [40].

4.3. Limitations of Finite-Precision Instantiations

While the theoretical definition of sanitization mechanisms for DP operates on real numbers $r \in \mathbb{R}$ (or integers $z \in \mathbb{Z}$), the implementations of such mechanisms have to approximate these mathematical abstractions by finite-precision representations due to the physical limitations of actual machines. This mismatch has been shown to give rise to several attacks, as pointed out by Mironov [45] and Gazeau et al. [46]. Mironov [45] shows that the irregularities of floating point implementations result in porous Laplace distributions,

thus undermining the privacy guarantees of floating point implementations of this sanitization mechanism. He proposes the *snapping mechanism* that truncates large values and rounds the final result so as to achieve DP of the implementation. Gazeau et al. [46] show that, in general, approximation errors of any kind of finite-precision representation of real numbers can lead to the disclosure of secrets. They provide a solution for fixing such privacy breaches for a large class of sanitization mechanisms. The solution is based on the concept of *closeness* and uses rounding and truncation to guarantee a limited (but acceptable) variant of DP.

We point out that the mitigation techniques proposed in these works rely on arithmetic operations that can be implemented using our arithmetic SMC protocols, thus allowing us to circumvent the known attacks based on finite-precision implementations. For the sake of simplicity, we omitted this extension from our presentation.

5. Security Analysis

In this section we state the security model, conduct a security analysis in the honest but curious setting, and discuss how to extend this result to a malicious setting.

We first recall the standard notion of t-secrecy for SMC, which is formulated as in [9] except for a small modification to accommodate the computing parties. The following definitions refer to computing parties $\mathcal{C} = \{C_1, \ldots, C_\beta\}$ engaging in a protocol Π that computes function $y = f(D)$, where $D = D_1, \ldots, D_n$, and D_i denotes the input of party P_i, and $y \in \mathcal{R}$ is the output.

Definition 4 (View) *The view of C_i consists of its shares $\{[D]\}_{C_i}$ and its internal random coin tosses r_i, as well as the messages M exchanged with the other parties during the protocol execution induced by the other parties' random coin tosses h, i.e.* $\mathsf{VIEW}_{\Pi(D,h)}(C_i) = (\{[D]\}_{C_i}, r_i, M)$. $\mathsf{VIEW}_{\Pi(D)}(C_i)$ *denotes the corresponding random function conditioned to the other parties' coin tosses.*

Definition 5 (t-secrecy) *A protocol Π is t-private in the presence of honest but curious adversaries if for each coalition $I = \{C_{i_1}, C_{i_2}, \ldots, C_{i_t}\} \subset \mathcal{C}$ of honest but curious computing parties of the size $t < \beta/2$, there exists a probabilistic polynomial time simulator S_I so that $\{S_I(\{[D]\}_I, f(D))\} \equiv \{\mathsf{VIEW}_{\Pi(D,h)}(I), y\}$. Here, $\{[D]\}_I = \bigcup_{C \in I}\{[D]\}_C$, $\equiv$ denotes indistinguishability, $\mathsf{VIEW}_{\Pi(D,h)}(I)$ the combined view of the parties in I, and h the coin tosses of the parties in $\mathcal{C}\backslash I$.*

Let $\mathcal{V}_\Pi$ be the set of all possible views for the protocol Π. We now formally define the notion of DDP for protocols, similar to the one introduced in [47]. Here, two vectors $D, D' \in \mathcal{D}^n$ are said to be *neighbors* if they differ in exactly one coordinate, which corresponds to the scenario in which exactly one user changes her input.

Definition 6 (ε-DDP) *We say that the data sanitization procedure implemented by a randomized protocol Π among β computing parties $\mathcal{C} = \{C_1, \ldots, C_\beta\}$ achieves ε-DDP with respect to a coalition $I \subset \mathcal{C}$ of honest but curious computing parties of the size t, if the following condition holds: for any neighboring input vectors $D, D' \in \mathcal{D}^n$ and any possible set $\mathcal{S} \subseteq \mathcal{V}_\Pi$, $Pr[\mathsf{VIEW}_{\Pi(D)}(I) \in \mathcal{S}] \leq e^\varepsilon Pr[\mathsf{VIEW}_{\Pi(D')}(I) \in \mathcal{S}]$ holds.*

For the malicious setting, the coalition I of honest but curious parties in Def. 5 and 6 is replaced by an equal-sized coalition I^M of malicious computationally-bounded (for a security parameter κ) parties and the protocol Π is replaced by a computational protocol Π^M fortified against malicious attackers with the same t-secrecy property. The above DDP relation changes to an indistinguishability-based computational DDP (IND-DDP) [7, 12] relation with a negligible function $negl(\kappa)$ so that $Pr[\mathsf{VIEW}_{\Pi^{M}(D)}(I^M) \in \mathcal{S}] \leq e^{\varepsilon} Pr[\mathsf{VIEW}_{\Pi^{M}(D')}(I^M) \in \mathcal{S}] + negl(\kappa)$.

We now state our main theorems on ε-DDP in the honest but curious model, and ε-IND-DDP in the malicious model.

Theorem 3 (ε-DDP) *Let $\varepsilon > 0$. In the honest but curious setting, our distributed LM and EM protocols achieve ε-DDP with respect to any honest but curious coalition $I \subset \mathcal{C}$ of the size $t < \beta$.*

Proof sketch. We start our analysis by proving t-secrecy for our protocols and then use this property to prove ε-DDP. The SMC arithmetic protocols over integers, fixed and floating point numbers internally use only two basic SMC primitives over the finite field $\mathbb{F}_q$, namely, the addition and multiplication primitives for shared secret values from $\mathbb{F}_q$. Assuming secure instances of distributed addition and multiplication protocols over $\mathbb{F}_q$ [43] (and secure protocols built on top of them), Aliasgari et al. [9] have proved the correctness and t-secrecy properties of the SMC arithmetic protocols employed in our mechanisms using Canetti's composition theorem [48]. More formally, they suggested that one can build a simulator for their arithmetic SMC protocols by invoking simulators for the corresponding building blocks so that the resulting environment would be indistinguishable from the real protocol execution of participants.

The proof of t-secrecy for our protocols follows along the same lines, building a simulator for each of the distributed DP mechanisms using the simulators for the underlying floating point arithmetic SMC protocols and the other building blocks, so that the corresponding environment is indistinguishable from the corresponding real distributed DP protocol execution.

The correctness and t-secrecy properties of our SMC protocols allow us to lift the DP analysis for the LM and EM algorithms from Sec. 3 to the corresponding SMC protocols. In particular, the correctness property ensures that the result is perturbed as specified by the LM and EM algorithms. The t-secrecy of the SMC arithmetic protocols ensures that no information about user inputs and the noise is available to the adversary controlling the t compromised computing parties.

Theorem 4 (ε-IND-DDP) *Let $\varepsilon > 0$ and κ be a sufficiently large security parameter. In the malicious setting, our distributed LM and EM protocols achieve ε-IND-DDP with respect to any malicious coalition $I^M \subset \mathcal{C}$ of the size $t < \beta$, under the strong RSA and decisional Diffie-Hellman assumptions for parameter κ.*

Proof sketch. As the computational verifiable secret sharing (VSS) scheme we use [44] enjoys the perfect secrecy property, the t-secrecy analysis for the SMC protocols in the malicious setting remains almost the same as in the honest but curious setting. Nevertheless, an active adversary can target the secure communication channels between honest parties, whose security relies on the decisional Diffie-Hellman assumption (or another stronger Diffie-Hellman variant). However, an active adversary can only break channel

secrecy and consequently the t-secrecy of SMC protocols with negligible probability (in κ).

The correctness of the computational SMC protocols is also maintained in the malicious setting up to a negligible probability in the security parameter κ. For a computational VSS scheme, correctness requires the discrete logarithm assumption [44], zero-knowledge range proofs require the strong RSA assumption [40], and finally, the ZK proofs for secure multiplication require the discrete logarithm assumption [43].

As a result, using the correctness and t-secrecy properties of the computational SMC schemes we can lift the DP analysis for the LM and EM algorithms from Sec. 3 to the corresponding SMC-based protocol by only introducing an additive negligible factor corresponding to the event that one of the discussed assumptions is broken.

6. Performance Analysis

Aliasgari et al. [9] microbenchmarked the performance for most of the required arithmetic SMC protocols in the honest but curious setting for three computing parties. However, we could not successfully execute several library functions and their library does not handle the malicious setting. Hence, we developed the complete SMC library for both the honest but curious and malicious settings from scratch. Here, we present our SMC implementation for integer, fixed point, and floating point arithmetic and measure the performance costs for the distributed LM and EM protocols in both settings.

6.1. Implementation

We implement all SMC protocols discussed in Sec. 1 as well as our DDP mechanisms as multi-threaded object-oriented C++ code to support any number (≥ 3) of computing parties in the honest but curious and malicious settings. Our implementation uses the GMP library [49] for all finite field computations, the Relic toolkit [50] for elliptic curve cryptographic constructions, and the Boost [51] and OpenSSL libraries for secure communication. Our numeric SMC libraries can be of independent interest to other distributed computation scenarios, and our complete code base is available online [52].

6.2. Experimental Setup

The experiments are performed using a 3.20 GHz (Intel i5) Linux machine with 16 GB RAM, and 1 Gbps LAN. We run experiments for the 3-party (i.e. $\beta = 3$ and $t = 1$), and 5-party (i.e. $\beta = 5$ and $t = 2$) computation setting. The floating point numbers employed in the experiments have a bit-length of $\ell = 32$ for significands and $k = 9$ for (signed) exponents, which gives a precision of up to 2^{-256}. For integers and fixed point numbers, we use a bit-length of $\gamma = 64$, where $f = 32$ for fixed point numbers. It gives a precision of 2^{-32} for the latter. The experiments use finite fields of the size of 177 bits for integers, 208 bits for fixed point numbers, and 113 bits for floating point significands. For floating point exponents, as well as sign and zero bits, significantly smaller fields suffice. In contrast to [9], we do not employ batching (which improves average computation times) as our distributed mechanisms call the individual arithmetic SMC functions only a few times. To determine an average performance, we run the experiments ten times for both parameter sets. In Table 3, we show our results for all required SMC functionalities in the

Table 3. Performance of a single 3-party and 5-party SMC operations measured in seconds

Type	Protocol	HbC		Malicious	
		$\beta = 3$, $t = 1$	$\beta = 5$, $t = 2$	$\beta = 3$, $t = 1$	$\beta = 5$, $t = 2$
Float	FLAdd	0.75	1.00	7.91	16.3
	FLMul	0.24	0.28	1.79	3.30
	FLScMul	0.24	0.29	1.80	3.29
	FLDiv	0.90	1.00	3.22	5.90
	FLLT	0.19	0.22	1.46	2.76
	FLRound	0.75	0.89	5.80	11.67
Convert	FP2FL	1.42	1.21	12.4	25.9
	Int2FL	1.07	1.44	12.4	26.0
	FL2Int	1.65	2.01	13.4	26.9
	FL2FP	1.67	2.08	13.6	27.5
Log	FLLog2	16.56	21.44	147	296
Exp	FLExp2	8.58	10.22	63.6	120

honest but curious and malicious settings. In particular, we include the computation time for single 3-party and 5-party arithmetic SMC operations measured in seconds. Note that as we employ Beaver's triple technique for multiplications [26], we have an offline (background) phase of generating those triples [23, 24], and a fast online phase for every multiplication. We only consider the online phase computation times here.

6.3. Cost Analysis (Honest but Curious Setting)

As expected, the SMC protocols for logarithm and exponentiation are the most expensive operations, and they will drive our distributed mechanism cost analysis. Our protocols also use Rec, IntAdd, FPAdd, RandInt, but we did not include them in Table 3 as they are local operations that can be performed significantly faster than the rest of the protocols. Next, we determine the average performance costs for our distributed LM and EM protocols for ($\beta = 3$, $t = 1$) computing parties and $100,000$ users. The distributed LM protocol has a computation cost of 22.5 sec. The good efficiency of the LM mechanism is due to the fact that we halved the number of costly logarithm operations FLLog2 and necessary follow-up operations by using the property $\ln r_x - \ln r_y = \ln \frac{r_x}{r_y}$. The computation cost of the distributed EM protocol linearly depends on the number $m = |\mathcal{R}|$ of result candidates. For instance, for $m = 5$, the cost of computation is 44.6 sec.

For larger numbers of computing parties β, one can extrapolate the performance from our analysis. Even for $\beta \approx 100$, we expect the distributed LM protocol to take about a few hundred seconds in the honest but curious setting. We also compared our experimental results with [9]. We could not reproduce their results, possibly due to the introduced memory management and correctness verifications.

6.4. Cost Analysis (Malicious Setting)

As expected, the computations times for the SMC operations secure against an active adversary are around an order of magnitude higher than those of the operations secure

against an honest but curious adversary. The average performance costs for our distributed LM and EM protocols for ($\beta = 3$, $t = 1$) computing parties and $100,000$ users in the malicious setting are the following. The distributed LM protocol has an average computation cost of $187\,\mathrm{sec}$. The cost of the distributed EM protocol, for $m = 5$ result candidates, is $316\,\mathrm{sec}$. Here, we observe that shifting to multiplication using Beaver's triple-based method [26] reduced the computation cost by a factor of two as compared to our earlier result [25].

We stress that these operations are performed by computation parties, and that there are no critical timing restrictions on DDP computations in most real-life scenarios, such as web analytics. Nevertheless, we expect one order of magnitude performance gain in the honest but curious setting as well as the malicious setting by employing high-performance computing servers. Furthermore, as users have to simply forward their shared values to the computing parties, which is an inexpensive operation ($< 1\,\mathrm{msec}$ in the honest but curious setting and a few milliseconds in the malicious setting), we believe that these numbers demonstrate the practicality of PrivaDA even in a setting where clients are equipped with computationally limited devices, such as smartphones.

7. Application Scenarios

We show the flexibility of our architecture by briefly discussing how PrivaDA can be used to improve the state of the art in two different application scenarios.

7.1. Web Analytics

Web analytics consist of the measurement, collection, analysis, and reporting of Internet data about users visiting a website. For instance, data can include user demographics, browsing behavior, and information about the clients' systems. This information is important for publishers, because it enables them to optimize their site content according to the users' interests; for advertisers, because it allows them to target a selected population; and many other parties, who we will refer to as *analysts*.

State of the art.   In order to obtain aggregated user information, websites commonly use third-party web analytics services called *aggregators* that track users' browsing behavior across the web, thereby violating their privacy. Newer systems, e.g. a series of non-tracking web analytics systems [11, 19] proposed by Chen et al., provide users with DP guarantees but rely on strong non-collusion assumptions. Should a collusion happen, not only the noise but also the individual user's data would be disclosed.

Protocol design in PrivaDA.   The computing parties are operated by third parties that are possibly paid by the aggregator. In order to avoid multiple responses by each user without relying on a public key infrastructure, which is unrealistic in this setting, we add an initial step to the protocol. The publisher signs and gives each visiting user a different token, along with one or more queries and an associated expiry time (within which the result has to be computed). The user sends the tokens to the computation parties, together with their answer shares, so that the computation parties would be able to detect duplicates and discard them before the aggregation. The users only have to submit their shares and can then go offline. Finally, the support for a variety of perturbation
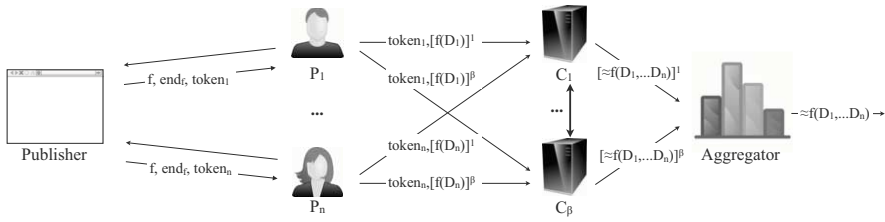
Figure 2. Protocol flow of privacy-preserving web analytics

mechanisms enables the execution of different kinds of analytical queries, for instance, our distributed exponential mechanism can be used to compute the average nationality and age group of visitors. The protocol is depicted in Fig. 2.

7.2. Anonymous Surveys

Next, let us consider anonymous surveys. In this setting, it is often reasonable to tolerate a little result perturbation in favor of strong privacy guarantees for the participating users.

State of the art. ANONIZE [53] is a recently proposed large-scale anonymous survey system. The authors exemplify it on an anonymous course evaluation service, in which students grade the courses they attend. However, ANONIZE does not address the problem that the survey result itself might still leak a lot of information about the individual user, which differential privacy aims at preventing.

Protocol design in PrivaDA. As compared to ANONIZE, the usage of PrivaDA yields differential privacy guarantees, besides avoiding the need to design and implement a complicated ad hoc protocol. We exemplify the usage of PrivaDA for anonymous surveys on the previously mentioned course evaluation service. Before submitting a grade for a certain course, students have to authenticate to prove their enrollment in that class. We envision a public key infrastructure maintained by the university, or an anonymous credential system used by the professor to grant her students access credentials. The computation parties will be implemented by organizations that are mutually distrustful, but are all interested in the results of the evaluation, such as the student association, or the university administration. The average grade is computed using the distributed Laplace mechanism.

References

[1] Andrés Molina-Markham, Prashant Shenoy, Kevin Fu, Emmanuel Cecchet, and David Irwin. Private Memoirs of a Smart Meter. In *BuildSys'10*, pages 61–66, 2010.

[2] Rui Wang, Yong Fuga Li, XiaoFeng Wang, Haixu Tang, and Xiaoyong Zhou. Learning Your Identity and Disease from Research Papers: Information Leaks in Genome Wide Association Study. In *CCS'09*, pages 534–544, 2009.

[3] Cynthia Dwork. Differential Privacy. In *ICALP'06*, pages 1–12, 2006.

[4] Cynthia Dwork, Frank McSherry, Kobbi Nissim, and Adam Smith. Calibrating Noise to Sensitivity in Private Data Analysis. In *TCC'06*, pages 265–284, 2006.

[5] Frank McSherry and Kunal Talwar. Mechanism Design via Differential Privacy. In *FOCS'07*, pages 94–103, 2007.

[6] Cynthia Dwork, Krishnaram Kenthapadi, Frank McSherry, Ilya Mironov, and Moni Naor. Our Data, Ourselves: Privacy Via Distributed Noise Generation. In *EUROCRYPT'06*, pages 486–503, 2006.

[7] Ilya Mironov, Omkant Pandey, Omer Reingold, and Salil P. Vadhan. Computational Differential Privacy. In *Crypto'09*, pages 126–142, 2009.

[8] Moritz Hardt and Guy N. Rothblum. A Multiplicative Weights Mechanism for Privacy-Preserving Data Analysis. In *FOCS'10*, pages 61–70, 2010.

[9] Mehrdad Aliasgari, Marina Blanton, Yihua Zhang, and Aaron Steele. Secure Computation on Floating Point Numbers. In *NDSS'13*, 2013.

[10] Gergely Ács and Claude Castelluccia. I have a DREAM! (DiffeRentially privatE smArt Metering). In *IH'11*, pages 118–132, 2011.

[11] Ruichuan Chen, Alexey Reznichenko, Paul Francis, and Johannes Gehrke. Towards Statistical Queries over Distributed Private User Data. In *NSDI'12*, pages 13–13, 2012.

[12] T.-H. Hubert Chan, Elaine Shi, and Dawn Song. Privacy-Preserving Stream Aggregation with Fault Tolerance. In *FC'12*, pages 200–214, 2012.

[13] Slawomir Goryczka, Li Xiong, and Vaidy Sunderam. Secure Multiparty Aggregation with Differential Privacy: A Comparative Study. In *EDBT/ICDT'13*, pages 155–163, 2013.

[14] Marek Jawurek and Florian Kerschbaum. Fault-Tolerant Privacy- Preserving Statistics. In *PETS'12*, pages 221–238, 2012.

[15] Vibhor Rastogi and Suman Nath. Differentially Private Aggregation of Distributed Time-Series with Transformation and Encryption. In *SIGMOD'10*, pages 735–746, 2010.

[16] Elaine Shi, T.-H. Hubert Chan, Eleanor G. Rieffel, Richard Chow, and Dawn Song. Privacy-Preserving Aggregation of Time-Series Data. In *NDSS'11*, 2011.

[17] George Danezis, Markulf Kohlweiss, and Alfredo Rial. Differentially Private Billing with Rebates. In *IH'11*, pages 148–162, 2011.

[18] Gilles Barthe, George Danezis, Benjaming Grégoire, César Kunz, and Santiago Zanella-Béguelin. Verified Computational Differential Privacy with Applications to Smart Metering. In *CSF'13*, pages 287–301, 2013.

[19] Istemi Ekin Akkus, Ruichuan Chen, Michaela Hardt, Paul Francis, and Johannes Gehrke. Non-tracking Web Analytics. In *CCS'12*, pages 687–698, 2012.

[20] Ruichuan Chen, Istemi Ekin Akkus, and Paul Francis. SplitX: High-Performance Private Analytics. In *SIGCOMM'13*, 2013. to appear.

[21] Octavian Catrina and Amitabh Saxena. Secure Computation With Fixed-Point Numbers. In *FC'10*, pages 35–50, 2010.

[22] Strange L. From and Thomas Jakobsen. Secure Multi-Party Computation on Integers. Master's thesis, University of Aarhus, 2006.

[23] Rikke Bendlin, Ivan Damgård, Claudio Orlandi, and Sarah Zakarias. Semi-homomorphic encryption and multiparty computation. In *EUROCRYPT'11*, pages 169–188, 2011.

[24] Ivan Damgård, Valerio Pastro, Nigel P. Smart, and Sarah Zakarias. Multiparty computation from somewhat homomorphic encryption. In *Crypto'12*, pages 643–662, 2012.

[25] Fabienne Eigner, Aniket Kate, Matteo Maffei, Francesca Pampaloni, and Ivan Pryvalov. Differentially Private Data Aggregation with Optimal Utility. In *ACSAC'14*, 2014.

[26] Donald Beaver. Efficient multiparty protocols using circuit randomization. In *Crypto'91*, pages 420–432, 1991.

[27] Andrew Chi-Chih Yao. Protocols for Secure Computations (Extended Abstract). In *FOCS'82*, pages 160–164, 1982.

[28] Ronald Cramer, Ivan Damgård, and Yuval Ishai. Share Conversion, Pseudorandom Secret-Sharing and Applications to Secure Computation. In *TCC'05*, pages 342–362, 2005.

[29] Assaf Ben-David, Noam Nisan, and Benny Pinkas. FairplayMP: A System for Secure Multi-party Computation. In *CCS'08*, pages 257–266, 2008.

[30] 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.

[31] Qinghua Li and Guohong Cao. Efficient Privacy-Preserving Stream Aggregation in Mobile Sensing with Low Aggregation Error. In *PETS'13*, pages 60–81, 2013.

[32] Dima Alhadidi, Noman Mohammed, Benjamin C. M. Fung, and Mourad Debbabi. Secure Distributed Framework for Achieving ε-Differential Privacy. In *PETS'12*, pages 120–139, 2012.

[33] Daniel Kifer and Ashwin Machanavajjhala. No Free Lunch in Data Privacy. In *SIGMOD'11*, pages

193–204, 2011.

[34] Andreas Haeberlen, Benjamin C. Pierce, and Arjun Narayan. Differential Privacy under Fire. In *USENIX'11*, 2011.

[35] S. P. Kasiviswanathan and A. Smith. A Note on Differential Privacy: Defining Resistance to Arbitrary Side Information. Report 2008/144, 2008.

[36] Raghav Bhaskar, Abhishek Bhowmick, Vipul Goyal, Srivatsan Laxman, and Abhradeep Thakurta. Noiseless Database Privacy. In *ASIACRYPT'11*, pages 215–232, 2011.

[37] Johannes Gehrke, Edward Lui, and Rafael Pass. Towards Privacy for Social Networks: A Zero-Knowledge Based Definition of Privacy. In *TCC'11*, pages 432–449, 2011.

[38] Liina Kamm and Jan Willemson. Secure Floating Point Arithmetic and Private Satellite Collision Analysis. *IJIS*, pages 1–18, 2014.

[39] Toomas Krips and Jan Willemson. Hybrid Model of Fixed and Floating Point Numbers in Secure Multiparty Computations. In *ISC'14*, pages 179–197, 2014.

[40] Fabrice Boudot. Efficient Proofs that a Committed Number Lies in an Interval. In *EUROCRYPT'00*, pages 431–444, 2000.

[41] Milton Abramowitz and Irene A. Stegun. *Handbook of Mathematical Functions with Formulas, Graphs, and Mathematical Tables*. Dover, 1964.

[42] Luc Devroye. Non-Uniform Random Variate Generation, 1986.

[43] Rosario Gennaro, Michael O. Rabin, and Tal Rabin. Simplified VSS and Fact-Track Multiparty Computations with Applications to Threshold Cryptography. In *PODC'98*, pages 101–111, 1998.

[44] T. P. Pedersen. Non-Interactive and Information-Theoretic Secure Verifiable Secret Sharing. In *Crypto'91*, pages 129–140, 1991.

[45] Ilya Mironov. On Significance of the Least Significant Bits for Differential Privacy. In *CCS'12*, pages 650–661, 2012.

[46] Ivan Gazeau, Dale Miller, and Catuscia Palamidessi. Preserving differential privacy under finite-precision semantics. In *QAPL'13*, pages 1–18, 2013.

[47] F. Eigner and M. Maffei. Differential Privacy by Typing in Security Protocols. In *CSF'13*, 2013.

[48] Ran Canetti. Security and Composition of Multiparty Cryptographic Protocols. *Journal of Cryptology*, 13(1):143–202, 2000.

[49] GMP: The GNU Multiple Precision Arithmetic Library. `http://gmplib.org`.

[50] D. F. Aranha and C. P. L. Gouvêa. RELIC is an Efficient LIbrary for Cryptography. `http://code.google.com/p/relic-toolkit/`.

[51] The Boost C++ Libraries. `http://www.boost.org`.

[52] PrivaDA Project Page: Full Version + Our SMPC Library. `http://crypsys.mmci.uni-saarland.de/projects/ArithmeticSMPC`.

[53] Susan Hohenberger, Steven Myers, Rafael Pass, and abhi shelat. ANONIZE: A Large-Scale Anonymous Survey System. In *S&P'14*, 2014.

Chapter 6
Oblivious Array Access for Secure Multiparty Computation

Peeter LAUD [a]

[a] *Cybernetica AS, Estonia*

Abstract. In this chapter, we describe efficient protocols for performing reads and writes in private arrays according to private indices. The protocols are implemented on top of the arithmetic black box (ABB) and can be composed freely to build larger privacy-preserving applications. We present two approaches to speed up private reads and writes — one based on precomputation and the other one on sorting. We show how several different problems become significantly more tractable while preserving the privacy of inputs. In particular, our second approach opens up a large class of parallel algorithms for adoption to run on SMC platforms.

Introduction

In secure multiparty computation (SMC), k parties compute $(y_1, \ldots, y_k) = f(x_1, \ldots, x_k)$, with the party P_i providing the input x_i and learning no more than the output y_i. For any functionality f, there is an SMC protocol [1,2]. A universally composable [3] abstraction for SMC is the arithmetic black box (ABB) [4]. The ideal functionality $\mathcal{F}_{ABB}$ allows the parties to store private data in it, perform computations with data inside the ABB, and reveal the results of computations. This means that the ABB does not leak anything about the results of the intermediate computations, but only the values whose declassification is explicitly requested by the parties. Hence, any secure implementation of ABB also protects the secrecy of inputs and intermediate computations. We discuss the basic techniques for SMC, and the ABB functionality in Chapters 1 and 2 of this book.

The existing ABB implementations may be quite efficient for realizing applications working with private data, if the control flow and the data access patterns of the application do not depend on private values. Simple methods for hiding data access patterns have overhead $\Theta(m)$ for accessing an element of an m-element vector. Smaller overheads can be obtained when techniques for *oblivious RAM* (ORAM) [5] are used in combination with SMC. In classical ORAM protocols, there are two parties: a memory-constrained *client* and a *server* which stores data on the client's behalf. The storage offered to the client by the server may be interpreted as a vector larger than the client's own memory. The client may access individual elements of this vector, but it wants to hide its access pattern. There has been a lot of interest in the topic of ORAM recently [6,7,8,9], with the best protocols achieving $O(\log m)$ overhead compared to public access patterns (or $O(\log^2 m)$ for general access patterns, if the size of an element of the vector is $o(\log^2 m)$).

In combination with SMC, the client's operations have to be executed securely, while the server's computations may run in public. The existing combinations of ORAM with SMC report at least $O(\log^3 m)$ overhead [10].

In this chapter, we propose two different methods for reading and writing data in SMC according to private addresses. The first is only suitable for reading according to a private index. The operations it performs are partitioned into the offline part that can be done without knowing the actual inputs, and the online part that requires the inputs. Moreover, in case of private lookup, it makes sense to consider three phases: offline, vector-only (where the vector to be read from, either public or private, is available) and online (where the private index is also available). In our protocols, the online phase requires only a constant number of costly SMC operations, while the bulk of the work is done offline and, depending on the protocol and the secrecy of the vector elements, in the vector-only phases. In cases where the main cost of SMC operations is communication between parties, the offline (and vector-only) computations can be performed using dedicated high-bandwidth high-latency channels.

Our second method is suitable for both reading and writing the elements of an array according to private indices. We note that SMC applications are often highly parallelized, because the protocols provided by ABB implementations often have significant latency. We exploit this parallelism in our second method, bundling together several data accesses. We provide two protocols on top of ABB: for reading the elements of an m-element vector n times, and for writing its elements n times. These protocols receive as an input a vector of indices (of the length n) and, in case of writing, a new vector of values. They return the vector of selected elements, or an updated original vector. The asymptotic complexity of both protocols is $O((m+n)\log m)$, with reasonable constants hidden in the O-notation (assuming that $n = O(m)$, which does not lessen the generality). These protocols can be interleaved with the rest of the SMC application in order to provide oblivious data access capability to it.

1. Related work

The reading of vector elements according to private indices is conceptually similar to private information retrieval (PIR) [11], for which there are protocols with $O(\log^2 n)$ communication and $O(n/\log n)$ public-key operations [12]. We know of no attempts to implement these techniques on top of SMC, though.

Protocols for oblivious RAM [5] have received significant attention during recent years [6,13,8]. The overhead of these protocols is around $O(\log^2 m)$ when accessing an element of a vector of the length m (with elements of the size $O(\log m)$). These ORAM constructions assume a client-server model, with the client accessing the memory held by the server, which remains oblivious to the access patterns. This model is simpler than the SMC model, because the computations of the client and the server are not shared among several parties.

In this chapter, we use SMC techniques to achieve oblivious data access in SMC applications. This goal has been studied before, by implementing the client's computations in an ORAM protocol on top of a secure two-party computation protocol set [14,15,16,17], or over an SMC protocol set [7,10]. For these protocol sets, the overhead of at least $O(\log^3 m)$ is reported. Recently, optimizing ORAM to perform well in the secure computation setting has become a goal of its own [18].

ORAM constructions often allow only sequential access to data, as the updating of the data structures maintained by the server cannot be parallelized. Recently, oblivious parallel RAM [19] has been proposed, which may be more suitable for SMC protocol sets where the computations have significant latency.

Our parallel reading protocol essentially builds and then applies an *oblivious extended permutation (OEP)* [20,21,22] (see Sec. 4.1 for details). Our OEP application protocol is more efficient (both in practice and asymptotically) than any other published construction built with SMC techniques.

2. Preliminaries

Universal composability (UC) [3] is a framework for expressing the security properties of systems. It considers an ideal functionality $\mathcal{F}$ and its implementation π with identical interfaces to the intended users. The latter is *at least as secure as* the former, if for any attacker $\mathcal{A}$ there is an attacker $\mathcal{A}_S$, so that $\pi\|\mathcal{A}$ and $\mathcal{F}\|\mathcal{A}_S$ are indistinguishable to any potential user of π or $\mathcal{F}$. The value of the framework lies in the composability theorem: if π is at least secure as $\mathcal{F}$, then ξ^π is at least as secure as $\xi^{\mathcal{F}}$ for any system ξ that uses π or $\mathcal{F}$. See Chapter 2, Sec. 5 for more details.

The *arithmetic black box* is an ideal functionality $\mathcal{F}_{\mathsf{ABB}}$. It allows its users (a fixed number of parties) to store and retrieve values securely, and to perform computations with them. See Chapter 2 for a thorough discussion of the $\mathcal{F}_{\mathsf{ABB}}$ functionality, as well as its possible implementations. In the protocols we present, we let $[\![x]\!]$ denote that some value has been stored in the ABB and is accessible under handle x. The notation $[\![z]\!] \leftarrow [\![x]\!] \otimes [\![y]\!]$ means that the operation $\otimes$ is performed on values stored under handles x and y, and the result is stored under handle z. In ABB implementations this involves the invocation of the protocol for $\otimes$.

All ABB implementations provide protocols for computing linear combinations of private values (with public coefficients) and for multiplying private values. The linear combination protocol is typically affordable and does not involve any communication and/or cheap operations with values. When estimating the (asymptotic) complexity of protocols built on top of ABB, it is typical to disregard the costs of computing linear combinations. Other operations, e.g. comparison, can be built on top of addition and multiplication [23,24], or the ABB implementation may have dedicated protocols for these operations [25]. In addition, our second method requires the ABB implementation to provide protocols for equality and comparison. These operations are also used by the applications we demonstrate.

2.1. Oblivious Shuffling

Our second method requires the ABB to provide *oblivious shuffles* — private permutations of values. For certain ABB implementations, these can be added as described in [26]. A more general approach is to use Waksman networks [27]. Given an oblivious shuffle $[\![\sigma]\!]$ for m elements, and a private vector $([\![v_1]\!], \ldots, [\![v_m]\!])$, it is possible to apply $[\![\sigma]\!]$ to this vector, permuting its elements and getting the vector $([\![v_{\sigma(1)}]\!], \ldots, [\![v_{\sigma(m)}]\!])$. It is also possible to *unapply* the shuffle to the obtained vector, and perform the inverse permutation of its elements. The complexity of the protocols implementing these ABB operations is either $O(m)$ or $O(m\log m)$ (for a constant number of parties).

We now describe how oblivious shuffles are implemented in the SHAREMIND SMC platform, tolerating one passively corrupted computing party out of a total of three. In the ABB implementation of SHAREMIND, a private value v from a finite ring R is additively shared: $[\![v]\!] = ([\![v]\!]_1, [\![v]\!]_2, [\![v]\!]_3)$, where party P_i knows $[\![v]\!]_i$ and $[\![v]\!]_1 + [\![v]\!]_2 + [\![v]\!]_3 = v$. A shuffle of n elements is an element σ of the symmetric group S_n. In SHAREMIND, following [26], we have $[\![\sigma]\!] = ([\![\sigma]\!]^1, [\![\sigma]\!]^2, [\![\sigma]\!]^3)$, where the components of $[\![\sigma]\!]$ are random elements of S_n satisfying $\sigma = [\![\sigma]\!]^1 \circ [\![\sigma]\!]^2 \circ [\![\sigma]\!]^3$, and each party knows two out of these three components (by convention: P_i does not know $[\![\sigma]\!]^{4-i}$). The protocol for applying a shuffle $[\![\sigma]\!]$ to a vector of private values $[\![\mathbf{v}]\!]$ is given in Alg. 1. It sequentially applies $[\![\sigma]\!]^1$, $[\![\sigma]\!]^2$, and $[\![\sigma]\!]^3$ to the vector $[\![\mathbf{v}]\!]$. To apply $[\![\sigma]\!]^i$, it reshares the current vector among the two parties that have $[\![\sigma]\!]^i$ (Alg. 2). The resharings for all elements of the vector can be done in parallel. Here and elsewhere, **foreach**-statements denote parallel executions. Clearly, this solution is secure against a single party. In fact, the messages received by each single party are just random elements of R. At the end of Alg. 1, the Reshare-operation (Alg. 1 in Chapter 2) adds a random sharing of 0 to each element of the vector. This does not change the values stored in the vector, but it does make the components of $[\![\mathbf{w}]\!]$ independent from the components of $[\![\mathbf{v}^{(3)}]\!]$. A protocol similar to Alg. 1 is also possible for ABBs based on Shamir's secret sharing [28].

Algorithm 1: Shuffling protocol in SHAREMIND

Data: A private vector $[\![\mathbf{v}]\!]$ of the length n
Data: A private shuffle $[\![\sigma]\!]$ of the size n
Result: A private vector $[\![\mathbf{w}]\!]$ satisfying $w_i = v_{\sigma(i)}$ for all i

1 **foreach** $i \in \{1, \ldots, n\}$ **do** $[\![v_i^{(1)}]\!] \leftarrow \mathsf{ReshareToTwo}([\![v_i]\!], P_3)$;
2 P_1 and P_2 reorder $[\![\mathbf{v}^{(1)}]\!]$ according to $[\![\sigma]\!]^1$
3 **foreach** $i \in \{1, \ldots, n\}$ **do** $[\![v_i^{(2)}]\!] \leftarrow \mathsf{ReshareToTwo}([\![v_i^{(1)}]\!], P_2)$;
4 P_1 and P_3 reorder $[\![\mathbf{v}^{(2)}]\!]$ according to $[\![\sigma]\!]^2$
5 **foreach** $i \in \{1, \ldots, n\}$ **do** $[\![v_i^{(3)}]\!] \leftarrow \mathsf{ReshareToTwo}([\![v_i^{(2)}]\!], P_1)$;
6 P_2 and P_3 reorder $[\![\mathbf{v}^{(3)}]\!]$ according to $[\![\sigma]\!]^3$
7 **foreach** $i \in \{1, \ldots, n\}$ **do** $[\![w_i]\!] \leftarrow \mathsf{Reshare}([\![v_i^{(3)}]\!])$;
8 **return** $[\![\mathbf{w}]\!]$

2.2. Private Sorting

With oblivious shuffles and comparison operations, vectors of private values (of the length m) can be sorted in $O(m \log m)$ time, where the size of the constants hidden in the O-notation is reasonable [28]. To sort a vector, we first generate a random shuffle of the length m and apply it to that vector. As this completely randomizes the order of the elements of the vector, it is safe to declassify the results of comparing them (as long as all elements of the vector are different, which must be ensured by the outer protocol). Hence, we can use any comparison-based sorting method to sort the shuffled vector.

In our protocols, we let $[\![\sigma]\!] \leftarrow \mathsf{sort}([\![\mathbf{v}]\!])$ denote the sorting operation applied to the private vector $[\![\mathbf{v}]\!]$. This operation does not actually reorder $\mathbf{v}$, but produces an oblivious

---

Algorithm 2: Protocol ReshareToTwo($[\![v]\!], P_k$) for resharing a private value $[\![v]\!]$ so that the share of party P_k is 0

    Data: A private value $[\![v]\!]$ and the name of party $P_k \in \{P_1, P_2, P_3\}$
    Result: A private value $[\![v']\!]$, so that $v = v'$ and $[\![v']\!]_k = 0$
1 P_k generates random $r \in R$ and computes $r' = [\![v]\!]_k - r$
2 P_k sends r to P_{k+1} and r' to P_{k-1}
3 P_{k-1} puts $[\![v']\!]_{k-1} = [\![v]\!]_{k-1} + r'$
4 P_k puts $[\![v']\!]_k = 0$
5 P_{k+1} puts $[\![v']\!]_{k+1} = [\![v]\!]_{k+1} + r$
6   **return** $[\![v']\!]$

---

shuffle $[\![\sigma]\!]$, the application of which to $\mathbf{v}$ would bring it to a sorted order. We require the sorting to be stable and the sorting protocol to be universally composable. In effect, this makes sort yet another operation provided by the ABB.

3. Private Lookup Based on Preprocessing

The following protocol requires the underlying ring R used by the ABB to be the field $\mathbb{F}$, where $|\mathbb{F}| \geq m + 1$. There are protocols for generating a uniformly random element of $\mathbb{F}$ inside the ABB (denote: $[\![r]\!] \overset{\$}{\leftarrow} \mathbb{F}$), and for generating a uniformly random nonzero element of $\mathbb{F}$ together with its inverse (denote: $([\![r]\!], [\![r^{-1}]\!]) \overset{\$}{\leftarrow} \mathbb{F}^*$). These protocols require a small constant number of multiplications on average for any ABB [23].

    Our protocol, depicted in Alg. 3, takes the handles to elements $v_{i_1}, \ldots, v_{i_m}$ (with arbitrary nonzero, mutually different indices $i_1, \ldots, i_m \in \mathbb{F}^*$) and the handle to the index j stored inside the ABB, and returns a handle to the element v_j. The protocol considers the vector of elements $(v_{i_1}, \ldots, v_{i_m})$ as a polynomial V over $\mathbb{F}$, satisfying $V(i_j) = v_{i_j}$ for all $j \in \{1, \ldots, m\}$. The lookup then amounts to the evaluation of the polynomial at a point. Similar ideas have appeared in [29] (for DFAs). We combine these ideas with a method to move offline most of the computations for the polynomial evaluation [30].

    There are Lagrange interpolation coefficients $\lambda_{j,k}^{\mathbf{I}} \in \mathbb{F}$ that depend only on the set $\mathbf{I} = \{i_1, \ldots, i_m\}$, so that $V(x) = \sum_{j=0}^{m-1} c_j x^j$, where $c_j = \sum_{k=1}^{m} \lambda_{j,k}^{\mathbf{I}} v_{i_k}$. These coefficients are public. For a given $\mathbf{I}$ of the size m they can be computed with $O(m^2)$ work [31].

Correctness.   The definition of c_k gives $\sum_{k=0}^{m-1} c_k l^k = v_l$ for all $l \in \{i_1, \ldots, j_m\}$. We can now verify that $w = \sum_{k=0}^{m-1} y_k z^k = \sum_{k=0}^{m-1} c_k r^k j^k r^{-k} = v_j$.

Security and privacy.   To discuss the security properties of a protocol in the $\mathcal{F}_{\mathsf{ABB}}$-hybrid model, we only have to consider which extra information the adversary may be able to obtain from the declassify-commands, and how it can affect the run of the protocol through the values it stores (the latter is significant only if the adversary is active). There are no store-commands in Alg. 3. The results of the declassify-commands are uniformly randomly distributed elements of $\mathbb{F}^*$, independent of anything else the adversary sees. These can be simulated without any access to $v_{i_1}, \ldots, v_{i_m}$ and j. Hence, Alg. 3 is secure and private against the same kinds of adversaries that the used implementation π_{ABB} of $\mathcal{F}_{\mathsf{ABB}}$ can tolerate.

Algorithm 3: Private lookup protocol

Data: Vector of indices $i_1, \ldots, i_m \in \mathbb{F} \setminus \{0\}$
Data: Vector of values $(\llbracket v_{i_1} \rrbracket, \ldots, \llbracket v_{i_m} \rrbracket)$ with $v_{i_1}, \ldots, v_{i_m} \in \mathbb{F}$
Data: Index $\llbracket j \rrbracket$ to be looked up, with $j \in \{i_1, \ldots, i_m\}$
Result: The looked-up value $\llbracket w \rrbracket = \llbracket v_j \rrbracket$
Offline phase

1 $(\llbracket r \rrbracket, \llbracket r^{-1} \rrbracket) \xleftarrow{\$} \mathbb{F}^*$
2 **for** $k = 2$ **to** $m - 1$ **do** $\llbracket r^k \rrbracket \leftarrow \llbracket r \rrbracket \cdot \llbracket r^{k-1} \rrbracket$;
3 Compute the coefficients $\lambda^{\mathbf{I}}_{j,k}$ from $i_1, \ldots, i_m$
   Vector-only phase

4 **foreach** $k \in \{0, \ldots, m-1\}$ **do** $\llbracket c_k \rrbracket \leftarrow \sum_{l=1}^{m} \lambda^{\mathbf{I}}_{k,l} \llbracket v_l \rrbracket$;
5 **foreach** $k \in \{0, \ldots, m-1\}$ **do** $\llbracket y_k \rrbracket \leftarrow \llbracket c_k \rrbracket \cdot \llbracket r^k \rrbracket$;
   Online phase

6 $z \leftarrow \mathsf{declassify}(\llbracket j \rrbracket \cdot \llbracket r^{-1} \rrbracket)$
7 $\llbracket w \rrbracket = \sum_{k=0}^{m-1} z^k \llbracket y_k \rrbracket$

Complexity.   In the offline stage, we perform $m - 2$ multiplications. We also generate one random invertible element together with its inverse; this generation costs the same as a couple of multiplications [23]. The round complexity of this computation, as presented in Alg. 3, is also $O(m)$, which would be detrimental to online computations. For offline computations, the acceptability of such round complexity mainly depends on the latency of the communication channels used. The offline phase can be performed in $O(1)$ rounds [23] at the cost of increasing the number of multiplications a few times. In the vector-only phase, the computation of the values $\llbracket c_k \rrbracket$ is free, while the computation of the values $\llbracket y_k \rrbracket$ requires $m - 1$ multiplications (the computation of $\llbracket y_0 \rrbracket$ is free). All these multiplications can be performed in parallel. If the vector v were public, then the computation of $\llbracket y_k \rrbracket$ would be free, too. The only costly operations in the online phase are a single multiplication and a single declassify-operation. These have similar complexities in existing ABB implementations.

3.1. Speeding up the Offline Phase

The preceding complexity analysis is valid for any implementation of ABB. Some implementations contain additional efficient operations that speed up certain phases of Alg. 3. If we use the additive secret sharing based implementation, as used in SHAREMIND [25], and a field $\mathbb{F}$ of characteristic 2 (a *binary* field), then we can reduce the complexity of the offline phase to $O(\sqrt{m})$, as shown below.

If the ring R is a binary field $\mathbb{F}$, then additive sharing is actually bit-wise secret sharing: $\llbracket v \rrbracket = \llbracket v \rrbracket_1 \oplus \llbracket v \rrbracket_2 \oplus \llbracket v \rrbracket_3$, where $\oplus$ denotes bit-wise exclusive OR. For such sharings, the usual arithmetic operations with shared values in $\mathbb{Z}_{2^n}$ are more costly, compared to additive sharings over $\mathbb{Z}_{2^n}$, but equality checks and comparisons are cheaper [25]. As most operations with array indices are expected to be comparisons, bit-wise secret sharing may be a good choice for them.

The multiplication protocol of SHAREMIND (Alg. 2 in Chapter 2) is based on the equality $(\llbracket u \rrbracket_1 + \llbracket u \rrbracket_2 + \llbracket u \rrbracket_3)(\llbracket v \rrbracket_1 + \llbracket v \rrbracket_2 + \llbracket v \rrbracket_3) = \sum_{i,j=1}^{3} \llbracket u \rrbracket_i \llbracket v \rrbracket_j$. After the party P_i has

sent $\llbracket u \rrbracket_i$ and $\llbracket v \rrbracket_i$ to party P_{i+1}, each of these nine components of the sum can be computed by one of the parties. The multiplication protocol is secure against one honest, but curious party [25, Theorem 2]. Indeed, as the sending of $\llbracket u \rrbracket_i$ from P_i to P_{i+1} takes place after resharing $\llbracket u \rrbracket$, the value $\llbracket u \rrbracket_i$ is a uniformly random number independent of all the other values P_{i+1} sees. Hence, the simulator for the view of P_{i+1} can itself generate this value. The same consideration also underlies the security proof of the specialized offline phase protocol given in Alg. 4, the properties of which we discuss below.

Algorithm 4: Computing $(\llbracket v^2 \rrbracket, \dots, \llbracket v^m \rrbracket)$ from $\llbracket v \rrbracket$ in SHAREMIND

    Data: $m \in \mathbb{N}$ and the value $\llbracket v \rrbracket$, where $v \in \mathbb{F}$, char $\mathbb{F} = 2$
    Result: Values $\llbracket u_0 \rrbracket, \dots, \llbracket u_m \rrbracket$, where $u_j = v^j$

1 $q \leftarrow \lceil \log \sqrt{m+1} \rceil$
2 $\llbracket u_0 \rrbracket \leftarrow (1,0,0)$
3 $\llbracket u_1 \rrbracket \leftarrow \mathsf{Reshare}(\llbracket v \rrbracket)$
4 Party P_i sends $\llbracket u_1 \rrbracket_i$ to party P_{i+1}
5 **for** $j = 1$ **to** $2^{q-1} - 1$ **do**
6   P_i computes $\llbracket u_{2j} \rrbracket_i \leftarrow \llbracket u_j \rrbracket_i^2$ and $\llbracket u_{2j} \rrbracket_{i-1} \leftarrow \llbracket u_j \rrbracket_{i-1}^2$
7   P_i computes $\llbracket t \rrbracket_i \leftarrow \llbracket u_j \rrbracket_i \cdot \llbracket u_{j+1} \rrbracket_i + \llbracket u_j \rrbracket_i \cdot \llbracket u_{j+1} \rrbracket_{i-1} + \llbracket u_j \rrbracket_{i-1} \cdot \llbracket u_{j+1} \rrbracket_i$
8      $\llbracket u_{2j+1} \rrbracket \leftarrow \mathsf{Reshare}(\llbracket t \rrbracket)$
9   Party P_i sends $\llbracket u_{2j+1} \rrbracket_i$ to party P_{i+1}
10 **foreach** $j \in \{2^q, \dots, m\}$ **do**
11   Let $(r,s) \in \{0, \dots, 2^q - 1\}$, so that $2^q r + s = j$
12   Party P_i computes $\llbracket t \rrbracket_i \leftarrow \llbracket u_r \rrbracket_i^{2^q} \cdot \llbracket u_s \rrbracket_i + \llbracket u_r \rrbracket_i^{2^q} \cdot \llbracket u_s \rrbracket_{i-1} + \llbracket u_r \rrbracket_{i-1}^{2^q} \cdot \llbracket u_s \rrbracket_i$
13     $\llbracket u_j \rrbracket \leftarrow \mathsf{Reshare}(\llbracket t_j \rrbracket)$

Correctness. We can use Alg. 4 only if $\mathbb{F}$ is a binary field. In this case squaring a shared value $\llbracket u \rrbracket$ is a local operation: $(\llbracket u \rrbracket_1 + \llbracket u \rrbracket_2 + \llbracket u \rrbracket_3)^2 = \llbracket u \rrbracket_1^2 + \llbracket u \rrbracket_2^2 + \llbracket u \rrbracket_3^2$ and the computation of $\llbracket u^2 \rrbracket_i = \llbracket u \rrbracket_i^2$ only requires the knowledge of $\llbracket u \rrbracket_i$. Note that the first loop of Alg. 4 satisfies the invariant that in the beginning of each iteration, each party P_i knows the values $\llbracket v^0 \rrbracket_i, \dots, \llbracket v^{2j-1} \rrbracket_i$ and also $\llbracket v^0 \rrbracket_{i-1}, \dots, \llbracket v^{2j-1} \rrbracket_{i-1}$. With these values, it can compute $\llbracket v^{2j} \rrbracket_i$ and $\llbracket v^{2j+1} \rrbracket_i$ (effectively, we are computing $v^{2j} = (v^j)^2$ and $v^{2j+1} = v^j \cdot v^{j+1}$). Party P_i can also compute $\llbracket v^{2j} \rrbracket_{i-1}$. It receives $\llbracket v^{2j+1} \rrbracket_{i-1}$ from P_{i-1}. In the second loop, we compute $v^j = (v^r)^{2^q} \cdot v^s$ for $j = 2^q \cdot r + s$. Again, $\llbracket (v^r)^{2^q} \rrbracket_i$ is locally computed from $\llbracket v^r \rrbracket_i$ by squaring it q times.

Privacy. We have to show that the view of a single party P_i can be simulated without access to shares held by other parties. Party P_i receives messages only on lines 4 and 9 of Alg. 4. In both cases, it receives a share of a freshly reshared value. Hence, this message can be simulated by a uniformly random number, as discussed in Chapter 2.

Complexity. We consider local computation and resharing to be free and hence, we have to count the number of messages sent by the parties. It is easy to see that a party sends at most $\sqrt{m+1}$ elements of the field $\mathbb{F}$ on lines 4 and 9 of Alg. 4. This is also the round complexity of Alg. 4. But by tracking the data dependencies in the first loop, we see that its iterations $2^{k-1}, \dots, 2^k - 1$ can be done in parallel for each $k \in \{1, \dots, q-1\}$. Hence, Alg. 4 can be executed in $O(\log m)$ rounds.

3.2. Speeding up the Vector-Only Phase

A different kind of optimization is available if the ABB implementation is based on Shamir's secret sharing [32], using the multiplication protocol by Gennaro et al. [33] (examples are VIFF [34] and SEPIA [35]). Such ABB implementations (for p parties), secure against t parties, can be given if $2t + 1 \leq p$ (for passive security) or $3t + 1 \leq p$ (for active security). In such implementations, a value $v \in \mathbb{F}$ for a field $\mathbb{F}$ of the size of at least $p + 1$ is stored in the ABB as $[\![v]\!] = ([\![v]\!]_1, \ldots, [\![v]\!]_p)$, so that there is a polynomial f over $\mathbb{F}$ with a degree of at most t and satisfying $f(0) = v$ and $f(c_i) = [\![v]\!]_i$ for all $i \in \{1, \ldots, p\}$, where $\mathbf{C} = \{c_1, \ldots, c_p\}$ is a set of mutually different, public, fixed, nonzero elements of $\mathbb{F}$. The share $[\![v]\!]_i$ is kept by the i-th party P_i.

Our optimization relies on the computation of a scalar product $[\![\sum_{i=1}^{k} u_i v_i]\!]$ from the values $[\![u_1]\!], \ldots, [\![u_k]\!]$ and $[\![v_1]\!], \ldots, [\![v_k]\!]$ stored inside the ABB, that costs as much as performing a single multiplication of stored values. For reference, Alg. 5 presents the scalar product protocol in the SSS-based ABB providing passive security (thus $2t + 1 \leq p$) [33]. The multiplication protocol can be obtained from it simply by letting the length of the vectors be 1. In this protocol, the values $\lambda_i^{\mathbf{C}}$ are the Lagrange interpolation coefficients satisfying $f(0) = \sum_{i=1}^{p} \lambda_i^{\mathbf{C}} f(c_i)$ for any polynomial f over $\mathbb{F}$ of a degree of at most $2t$. The protocols providing active security are much more complex [36], but similarly have equal costs for multiplication and scalar product.

Algorithm 5: Scalar product protocol in an SSS-based ABB [33]

> **Data**: Vectors $([\![u_1]\!], \ldots, [\![u_n]\!])$ and $([\![v_1]\!], \ldots, [\![v_n]\!])$
> **Result**: Value $[\![w]\!]$, so that $w = \sum_{j=1}^{n} u_j v_j$
> Party P_i computes $d_i \leftarrow \sum_{j=1}^{n} [\![u_j]\!]_i [\![v_j]\!]_i$
> Party P_i picks a random polynomial f_i of a degree of at most t, so that $f_i(0) = d_i$.
> Party P_i sends $f_i(c_j)$ to P_j
> Party P_i computes $[\![w]\!]_i \leftarrow \sum_{j=1}^{p} \lambda_j^{\mathbf{C}} f_j(c_i)$.

Our optimization consists of a reordering of the operations of the vector-only and online phases of the private lookup protocol, as depicted in Alg. 6. We see that compared to Alg. 3, 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 online phase becomes more complex, but only by a single scalar product that costs as much as a single multiplication. The correctness and privacy arguments for Alg. 6 are the same as for Alg. 3.

Unfortunately, we cannot use the optimizations of both Sec. 3.1 and Sec. 3.2 at the same time, as the cost of converting from one representation to the other cancels out any

Table 1. Communication costs (in elements of $\mathbb{F}$) of different private lookup protocols

Sharing	offline	vec-only	online
additive	$3\sqrt{m}$	$6m$	12
(public **v**)	$3\sqrt{m}$	0	12
Shamir's	$6m$	0	15
(public **v**)	$6m$	0	9

Algorithm 6: Improved vector-only and online phases of the private lookup protocol

---

Data: Lagrange interpolation coefficients $\lambda^{\mathbf{I}}_{j,k}$

Data: Random nonzero $[\![r]\!]$ 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 $[\![w]\!] = [\![v_j]\!]$.

Vector-only phase

1 **foreach** $k \in \{0, \ldots, m-1\}$ **do** $[\![c_k]\!] \leftarrow \sum_{l=1}^{m} \lambda^{\mathbf{I}}_{k,l} [\![v_l]\!]$;

Online phase

2 $z \leftarrow \mathsf{declassify}([\![j]\!] \cdot [\![r^{-1}]\!])$

3 **foreach** $j \in \{0, \ldots, m-1\}$ **do** $[\![\zeta_j]\!] \leftarrow z^j [\![r^j]\!]$;

4 $[\![w]\!] = ([\![c_0]\!], \ldots, [\![c_{m-1}]\!]) \cdot ([\![\zeta_0]\!], \ldots, [\![\zeta_{m-1}]\!])$

---

efficiency gains. If we have three parties and seek passive security against one of them, then our choices are given in Table 1. Recall that multiplication in both representations and declassification in the additive representation requires the communication of six field elements in total. Declassification in the representation based on Shamir's secret sharing requires three field elements to be sent.

4. Oblivious Data Access

4.1. Protocol for Reading

In Alg. 7, we present our protocol for obliviously reading several elements of an array. Given a vector $\mathbf{v}$ of the length m, let $\mathsf{prefixsum}(\mathbf{v})$ denote a vector $\mathbf{w}$, also of the length m, where $w_i = \sum_{j=1}^{i} v_j$ for all $j \in \{1, \ldots, m\}$. Computing $\mathsf{prefixsum}([\![\mathbf{v}]\!])$ is a free operation in existing ABB implementations, because addition of elements, not requiring any communication between the parties, is counted as having negligible complexity. We can also define the inverse operation $\mathsf{prefixsum}^{-1}$: if $\mathbf{w} = \mathsf{prefixsum}(\mathbf{v})$ then $\mathbf{v} = \mathsf{prefixsum}^{-1}(\mathbf{w})$. The inverse operation is even easier to compute: $v_1 = w_1$ and $v_i = w_i - w_{i-1}$ for all $i \in \{2, \ldots, m\}$.

We see that in Alg. 7, permutation σ orders the indices that we want to read, as well as the indices $1, \ldots, n$ of the original array $\mathbf{v}$. Due to the stability of the sorting algorithm, each index of the original array ends up before the reading indices equal to it. In $\mathsf{apply}(\sigma, \mathbf{u})$, each element v'_i of $\mathbf{v}'$, located in the same position as the index i of the original array in sorted $\mathbf{t}$, is followed by zero or more zeros. Prefix summing restores the elements of $\mathbf{v}$, with the zeros also replaced by the elements that precede them. Unapplying σ restores the original order of $\mathbf{u}$ and we can read the elements of $\mathbf{v}$ from the latter half of $\mathbf{u}'$.

The protocol presented in Alg. 7 clearly preserves the security guarantees of the implementation of the underlying ABB, as it applies only ABB operations, classifies only public constants and declassifies nothing. Its complexity is dominated by the complexity of the sorting operation, which is $O((m+n)\log(m+n))$. We also note that the round complexity of Alg. 7 is $O(\log(m+n))$.

Algorithm 7: Obliviously reading n values from the private array

---

Data: A private vector $[\![\mathbf{v}]\!]$ of the length m
Data: A private vector $[\![\mathbf{z}]\!]$ of the length n, with $1 \le z_i \le m$ for all i
Result: A private vector $[\![\mathbf{w}]\!]$ of the length n, with $w_i = v_{z_i}$ for all i
Preparation phase
1 **foreach** $i \in \{1, \dots, m\}$ **do** $[\![t_i]\!] \leftarrow i$;
2 **foreach** $i \in \{1, \dots, n\}$ **do** $[\![t_{m+i}]\!] \leftarrow [\![z_i]\!]$;
3 $[\![\sigma]\!] \leftarrow \mathsf{sort}([\![\mathbf{t}]\!])$
Application phase
4 $[\![\mathbf{v}']\!] \leftarrow \mathsf{prefixsum}^{-1}([\![\mathbf{v}]\!])$
5 **foreach** $i \in \{1, \dots, m\}$ **do** $[\![u_i]\!] \leftarrow [\![v_i']\!]$;
6 **foreach** $i \in \{1, \dots, n\}$ **do** $[\![u_{m+i}]\!] \leftarrow 0$;
7 $[\![\mathbf{u}']\!] \leftarrow \mathsf{unapply}([\![\sigma]\!]; \mathsf{prefixsum}(\mathsf{apply}([\![\sigma]\!]; [\![\mathbf{u}]\!])))$
8 **foreach** $i \in \{1, \dots, n\}$ **do** $[\![w_i]\!] \leftarrow [\![u_{m+i}']\!]$;
9 **return** $[\![\mathbf{w}]\!]$

Instead of reading elements from an array, the elements of which are indexed with $1, \dots, m$, the protocol presented could also be used to read the private values from a dictionary, the elements of which are indexed with (private) $[\![j_1]\!], \dots, [\![j_m]\!]$. In this case, on line 1, $[\![t_i]\!]$ is not initialized with i, but with $[\![j_i]\!]$. Note that the algorithm has to be modified to detect if all the indices that we attempt to read are present in the dictionary.

In Alg. 7, the argument $[\![\mathbf{v}]\!]$ is only used in the application phase. At the same time, the step that dominates the complexity of the protocol — sorting of $[\![\mathbf{t}]\!]$ on line 3 — takes place in the preparation phase. Hence, if we read the same positions of several vectors, the preparation phase of Alg. 7 can be executed only once and the application phase as many times as necessary. In Sec. 5.2.2, we will denote the preparation phase with `prepareRead` (with inputs $[\![\mathbf{z}]\!]$ and m, and output $[\![\sigma]\!]$), and the application phase with `applyRead` (with inputs $[\![\mathbf{v}]\!]$ and $[\![\sigma]\!]$).

4.2. Protocol for Writing

So as to specify the parallel writing protocol, we have to determine how to resolve multiple attempts to write to the same field. We require each writing request to come with a numeric *priority*. The request with the highest priority goes through (if it is not unique, then one is selected arbitrarily). We can also give priorities to the existing elements of the array. Normally they should have the lowest priority (if any attempt to write them actually means that they must be overwritten), but in some cases it makes sense to assign priorities differently. For example, in the Bellman-Ford algorithm for computing shortest distances, the current distance $d[v]$ of some vertex v from the source vertex s is always updated as $d[v] \leftarrow \min(d[v], L)$ for some L representing the length of some path from s to v. The reading of $d[v]$, the computation of the minimum, and the assignment to $d[v]$ can be implemented as a single assignment if the priority of each value is equal to the negation of that value.

Assume that there is an algorithm `compute_priority` which, when applied to an element $[\![w_i]\!]$ of the vector $[\![\mathbf{w}]\!]$, as well as to its index i, returns the priority of keeping the current value of the i-th element of **w**. The parallel writing protocol is given in Alg. 8.

The writing algorithm receives a vector of values $[\![\mathbf{v}]\!]$ to be written together with the indices $[\![\mathbf{j}]\!]$ showing where they have to be written, and the writing priorities $[\![\mathbf{p}]\!]$. Alg. 8 transforms the current vector $[\![\mathbf{w}]\!]$ (its indices and priorities) into the same form and concatenates it with the indices and priorities of the write requests. The data is then sorted according to indices and priorities (with the higher-priority elements coming first). The operation zip on two vectors of equal length transforms them into a vector of pairs. The ordering on these pairs is determined lexicographically. The vector $[\![\mathbf{b}]\!]$ is used to indicate the highest-priority position for each index: $b_i = 0$ if the i-th element in the vector $\mathbf{j}'$ is the first (and hence, the highest-priority) value equal to j'_i. All equality checks on line 10 can be done in parallel. Performing the sort on line 11 moves the highest-priority values to the first m positions. The sorting is stable and hence, the values correspond to the indices $1,\ldots,m$ in this order. Thus, we have to apply the shuffles induced by both sorts to the vector of values $\mathbf{v}' = \mathbf{v}\|\mathbf{w}$, and take the first m elements of the result.

Algorithm 8: Obliviously writing n values to a private array

Data: Private vectors $[\![\mathbf{j}]\!]$, $[\![\mathbf{v}]\!]$, $[\![\mathbf{p}]\!]$ of the length n, where $1 \le j_i \le m$ for all i
Data: Private array $[\![\mathbf{w}]\!]$ of the length m
Result: Updated $\mathbf{w}$: values in $\mathbf{v}$ written to indices in $\mathbf{j}$, if priorities in $\mathbf{p}$ are high
      enough
Preparation phase
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 $\quad [\![p'_{n+i}]\!] \leftarrow -\mathsf{compute\_priority}(i,[\![w_i]\!])$
7 $[\![\sigma]\!] \leftarrow \mathsf{sort}(\mathsf{zip}([\![\mathbf{j}']\!],[\![\mathbf{p}']\!]))$
8 $[\![\mathbf{j}'']\!] \leftarrow \mathsf{apply}([\![\sigma]\!];[\![\mathbf{j}']\!])$
9 $[\![b_1]\!] \leftarrow 0$
10 **foreach** $i \in \{2,\ldots,N\}$ **do** $[\![b_i]\!] \leftarrow [\![j''_i]\!] \stackrel{?}{=} [\![j''_{i-1}]\!];$
11 $[\![\tau]\!] \leftarrow \mathsf{sort}([\![\mathbf{b}]\!])$
Application phase
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 $[\![\mathbf{w}']\!] \leftarrow \mathsf{apply}([\![\tau]\!];\mathsf{apply}([\![\sigma]\!];[\![\mathbf{v}']\!]))$
15 **foreach** $i \in \{1,\ldots,m\}$ **do** $[\![w_i]\!] \leftarrow [\![w'_i]\!];$
16 **return** $[\![\mathbf{w}]\!]$

The writing protocol is secure for the same reasons as the reading protocol. Its complexity is dominated by the two sorting operations; it is $O((m+n)\log(m+n))$, with the round complexity being $O(\log(m+n))$. Similarly to the reading protocol, the writing protocol can be adapted to write into a dictionary instead. Another similarity is the split into two phases — the complex sorting operations in the preparation phase only use indices and priorities, while the actual values are used solely in cheap operations in the application phase. For the purposes of Sec. 5.2.2, we introduce the protocols prepareWrite

executing the preparation phase, and `applyWrite` executing the application phase. The protocol `prepareWrite` receives as inputs $\mathbf{j}$, $\mathbf{p}$, and the length m of $\mathbf{w}$. It lets the existing elements of $\mathbf{w}$ have the least possible priority, i.e. they will be definitely overwritten if there is at least one request to do so (prioritizing the existing elements of $\mathbf{w}$ is not needed in Sec. 5.2.2). The output of `prepareWrite` is the pair of oblivious shuffles $([\![\sigma]\!], [\![\tau]\!])$. These are input to `applyWrite` together with $[\![\mathbf{v}]\!]$ and $[\![\mathbf{w}]\!]$.

4.3. Sorting Bits

Alg. 8 makes two calls to the sorting protocol. While the first is a rather general sorting protocol, the second one on line 11 only performs a stable sort on bits, ordering the 0-bits before the 1-bits (and the sort does not have to be stable on the 1-bits). In the following, we show that the second sort can be performed with complexity similar to that of a random shuffle instead of a full sort. Our method leaks the number of zeros among the bits, but this information was already public in Alg. 8 (being equal to the length of $\mathbf{w}$). The sorting protocol is given in Alg. 9. Here, $\mathsf{random\_shuffle}(n)$ generates an oblivious random shuffle for vectors of the length n. The protocol ends with a composition of an oblivious and a public shuffle. This operation, as well as the generation of a random shuffle, is supported by existing implementations of shuffles.

Algorithm 9: Stable sorting of 0-bits in a bit-vector

    Data: Vector of private values $[\![\mathbf{b}]\!]$ of the length m, where each $b_i \in \{0, 1\}$
    Result: Oblivious shuffle $[\![\sigma]\!]$, so that $\mathsf{apply}([\![\sigma]\!]; [\![\mathbf{b}]\!])$ is sorted and the order of
            0-bits does not change
    Leaks: The number of 0-bits in $[\![\mathbf{b}]\!]$
1 **foreach** $i \in \{1, \ldots, m\}$ **do** $[\![c_i]\!] \leftarrow 1 - [\![b_i]\!]$;
2 $[\![\mathbf{x}]\!] \leftarrow \mathsf{prefixsum}([\![\mathbf{c}]\!])$
3 $[\![\tau]\!] \leftarrow \mathsf{random\_shuffle}(m)$
4 $\mathbf{b}' \leftarrow \mathsf{declassify}(\mathsf{apply}([\![\tau]\!]; [\![\mathbf{b}]\!]))$
5 $[\![\mathbf{x}']\!] \leftarrow \mathsf{apply}([\![\tau]\!]; [\![\mathbf{x}]\!])$
6 **foreach** $i \in \{1, \ldots, m\}$ **do**
7     $\lfloor \; y_i \leftarrow$ **if** $b'_i = 0$ **then** $\mathsf{declassify}([\![x'_i]\!])$ **else** $m + 1$
8 Let ξ be a public shuffle that sorts $\mathbf{y}$
9 $[\![\sigma]\!] \leftarrow [\![\tau]\!] \circ \xi$
10 **return** $[\![\sigma]\!]$

We see that the most complex operations of Alg. 9 are the applications of oblivious shuffle $[\![\tau]\!]$. If the communication complexity of these is $O(m)$ and the round complexity of these is $O(1)$, then this is also the complexity of the entire protocol. The protocol declassifies a number of things and thus it is important to verify that the declassified values can be simulated. The vector $\mathbf{b}'$ is a random permutation of zeros and ones, where the number of 0-bits and 1-bits is the same as in $[\![\mathbf{b}]\!]$. Hence, the number of 0-bits is leaked. However, nothing else is leaked: if the simulator knows the number n of 0-bits, then $\mathbf{b}'$ is a uniformly randomly chosen bit-vector with n bits with the value 0 and $(m - n)$ bits with the value 1.

The vector $\mathbf{y}$ (computed in a constant number of rounds, as all declassifications can be done in parallel) is a random vector of numbers, so that $(m-n)$ of its entries equal $(m+1)$, and the rest are a uniformly random permutation of $\{1,\ldots,n\}$. The numbers $\{1,\ldots,n\}$ in $\mathbf{y}$ are located at the same places as the 0-bits in $\mathbf{b}'$. Hence, the simulator can generate $\mathbf{y}$ after generating $\mathbf{b}'$. The sorting protocol does not declassify anything else, apart from $\mathbf{b}'$ and $\mathbf{y}$. The rest of Alg. 9 consists of invoking the functionality of the ABB or manipulating public data.

4.4. Performance and Applicability

Using our algorithms, 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 read values. Dividing by n, we learn that the cost of one access is $O((1+\frac{m}{n})\log(m+n))$. In practice, the cost will depend greatly 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 for those where the ABB implementation used is based on secret sharing and requires ongoing communication between the parties. Parallelization allows decreasing the number of communication rounds necessary for the application, thereby 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 $1/f$ of the memory it uses, where f is actually a function of the total memory size m of the application, and $1 \le f(m) \le m$. 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. Recall that for ORAM implementations 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.

Parallel random access machines (PRAM) are a theoretical model for parallel computations, for which a sizable body of efficient algorithms exists. PRAM models differ in the permissiveness of simultaneous access to the same memory cell by different processors. Commonly considered models are EREW (exclusive read, exclusive write), CREW (common read, exclusive write) and CRCW (common read, common write). The latter has several subclasses regulating the handling of simultaneous writes to the same memory cell. Using our parallel reading and writing protocols, any algorithm for priority-CRCW PRAM (PRAM, where many processors can read or write the same memory cell in parallel, with priorities determining which write goes through) 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 are 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 an algorithm on top of SMC using the reading and writing protocols presented in this chapter, we can say that the overhead of these protocols is $O(\log^2 m)$. 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, to which the optimization described above also applies), the overhead is even smaller.

5. Applications

We have implemented the protocols described in Sec. 3 and Sec. 4 on the SHAREMIND secure multiparty computation platform and used them to implement a number of applications. The private lookup in Sec. 3 has been used for the privacy-preserving execution of deterministic finite automata (discussed in Chapter 7) and for computing single-source shortest distances (SSSD) in sparse graphs. The oblivious access protocols in Sec. 4 have been used to find the minimum spanning tree (MST) of a graph in a privacy-preserving manner. All algorithms have been benchmarked on three computing nodes, each of which was deployed on a separate machine. The computers in the cluster were connected by an Ethernet local area network with the link speed of 1 Gbps. Each computer in the cluster had 48 GB of RAM and a 12-core 3 GHz CPU with Hyper Threading.

Privacy-preserving graph algorithms have been studied in [37] in a non-composable manner. Composable SSSD protocols for dense graphs have been studied in [38]. Recently, ORAM-with-SMC techniques have been used to implement Dijkstra's algorithm for sparse graphs [10]. Non-composable privacy-preserving implementations of Kruskal's and Prim's algorithms are discussed in [37].

5.1. Algorithm for Single-Source Shortest Distances

Let $G = (V, E)$ be a directed graph with $s, t : E \to V$ denoting the source and target, and $w : E \to \mathbb{N}$ denoting the length of each edge. Let $v_0 \in V$. The Bellman-Ford (BF) algorithm for SSSD starts by defining $d_0[v] = 0$, if $v = v_0$, and $d_0[v] = \infty$ for $v \in V \setminus \{v_0\}$. It then computes $d_{i+1}[v] = \min(d_i[v], \min_{e \in t^{-1}(v)} d_i[s(e)] + w(e))$ for all $v \in V$ and $i \in \{0, \ldots, |V| - 2\}$. The vector $d_{|V|-1}$ is the result of the algorithm.

We have implemented the BF algorithm on top of the SHAREMIND platform, hiding the structure of the graph, as well as the lengths of edges. In our implementation, the numbers $n = |V|$ and $m = |E|$ are public, and so are the in-degrees of vertices (obviously, these could be hidden by using suitable paddings). In effect, the mapping t in the definition of the graph is public, while the mappings s and w are private. During the execution, we use private lookup to find $d_i[s(e)]$. As the vectors d_i have to be computed one after another, but the elements of the same vector can be computed in parallel, our implementation has $O(n)$ rounds in the online phase.

As the vector d_i is not yet available at the start of computation, we use the optimized vector-only phase to avoid an $O(n)$ factor during the execution of the BF algorithm. Hence, we use the ABB implementation based on Shamir's secret sharing. We have to perform arithmetic and comparisons with secret values and hence, we must use a prime field as the field $\mathbb{F}$ (we use $GF(p)$ with $p = 2^{32} - 5$).

For 1,000 vertices and 6,000 edges, the online phase of our implementation requires around 10 minutes. See [39] for the performance of the implementation on tasks of other sizes.

5.2. Algorithms for Finding the Minimum Spanning Tree

Let $G = (V, E)$ be an undirected graph, where the set of vertices V is identified with the set $\{1, \ldots, |V|\}$ and the set of edges E with a subset of $V \times V$ (the edge between vertices u and v occurs in E both as (u, v) and as (v, u)). We assume that the graph is

connected. Let $\omega : E \to \mathbb{N}$ give the weights of the edges (ω must be symmetric). A *minimum spanning tree* (MST) of G is a graph $T = (V, E')$ that is connected and for which the sum $\sum_{e \in E'} \omega(e)$ takes the smallest possible value.

Kruskal's and Prim's algorithms are the two most well-known algorithms for finding the MST of a graph. These algorithms work in time $O(|E| \log |V|)$ or $O(|E| + |V| \log |V|)$ [40]. They are inherently sequential and, therefore, unsuitable for an SMC implementation.

Other algorithms for MST have been proposed. Borůvka's algorithm [41] works in iterations. At the beginning of each iteration, the set of vertices V has been partitioned into $V_1 \dot\cup \cdots \dot\cup V_k$ and for each V_i, the minimum spanning tree has already been found (at the start of the algorithm, each vertex is a separate part). For each i, let e_i be a minimum-weight edge connecting a vertex in V_i with a vertex in $V \backslash V_i$. We add all edges e_i to the MST we are constructing and join the parts V_i that are now connected. We iterate until all vertices are in the same part. Clearly, the number of iterations is at most $\log_2 |V|$, because the number of parts drops to at most half during each iteration.

5.2.1. Parallel Algorithms for Finding the Minimum Spanning Tree

Borůvka's algorithm seems amenable to parallelization, as the edges e_i can all be found in parallel. Parallelizing the joining of parts is more complicated. Awerbuch and Shiloach [42] have proposed a parallel variant of Borůvka's algorithm that introduces data structures to keep track of the parts of V, and delays the joining of some parts. Due to the delays, the number of iterations may increase, but this increase is shown to be at most $\log_{3/2} |V|$. Each iteration requires constant time when executed by $|E|$ processors on priority-CRCW PRAM. The algorithm assumes that all the edges are of different weights (this does not lessen the generality). Let us describe the algorithm in more detail.

The Awerbuch-Shiloach algorithm [42] is presented in Alg. 10. It uses the array $\mathcal{T}$ to record which edges have been included in the MST. To keep track of the partitioning of V, Alg. 10 uses a union-find data structure F that records the current parent for each vertex. At each moment, F defines a forest of rooted trees, with the vertices v satisfying $v = F[v]$ being the roots. The trees of this forest correspond to the parts in the current partitioning. A rooted tree is called a *star* if its height is at most 1.

The array $\mathcal{A}$ records which edges are *active*. The algorithm iterates as long as any active edges remain. The body of the **while**-loop is multi-threaded, creating one thread for each active edge. The changes a thread makes in common data are not visible to other threads until the next **Synchronize**-statement. In particular, the reads and writes of F by different threads on line 10 do not interfere with each other.

The Awerbuch-Shiloach algorithm joins two parts in the current partitioning of V, or two rooted trees in the forest defined by F only if at least one of them is a star. Computing which vertices belong in stars can be done in constant time with $|V|$ processors. At each iteration of Alg. 10, before executing lines 9 and 16, the algorithm Alg. 11 is invoked and its output is used to check whether the vertex u belongs to a star.

An iteration of Alg. 10 can be seen as a sequence of three *steps*, separated by the **Synchronize**-statements. In the first step, the edges to be added to the tree are selected. For each star with root $r \in V$, the lightest outgoing edge is selected and stored in $\mathcal{W}[r]$. This selection crucially depends on prioritized writing: the writing with the smallest priority will go through. In addition, the star is connected to another tree by changing the F-ancestor of r. In the second step, we break the F-cycles of the length 2 that may

---

Algorithm 10: MST algorithm by Awerbuch and Shiloach

    Data: Connected graph $G = (V, E)$, edge weights ω
    Result: $E' \subseteq E$, so that (V, E') is the MST of G

1 **foreach** $(u, v) \in E$ **do**
2       $\mathcal{T}[\{u, v\}] \leftarrow$ false
3       $\mathcal{A}[(u, v)] \leftarrow$ true
4 **foreach** $v \in V$ **do**
5       $F[v] \leftarrow v$
6       $\mathcal{W}[v] \leftarrow$ NIL
7 **while** $\exists (u, v) \in E : \mathcal{A}[(u, v)]$ **do**
8   **foreach** $(u, v) \in E$ *where* $\mathcal{A}[(u, v)]$ **do**
9   **if** in_star$(u) \wedge F[u] \neq F[v]$ **then**
10              $F[F[u]] \leftarrow F[v]$ <u>with priority</u> $\omega(u, v)$
11              $\mathcal{W}[F[u]] \leftarrow \{u, v\}$ <u>with priority</u> $\omega(u, v)$
12   **Synchronize**
13   **if** $\mathcal{W}[F[u]] = \{u, v\}$ **then** $\mathcal{T}[\{u, v\}] \leftarrow$ true;
14   **if** $u < F[u] \wedge u = F[F[u]]$ **then** $F[u] \leftarrow u$;
15   **Synchronize**
16   **if** in_star(u) **then**
17              $\mathcal{A}[(u, v)] \leftarrow$ false
18   **else**
19              $F[F[u]] \leftarrow F[u]$

20 **return** $\{(u, v) \in E \mid \mathcal{T}[\{u, v\}]\}$

Algorithm 11: Checking for stars in Alg. 10

    Data: A set V, a mapping $F : V \rightarrow V$
    Result: Predicate St on V, indicating which elements of V belong to stars

1 **foreach** $v \in V$ **do** $St[v] \leftarrow$ true;
2 **foreach** $v \in V$ **do**
3   **if** $F[v] \neq F[F[v]]$ **then**
4           $St[v] \leftarrow$ false
5           $St[F[F[v]]] \leftarrow$ false
6 **foreach** $v \in V$ **do** $St[v] \leftarrow St[v] \wedge St[F[v]]$;
7 **return** St

---

have resulted from joining two stars. Independently, we also record the edges added to the MST. In the third step, we decrease the height of F-trees and deactivate the edges attached to a component that is still a star at this step. These edges definitely cannot end up in the MST.

Alg. 11 for checking which vertices belong in stars is straightforward. If the parent and the grandparent of a vertex differ, then neither this vertex nor its grandparent is in a

star. Also, if a parent of some vertex is not in a star, then the same holds for this vertex.

5.2.2. Privacy-Preserving Minimum Spanning Tree

Let the ABB store information about the structure of a graph and the weights of its edges. This means that the size of the graph — the number of vertices n and the number of edges m — is public. We identify the vertices with numbers $1, 2, \ldots, n$. The structure of the graph is private. The ABB stores m pairs of values, each one between 1 and n, giving the endpoints of the edges. For each edge, the ABB also stores its weight.

Because we are working in the ABB model, it is not necessary to specify which parties originally hold which parts of the description of the graph. No matter how they are held, they are first input to the ABB, after which the privacy-preserving MST algorithm is executed. Even more generally, some data about the graph might initially be held by no party at all, but be computed by some previously executed protocol. The strong composability results it provides are a benefit of working in the ABB model.

The algorithms in Sec. 4 can be used to implement Alg. 10 in a privacy-preserving manner, if the dependencies of the control flow from the private data (there is a significant amount of such dependencies, mostly through the array $\mathcal{A}$) are eliminated without penalizing the performance too much. Also, when implementing the algorithm, we would like to minimize the number of calls to prepareRead and prepareWrite algorithms, due to their overheads.

Let us first describe the privacy-preserving algorithm for checking for stars, as Alg. 11 has a relatively simple structure. Its privacy-preserving version is depicted in Alg. 12. It receives the same mapping F as an input, now represented as a private vector $[\![\mathbf{F}]\!]$. As the first step, the protocol finds $F \circ F$ in a privacy-preserving manner, and stores it in $[\![\mathbf{G}]\!]$. To find $[\![\mathbf{G}]\!]$, we have to read from the array $[\![\mathbf{F}]\!]$ according to the indices also stored in $[\![\mathbf{F}]\!]$. This takes place on lines 1–2 of Alg. 12. We can now privately compare whether the parent and grandparent of a vertex are equal (in parallel for all i, as denoted by the use of **foreach**). The result is stored in $[\![\mathbf{b}]\!]$, which serves as an intermediate value for the final result $[\![\overrightarrow{St}]\!]$. After line 3 of Alg. 12, the value of $[\![\mathbf{b}]\!]$ is the same as the value of $\overrightarrow{St}$ after the assignments on lines 1 and 4 of Alg. 11.

Algorithm 12: Privacy-preserving checking for stars

    Data: Private vector $[\![\mathbf{F}]\!]$ of the length n, where $1 \leq F_i \leq n$

    Result: Private predicate $[\![\overrightarrow{St}]\!]$, indicating which elements of $\{1, \ldots, n\}$ belong to
          stars according to $\mathbf{F}$

1   $[\![\sigma]\!] \leftarrow \texttt{prepareRead}([\![\mathbf{F}]\!], n)$

2   $[\![\mathbf{G}]\!] \leftarrow \texttt{applyRead}([\![\mathbf{F}]\!], [\![\sigma]\!])$

3   **foreach** $i \in \{1, \ldots, n\}$ **do** $[\![b_i]\!] \leftarrow [\![F_i]\!] \overset{?}{=} [\![G_i]\!]$;

4   $[\![b_{n+1}]\!] \leftarrow \mathsf{false}$

5   **foreach** $i \in \{1, \ldots, n\}$ **do** $[\![a_i]\!] \leftarrow [\![b_i]\!] ? (n+1) : [\![G_i]\!]$;

6   $[\![\mathbf{b}']\!] \leftarrow \texttt{obliviousWrite}([\![\mathbf{a}]\!], \overrightarrow{\mathsf{false}}, \mathbf{1}, [\![\mathbf{b}]\!])$

7   $[\![\mathbf{p}]\!] \leftarrow \texttt{applyRead}([\![\mathbf{b}']\!], [\![\sigma]\!])$ `// Ignore` b'_{n+1}

8   **foreach** $i \in \{1, \ldots, n\}$ **do** $[\![St_i]\!] \leftarrow [\![b'_i]\!] \wedge [\![p_i]\!]$;

9   **return** $[\![\overrightarrow{St}]\!]$

Next, we prepare for privately performing the assignment on line 4 of Alg. 11. We only want to perform the assignment if $[\![F_i]\!] \neq [\![G_i]\!]$ and hence, the number of assignments we want to perform depends on private data. Alg. 8 presumes that the number of writes is public. We overcome this dependency by assigning to a dummy position each time Alg. 11 would have avoided the assignment on line 5. We let the vector $[\![\mathbf{b}]\!]$ have an extra element at the end and assign to this element for each dummy assignment. On line 5 we compute the indices of vector $[\![\mathbf{b}]\!]$ where false has to be assigned. Here, the operation ? : has the same meaning as in C/C++/Java — it returns its second argument if its first argument is true (1), and its third argument if the first argument is false (0). It can be implemented easily in any ABB: $[\![b]\!] ? [\![x]\!] : [\![y]\!]$ is computed as $[\![b]\!] \cdot ([\![x]\!] - [\![y]\!]) + [\![y]\!]$.

On line 6 of Alg. 12, oblivious write is performed. The arguments to the protocol `obliviousWrite` are in the same order as in the preamble of Alg. 8: the vector of addresses, the vector of values to be written, the vector of writing priorities, and the original array. All arguments can be private values. All public values are assumed to be automatically classified. On line 6, all values to be written are equal to false, as in Alg. 11. Hence, the priorities really do not matter, as we make them all equal to 1 (with the assumption that the priorities for existing elements of $[\![\mathbf{b}]\!]$, output by `compute_priority` on line 6 of Alg. 8, are equal to 0). The result of the writing is a private vector $[\![\mathbf{b'}]\!]$ of the length $n+1$ that is equal to $[\![\mathbf{b}]\!]$ in positions that were not overwritten.

Lines 7 and 8 of Alg. 12 correspond to the assignment on line 6 of Alg. 11. First we compute $St[F[v]]$ for all v (in terms of Alg. 11) by reading from $[\![\mathbf{b}]\!]$ according to the indices in $[\![\mathbf{F}]\!]$. On line 1 we prepared the reading according to these indices. As $[\![\mathbf{F}]\!]$ has not changed in the meantime, this preparation is still valid and can be reused. Hence, we apply `applyRead` to the first n elements of $[\![\mathbf{b'}]\!]$. The conjunction is computed on line 8.

The privacy-preserving MST protocol is given in Alg. 13. To adapt Alg. 10 for execution on an ABB, we have to first simplify its control flow. Fortunately, it appears that it is not necessary to keep track of the edges that are still active. The outcome of Alg. 10 does not change if all edges are assumed to be active all the time. In this case, only the stopping criterion of Alg. 10 (that there are no more active edges) has to be changed to something more suitable. One could keep track of the number of edges already added to the MST, or execute the main loop of the algorithm for a sufficient number of times. We opt for the second solution, as otherwise we could leak something about the graph through the running time of the algorithm.

Alg. 13 first copies around some input data, effectively making the set of edges E symmetric. Throughout this algorithm we assume that x mod m returns a value between 1 and m. On line 2, we negate the weights, as a lower weight of some edge means that it has higher priority of being included in the MST. On lines 4 to 7 we initialize $[\![\mathbf{F}]\!]$, $[\![\mathcal{W}]\!]$ and $[\![\mathcal{T}]\!]$ similarly to Alg. 10. All these vectors have an extra element in order to accommodate dummy assignments. In $[\![\mathcal{W}]\!]$, the value $(m+1)$ corresponds to the value NIL in Alg. 10 — the elements of $[\![\mathcal{W}]\!]$ are used below as addresses to write into $[\![\mathcal{T}]\!]$ and $(m+1)$ indicates a dummy assignment.

Before starting the iterative part of the algorithm, on line 8, we prepare for reading according to the endpoints of edges. The actual reads are performed in each iteration.

The number of iterations of Alg. 13 (line 9) will be sufficient for all edges of the MST to be found. As discussed before, $\lfloor \log_{3/2} n \rfloor$ is a suitable number. All iterations are identical, as the computations do not depend on the sequence number of the current iteration.

Algorithm 13: Privacy-preserving minimum spanning tree

Data: Number of vertices n, number of edges m

Data: Private vector $[\![\mathbf{E}]\!]$ of the length $2m$ (endpoints of edges, i-th edge is (E_i, E_{i+m}))

Data: Private vector $[\![\omega]\!]$ of the length m (edge weights)

Result: Private boolean vector $[\![\mathcal{T}]\!]$ of length m, indicating which edge belongs to the MST

1   **foreach** $i \in \{1, \dots, 2m\}$ **do**

2     $[\![\omega'_i]\!] \leftarrow -[\![\omega_{i \bmod m}]\!]$

3     $[\![E'_i]\!] \leftarrow [\![E_{(i+m) \bmod 2m}]\!]$

4   **foreach** $i \in \{1, \dots, n+1\}$ **do**

5     $[\![F_i]\!] \leftarrow i$

6     $[\![\mathcal{W}_i]\!] \leftarrow (m+1)$

7   **foreach** $i \in \{1, \dots, m+1\}$ **do** $[\![\mathcal{T}_i]\!] \leftarrow$ false;

8   $[\![\sigma^e]\!] \leftarrow \texttt{prepareRead}([\![\mathbf{E}]\!], n)$

9   **for** *iteration_number* $:= 1$ **to** $\lfloor \log_{3/2} n \rfloor$ **do**

10     $[\![\overrightarrow{St}]\!] \leftarrow \texttt{StarCheck}([\![\mathbf{F}]\!])$

11     $[\![\mathbf{F}^e]\!] \leftarrow \texttt{applyRead}([\![\mathbf{F}]\!], [\![\sigma^e]\!])$ // Ignore F_{n+1}

12     $[\![\overrightarrow{St}^e]\!] \leftarrow \texttt{applyRead}([\![\overrightarrow{St}]\!], [\![\sigma^e]\!])$

13   **foreach** $i \in \{1, \dots, m\}$ **do** $[\![d_i]\!] \leftarrow [\![F_i^e]\!] \overset{?}{=} [\![F_{i+m}^e]\!]$;

14   **foreach** $i \in \{1, \dots, 2m\}$ **do**

15       $[\![a_i]\!] \leftarrow [\![St_i^e]\!] \wedge \neg[\![d_{i \bmod m}]\!] \, ? \, [\![F_i^e]\!] : (n+1)$

16     $([\![\sigma^v]\!], [\![\tau^v]\!]) \leftarrow \texttt{prepareWrite}([\![\mathbf{a}]\!], [\![\omega']\!], n+1)$

17     $[\![\mathbf{F}]\!] := \texttt{applyWrite}([\![\sigma^v]\!], [\![\tau^v]\!], [\![\mathbf{E'}]\!], [\![\mathbf{F}]\!])$

18     $[\![\mathcal{W}]\!] := \texttt{applyWrite}([\![\sigma^v]\!], [\![\tau^v]\!], (i \bmod m)_{i=1}^{2m}, [\![\mathcal{W}]\!])$

19     $[\![\mathcal{T}]\!] := \texttt{obliviousWrite}([\![\mathcal{W}]\!], \overrightarrow{\text{true}}, \mathbf{1}, [\![\mathcal{T}]\!])$

20     $[\![\sigma^f]\!] \leftarrow \texttt{prepareRead}([\![\mathbf{F}]\!], n+1)$

21     $[\![\mathbf{G}]\!] \leftarrow \texttt{applyRead}([\![\mathbf{F}]\!], [\![\sigma^f]\!])$

22     $[\![\mathbf{H}]\!] \leftarrow \texttt{applyRead}([\![\mathbf{G}]\!], [\![\sigma^f]\!])$

23   **foreach** $i \in \{1, \dots, n\}$ **do**

24       $[\![c_i^{(1)}]\!] \leftarrow i \overset{?}{=} [\![G_i]\!]$

25       $[\![c_i^{(2)}]\!] \leftarrow i \overset{?}{<} [\![F_i]\!]$

26       $[\![c_i^{(3)}]\!] \leftarrow [\![F_i]\!] \overset{?}{=} [\![H_i]\!] \wedge [\![F_i]\!] \overset{?}{<} [\![G_i]\!]$

27       $[\![F_i]\!] := \begin{cases} i, & \text{if } [\![c_i^{(1)}]\!] \wedge [\![c_i^{(2)}]\!] \\ [\![F_i]\!], & \text{if } [\![c_i^{(1)}]\!] \wedge \neg[\![c_i^{(2)}]\!] \\ [\![F_i]\!], & \text{if } \neg[\![c_i^{(1)}]\!] \wedge [\![c_i^{(3)}]\!] \\ [\![G_i]\!] & \text{if } \neg[\![c_i^{(1)}]\!] \wedge \neg[\![c_i^{(3)}]\!] \end{cases}$

28   **return** $([\![\mathcal{T}_1]\!], \dots, [\![\mathcal{T}_m]\!])$

An iteration starts very similarly to Alg. 10, running the algorithm used to check for stars, and finding for each endpoint u of each edge e the values $F[u]$ and $St[u]$ (in terms of Alg. 10). On line 9 of Alg. 10, a decision is made whether to update an element of $\mathbf{F}$ and an element of $\mathcal{W}$. The same decision is made on lines 13–15 of Alg. 13: we choose the address of the element to update. If the update should be made then this address is $[\![F_i^e]\!]$. Otherwise, it is the dummy address $n+1$. On lines 16–18 the actual update is made. As the writes to both $[\![\mathbf{F}]\!]$ and $[\![\mathcal{W}]\!]$ are according to the same indices $[\![\mathbf{a}]\!]$ and priorities $[\![\omega']\!]$, their preparation phase has to be executed only once. If the write has to be performed, we write the other endpoint of the edge to $\mathbf{F}$ and the index of the edge to $\mathcal{W}$. On line 19 we update $[\![\mathcal{T}]\!]$ similarly to line 13 of Alg. 10.

Compared to Alg. 10, we have redesigned the breaking of F-cycles and decreasing the height of F-trees in order to reduce the number of calls to algorithms in Sec. 4. In Alg. 10, the cycles are broken, which requires data to be read according to the indices in $\mathbf{F}$, and thus the invocation of `prepareRead`, as $\mathbf{F}$ has just been updated. Next, the F-grandparent of each vertex is taken to be its F-parent, which again requires a read according to $\mathbf{F}$ and another invocation of `prepareRead`. Instead, we directly compute what will be the F-grandparent of each vertex after breaking the F-cycles, and take this to be its new F-parent.

For this computation, we need the F-parents of each vertex, which we already have in the vector $\mathbf{F}$. We also need their F-grandparents which we store in $\mathbf{G}$, and F-great-grandparents, which we store in $\mathbf{H}$. We only need a single call to `prepareRead` to find both $[\![\mathbf{G}]\!]$ and $[\![\mathbf{H}]\!]$. After breaking the cycles, the F-grandparent of the vertex i can be either i, F_i or G_i. It is straightforward to see that the computation on lines 24–27 finds the F-grandparent of i and assigns it to $[\![F_i]\!]$. As before, the computations for different vertices are made in parallel. The *case*-construction on line 27 is implemented as a composition of ? : operations.

Finally, we return the private Boolean vector $[\![\mathcal{T}]\!]$ indicating which edges belong to the MST, except for its final dummy element $[\![\mathcal{T}_{m+1}]\!]$.

Alg. 13 is UC-secure for the same reasons as Alg. 7 and Alg. 8. It only applies the operations of ABB, classifies only public constants and declassifies nothing. The amount of work it performs (or the amount of communication it requires for typical ABB implementations) is $O(|E|\log^2|V|)$. Indeed, it performs $O(\log|V|)$ iterations, the complexity of which is dominated by reading and writing preparations requiring $O(\log|E|) = O(\log|V|)$ work. For typical ABB implementations, the round complexity of Alg. 13 is $O(\log^2|V|)$, as each private reading or writing preparation requires $O(\log|V|)$ rounds.

The Awerbuch-Shiloach algorithm (Alg. 10) accesses all of its memory during each of its iterations. Hence, we can say that for this algorithm the overhead of our private data access techniques is only $O(\log m)$.

We have also implemented Alg. 13 (and Alg. 12) on the SHAREMIND SMC platform and tested its performance on the same setup as described in Sec. 4.4. We have varied the number of vertices n and selected the number of edges m based on it and on the most likely applications of our private MST protocol. Our implementation requires less than 100 seconds for a graph with 1,000 vertices and 6,000 edges. For a graph with 200,000 vertices and 1.2 million edges, the execution time is less than nine hours. See [43] for the running times of tasks of other sizes.

References

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

[2] 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.

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

[4] 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.

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

[6] Elaine Shi, T.-H. Hubert Chan, Emil Stefanov, and Mingfei Li. Oblivious RAM with $O((\log N)^3)$ worst-case cost. In Dong Hoon Lee and Xiaoyun Wang, editors, *Advances in Cryptology - ASIACRYPT 2011 - 17th International Conference on the Theory and Application of Cryptology and Information Security, Seoul, South Korea, December 4-8, 2011. Proceedings*, volume 7073 of *Lecture Notes in Computer Science*, pages 197–214. Springer, 2011.

[7] 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.

[8] Emil Stefanov, Marten van Dijk, Elaine Shi, Christopher W. Fletcher, Ling Ren, Xiangyao Yu, and Srinivas Devadas. Path ORAM: an extremely simple oblivious RAM protocol. In Ahmad-Reza Sadeghi, Virgil D. Gligor, and Moti Yung, editors, *ACM Conference on Computer and Communications Security*, pages 299–310. ACM, 2013.

[9] Xiao Shaun Wang, Yan Huang, T.-H. Hubert Chan, Abhi Shelat, and Elaine Shi. SCORAM: oblivious RAM for secure computation. In Gail-Joon Ahn, Moti Yung, and Ninghui Li, editors, *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security, Scottsdale, AZ, USA, November 3-7, 2014*, pages 191–202. ACM, 2014.

[10] 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.

[11] Eyal Kushilevitz and Rafail Ostrovsky. Replication is NOT Needed: SINGLE Database, Computationally-Private Information Retrieval. In *FOCS*, pages 364–373. IEEE Computer Society, 1997.

[12] Helger Lipmaa. First CPIR Protocol with Data-Dependent Computation. In Donghoon Lee and Seokhie Hong, editors, *ICISC*, volume 5984 of *Lecture Notes in Computer Science*, pages 193–210. Springer, 2009.

[13] Eyal Kushilevitz, Steve Lu, and Rafail Ostrovsky. On the (in)security of hash-based oblivious RAM and a new balancing scheme. In Yuval Rabani, editor, *Proceedings of the Twenty-Third Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2012, Kyoto, Japan, January 17-19, 2012*, pages 143–156. SIAM, 2012.

[14] 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.

[15] 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.

[16] Chang Liu, Yan Huang, Elaine Shi, Jonathan Katz, and Michael Hicks. Automating Efficient RAM-Model Secure Computation. In *Proceedings of 2014 IEEE Symposium on Security and Privacy*. IEEE, 2014.

[17] 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/`.

[18] Xiao Shaun Wang, Yan Huang, T-H. Hubert Chan, abhi shelat, and Elaine Shi. Scoram: Oblivious ram for secure computation. Cryptology ePrint Archive, Report 2014/671, 2014. `http://eprint.iacr.org/`, to appear at ACM CCS '14.

[19] Elette Boyle, Kai-Min Chung, and Rafael Pass. Oblivious parallel ram. Cryptology ePrint Archive, Report 2014/594, 2014. `http://eprint.iacr.org/`.

[20] 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.

[21] Yan Huang, David Evans, and Jonathan Katz. Private set intersection: Are garbled circuits better than custom protocols? In *NDSS*. The Internet Society, 2012.

[22] 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.

[23] 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.

[24] Takashi Nishide and Kazuo Ohta. Multiparty computation for interval, equality, and comparison without bit-decomposition protocol. In Tatsuaki Okamoto and Xiaoyun Wang, editors, *Public Key Cryptography*, volume 4450 of *Lecture Notes in Computer Science*, pages 343–360. Springer, 2007.

[25] 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.

[26] Sven Laur, Jan Willemson, and Bingsheng Zhang. Round-Efficient Oblivious Database Manipulation. In *Proceedings of ISC 2011*, pages 262–277, 2011.

[27] Abraham Waksman. A permutation network. *J. ACM*, 15(1):159–163, 1968.

[28] 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.

[29] 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.

[30] 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.

[31] A Eisinberg and G Fedele. On the inversion of the Vandermonde matrix. *Applied mathematics and computation*, 174(2):1384–1397, 2006.

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

[33] 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.

[34] 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.

[35] 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.

[36] Ronald Cramer and Ivan Damgård. Multiparty computation, an introduction. In *Contemporary Cryptology*, Advanced Courses in Mathematics - CRM Barcelona, pages 41–87. Birkhäuser Basel, 2005.

[37] Justin Brickell and Vitaly Shmatikov. Privacy-preserving graph algorithms in the semi-honest model. In Bimal K. Roy, editor, *ASIACRYPT*, volume 3788 of *Lecture Notes in Computer Science*, pages 236–252. Springer, 2005.

[38] Abdelrahaman Aly, Edouard Cuvelier, Sophie Mawet, Olivier Pereira, and Mathieu Van Vyve. Securely solving simple combinatorial graph problems. In Ahmad-Reza Sadeghi, editor, *Financial Cryptography*, volume 7859 of *Lecture Notes in Computer Science*, pages 239–257. Springer, 2013.

[39] 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.

[40] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to Algorithms*, chapter 23.2 The algorithms of Kruskal and Prim, pages 567–574. MIT Press and McGraw-Hill, 2nd edition, 2001.

[41] Jaroslav Nešetřil, Eva Milkovà, and Helena Nešetřilovà. Otakar Borůvka on minimum spanning tree problem; Translation of both the 1926 papers, comments, history. *Discrete Mathematics*, 233(1-3):3–36, 2001.

[42] Baruch Awerbuch and Yossi Shiloach. New connectivity and MSF algorithms for shuffle-exchange network and PRAM. *IEEE Trans. Computers*, 36(10):1258–1263, 1987.

[43] Peeter Laud. Privacy-Preserving Minimum Spanning Trees through Oblivious Parallel RAM for Secure Multiparty Computation. Cryptology ePrint Archive, Report 2014/630, 2014. `http://eprint.iacr.org/`.

Chapter 7
Business Process Engineering and Secure Multiparty Computation

Roberto GUANCIALE [a], Dilian GUROV [a] and Peeter LAUD [b]

[a] *KTH Royal Institute of Technology, Sweden*
[b] *Cybernetica AS, Estonia*

Abstract. In this chapter we use secure multiparty computation (SMC) to enable privacy-preserving engineering of inter-organizational business processes. Business processes often involve structuring the activities of several organizations, for example when several potentially competitive enterprises share their skills to form a temporary alliance.

One of the main obstacles to engineering the processes that govern such collaborations is the perceived threat to the participants' autonomy. In particular, the participants can be reluctant to expose their internal processes or logs, as this knowledge can be analyzed by other participants to reveal sensitive information.

We use SMC techniques to handle two problems in this context: *process fusion* and *log auditing*. *Process fusion* enables the constituents of a collaboration to discover their local views of the inter-organizational workflow, enabling each company to re-shape, optimize and analyze their local flows. *Log auditing* enables a participant that owns a business process to check if the partner's logs match its business process, thus enabling the two partners to discover failures, errors and inefficiencies.

Keywords. Business process engineering, regular languages, finite automata, secure multiparty computation

Introduction

A business process is a collection of structured activities and events that allows an enterprise to achieve a business goal. Business processes are operated by business functional units whose tasks are either performed manually or are computer-based. Unambiguous specification of the enterprise's business processes is necessary to monitor the process performance, to automate the computer-aided tasks and to re-engineer the structure of the enterprise. For this reason several high-level modeling languages have been proposed (e.g. BPMN [1] and EPC [2]). There is a general agreement (see, e.g. [3]) that well-formed business processes correspond to bounded Petri nets (or more specifically, sound workflow nets), and several proposals (e.g. [4]) demonstrate techniques for converting high-level models (such as BPMN) into Petri nets.

Business processes often involve structuring the activities of several organizations. This is the case when quite a few potentially competitive enterprises can share their knowledge and skills to form a temporary alliance, usually called a virtual enterprise

(VE), in order to take advantage of new business opportunities. Virtual enterprises can be part of long-term strategic alliances or short-term collaborations. To effectively manage a virtual enterprise, well-founded support from business process engineering techniques is critical.

One of the main obstacles to such business process engineering is the perceived threat to the participants' autonomy. In particular, the participants can be reluctant to expose their internal processes or logs, as this knowledge can be analyzed by the other participants to reveal sensitive information such as efficiency secrets or weaknesses in responding to market demand. Moreover, the value of confidentiality of business processes is widely recognized.

In this chapter we use SMC techniques to handle two related problems: *VE process fusion* and *log auditing*.

VE process fusion.   The dependencies among the activities of a prospective VE cross the boundaries of the VE constituents. It is, therefore, crucial to allow the VE constituents to discover their local views of the inter-organizational workflow, enabling each company to re-shape, optimize and analyze the possible local flows that are consistent with the processes of the other VE constituents. We refer to this problem as *VE process fusion*. Even if it has been widely investigated, no previous work addresses VE process fusion in the presence of privacy constraints.

Log auditing.   The process of auditing business partners is carried out to determine whether their activities are executed within certain boundaries. Log auditing enables a participant that owns a business process to check if the partner's logs match its business process. By spotting unexpected behavior, log auditing allows two partners to discover failures, errors and inefficiencies.

These two applications are built on top of two more general results: *private intersection of regular languages* and *private lookup* of an element of an array. In particular, we use the universally composable arithmetic black box (ABB) [5] as an abstraction of SMC. Here, our goal is to use the ABB to derive efficient privacy-preserving implementations for the two computational tasks.

The chapter is organized as follows. In Sec. 1 we recall some well-known notions and results from formal languages, automata theory, Petri nets and secure multiparty computation. In Sec. 2 we present our main result, an SMC protocol for private intersection of regular languages. In Sec. 3 and 4 we use the new protocols to handle VE process fusion and log auditing in a privacy-preserving setting.

1. Background

Let us recall several standard notions from the theories of formal languages, automata, Petri nets and secure multiparty computation. For a more detailed introduction and proofs of the propositions, we refer the reader to standard textbooks such as [6] and [7].

1.1. Formal Languages

Let $\mathcal{L}$ be a language over an alphabet Σ. Then for $\Sigma' \subseteq \Sigma$, let $\mathbf{proj}_{\Sigma'}(\mathcal{L})$ denote the *projection* of $\mathcal{L}$ onto the alphabet Σ', defined through deleting in every string of $\mathcal{L}$ the let-

ters not in Σ', and for $\Sigma' \supseteq \Sigma$, let $\mathbf{proj}_{\Sigma'}^{-1}(\mathcal{L})$ denote the *inverse projection* of $\mathcal{L}$ onto the alphabet Σ', defined as the greatest language over Σ' such that its projection onto Σ is $\mathcal{L}$. Our formalization uses the language product operator $\times^L$ introduced in [8] in the context of modular distributed monitoring.

Definition 1 *The* product *of two languages $\mathcal{L}_1$ and $\mathcal{L}_2$ over alphabets Σ_1 and Σ_2, respectively, is the largest language $\mathcal{L} = \mathcal{L}_1 \times^L \mathcal{L}_2$ over $\Sigma_1 \cup \Sigma_2$ such that $\mathbf{proj}_{\Sigma_1}(\mathcal{L}) = \mathcal{L}_1$ and $\mathbf{proj}_{\Sigma_2}(\mathcal{L}) = \mathcal{L}_2$:*

$$\mathcal{L}_1 \times^L \mathcal{L}_2 = \mathbf{proj}_{\Sigma_1 \cup \Sigma_2}^{-1}(\mathcal{L}_1) \cap \mathbf{proj}_{\Sigma_1 \cup \Sigma_2}^{-1}(\mathcal{L}_2) \ .$$

Later we will need the following two properties of language product and projection, established in [9].

Proposition 1 *Let $\mathcal{L}_1$ and $\mathcal{L}_2$ be two languages over alphabets Σ_1 and Σ_2, respectively, and let $\mathcal{L}$ be the language $\mathbf{proj}_{\Sigma_1 \cap \Sigma_2}(\mathcal{L}_1) \cap \mathbf{proj}_{\Sigma_1 \cap \Sigma_2}(\mathcal{L}_2)$. Then:*

$$\mathbf{proj}_{\Sigma_i}(\mathcal{L}_1 \times^L \mathcal{L}_2) = \mathcal{L}_i \cap \mathbf{proj}_{\Sigma_i}^{-1}(\mathcal{L}) \qquad (i \in \{1,2\}) \ .$$

Proposition 2 *Let $\mathcal{L}_1$ and $\mathcal{L}_2$ be two languages over alphabets Σ_1 and Σ_2, respectively, and let $\mathcal{L}$ be as above. Then:*

$$\mathcal{L} = \mathbf{proj}_{\Sigma_1 \cap \Sigma_2}(\mathbf{proj}_{\Sigma_i}(\mathcal{L}_1 \times^L \mathcal{L}_2)) \qquad (i \in \{1,2\}) \ .$$

1.2. Finite Automata

A *deterministic finite automaton* (DFA) is a quintuple $A = (Q, \Sigma, \delta, q_0, F)$, where Q is a finite set of *states*, Σ an *alphabet*, $\delta : Q \times \Sigma \to Q$ a *transition function*, $q_0 \in Q$ an *initial* state, and $F \subseteq Q$ a set of *final* (or *accepting*) states.

The transition function is lifted to strings in the natural fashion, the lifted version is denoted by $\hat{\delta}$. Namely, $\hat{\delta}(q, \varepsilon) = q$ and $\hat{\delta}(q, a \cdot \sigma) = \hat{\delta}(\delta(q,a), \sigma)$, where $a \in \Sigma$ and $\sigma \in \Sigma^*$. A string σ is *accepted* by the automaton A if $\hat{\delta}(q_0, \sigma) \in F$. The set of strings that A accepts is called the *language* of (or *recognized* by) A, and is denoted by $\mathcal{L}(A)$. The class of languages recognized by deterministic finite automata is the class of *regular languages*.

Equivalently, regular languages can be represented by *nondeterministic finite automata* (NFA) that only differ from DFA due to the fact that they have a set of start states (rather than exactly one), and as a codomain of the transition function the set 2^Q (rather than Q), thus, specifying a set of possible transitions from a given state on a given input symbol. Thus, every DFA can also be seen as an NFA. Conversely, every NFA A can be converted into an equivalent DFA (i.e. accepting the same language) by means of the standard *subset* construction. We denote the resulting automaton by $SC(A)$. In general, an NFA can be exponentially more succinct in representing a regular language than a DFA.

Regular languages are closed under reverse and intersection, among other operations, as the standard operations of *reverse* and *product* on finite automata have this effect on their languages. Let A^{-1} denote the reverse automaton of A (obtained simply by reversing the transitions and by swapping the initial and final states), and let $A_1 \times A_2$ denote the product of A_1 and A_2.

1.3. Minimization of Deterministic Finite Automata

For every regular language there is a unique (up to isomorphism) minimal DFA accepting it (i.e. an automaton with a minimum number of states). However, there is no canonical minimal NFA for a given regular language. Let us use $min(\mathcal{L})$ and $min(A)$ to denote the minimal deterministic finite automata that recognize $\mathcal{L}$ and $\mathcal{L}(A)$, respectively.

Standard algorithms for DFA minimization, e.g. those by Moore [10], Hopcroft [11] and Watson [12], are based on partitioning the states of the DFA into equivalence classes that are not distinguishable by any string, and then constructing a *quotient automaton* w.r.t. the equivalence. Notice that the uniqueness result assumes that the automaton has no unreachable states.

The equivalence $\approx$ is computed iteratively, approximating it with a sequence of equivalences capturing the indistinguishability of strings up to a given length. Below we give an account of partition refinement that is suitable for our implementation.

Let $A = (Q, \Sigma, \delta, q_0, F)$ be a DFA. Consider the family of relations $\approx_n \subseteq Q \times Q$ defined as follows:

$$\approx_0 = F^2 \cup (Q \backslash F)^2$$
$$\approx_{i+1} = \approx_i \cap \bigcap_{a \in \Sigma} \delta^{-1}(\approx_i, a)$$

where the inverse of δ is lifted over pairs of states. The family defines a sequence of equivalences, or *partition refinements*, that in no more than $|Q|$ steps stabilizes in $\approx$.

One standard representation of partitions is the *kernel* relation of a mapping. Recall that any mapping $f : A \to B$ induces an equivalence relation κ_f on A, called the kernel of f and defined as $(a_1, a_2) \in \kappa_f$ whenever $f(a_1) = f(a_2)$. Applying this representation to partition refinement of the set of states Q of a given DFA, we define the family of index sets:

$$I_0 = \{0, 1\}$$
$$I_{i+1} = I_i \times [\Sigma \to I_i]$$

to be used as codomains, where $[A \to B]$ denotes the space of (total) mappings from A to B. Next, we define the family of mappings $\rho_n : Q \to I_n$ as follows:

$$\rho_0(q) = \begin{cases} 0 & \text{if } q \in F \\ 1 & \text{if } q \in Q \backslash F \end{cases}$$

$$\rho_{i+1}(q) = (\rho_i(q), \ \lambda a \in \Sigma. \ \rho_i(\delta(q, a))) \ .$$

It is easy to show by mathematical induction on n that $\kappa_{\rho_n} = \approx_n$ under the standard notions of equality on pairs and mappings, i.e. that ρ_n represents the n-th approximant in the above partition refinement sequence.

Notice that for representing a partition on Q, a codomain of the same cardinality as Q suffices. Therefore, we can, at every stage of an actual computation of the approximation sequence, normalize the codomain to the set Q itself. For example, let $\chi_{\rho_n} : rng(\rho_n) \to Q$ be a family of injective mappings from the ranges (i.e. active codomains) of ρ_n into Q. We then obtain a family of mappings $\pi_n : Q \to Q$ defined by $\pi_n = \chi_{\rho_n} \circ \rho_n$ that induce the

same kernel, i.e. $\kappa_{\pi_n} = \kappa_{\rho_n}$. Now, as $\rho_n = \chi_{\rho_n}^{-1} \circ \pi_n$, we can re-express the above partition refinement sequence via the latter mappings as follows:

$$\pi_0(q) = \chi_{\rho_0}(\rho_0(q))$$
$$\pi_{i+1}(q) = \chi_{\rho_{i+1}}(\chi_{\rho_i}^{-1}(\pi_i(q)), \lambda a \in \Sigma. \chi_{\rho_i}^{-1}(\pi_i(\delta(q,a)))) .$$

Therefore, for an actual implementation, it only remains to define the injections χ_{ρ_n} in a suitable fashion. The approach we adopt in our algorithm (see Sec. 2) is to assume an ordering on the two elements of I_0 and on the symbols of the input alphabet Σ. These orderings induce a (lexicographical) ordering on I_n for all $n > 0$. Now, assuming also an ordering on the states Q, we define χ_{ρ_n} as the least order-preserving injection of $rng(\rho_n)$ into Q, i.e. the injection that maps the least element of $rng(\rho_n)$ to the least element of Q, and so on.

1.4. Petri Nets

A *marked labeled Petri net*, or simply a net, is a tuple $N = (P, T, W, \Sigma, \lambda, m_0)$, where P is a set of *places*, T a set of *transitions*, $W : (P \times T) \cup (T \times P) \to \mathbb{N}$ a weighted *incidence function*, Σ an alphabet, $\lambda : T \rightharpoonup \Sigma$ a partial *transition labeling function*, and $m_0 : P \to \mathbb{N}$ an *initial marking*. Transitions whose labeling is undefined are termed *hidden*. Fig. 1 in Sec. 3 depicts the Petri nets that model the business processes of our running example, where black boxes represent hidden transitions and the weight of all the depicted edges is one.

The dynamics (or behavior) of a Petri net is defined in terms of its *markings*, conceptually corresponding to the states of finite automata, and the *firing* of transitions. A marking $m : P \to N$ assigns to each place of the net a number of *tokens*. To define the effect of firing a transition, we need the notions of pre- and postset of a place or transition. For each node $n \in P \cup T$ of the net, the *preset* of x is the set $\bullet x = \{y \mid W(y,x) > 0\}$, while the *postset* of x is $x \bullet = \{y \mid W(x,y) > 0\}$. Given a marking m, a transition t is *enabled* in m if $\forall p \in \bullet t : m(p) \geq W(p,t)$. Then, the enabled transition t can *fire* and produce a *follower* marking m' defined by $\forall p : m'(p) = m(p) - W(p,t) + W(t,p)$. This fact is denoted as $m[t\rangle m'$. Thus, when firing, a transition consumes tokens from its preset of places and produces tokens in its postset of places according to the weights of the incidence function.

Given a sequence of transitions $tt = t_1 \ldots t_n \in T^*$, we write $m_0[tt\rangle m_n$ if there are markings $m_1, \ldots, m_n$ so that $m_{i-1}[t_i\rangle m_i$ for all $i \in 1, \ldots, n$. The set of *reachable markings* of a net N is defined as $M(N) = \{m \mid \exists tt \in T^* : m_0[tt\rangle m\}$. The *reachability graph* of a Petri net N is the graph $G(M(N), E)$ where $(m, m') \in E$ whenever there is a transition t so that $m[t\rangle m'$. A Petri net N is *bounded* if there exists $k \geq 0$ so that $m(p) \leq k$ for every place p and reachable marking $m \in M(N)$. Clearly, if a Petri net is bounded then its reachability graph is finite.

With each execution (i.e. fireable transition sequence) of a Petri net, we can associate a *trace*, which is a word in Σ^*. If $m_o[t_1 \ldots t_n\rangle m$ then $\lambda(t_1) : \cdots : \lambda(t_n)$ is a trace of the Petri net. If the label associated with a transition in the sequence is undefined, then the associated word is the empty one. The set of all traces of a Petri net N is called the *language* of N and is denoted by $\mathcal{L}(N)$.

Definition 2 *Two labeled Petri nets N_1 and N_2 are* trace equivalent*, denoted $N_1 \sim N_2$, whenever $\mathcal{L}(N_1) = \mathcal{L}(N_2)$.*

The following result allows the reachability graph of a Petri net to be viewed as a nondeterminstic finite automaton accepting the same language.

Proposition 3 *Let N be a bounded labeled Petri net, and let $NFA(N)$ be the nondeterminstic finite automaton $(M(N), \Sigma, \delta, m_0, M(N))$ so that $m' \in \delta(m, a)$ whenever $m[t\rangle m'$ and $\lambda(t) = a$. Then $\mathcal{L}(N) = \mathcal{L}(NFA(N))$.*

There are several proposals for synthesizing (up to language equivalence) a labeled Petri net from an arbitrary DFA. Hereafter, we use *Reg* to represent the algorithm based on regions [13].

Proposition 4 *Let A be a DFA and $N = Reg(A)$. Then $\mathcal{L}(N) = \mathcal{L}(A)$.*

We will now lift the notions of product and projection to the domain of Petri nets. The *product* of two labeled Petri nets is a labeled Petri net that synchronizes the transitions of the original two nets that carry the same label. The Petri net product offers a natural way to model complex systems by composing elementary components. We use the product construction introduced in [7]. Formally, let $N_1 = (P_1, T_1, W_1, \Sigma_1, \lambda_1, m_1)$ and $N_2 = (P_2, T_2, W_2, \Sigma_2, \lambda_2, m_2)$ be two labeled nets so that $\Sigma = \Sigma_1 \cap \Sigma_2$ and P_1, P_2, T_1, T_2 and Σ are pairwise disjoint. Their product $N_1 \times N_2$ is defined as the net $(P, T_1 \cup T_2, W, \Sigma_1 \cup \Sigma_2, \lambda, m)$, where:

- $P = P_1 \cup P_2 \cup \{\star\} \cup \Sigma$
- $m = m_1 \uplus m_2 \uplus \{\star \to 1\}$
- $W = W_1 \uplus W_2 \uplus$
  $$\uplus_{t \in T_1. \lambda_1(t) \in \Sigma} \{(t, \star) \to 1, (\lambda_2(t), t) \to 1\} \uplus$$
  $$\uplus_{t \in T_2. \lambda_2(t) \in \Sigma} \{(\star, t) \to 1, (t, \lambda_1(t)) \to 1\}$$
- $\lambda(t) = \lambda_1(t)$ if $t \in T_1$ and $\lambda(t) = \lambda_2(t)$ if $t \in T_2$ and $\lambda_2(t) \notin \Sigma$

We add a new place $\star$ (having one token in the initial marking) and, for each symbol $l \in \Sigma$, a new place l (having no tokens in the initial marking). For each transition of N_1 that must be synchronized, we add $\star$ to its postset and l to its preset. Conversely, for each transition of N_2 that must be synchronized, we add l to its postset and $\star$ to its preset. Finally, every transition of N_2 that must be synchronized, is hidden. This arrangement forces each activation of a transition of N_2 (having a label in Σ) to be necessarily followed by the activation of a transition of N_1 having the same label. Informally, each place l tracks the last symbol fired by N_2 and forces N_1 to repeat this symbol before returning the token to $\star$.

Proposition 5 *Let N_1 and N_2 be two labeled Petri nets. Then, $\mathcal{L}(N_1 \times N_2) = \mathcal{L}(N_1) \times_L \mathcal{L}(N_2)$.*

The *projection* of a labeled Petri net on an alphabet $\Sigma' \subseteq \Sigma$ is obtained by hiding all transitions that are labeled with symbols not contained in Σ'. That is, $\mathbf{Proj}_{\Sigma'}((P, T, W, \Sigma, \lambda, m_0)) = (P, T, W, \Sigma', \lambda', m_0)$, where $\lambda'(t) = \lambda(t)$ if $\lambda(t) \in \Sigma'$ and is undefined otherwise.

Proposition 6 *Let N be a labeled Petri net, and let $\Sigma' \subseteq \Sigma$. We then have $\mathcal{L}(\mathbf{Proj}_{\Sigma'}(N)) = \mathbf{Proj}_{\Sigma'}(\mathcal{L}(N))$.*

1.5. Arithmetic Black Boxes

A highly suitable abstraction of secure multiparty computation (SMC) is the universally composable arithmetic black box (ABB) [5] (see Chapter 2), the use of which allows very simple security proofs for higher-level SMC applications.

An arithmetic black box is an ideal functionality $\mathcal{F}_{ABB}$. It allows its users to store and retrieve values securely, and to perform computations on them. When a party sends the command store(v) to $\mathcal{F}_{ABB}$, 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 sends the command retrieve(h) to $\mathcal{F}_{ABB}$, 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 compute($op; h_1, \ldots, h_k; params$) to $\mathcal{F}_{ABB}$, 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.

Typically, the ABB performs computations with values v from some ring $\mathbb{R}$. 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 into a protocol in the implementation π_{ABB}).

2. Private Regular Language Intersection

In [9] we consider two mutually distrustful parties, each of whom knows a regular language and wishes to obtain their combined language. Both parties are reluctant to reveal any information about their own languages that is not strictly deducible from the combined result. Moreover, we assume that there is no trusted third party. The combined language we consider here is the *intersection* of the two languages. An intersection of formal languages is a primitive used for a wide range of applications. Cases of usage include: (i) enterprises that are building a cooperation and want to establish a cross-organizational business process (see Sec. 3), (ii) competitive service providers that collaborate to detect intrusions, and (iii) agencies that intersect their compressed databases.

In some restricted cases, private language intersection can be achieved using existing protocols to compute private set intersection (see e.g. [14]). However, this requires the input languages to respect two constraints: the words of the two languages have to be subsets of a given domain of finite size, and the languages themselves must be finite. In this chapter, we go beyond finite sets and propose a technique for privacy-preserving intersection of regular languages. Our approach is based on: (i) representing the input languages as finite automata, (ii) a well-known result that for every regular language there is a unique (up to isomorphism) DFA accepting it, and (iii) designing privacy-preserving versions of some classic algorithms on automata: product, trimming and minimization.

Let two parties P_1 and P_2 hold their respective regular languages $\mathcal{L}_1$ and $\mathcal{L}_2$, both defined over a common and public alphabet Σ. Private regular language intersection allows the two parties to compute the regular language $\mathcal{L} = \mathcal{L}_1 \cap \mathcal{L}_2$ in a privacy-preserving manner, that is, without leaking elements of $\mathcal{L}_1 \setminus \mathcal{L}$ or $\mathcal{L}_2 \setminus \mathcal{L}$.

Here we assume a semi-honest adversary model, where every involved agent correctly follows the protocol, but might also record intermediate messages to infer additional information.

Regular languages can be infinite, so we need to choose a finite representation. We use finite automata, though another obvious choice is to use regular expressions. The two involved parties represent their languages $\mathcal{L}_1$ and $\mathcal{L}_2$ by means of two (deterministic or nondeterministic) automata A_1 and A_2 recognizing these; formally, $\mathcal{L}(A_1) = \mathcal{L}_1$ and $\mathcal{L}(A_2) = \mathcal{L}_2$. To make our functionality deterministic, we must yield a canonical representative automaton recognizing the language $\mathcal{L} = \mathcal{L}_1 \cap \mathcal{L}_2$. As such a representative we choose the unique (up to isomorphism) minimal DFA A recognizing $\mathcal{L}$ (see Subsec. 1.3). Finally, in order to enable standard SMC constructions to operate on the DFA, we relax the privacy constraint by considering as public information the upper limit on the number of states of the two input automata. That is, it is publicly known that the languages $\mathcal{L}_1$ and $\mathcal{L}_2$ are in the set of languages accepted by deterministic finite automata that respect this limit (see [15] for the analysis of the size of this set).

As shown in Sec. 1.3, the automaton $min(\mathcal{L}_1 \cap \mathcal{L}_2)$ can be obtained by using four composable sub-protocols to: (i) compute the product of the automata A_1 and A_2, (ii) trim the non-reachable states from the resulting automata, (iii) refine the partitions, and (iv) compute the quotient automaton. Here, the most costly step is partition refinement, which requires $|A_1| \cdot |A_2|$ iterations of a sub-protocol that, starting from a partition π_i and the transition function $\delta = \delta(A_1) \times \delta(A_2)$, yields the new partition π_{i+1}.

As we are not interested in the identity of the states of the involved automata, let us assume that the automata states Q are represented through the natural numbers of the interval $[1 \ldots |Q|]$. Each partition π_i, as well as the transition function associated to a symbol (i.e. $\delta_a(q) = \delta(q,a)$), can be represented as mappings of type $Q \to Q$. Thus, each partition refinement step can be implemented as follows:

1. For each state $q \in Q$ and symbol $a \in \Sigma$, compute the mapping composition $x_a(q) = \pi_i(\delta_a(q))$.
2. Generate π_{i+1} so that $\pi_{i+1}(q) = \pi_{i+1}(q')$ whenever $\pi_i(q) = \pi_i(q')$ and $x_a(q) = x_a(q')$ for all $a \in \Sigma$.

2.1. Composition of Mappings

Mappings representing symbol transition functions can be represented as *graphs*, or matrices of the size $|Q| \times |Q|$ with entries in $\{0,1\}$. The current partition can be represented as a vector of the length $|Q|$. Hence, the mappings x_a can be computed as products of a matrix with a vector, utilizing a protocol for this operation.

Alternatively, certain protocol sets for secure multiparty computation allow the symbol transition functions δ_a to be represented in a manner that preserves their privacy. Efficient protocols exist for computing $\delta_a \circ \pi$ from a mapping π, where both π and $\delta_a \circ \pi$ are represented as vectors of the length $|Q|$. For example, this is the case for the *additive three-party* protocol set used in SHAREMIND [16,17]. If δ_a is a permutation of Q, then we can use the protocols described in [18]. If δ_a is not a permutation, then we can use protocols for oblivious extended permutations [19,20]. Related protocols are also described in Chapter 6.

2.2. Generation of New Partition IDs

The second step of partition refinement can be achieved by employing any composable protocol that implements the following straightforward algorithm:

1. Initialize the new mapping as the identity mapping $\forall q.\pi_{i+1}(q) = q$.
2. For each state q, update the mapping of every other state q' as: $\pi_{i+1}(q') = \pi_{i+1}(q)$ whenever $\pi_i(q') = \pi_i(q)$ and $\bigwedge_a x_a(q') = x_a(q)$.

The representation of Q through the natural numbers proposed above provides us with a natural lexicographical ordering on $(\pi_i(q), \lambda a.x_a(q))$. This allows the second step of the partition refinement to be achieved by composing the following sub-steps:

1. Generate a matrix M of $|Q| \times (|\Sigma| + 3)$ elements, so that $M[q,0] = 0$, $M[q,1] = q$, $M[q,2] = \pi_i(q)$ and $M[q,3+a] = x_a(q)$.
2. Sort the matrix lexicographically on the last $|\Sigma| + 1$ columns.
3. Iterate the matrix and update the zeroth column with a counter; the counter is increased if at least one of the last $|\Sigma| + 1$ columns of the current row differs from the corresponding value of the previous row.
4. Invert the sort, i.e. sort the matrix on the first column again; alternatively, the additive three-party protocol set used by SHAREMIND allows the sorting permutation to be remembered in a privacy-preserving manner and its inverse to be applied effciently to the rows of the matrix.
5. Set $\pi_{i+1}(q) = M[q,0]$.

2.3. Discussion of Alternative Approaches

To justify further our approach based on the minimal DFA, we briefly discuss below why some common strategies for speeding up the private computation of a function f are not applicable as alternative solutions to the problem addressed here (i.e. f yielding the intersection of two regular languages).

Randomization of the result. If $f = g \circ f'$, a common approach is to first privately compute f', and then privately and randomly construct a public result that is g-equivalent to the output of f' so that the distribution of the result is uniform. Any automaton A recognizing the intersection language $\mathcal{L}$ is equivalent to the automaton $A_1 \times A_2$. This observation leads to a straightforward strategy for private regular language intersection: (i) use a composable protocol to compute $f' = A_1 \times A_2$, and then (ii) build an efficient composable protocol that randomly constructs an equivalent automaton (to $A_1 \times A_2$). However, there is no efficient way to generate a uniform distribution over all automata equivalent to a given one. Instead, we essentially construct an equivalent automaton with a degenerate distribution, namely the minimal deterministic one.

Problem transformation. Another common approach is to generate a new "random" problem f' by transforming f in a way that (i) there is no relation between the random distribution of f' and the original problem, and (ii) it is possible to compute the solution of f knowing the solution of f' and the applied transformation. This allows (i) to generate and apply the random transformation in a privacy-preserving way, (ii) to offload the computation of f' to a non-trusted third party, and (iii) to compute in a privacy-preserving way the expected output from the solution of f' and the generated transformation. This

approach has been used to handle problems (see e.g. [21]) where the inputs are repre-sented as matrices and the transformation can be easily generated as random invertible matrices. Again, however, there is no mechanism to generate a random and invertible transformation such that, starting from an arbitrary input automaton, the distribution of the resulting transformed automaton is uniform.

Generation of a combinatorial problem.   This approach has been used successfully for scientific computations (see e.g. [22]). The idea is as follows. First, one of the two parties decomposes the problem into a set of n subtasks, so that knowing the solution of each subtask would allow the party to discover the intended private output only. Second, the same party generates $m-1$ other (random) problems and the corresponding n subtasks. Next, the other party locally solves $m \cdot n$ subtasks, and finally, the two parties execute an n-out-of-$(m \cdot n)$ oblivious transfer, allowing the first party to compute the intended private output. The security of the approach depends on the fact that if the $m \cdot n$ subtasks are indistinguishable, then the attacker needs to solve a combinatorial problem $\binom{m \cdot n}{n}$ to discover the original inputs. In the context of private language intersection this amounts to performing the following steps.

- The first party generates $A_1^{1,1} \ldots A_1^{1,n}$, so that $\mathcal{L}_1 = \cup_i \mathcal{L}(A_1^{1,i})$.
- The first party generates $\mathcal{L}_1^2 \ldots \mathcal{L}_1^m$ and the automata $A_1^{k,n}$, so that $\forall k. \mathcal{L}_1^k = \cup_i \mathcal{L}(A_1^{k,i})$.
- The second party solves the problems $\mathcal{L}^{k,i} = \mathcal{L}(A_1^{k,i}) \cap \mathcal{L}_2$.
- The two parties use oblivious transfer to allow the first party to obtain the solutions $\mathcal{L}^{1,1} \ldots \mathcal{L}^{1,n}$.
- The first party computes the final result $\mathcal{L} = \cup_i \mathcal{L}^{1,i}$.

Again, the main obstacle to applying this approach to our problem is that there is no mechanism to obtain a uniform distribution of random regular languages (or automata having an arbitrary number of states). The existing random generators [23] that pick one automaton from a set of automata that have up to a fixed number of states cannot be used here. In fact, this prevents us from generating $m \cdot n$ indistinguishable subtasks while satisfying the constraints (i) and (ii) of problem transformation, thus allowing the attacker to avoid solving the combinatorial problem.

Incremental construction.   Yet another common approach is to compute the result of a function f by composing simpler functionalities that incrementally approximate the result, so that each intermediate result is part of the final output. For example, starting from an initial empty approximation $A_0 = \emptyset$, the union of two finite sets P_1 and P_2 can be incrementally built by iterating a protocol that yields the minimum of two elements: $A_i = A_{i-1} \cup min(min(P_1 \setminus A_i), min(P_2 \setminus A_i))$. Such an approach has been used to privately compute all-pairs shortest distances [24] and minimum spanning trees. However, we did not find in the literature such a construction to compute the minimal automaton A starting from A_1 and A_2, with the exception of the dynamic minimization algorithms that can be used only if one of the two input automata recognizes a finite language.

2.4. Implementation

The implementation of Moore's algorithm consists of the following three steps: (i) a product construction, (ii) determining the reachable states, and (iii) determining the

equivalent states. Steps (ii) and (iii) are independent of each other, as we do not want to leak the size of the automaton resulting from one of these steps. We have used the SHAREMIND platform and its three-party additively secret shared protection domain for secure multiparty computations, offering efficient vectorized arithmetic operations, as well as array permutations and oblivious reads (see Chapter 6). In this set-up, there are three *computing nodes*, into which the i-th *client party* holding $A_i = (Q_i, \Sigma, \delta_i, q_{0i}, F_i)$ uploads its automaton in shared form [25]. The computing nodes find a shared representation of the minimization of $A = A_1 \times A_2$ and reveal it to the clients. The platform is secure against a semi-honest adversary that can adaptively corrupt one of the computing nodes.

In our implementation, the size of the task (i.e. the values $|Q_1|$, $|Q_2|$ and $|\Sigma|$) is public. As each party can permute its Q_i, q_{01} and q_{02} are public without loss of generality. The set F_i is represented as a bit-vector χ_i of length $|Q_i|$. This vector is uploaded to computing nodes in shared form $[\![\chi_i]\!] = ([\![\chi_i]\!]_1, [\![\chi_i]\!]_2, [\![\chi_i]\!]_3)$ with $[\![\chi_i]\!]_1 \oplus [\![\chi_i]\!]_2 \oplus [\![\chi_i]\!]_3 = \chi_i$. The symbol transition functions $\delta_{i,a} = \delta_i(\cdot, a) : Q_i \to Q_i$ are represented as discussed in Sec. 2.1: shared mappings $[\![\delta_{i,a}]\!]$, allowing $\mathsf{rearr}([\![\delta_{i,a}]\!], [\![\mathbf{x}]\!]) = ([\![x_{\delta_{i,a}(1)}]\!], \ldots, [\![x_{\delta_{i,a}(|Q_i|)}]\!])$ to be computed efficiently from $[\![\mathbf{x}]\!] = ([\![x_1]\!], \ldots, [\![x_{|Q_i|}]\!])$. Here rearr is the oblivious parallel reading operation described in Chapter 6, Sec. 4.1.

Step (i) of our intersection algorithm is essentially a no-op, as computing $F = F_1 \times F_2$ is trivial and there is no efficient way to compute the products $\delta_a = \delta_{1,a} \times \delta_{2,a}$. Instead, to compute $\mathsf{rearr}([\![\delta_a]\!], [\![\mathbf{x}]\!])$ in steps (ii) and (iii) for $\mathbf{x}$ indexed with elements of $Q = Q_1 \times Q_2$, we organize $[\![\mathbf{x}]\!]$ as an array with $|Q_1|$ rows and $|Q_2|$ columns. We will then apply $\mathsf{rearr}([\![\delta_{1,a}]\!], \cdot)$ to the rows of $\mathbf{x}$, and $\mathsf{rearr}([\![\delta_{2,a}]\!], \cdot)$ to the columns of the result.

The implementation of Moore's algorithm in step (iii) of our intersection algorithm is iterative, following Sec. 2.2. One iteration, constructing the shared partition $[\![\pi_{i+1}]\!]$ from $[\![\pi_i]\!]$, is given in Alg. 1. Here $[\![\iota]\!]$ is a vector of the length $|Q|$, with $\iota_i = i$. All vectors are considered to be column vectors. All **foreach**-loops are parallelized.

Algorithm 1: One iteration of Moore's algorithm

Data: Private symbol transition functions $[\![\delta_a]\!]$ for all $a \in \Sigma$
Data: Private partition $[\![\pi_i]\!]$ from the previous iteration
Result: Refined private partition $[\![\pi_{i+1}]\!]$
foreach $a \in \Sigma$ **do** $[\![\pi_{i,a}]\!] := \mathsf{rearr}([\![\delta_a]\!], [\![\pi_i]\!])$;
$[\![\hat{\Pi}_i]\!] := ([\![\pi]\!] | [\![\pi_{i,a_1}]\!] | \cdots | [\![\pi_{i,a_{|\Sigma|}}]\!] | [\![\iota]\!])$
$([\![\Pi_i]\!], [\![\sigma]\!]) := \mathsf{sort\_rows}([\![\hat{\Pi}_i]\!])$
$[\![c_1]\!] := 1$
foreach $j \in \{2, \ldots, |Q|\}$ **do**
    foreach $a \in \Sigma$ **do**
        $[\![c_{j,a}]\!] := ([\![\Pi_i[j,a]]\!] \neq [\![\Pi_i[j-1,a]]\!])$
    $[\![c_j]\!] := ([\![\Pi_i[j,0]]\!] \neq [\![\Pi_i[j-1,0]]\!]) \vee \bigvee_{a \in \Sigma} [\![c_{j,a}]\!]$
foreach $j \in |Q|$ **do** $[\![\hat{\pi}_{i+1,j}]\!] := \sum_{k=1}^{j} [\![c_k]\!]$;
$[\![\pi_{i+1}]\!] := \mathsf{unshuffle}([\![\sigma]\!], [\![\hat{\pi}_{i+1}]\!])$

This algorithm uses the following functionality from the SHAREMIND platform. The function $\mathsf{sort\_rows}$ returns both its arguments (rows sorted lexicographically), and a

shared permutation $[\![\sigma]\!]$ that brought the rows of the original matrix to the sorted order. The function is not completely privacy-preserving, as it leaks information on specific rows that were equal to other rows. To prevent this leakage, we have used the extra column ι. Boolean values $\{\mathsf{true}, \mathsf{false}\}$ are identified with integers $\{1,0\}$. The large fan-in disjunction is implemented by first adding up the inputs and then comparing the result to 0 [18]. The function unshuffle applies the inverse of σ to the vector $\hat{\pi}_{i+1}$, thereby undoing the permutation from sorting.

The iterations have to be run until the number of parts in π remains the same. Making the results of sameness checks public leaks the number of iterations. This number can be derived from the minimized automaton and hence, the leak is acceptable if the final automaton is public. Otherwise, an application-specific bound must be provided by the parties holding A_1 and A_2, as the worst-case bound is almost $|Q| = |Q_1| \cdot |Q_2|$.

In step (ii), reachable states can be computed in various ways. One can find the reflexive-transitive closure of the adjacency matrix M of A. This requires $\log D$ multiplications of matrices of the size $|Q| \cdot |Q|$, where $D \le |Q|$ is an upper bound on the distance of the states of $A_1 \times A_2$ from the starting state. Using SHAREMIND, one multiplication of $n \times n$ matrices requires $O(n^3)$ local work, but only $O(n^2)$ communication. Still, for larger values of n, this is too much. Instead, we find the reachable states iteratively: let $R_0 = \{q_0\}$ and $R_{i+1} = R_i \cup R'_i$, where $R'_i = \{q \in Q \,|\, \exists q' \in R_i, a \in \Sigma : \delta(q',a) = q\}$ (represented as 0/1-vectors $\mathbf{r}_i$). If the diameter of A is small, and we have a good (application-specific) upper bound D' for it, then this approach may require much less computational effort.

The vector $\mathbf{r}'_i$ can be found from $\mathbf{r}_i$ by multiplying it with the the matrix M. Using SHAREMIND, this requires $O(n^2)$ communication and $O(n^2 D')$ local computation due to the size of M for the computation of $\mathbf{r}_{D'}$ from $\mathbf{r}_0$. With oblivious parallel reads, we can bring both costs down to $O(nD')$.

When computing $\mathbf{r}'_i$ from $\mathbf{r}_i$, we have to apply $[\![\delta_a]\!]$ to $\mathbf{r}_i$ in the opposite direction compared to step (iii): $r'_{ij} = 1$ iff $r_{ik} = 1$ and $\delta(q_k, a) = q_j$ for some k and a. Using the techniques described in Chapter 6, we can implement the function $[\![\mathbf{y}]\!] = \mathsf{rearr}^{-1}([\![f]\!], [\![\mathbf{x}]\!])$, satisfying $y_i = \sum_{j \in f^{-1}(i)} x_j$ in a privacy-preserving manner and access it from applications built on top of SHAREMIND. This function, with performance equal to rearr, suffices for finding reachable states. Alg. 2 gives the computation of $[\![\mathbf{r}_{i+1}]\!]$ from $[\![\mathbf{r}_i]\!]$.

Algorithm 2: One iteration in finding reachable states

Data: Private symbol transition functions $[\![\delta_a]\!]$ for all $a \in \Sigma$
Data: Private Boolean vector $[\![\mathbf{r}_i]\!]$ indicating the states that already have been found to be reachable
Result: Vector $[\![\mathbf{r}_{i+1}]\!]$ updating $[\![\mathbf{r}_i]\!]$ with additional states have found to be reachable
foreach $a \in \Sigma$ **do** $[\![\mathbf{s}_a]\!] := \mathsf{rearr}^{-1}([\![\delta_a]\!], [\![\mathbf{r}_i]\!])$;
foreach $j \in \{1, \ldots, |Q|\}$ **do**
$\quad [\![s_j]\!] := \sum_{a \in \Sigma} [\![s_{a,j}]\!]$
$\quad [\![r_{i+1,j}]\!] := ([\![s_j]\!] \neq 0) \vee [\![r_{i,j}]\!]$

Our SHAREMIND cluster consists of three computers with 48 GB of RAM and a 12-core 3 GHz CPU with Hyper Threading running Linux (kernel v.3.2.0-3-amd64), connected by an Ethernet local area network with a link speed of 1 Gbps. On this cluster, we have benchmarked the execution time of Alg. 1 and 2. If $|\Sigma| = 10$, $|Q_1| = |Q_2| = 100$, then one iteration in determining reachable states (Alg. 2) requires approximately 0.9 s, while one iteration in Moore's algorithm (Alg. 1) requires approximately 4.5 s. For $|\Sigma| = 10$, $|Q_1| = |Q_2| = 300$, these times are 6.2 s and 40 s, respectively. In the worst case, algorithms converge in $|Q_1| \cdot |Q_2|$ iterations. While these are the execution times of single iterations, our experiments show that privacy-preserving minimization of DFA is feasible even for automata with 100,000 states, if the application producing these automata allows us to give reasonable bounds on the number of iterations necessary for these algorithms to converge.

3. Virtual Enterprise Process Fusion

In [26] we investigate a mechanism for establishing cross-organizational business processes, or more precisely, for identifying in case of each participant of a virtual enterprise (VE), which operations can be performed locally. In other words, we need to compute the contributing subset of the existing local business process that is consistent with the processes of the other VE constituents. We refer to this problem as *VE process fusion*.

Here we consider two mutually distrustful parties, each of whom follows a local business process and wishes to compute their local view of the VE process, assuming no trusted third party is available. The VE process is modeled as the synchronous composition of the participant work-flows, and each participant's local view is represented by a process that is trace equivalent to the VE process up to transitions that are not observable by the participant itself. The two parties are reluctant to reveal any information about their own business process that is not strictly deducible from the local view of the other party.

Assume two enterprises a and b with their own business processes that cooperate to build a VE. For each of the two enterprises we are given a local alphabet, Σ_a and Σ_b, respectively. Each enterprise also owns a local business process representing all the possible executions allowed, which is given as a bounded labeled Petri net (N_a and N_b, respectively) that is defined over the corresponding local alphabet. For example, consider the Petri nets depicted in Fig. 1a and 1b. The symbols of the alphabets can represent various types of actions or events:

1. An internal task of the enterprises (the boxes labeled E, D, G and H, standing for tasks such as the packaging of goods and the like).
2. An interaction between the two enterprises (A and B, representing tasks such as the exchange of electronic documents).
3. An event observed by one of the enterprises only (P, the receipt of a payment).
4. An event observed by both enterprises (C, the departure of a carrier from the harbor).
5. A silent event (black boxes, usually used to simplify net structure).

The problem of *VE process fusion* can be defined as computing the mapping:

$$N_i \mapsto N_i' \qquad (i \in \{a, b\})$$

(a) N_a

(b) N_b (c) N_b'
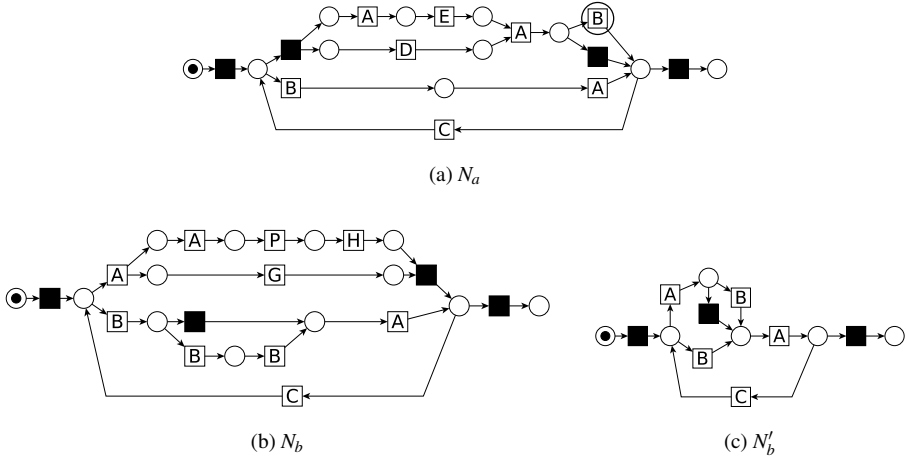
Figure 1. Business processes modeled with Petri nets
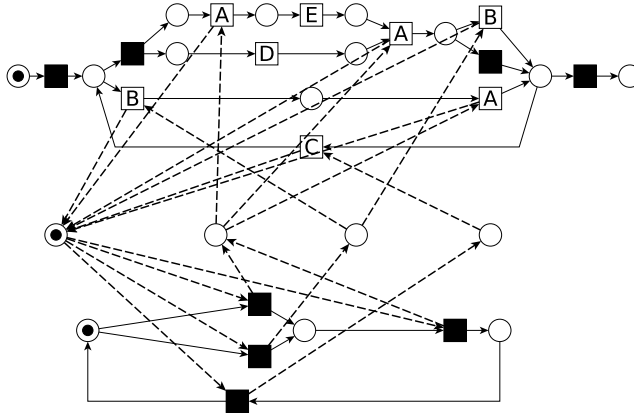
Figure 2. The local view of *a* after its fusion with *b*

, where $N_i' \sim \mathbf{proj}_{\Sigma_i}(N_a \times N_b)$. This describes the problem of each participant computing from its local business process N_i a new process, consistent with the global VE business process represented by the synchronous composition. If there are no privacy constraints, then implementing VE process fusion is straightforward: the two parties can simply (1) exchange their Petri nets, (2) compute the product Petri net $N_a \times N_b$, and finally (3) project $N_a \times N_b$ on their respective alphabet Σ_i.

To illustrate VE process fusion, consider the following running example. Let *a* and *b* be two enterprises, with business processes as shown in Fig. 1a and 1b, respectively. Intuitively, when enterprise *a* is fused with enterprise *b*, its business process must be updated so as to satisfy the partner's constraints. For instance, an analysis of the fusion suggested above will reveal that the encircled activity *B* in Fig. 1a can no longer be executed after the fusion.

Here we are interested in preserving the participants' privacy. In particular, we wish the two participants to obtain N_a' and N_b', respectively, without being able to learn about

the other enterprise's processes more than what can be deduced from their own process (i.e. the *private input*) and the obtained result (i.e. the *private output*). Apart from the processes, we also consider the alphabet differences to be private. That is, we consider public just the common alphabet $\Sigma_a \cap \Sigma_b$ (i.e. the events of type 2 and 4). For example, regardless of whether enterprise b owns the business process N_b or N'_b from Fig. 1, the sub-process of N_a that is consistent with the observable partner's constraints is one and the same, namely the Petri net shown in Fig. 2 (presented for convenience as the superimposition of the observable external constraints to the original process of enterprise a). Therefore, the mechanism used to implement process fusion should not allow party a to distinguish between N_b and N'_b or any other partner process that gives rise to the same local view of a.

To compute the VE process fusion without compromising the participants' privacy we use a SMC protocol for private intersection of regular languages described in Sec. 2. Alg. 3 gives the protocol executed by the participant $i \in \{a, b\}$.

Algorithm 3: Protocol for privacy-preserving process fusion

Data: Participant identity $i \in \{a, b\}$
Data: Business process of participant i, represented as Petri Net N_i
Result: Local process of participant i, consistent with the global VE business
process

1   $N_i^p := \mathbf{proj}_{\Sigma_a \cap \Sigma_b}(N_i)$ // project the private net on the common alphabet
2   $A_i^p := NFA(N_i^p)$ // obtain the equivalent nondeterministic automaton
           // by computing the reachability graph of the net
3   $A_i^d := SC(A_i^p)$ // determinize the automaton
4   send A_i^d to the privacy-preserving language intersection protocol described in
Sec. 2
5   receive $A := SMC_\times(A_a^d, A_b^d)$
6   $N := Reg(A)$ // synthesize the corresponding Petri net
7   **return** $N_i \times N$ // apply the external constraints to the initial net

---

Our protocol is built on two main ideas. As the input Petri nets are bound, their reachability graphs are finite and can be used to compute the deterministic finite automata representing the net languages (steps 1, 2 and 3). Moreover, as proved by Thm. 8, disclosing the intermediate language (step 5) does not leak any information that cannot be directly deduced from private output.

The following result establishes the *correctness* of the protocol, in the sense that the resulting network correctly represents the executions obtained by the synchronous product of the two business processes after hiding all internal transitions of the other participant.

Theorem 7 $Protocol_i(N_a, N_b) \sim \mathbf{proj}_{\Sigma_i}(N_a \times N_b)$.

The next result shows that the protocol preserves the participants' *privacy*, namely that the two participants are not able to learn more about the other enterprise's processes than can be deduced from their own processes and private output.
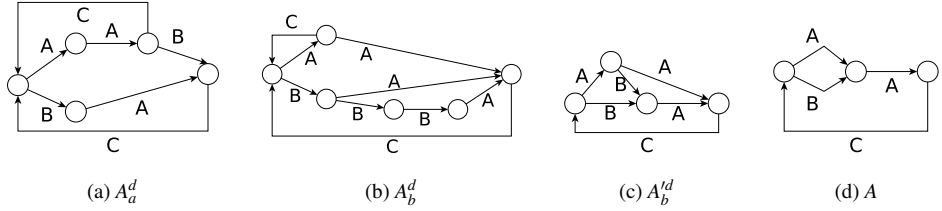
Figure 3. Deterministic finite automata
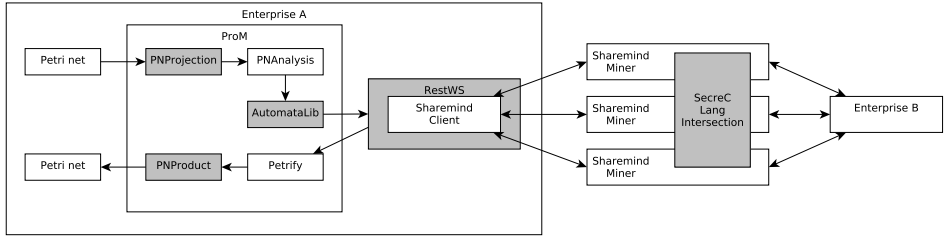
Figure 4. Architecture

Theorem 8 *Let N_a, N_b and N'_b be three labeled Petri nets defined over the alphabets Σ_a, Σ_b and Σ'_b, respectively. If $\mathbf{proj}_{\Sigma_a}(N_a \times N_b) \sim \mathbf{proj}_{\Sigma_a}(N_a \times N'_b)$ and $\Sigma_a \cap \Sigma_b = \Sigma_a \cap \Sigma'_b$ then the automata obtained from step 5 of $Protocol_a(N_a, N_b)$ and $Protocol_a(N_a, N'_b)$ are the same.*

Thm. 8 guarantees that $Protocol_a(N_a, N_b)$ is indistinguishable from $Protocol_a(N_a, N'_b)$, where indistinguishability means that the messages received by the participant a are the same.

Example.   We demonstrate the protocol using our running example. Starting from the input Petri nets N_a and N_b depicted on Fig. 1, the two participants compute the deterministic finite automata A^d_a and A^d_b of Fig. 3 (steps 1, 2 and 3), hiding all transitions whose labels are not in $\Sigma_a \cap \Sigma_b$, computing the reachability graph, and then determinizing the result. Step 4 requires the execution of the secure multiparty protocol for regular language intersection of Sec. 2, which returns the automaton A to both participants. After the termination of the SMC protocol, the participants can proceed independently. Fig. 2 depicts the final network obtained by the participant a, which is computed by synthesizing a Petri net from A, and by using the product operator $\times$.

Now, if the participant b owns the Petri net N'_b, then the computed intermediate automaton is A'^d_b. Notice that the SMC protocol for regular language intersection still yields the automaton A. In fact, A is the minimal automaton of both $A^d_a \times A^d_b$ and $A^d_a \times A'^d_b$. This guarantees that the participant a learns nothing more than the expected result.

3.1. Implementation

We developed a prototype implementation by integrating a well-known business process analysis platform, PROM [27], with the SMC platform SHAREMIND [16]. Fig. 4 shows the architecture of our prototype, where gray boxes represent the components we developed ourselves.

Each enterprise hosts an instance of PROM. The enterprise business process can be imported either by using the native PROM support for standard Petri net file formats, or by using existing plug-ins (see e.g. [28,29]) that transform BPMN diagrams into Petri nets. Steps 1, 2 and 3 of Alg. 3 are executed in PROM: (i) the new PNProjection plug-in hides the transitions that are labeled with symbols not shared between the two enterprises, (ii) the existing PNAnalysis plug-in is used to compute the marking graph and the corresponding NFA, and (iii) a new plug-in encapsulates the functionality of the JAutomata Library [30].

Each enterprise hosts a private SHAREMIND client. To initiate the regular language intersection protocol, SHAREMIND clients share the automata computed at step 3 among the three miners, using the SHAREMIND network protocol. When the protocol is completed, the three miners deliver back to the two clients the computed shares, enabling the enterprises to reconstruct the final result. To enable PROM (developed in Java) to interact with the SHAREMIND client (developed in C++), we encapsulated the latter in a REST web service and used standard libraries to implement the HTTP protocol and data serialization.

Step 6 is implemented using the existing TSPetrinet PROM plug-in. This plug-in can synthesize labeled Petri nets starting from arbitrary finite transition systems using regions. Finally, we developed the PNProduct plug-in based on the product $\times$ construction introduced in Sec. 1.4.

4. Log Auditing

Identifying errors in activities that involve several business partners and cross enterprise boundaries is challenging. These activities must respect the policies of all the partners involved. From a behavioral point of view, such constraints can be rephrased to require that all process instances be consistent with the business processes of all partners.

Log auditing consists of checking whether the log of a completed process instance matches a predefined business process, and identifying possible mismatches. In this work we consider two mutually distrustful parties, with one party owning the log and the other one owning the business process, assuming no trusted third party is available. The two parties are reluctant to reveal any information about their log and business process that is not strictly deducible from the matching result.

Assume two enterprises a and b. For each of the two enterprises we are given local alphabets, Σ_a and Σ_b, respectively. As before, we assume that $\Sigma = \Sigma_a \cap \Sigma_b$ are events and interactions shared between the two enterprises, while $(\Sigma_a \cup \Sigma_b) \setminus \Sigma$ are internal tasks. Enterprise a recorded the log of a process instance $\omega \in \Sigma_a^*$ by monitoring enterprise activities. Enterprise b owns a local business process representing all possible licit executions that is given as a bounded labeled Petri net N_b defined over the corresponding local alphabet.

Formally, the problem of *log auditing* is to compute whether

$$\mathbf{proj}_{\Sigma_a \cap \Sigma_b}(\omega) \in \mathbf{proj}_{\Sigma_a \cap \Sigma_b}(\mathcal{L}_b) \ .$$

If there are no privacy constraints, then log auditing is simple: the two parties can simply exchange their logs and processes, and replay the string ω on the Petri net.

Here we wish the two participants to check the match without being able to learn about the other enterprise's process or log more than can be deduced from their own input and the obtained result. Apart from the process and the log, we also consider the alphabet differences as private values. Hence, we consider only the common alphabet $\Sigma_a \cap \Sigma_b$ as a public value.

To execute log auditing without compromising the participants' privacy we use the SMC protocol for private lookup described in Chapter 6, Sec. 3.

Algorithm 4: Log auditing by the owner b of the business process

Data: Business process, represented as a Petri net N_b

Result: A flag indicating whether the log of the other party matches this process

1 $N^p := \mathbf{proj}_{\Sigma_a \cap \Sigma_b}(N_b)$ // project the private net onto the common alphabet

2 $A^p := NFA(N^p)$ // obtain the equivalent nondeterministic automaton
               // by computing the reachability graph of the net

3 $A^d := SC(A^p)$ // determinize the automaton

4 send A^d to the DFA execution protocol in Alg. 5

Algorithm 5: DFA execution

Data: Alphabet Σ, and the set of states Q of a DFA A

Data: Private transition function $[\![\delta]\!]$ of A

Data: Private characteristic vector $[\![\mathbf{F}]\!]$ of the final states of A

Data: Private string $[\![\omega_a]\!]$ with $\omega_a \in \Sigma^*$

Result: A private flag indicating whether ω_a is accepted by A

$[\![q]\!] = [\![0]\!]$

for $i = 1$ **to** $|\omega'_a|$ **do**

     $[\![w_i]\!] = [\![\omega'_{a,i}]\!]$

     $[\![q]\!] = \mathsf{lookup}([\![\delta]\!], |\Sigma| \cdot [\![q]\!] + [\![w_i]\!])$

$[\![match]\!] = \sum_{i \in |Q|}(([\![q]\!] == [\![i]\!]) \wedge [\![F_i]\!])$

return $[\![match]\!]$

Alg. 4 is used by the process owner to compute a DFA recognizing the language (over the common alphabet Σ) of the original business process. The process is delivered to the ABB that executes Alg. 5. The other input of the ABB ($[\![\omega'_a]\!]$) is provided by the other participant by simply projecting the log ω_a over Σ. Alg. 5 is implemented using the ABB, where $\mathsf{lookup}([\![\delta]\!], |\Sigma| \cdot [\![q]\!] + [\![w_i]\!])$ is a private lookup that implements the transition function δ of the DFA. It is represented as a table of $|Q| \cdot |\Sigma|$ private values. We compute the private index from $[\![q_{i-1}]\!]$ and $[\![w_i]\!]$ and use it to find $[\![q_i]\!]$ from this table, seen as a vector of the length $N = mn$. In our implementation, the size of the problem (values $|Q| = m$, $|\Sigma_a \cap \Sigma_b| = n$ and $|\omega| = \ell$) is public, but δ and F are private.

4.1. Implementation

We need private lookup to implement δ. It is represented as a table of $|Q| \cdot |\Sigma|$ private values. We compute the private index from $[\![q_{i-1}]\!]$ and $[\![w_i]\!]$, and use it to find $[\![q_i]\!]$ from

this table, seen as a vector of the length $N = mn$. We have implemented DFA execution using both additive sharing and Shamir's sharing, in fields $GF(p)$ for $p = 2^{32} - 5$ and $GF(2^{32})$.

Our implementation is suboptimal in the round complexity (which is $O(\ell)$), as it faithfully implements the definition of the DFA execution. Hence, the running time of the online phase is currently dominated by the latency of the network. On the other hand, this also implies that many instances of DFA execution run in parallel would have almost the same runtime (for the online phase) as the single instance. It is well-known that the DFA execution can be implemented in a parallel fashion, using $O(\log \ell)$ time (or SMC rounds). This, however, increases the total work performed by the algorithm by a factor of $O(m)$.

We see that for the vector-only phase of the private lookup, we need the description of δ, but not yet the string w. This corresponds very well to certain envisioned cloud services, in particular to privacy-preserving spam filtering, where regular expressions are used to decide whether a particular message is spam.

The online phase of our implementation (on the cluster described in Sec. 2.4) requires about three seconds to match a string of the length of $2,000$ over an alphabet of the size of 30 on a DFA with $1,000$ states, when using the field $GF(p)$. With the optimization to vector-only phase, the time spent in the online phase increases to around 4.5 seconds. See [31] for the performance of other phases, and of the tasks of other sizes.

5. Concluding Remarks

Organizational activities that cross enterprise borders are an important playground for SMC techniques. In fact, it is critical to be able to effectively manage and coordinate several competitive partners without compromising the confidentiality of their classified information. In this chapter we demonstrated two results of the UaESMC project by applying these to the context of business processes.

The method proposed in Sec. 2 allows privately intersecting regular languages in the presence of privacy constraints. Our approach goes beyond the existing techniques for private set intersection as regular languages can be infinite. We use finite representations of regular languages as deterministic finite automata, and our method is based on computing, in a secure domain, the minimization of automata. Since minimal deterministic finite automata are unique (up to isomorphism) for a given regular language, the result of the computation can be revealed to the parties without leaking any additional information other than the intersection language itself. The same approach can be used for any binary operation on regular languages that has its counterpart in the class of finite automata: one needs merely to replace the product construction with the corresponding one, and then proceed with minimization as above.

The private intersection of regular languages is the key block to handling process fusion. In Sec. 3 we use the protocol to allow competitive parties to discover their local views of the composition of their business processes. Even if the composition of work-flows has been widely studied, no previous work takes into account privacy from a workflow perspective.

Sec. 4 presents an additional application of SMC techniques to the context of business processes. There, we use the protocols described in Chapter 6 to implement the exe-

cution of deterministic finite automata, which allows competitive parties to audit the logs of their respective business partners.

Analysis of the composition of business processes in cases where classified information is needed is an interesting open issue. For example, when two enterprises collaborate in order to build a virtual enterprise, the composition of their workflows can yield unsound processes, such as interorganizational interactions that manifest deadlocks and livelocks.

References

[1] Stephen A White. Introduction to BPMN. *IBM Cooperation*, 2, 2004.

[2] Wil MP van der Aalst. Formalization and verification of event-driven process chains. *Information and Software technology*, 41(10):639–650, 1999.

[3] Wil MP van der Aalst. The application of Petri nets to workflow management. *Journal of circuits, systems, and computers*, 8(01):21–66, 1998.

[4] Remco M Dijkman, Marlon Dumas, and Chun Ouyang. Semantics and analysis of business process models in BPMN. *Information and Software Technology*, 50(12):1281–1294, 2008.

[5] 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.

[6] Dexter C Kozen. *Automata and computability*. Springer-Verlag New York, Inc., 1997.

[7] Michel Hack. Petri net language. Technical report, 1976.

[8] Eric Fabre and Albert Benveniste. Partial order techniques for distributed discrete event systems: Why you cannot avoid using them. *Discrete Event Dynamic Systems*, 17(3):355–403, 2007.

[9] Roberto Guanciale, Dilian Gurov, and Peeter Laud. Private intersection of regular languages. In *PST*, pages 112–120. IEEE, 2014.

[10] Edward F Moore. Gedanken-experiments on sequential machines. *Automata studies*, 34:129–153, 1956.

[11] John Hopcroft. An n log n algorithm for minimizing states in a finite automaton. Technical report, DTIC Document, 1971.

[12] Bruce W Watson. A taxonomy of finite automata minimization algorithms. Technical report, 1993.

[13] Jordi Cortadella, Michael Kishinevsky, Luciano Lavagno, and Alexandre Yakovlev. Deriving Petri nets from finite transition systems. *Computers, IEEE Transactions on*, 47(8):859–882, 1998.

[14] Michael J Freedman, Kobbi Nissim, and Benny Pinkas. Efficient private matching and set intersection. In *EUROCRYPT*, pages 1–19. Springer, 2004.

[15] Michael Domaratzki, Derek Kisman, and Jeffrey Shallit. On the number of distinct languages accepted by finite automata with n states. *Journal of Automata, Languages and Combinatorics*, 7(4):469–486, 2002.

[16] 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.

[17] Dan Bogdanov. *Sharemind: programmable secure computations with practical applications*. PhD thesis, University of Tartu, 2013.

[18] Sven Laur, Jan Willemson, and Bingsheng Zhang. Round-Efficient Oblivious Database Manipulation. In *Proceedings of ISC 2011*, pages 262–277, 2011.

[19] Peeter Laud and Jan Willemson. Composable Oblivious Extended Permutations. Cryptology ePrint Archive, Report 2014/400, 2014. `http://eprint.iacr.org/`.

[20] Peeter Laud. Privacy-Preserving Minimum Spanning Trees through Oblivious Parallel RAM for Secure Multiparty Computation. Cryptology ePrint Archive, Report 2014/630, 2014. `http://eprint.iacr.org/`.

[21] Wenliang Du and Mikhail J Atallah. Privacy-preserving cooperative scientific computations. In *Computer Security Foundations Workshop, IEEE*, pages 0273–0273. IEEE Computer Society, 2001.

[22] 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.

[23] Frédérique Bassino and Cyril Nicaud. Enumeration and random generation of accessible automata. *Theoretical Computer Science*, 381(1):86–104, 2007.

[24] Justin Brickell and Vitaly Shmatikov. Privacy-preserving graph algorithms in the semi-honest model. In *ASIACRYPT*, pages 236–252. Springer, 2005.

[25] Riivo Talviste. Deploying secure multiparty computation for joint data analysis - a case study. Master's thesis, University of Tartu, 2011.

[26] Roberto Guanciale and Dilian Gurov. Privacy preserving business process fusion. In *SBP*. Springer, 2014.

[27] Boudewijn F van Dongen, Ana Karla A de Medeiros, HMW Verbeek, AJMM Weijters, and Wil MP van der Aalst. The prom framework: A new era in process mining tool support. In *Applications and Theory of Petri Nets*, pages 444–454. Springer, 2005.

[28] Roberto Bruni, Andrea Corradini, Gian Luigi Ferrari, Tito Flagella, Roberto Guanciale, and Giorgio Spagnolo. Applying process analysis to the italian egovernment enterprise architecture. In *WS-FM*, pages 111–127. Springer, 2011.

[29] Anna Kalenkova, Massimiliano de Leoni, and Wil M. P. van der Aalst. Discovering, analyzing and enhancing BPMN models using prom. In Lior Limonad and Barbara Weber, editors, *BPM Demo Sessions*, page 36. CEUR-WS.org, 2014.

[30] Arnaud Bailly. Jautomata library. `http://jautomata.sourceforge.net/`, 2006.

[31] 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.

Chapter 8
Mechanism Design and
Strong Truthfulness

Yiannis GIANNAKOPOULOS [a]

[a] *University of Oxford, UK*

Abstract. In this chapter we give a very brief overview of some fundamentals from mechanism design, the branch of game theory dealing with designing protocols to cope with agents' private incentives and selfish behavior. We also present recent results involving a new, extended utilities model that can incorporate *externalities*, such as malicious and spiteful behavior of the participating players. A new notion of *strong truthfulness* is proposed and analyzed. It is based on the principle of punishing players that lie. Due to this, strongly truthful mechanisms can serve as subcomponents in bigger mechanism protocols in order to boost truthfulness. The related solution concept equilibria are discussed and the power of the decomposability scheme is demonstrated by an application in the case of the well-known mechanism design problem of scheduling tasks to machines for minimizing the makespan.

Keywords. Mechanism design, incentive compatibility, strong truthfulness, VCG mechanisms

Introduction

Game theory [1] is a discipline that studies the strategic behavior of players that interact with each other and are completely rational and selfish: they only care about maximizing their own personal utilities. Extensive work has been done in this field over the last 60 years and various solution concepts and models have been proposed that try to explain and predict the outcome of such games. The branch of game theory called *mechanism design* (MD) was developed to have, in a way, a completely inverse goal: if we as algorithm designers want to enforce a certain outcome to such a system of interacting selfish entities, what are the game-playing rules we should impose? Therefore, it seems that MD and SMC share some very important common fundamental priors, that is, trying to optimize a joint objective while dealing with players that have incentives to manipulate our protocols. In fact, there has been a long and fruitful interaction between ideas from game theory and cryptography, see e.g. [2,3,4,5,6].

First, we give a short introduction to the basic ideas of MD, mostly using auctions as a working example, and discuss important MD paradigms, like the VCG mechanism and various optimization objectives, like maximizing the social welfare or the seller's revenue. Next, we devote the remaining chapter to discussing some exciting recent results from [7]. We believe that these results are particularly relevant to cryptographic

considerations, as they involve a new, extended utilities model for MD settings which can incorporate *externalities*, such as malicious and spiteful behavior of the participating agents. A new notion of *strong* truthfulness is presented and analyzed. It is based on the principle of punishing players that lie. Due to this, strongly truthful mechanisms can serve as subcomponents in bigger mechanism protocols in order to boost truthfulness in settings with externalities and achieve a kind of externalities-resistant performance. This decomposability scheme is rather general and powerful, and we show how it can be also adapted to the case of the well-known MD problem of scheduling tasks to unrelated parallel machines in order to minimize the required execution time (makespan).

1. Foundations

In this section we are going to present some fundamental notions from MD, upon which the exposition and results of subsequent sections are going to be built. We deliberately choose to make this presentation based on a simple, single-item auction paradigm, in order to demonstrate more clearly both the intuition and the essence of these notions and not get lost in the technicalities of the general models which, after all, are not essential for the needs of this book. The reader, of course, can find many good references including more formal and general introductions to the fascinating area of algorithmic mechanism design (see e.g. [8,9]).

1.1. Basic Notions from Game Theory and Mechanism Design

Assume the following traditional MD setting, in particular a single-item auction scenario. We have n players (also called agents), each of whom is willing to pay t_i, $i = 1, 2, \ldots, m$, in order to get the item. These are called the agents' *types*, and $\mathbf{t} = (t_1, t_2, \ldots, t_n)$ is the *type profile*. Let us assume that all these belong to some domain T, where in our particular auction setting it is a natural assumption to consider $T = \mathbb{R}_+$. This is *private* information of the players, who report it to the auctioneer in the form of *bids* b_i, $i = 1, 2, \ldots, m$. The reason we discriminate between types and bids is that the players, as we will see shortly, may have reason to lie about their true types and misreport some $b_i \neq t_i$. Given the input by the players, i.e. the bid profile $\mathbf{b} = (b_1, b_2, \ldots, b_n)$, the auctioneer needs to decide who gets the item and how much she is going to pay for it.

More formally, a *mechanism* $\mathcal{M} = (\mathbf{a}, \mathbf{p})$ consists of two vectors: an *allocation* vector $\mathbf{a} = \mathbf{a}(\mathbf{b}) = (a_1(\mathbf{b}), a_2(\mathbf{b}), \ldots, a_n(\mathbf{b})) \in [0, 1]^n$ and a *payment* vector $\mathbf{p} = \mathbf{p}(\mathbf{b}) = (p_1(\mathbf{b}), p_2(\mathbf{b}), \ldots, p_n(\mathbf{b})) \in \mathbb{R}_+^n$. If our mechanism is deterministic, $a_i(\mathbf{b})$ is simply an indicator variable of the value 0 or 1, denoting whether player i wins the item or not. If we allow for randomized mechanisms, $a_i(\mathbf{b})$ denotes the probability of player i winning the item. In the latter case, we must make sure that $\sum_{i=1}^{n} a_i(\mathbf{b}) \leq 1$ for all $\mathbf{b} \in T^n$. Also, agent i will have to submit a payment of $p_i(\mathbf{b})$ to the mechanism. We define the *utility* of player i to be his total happiness after taking part in the auction, known as his *valuation* $v_i(\mathbf{a}, t_i) = a_i \cdot t_i$, minus the payment p_i he has submitted. Formally, utility is defined as:

$$u_i(\mathbf{b}|t_i) = v_i(\mathbf{a}(\mathbf{b}), t_i) - p_i(\mathbf{b}) = a_i(\mathbf{b}) \cdot t_i - p_i(\mathbf{b}). \tag{1}$$

Notice the notation $u_i(\mathbf{b}|t_i)$ and the different usage of bids and types in expression (1). In case of *truth-telling*, i.e. honest reporting of $b_i = t_i$, we simplify the notation to

$$u_i(\mathbf{b}) = a_i(\mathbf{b}) \cdot b_i - p_i(\mathbf{b}). \qquad (2)$$

We call these utilities *quasilinear*, due to the special form of these expressions, and in particular the linear-form connection between a player's utility and the player's own type t_i. The MD model can be defined in a more general way. The allocation function can belong to an arbitrary set of *outcomes A*. For our special case of single-item bidders, the outcomes are just the allocation vectors belonging to $A = [0,1]^n$. A player's valuation v_i is then defined over the set of possible outcomes and her true type t_i, $v_i(\mathbf{a}, t_i)$, and does not need to have the special linear form for the case of a single-item auction described previously. See Sec. 1.2.2 for a more general example of a MD setting.

If we look a little closer, we can see that we have already formed a *game* (see e.g. [10]): the players' strategies are exactly their valuations and each one of them is completely rational, and their goal is to *selfishly* maximize her own utility. So, we can use standard solution concepts and in particular *equilibria* in order to talk about possible stable states of our auctions. For example, the most fundamental notion that underlies the entire area of MD is that of *truthfulness* (also called incentive compatibility or strategy-proofness). Intuitively, we will say that a mechanism is truthful if it makes sure that no agent has an incentive to lie about her true type.

Definition 1 (Truthfulness). *A mechanism $\mathcal{M} = (\mathbf{a}, \mathbf{p})$ is called truthful if truth-telling is a* dominant-strategy equilibrium *of the underlying game, i.e. $u_i(b_i, \mathbf{b}_{-i}|b_i) \geq u_i(\tilde{b}_i, \mathbf{b}_{-i}|b_i)$ for every player i, all possible bid profiles $\mathbf{b} \in T^n$ and all possible misreports $\tilde{b}_i \in T$.*

In case of randomized mechanisms, the above definitions are naturally extended by taking expectations of the utilities (*truthful in expectation* mechanisms). Here we have used standard game-theoretic notation, $\mathbf{b}_{-i}$ denoting the result of removing the i-th coordinate of $\mathbf{b}$. This vector is of one dimension lower than $\mathbf{b}$. This notation is very useful for modifying vectors at certain coordinates. For example $(x, \mathbf{b}_{-i})$ is the vector we get if we replace the i-th coordinate b_i of $\mathbf{b}$ with a new value of x. In particular, notice that this means that $(b_i, \mathbf{b}_{-i}) = \mathbf{b}$.

Being implemented in dominant strategies, truthfulness is a very stable and desirable property, which we want all our auction mechanisms to satisfy. It allows us to extract the truth from the participating parties and, thus, be able to accurately design the protocols for the goals we want to achieve. A celebrated result by Myerson gives us a powerful and simple characterization of truthful mechanisms and also helps us completely determine a mechanism simply by giving its allocation function $\mathbf{a}$.

Theorem 1 (Myerson [11]). *A mechanism $\mathcal{M} = (\mathbf{a}, p)$ is truthful if and only if:*

1. *Its allocation functions are monotone nondecreasing, in the sense that*

$$b_i \leq b_i' \implies a_i(b_i, \mathbf{b}_{-i}) \leq a_i(b_i', \mathbf{b}_{-i})$$

   for every player i, all valuation profiles $\mathbf{b}_{-i}$ and all valuations b_i, b_i'.
2. *The payment functions are given by*

$$p_i(\mathbf{b}) = a_i(\mathbf{b})b_i - \int_0^{b_i} a_i(x, \mathbf{b}_{-i})\,dx. \qquad (3)$$

Based on this, one can show another elegant and concise analytic characterization of truthful mechanisms.

Theorem 2. *[Rochet [12]] A mechanism is truthful if and only if for every player i, it induces a utility function u_i (over T^n) which is* convex *with respect to its i-th component.*

1.2. Fundamental MD Problems

In this section we briefly present two of the most fundamental MD domains, namely *additive auctions* (Sec. 1.2.1) and *scheduling unrelated machines* (Sec. 1.2.3), as well as the associated optimization objectives. These are the predominant motivating examples that move the entire field forward and also serve as the basis for our exposition in this chapter.

1.2.1. Welfare and Revenue in Auctions

The fundamental single-item auction introduced in Sec. 1.1 can be generalized to the following *m*-items *additive* valuations auction setting, where now the allocation **a** of a mechanism $\mathcal{M} = (\mathbf{a}, \mathbf{p})$ is an $n \times m$ matrix $\mathbf{a} = \{a_{ij}\} \subseteq [0,1]^{n \times m}$, where a_{ij} represents the probability of agent i getting item j, $i = 1, 2, \ldots, n$, $j = 1, 2, \ldots, m$. Inputs to the mechanism are bid profiles $\mathbf{b} \in T^{n \times m}$, $T = \mathbb{R}_+$, where b_{ij} is the bid of player i for item j. Of course, we must make sure that for every possible input **b** the selected outcome $\mathbf{a}(\mathbf{b})$ must not assign any item to more than one agent, i.e.

$$\sum_{i=1}^{n} a_{ij}(\mathbf{b}) \leq 1 \quad \text{for all items} \ \ j = 1, 2, \ldots, m. \tag{4}$$

When we design auction mechanisms we are usually interested in maximizing either the combined happiness of our society, meaning the sum of the valuations of the players that receive items, or the auctioneer's profit, i.e. the sum of the payments he collects from the participating agents. So, we define the social *welfare* of mechanism $\mathcal{M}$ on input **b** to be

$$W(\mathbf{b}) \equiv \sum_{i=1}^{n} v_i(\mathbf{a}(\mathbf{b}), \mathbf{b}_i) = \sum_{i=1}^{n} \sum_{j=1}^{m} a_{ij}(\mathbf{b}) b_{ij} \ .$$

Here we assume that the players have *additive* valuations, that is, the valuation for receiving a subset of items is just the sum of the valuations of the items in the bundle, and its *revenue*

$$R(\mathbf{b}) \equiv \sum_{i=1}^{n} p_i(\mathbf{b}).$$

The most well-known auction is without doubt the VCG auction, named after the work of Vickrey [13], Clarke [14] and Groves [15]. In the simple single-item setting, this mechanism reduces to the Vickrey second-price auction that gives the item to the highest bidding agent but collects as payment the second highest bid. In that way, it ensures truthfulness by not giving an incentive to the wining agent to misreport a lower bid, as

her payment is independent of her own bid and depends only on the bids of the other players. And, above all, this mechanism maximizes social welfare by definition.

Formally, by generalizing these ideas to the setting of m items, the VCG auction is a mechanism with the allocation rule

$$\mathbf{a}(\mathbf{b}) = \underset{\alpha \in A}{\operatorname{argmax}} W(\alpha(\mathbf{b})) = \underset{\alpha \in A}{\operatorname{argmax}} \sum_{i=1}^{n} v_i(\alpha, \mathbf{b}_i) \quad \text{for all } \mathbf{b} \in T^{n \times m}, \qquad (5)$$

and payments

$$p_i(\mathbf{b}) = \max_{\alpha \in A} \sum_{j \neq i} v_j(\alpha(\mathbf{b}), \mathbf{b}_j) - \sum_{j \neq i} v_j(a(\mathbf{b}), \mathbf{b}_j) \quad \text{for all } \mathbf{b} \in T^{n \times m}. \qquad (6)$$

The underlying idea is that first of all, the allocation rule (5) ensures that social welfare is the optimal one, and the payments (6) are such that they internalize the externalities of every player i. That is, intuitively, we charge player i for the harm that her participation in the auction causes to the rest of the society.

1.2.2. VCG Example: Buying Paths of a Network

We demonstrate the allocation and payments of the VCG mechanism through the following example, which is slightly more complex than the single-item auction we have been focusing on thus far. Let $G = (V, E)$ be a 2-edge connected directed graph. Let $s, t \in V$ be two nodes of the graph. Each edge $e \in E$ is a player that has as type an edge-cost (latency) c_e. We want to send a message from s to t, so the set of outcomes A in our setting are all possible paths π from s to t in G. We will denote such a path compactly as $\pi : s \to t$. If a player e is selected in an outcome-path, she incurs a damage of c_e, so the valuations are modeled by

$$v_e = v_e(\pi) = \begin{cases} -c_e, & \text{if } e \in \pi, \\ 0, & \text{otherwise.} \end{cases}$$

for all $\pi : s \to t$. Hence, maximizing social welfare $\sum_e v_e(\pi)$ in our problem is equivalent to selecting the shortest path from s to t in the weighted graph G with the edge weights c_e.

Consider the graph on Fig. 1. The outcome of our mechanism would be the shortest path from s to t, here $s \to a \to t$ that has the length of $5 + 1 = 6$. Let us look at what players (s, a) and (a, t) have to pay to get allocated. If player (s, a) was not present, the shortest path would be $s \to b \to t$ for social welfare of $(-4) + (-3) = -7$. As she is present, the welfare of the other players is $v_{(a,t)} + v_{(s,b)} + v_{(b,t)} = -1 + 0 + 0 = -1$. So, she should pay $p_{(s,a)} = -7 - (-1) = -6$. The negative sign means that we should give money to this edge/player for using it instead of asking money from it. Similarly, the monetary compensation to the other participating edge should be $p_{(a,t)} = -2$.

1.2.3. Scheduling and Minimizing Makespan

The scheduling domain is essentially an additive multi item auction setting (see Sec. 1.2.1) with the modification that players are not trying to maximize their utilities,
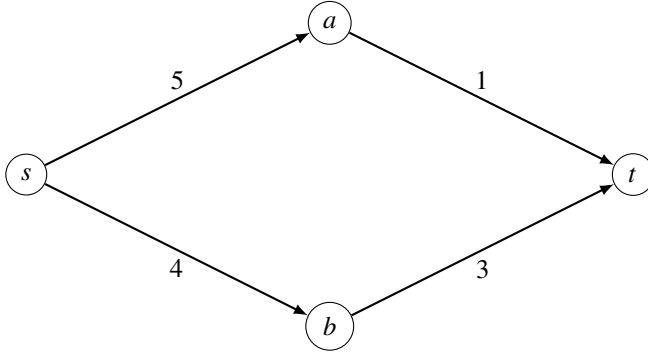
Figure 1. An example routing graph demonstrating the execution of the VCG mechanism

but to *minimize* them as they are cost functions. This results in a different model, and the exact nature of the relation between the two domains is not completely clear at the moment. The MD version of the problem was first studied by Nisan and Ronen in their seminal paper [8] that has arguably been the most influential paper in the area of algorithmic MD. In the scheduling setting we have n machines (players) and m tasks (items). Each machine reports to the mechanism designer the time she would need to process every item, in the form of a time matrix (type profile) $\mathbf{t} = \{t_{ij}\} \subseteq T^{n \times m}$, $T = \mathbb{R}_+$, where t_{ij} is the processing time of machine i for task j. A feasible allocation is an assignment of tasks to players, given (as in the case of additive auctions) by an allocation matrix $\mathbf{a} = \{a_{ij}\} \in \{0,1\}^{n \times m}$, $a_{ij} = 1$ if and only if machine i executes task j. The total valuation of machine i is the sum of the processing times for each individual task assigned (additive valuations) $\sum_{j=1}^{m} a_{ij} t_{ij}$. Each machine's resulting cost is $c_i(\mathbf{t}) = \sum_{j=1}^{m} a_{ij}(\mathbf{t}) t_{ij} - p_i(\mathbf{t})$, where $p_i(\mathbf{t})$ represents the payments with which we compensate machine i in order to motivate it to take part in the execution.

Especially with the emergence of the Internet as the predominant computing paradigm, it is natural to assume that these machines will act selfishly and care only about minimizing their own costs $c_i(\mathbf{t})$, and possibly misreport their true processing times to this end. Hence, a game theoretic approach to the classical task allocation problem under the *added truthfulness* constraint is both interesting and necessary. The standard objective is to design truthful mechanisms that minimize the *makespan*

$$\text{Makespan}(\mathbf{t}) = \max_i \sum_{j=1}^{m} a_{ij}(\mathbf{t}) t_{ij},$$

that is, the time it would take the slowest machine to finish the processing. Again, we will also require no task to be unprocessed, and thus (4) becomes

$$\sum_{i=1}^{n} a_{ij}(\mathbf{t}) = 1 \quad \text{for all tasks } j \text{ and time matrices } \mathbf{t} \in T^{n \times m}. \tag{7}$$

This is known as the *scheduling problem in parallel unrelated machines* [8,16]. When we consider the *fractional allocations* variant of the scheduling problem [17], we allow $\{a_{ij}\} \in [0,1]^{n \times m}$, while still demanding condition (7). Fractional allocations are essentially randomized mechanisms for the non-fractional version of the problem.

2. Externalities and Challenges

In this chapter we describe the notion of *externalities* in MD settings and present the game theoretic and algorithmic challenges that they introduce. The discussion is inevitably intertwined with challenging the fundamental notion of traditional *truthfulness* itself, and thus, this section essentially forms the motivation for the main results of this chapter, presented in Sec. 3.

2.1. Externalities in Mechanism Design

A fundamental assumption throughout game theory and mechanism design is that all participating players (agents) are fully rational and act selfishly: they have their own well-defined utility function (see e.g. (2)) and they only care about optimizing this function. This can be maximizing satisfaction when they want to buy an item at an auction, or minimizing their cost when they are machines that get to execute tasks (as in the scheduling problem described in Sec. 1.2.3). Usually, this utility function optimization is considered myopic, in the sense that players do not care about variations on the achieved utilities of other players as far as these variations do not affect their own utility levels. For example, in a standard single-item auction setting, if some player does not get the item (thus achieving zero utility), she is indifferent (i.e. her utility does not change) towards how much the winning bidder is going to pay for the item and, even more, she does not care about the bidder's identity.

However, it can be debated whether this really is natural or expected behavior when we think of everyday social interactions. Experiments show that bidders can overbid, possibly risking ending up with negative utilities, just for the joy of winning in case they get the item. Or, on the other hand, if they do not manage to win the item, overbidding will drive prices up in order to harm the other winning agent(s), which is arguably spiteful behavior.

In these examples, where participants in a mechanism behave seemingly irrationally, their happiness is not only a function of their own core utility, but is also affected in complex ways by the other players' utilities. We will call such effects on the modeling of our agents' happiness *externalities*, to emphasize the third party nature of the interaction. Of course, externalities are not only *negative* like in these examples. They can be *positive*, altruistic in nature, e.g. a loving couple taking part in the same auction: one partner may be happy to lose the item if it means that the other one wins it.

Externalities have been heavily studied in economics, not just game theory, and the literature is extensive and diverse. For our purposes, we will approach externalities in the context of the informal definition we gave in the previous paragraphs.

2.2. Impact on Truthfulness

Under the scope of externalities, let us revisit the canonical example of a single-item auction. We know that the second-price paradigm, in particular the Vickrey auction that gives the item to the highest bidding agent but collects as payment the second-highest bid, is optimal for social welfare while also being truthful as no agent has an incentive to lie about her true valuation no matter what the other players report (i.e. truth-telling is a dominant strategy equilibrium). This result allows us not only to maximize the collective

happiness of the participants, but also ensure the integrity of the extracted information, here the agents' bids. Furthermore, this is backed up by a very powerful notion of stability for our solution, that of dominant strategies implementation.

However, if we consider spiteful behavior, this does not hold as the second highest bidder, who is going to lose the item anyway, can harm the winning agent by declaring an (untruthful) higher bid that immediately (externally) affects the payment of the other players, and in particular increases the payment for the winning player. Our aim is to study, model, and, if possible, prevent this phenomenon. One simple suggestion would be to encode such possible externalities-behavior into each player's type (some kind of generalized valuation) and then run the powerful VCG mechanisms (see Sec. 1.2.1) on the new extended type-profile space of all agents to get a socially efficient and dominant strategy truthful mechanism. However, there is a fundamental problem with this approach that conflicts all existing tools in the area of mechanism design: the players' utilities now also depend on *other* agents' payments[1]. To be more precise, utilites are no more *quasilinear* functions with respect to payments.

2.3. Challenges

Given the above discussion, the new challenges that are the main motive for this chapter, are the following:

- How to incorporate these externalities properly into the existing standard utility maximization framework of MD? We need a new model for the utility functions, taking into consideration both the core and external utilities of each player.
- Is the standard notion of truthfulness, and in particular that of implementation in dominant strategies, computationally and descriptively appropriate to model this new complex framework of interactions? Can we propose a new notion of empowered truthfulness?
- Utilizing stronger notions of truthfulness, can we design mechanisms that manage to somehow resist externalities that threaten the integrity of traditional truthfulness, the building block of the entire area of MD?
- Is there a general MD paradigm that provides construction of such externality-resistant mechanisms?

3. Strong Truthfulness and Externality Resistance

In this section we start dealing with the challenges discussed in Sec. 2.3. For the remainder of this chapter our exposition is based on the main results of [7]. Further details and missing proofs can be found in that paper.

First, under these new complex underlying interactions among players, we need to have a solid building-block, stronger than the traditional notion of truthfulness which, as we already have seen in Sec. 2.1, even on the simplest example of single-item auctions, can make the powerful VCG second-price auction fail. The intuition is that we would like to introduce a more strict notion of truthfulness, where not only players have no reason to

---

[1]This is in fact a more appropriate definition, economics-wise, of externalities themselves: utilities are *externally* affected by payments to third parties, over which we have no direct control.

lie but are punished for misreports. We want greater deviations from the true valuations to result in greater reductions to the resulting utilities for the players who deviate.

3.1. The Notion of Strong Truthfulness

Definition 2 (Strong truthfulness). *A mechanism is called c-strongly truthful, for some $c \in \mathbb{R}$, if it induces utilities u_i such that*

$$u_i(b_i, \mathbf{b}_{-i}) - u_i(\tilde{b}_i, \mathbf{b}_{-i}) \geq \frac{1}{2}c|\tilde{b}_i - b_i|, \tag{8}$$

for every player i, bids b_i, $\tilde{b}_i$ and all bid profiles of the other players $\mathbf{v}_{-i}$.

This notion of strong truthfulness is a generalization of the standard notion of truthfulness, achieved by setting $c = 0$. Under these definitions we can now prove an analytic characterization of strong truthfulness, in the spirit of Thm. 2:

Theorem 3. *A mechanism is c-strongly truthful if and only if the utility functions it induces for every player are all c-strongly convex functions*[2].

3.1.1. Strongly Truthful Auctions

Let us give an example of a strongly truthful auction in the simplest case of a single-buyer, single-item auction. Assume a single player with a valuation for the item drawn from some bounded real interval $[L, H]$. We define the following mechanism, which we will call *linear*.

Definition 3 (Linear mechanism). *The linear mechanism (LM) for the single-buyer, single-item auction setting has the allocation*

$$a(b) = \frac{b - L}{H - L} \quad,$$

where the buyer's bid b for the item ranges over a fixed interval $[L, H]$.

It turns out that this mechanism is in fact the strongest possible one we can hope for in this setting.

Theorem 4. *The linear mechanism is $\frac{1}{H-L}$-strongly truthful for the single-buyer, single-item auction setting and this is optimal*[3] *among all mechanisms in this setting.*

Notice that Thm. 4 holds only in cases where the valuations domain is a (real) interval $T = [L, H]$. If we want to deal with unbounded domains, e.g. $T = \mathbb{R}_+$, we need to define a more flexible notion of *relative* strong truthfulness (see [7, Sec. 2.2]).

---

[2]A function $f : \mathbb{R}^n \longrightarrow \mathbb{R}$ is called *c-strongly convex*, where c is a nonnegative real parameter, if for all $\mathbf{x}, \mathbf{y} \in \mathbb{R}^n$: $f(\mathbf{x}) - f(\mathbf{x}) \geq \nabla f(\mathbf{x}) \cdot (\mathbf{y} - \mathbf{x}) + \frac{c}{2}\|\mathbf{y} - \mathbf{x}\|^2$, where the standard dot inner product and Euclidean norm are used.

[3]Formally, for every *c*-strongly truthful mechanism in this setting $c \leq \frac{1}{H-L}$.

3.2. External Utilities Model

In order to model externalities, both positive (altruism) and negative (spite), we are extending our agents' types to include not only their valuations but also some parameters γ that try to quantify the interest/external-effect each agent has upon others. Formally, for every agent $i = 1, 2, \ldots, n$ we redefine its type t_i to be $t_i = (v_i, \gamma_i)$, where $\gamma_i = (\gamma_{i1}, \gamma_{i2}, \ldots, \gamma_{in})$ with $\gamma_{ij} \in \mathbb{R}$, $j = 1, 2, \ldots, n$ being the *externality parameter* of player i with respect to player j. In fact, parameter γ_{ii} is not needed and we can safely ignore it, setting $\gamma_{ii} = 0$ for all players i, but we still keep it in the formation of γ_i in order to have a vector-consistent notation. The usage of these parameters becomes apparent when we define the utilities in our new externalities model. Intuitively, negative values of γ_{ij} correspond to player i demonstrating spiteful behavior towards player j, positive values correspond to altruistic behavior, and a value of zero corresponds to lack of externalities towards player j (i.e. player i demonstrates standard game-theoretic selfish behavior).

Moving on to defining utilities, let $\mathcal{M} = (\mathbf{a}, \mathbf{p})$ be a mechanism for our new externalities model. Our players have true types $t_i = (v_i, \gamma_i)$ and they submit to the mechanism *bids* (also called *values*) which are (possibly mis-)reports of the valuation component v_i of their true type t_i. Given such a bid profile $\mathbf{b}$, mechanism $\mathcal{M}$ again computes the allocation $\mathbf{a}(\mathbf{b}) = (a_1(\mathbf{b}), a_2(\mathbf{b}), \ldots, a_n(\mathbf{b}))$, where $a_i(\mathbf{b}) \in [0, 1]$ is the probability that agent i receives the service[4], and the payment vector $\mathbf{p}(\mathbf{b}) = (p_1(\mathbf{b}), p_2(\mathbf{b}), \ldots, \mathbf{p}_n(\mathbf{b}))$, where $p_i(\mathbf{b})$ is the payment extracted *from* player i.

In this setting, we define the *base utility* of player i under mechanism $\mathcal{M}$, given the (true) type $t_i = (v_i, \gamma_i)$ and the reported bid vector $\mathbf{b}$, to be the standard utility (1)

$$u_i(\mathbf{b}|t_i) = u_i(\mathbf{b}|v_i) = a_i(\mathbf{b}) \cdot v_i - p_i(\mathbf{b})$$

and then we define her *externality-modified utility*, given also the other players' (true) type profiles $\mathbf{t}_{-i}$, to be

$$\hat{u}_i(\mathbf{b}|\mathbf{t}) = \hat{u}_i(\mathbf{b}|\mathbf{v}, t_i) = u_i(\mathbf{b}|v_i) + \sum_{j \neq i} \gamma_{ij} u_j(\mathbf{b}|t_j) \ . \tag{9}$$

From now on we will refer to this externality-modified utility simply as utility, as this is going to be the utility notion in our new externalities-included model upon which we will also build our new notions of truthfulness and resistant mechanisms. We observe two important things about these definitions.

First, as expected, the base utility $u_i(\mathbf{b}|t_i)$ only depends on type $\mathbf{t}_i$ of player i and not on the other players' types $\mathbf{t}_{-i}$ and, in particular, it depends just on the valuation component v_i of $\mathbf{t}_i$ (i.e. the externality parameters $\gamma_i = (\gamma_{i1}, \gamma_{i2}, \ldots, \gamma_{in})$ do not play any part in these base utilities). Therefore, we can also use the slightly lighter notation $u_i(\mathbf{b}|v_i)$ to denote the basic utility. Essentially, this is the component of our new utility that corresponds exactly to the standard definition of utilities in the traditional no-externalities setting of MD (see Sec. 1.1).

Second, the externality-modified utility needs to depend on the entire (true) type profile $\mathbf{t}$ and not just the component t_i of i, because the externalities-induced term of equation (9) comprises of a sum that ranges across all other players. Furthermore, unlike

---

[4]In the single-item auction paradigm, this means agent i gets the item.

for the base utilities, we do not just need the valuations v_i but all parameters γ_{ij} for all $j \neq i$. However, from the parameters profile vector **t** we only need the externalities parameters of player i so to make the notation more straightforward, we can write $\hat{u}_i(\mathbf{b}|\mathbf{v}, \gamma_i)$ instead of $\hat{u}_i(\mathbf{b}|\mathbf{t})$.

Naturally enough, if $b_i = v_i$, we can denote the base utilities $u_i(\mathbf{b}|\mathbf{t}_i)$ by $u_i(\mathbf{b})$, and if $\mathbf{b} = \mathbf{t}$, we can use $\hat{u}_i(\mathbf{b})$ instead of $\hat{u}_i(\mathbf{b}|\mathbf{t})$ for the externality-modified utilities.

3.3. Externality Resistant Mechanisms

Definition 4 (Externality-resistant VCG). *The externality resistant-VCG mechanism for our MD setting with externalities, denoted by* $\text{rVCG}(\delta)$ *and parametrized by some* $\delta \in [0, 1]$ *is the following protocol:*

- *Ask all n players to report their bids* b_i, $i = 1, 2, \ldots, n$.
- *With the probability of* $\frac{\delta}{n}$ *for every player i, single out player i and run the* LM *mechanism (Def. 3).*
- *With complementary probability* $1 - \delta$, *run the standard VCG mechanism.*

Theorem 5. *Consider a MD setting with externalities where players' valuations are drawn from the unit real interval, i.e.* $v_i \in [0, 1]$ *for all* $i = 1, 2, \ldots, m$. *Then, for every* $\delta, \varepsilon > 0$, *if the the externality parameters of our agents satisfy*

$$\max_{i,j} \gamma_{ij} < \frac{\varepsilon \delta}{8(1 - \delta)^2 n^3} \;\;,$$

the externality-resistant VCG mechanism (Def. 4) induces utilities so that

$$u_i^{\text{rVCG}(\delta)}(\mathbf{b}) \geq (1 - \delta) u_i^{\text{VCG}}(\mathbf{v}) - \varepsilon$$

for every player $i = 1, 2, \ldots, n$ *and all* undominated *strategy (bid) profiles* **b** *of the players.*

The intuition behind this result is that, by randomizing over components or mechanisms that are known to work in simpler settings, the rVCG mechanism manages to achieve (base) utilities that are very close to those of the corresponding VCG mechanism in the no-externalities setting (see Sec. 1.2.1). Of course, this cannot be implemented in dominant strategies (we have seen that truthfulness under such strong solution concepts is doomed to fail) but under a weaker solution concept, that of undominated strategies (see Sec. 3.4).

As an immediate result of this virtual simulation of ideal behavior where only base utilities are taken into consideration, rVCG manages to approximate both optimal social welfare as well as the revenue of the traditional VCG (run on base utilities).

Theorem 6. *In every outcome of* $\text{rVCG}(\delta)$ *implemented in undominated strategies, the social welfare of* $\text{rVCG}(\delta)$ *is within an additive error of* $n\eta$ *from the optimal social welfare and within an additive error of* $2n\eta$ *of the revenue achieved by the standard* VCG *mechanism (run on base utilies without externalities), where n is the number of players and* η *is a parameter so that*

$$\eta \leq \frac{4(1 - \delta)}{\delta} n^2 \gamma$$

(where $\gamma = \max_{i,j} \gamma_{ij}$).

3.4. Implementation in Undominated Strategies

We briefly describe the solution concept we use in our model and under which our notion of externality resistance is realized, namely *implementation in undominated strategies*. Intuitively, we say that a given property P (here, externality resistance) is implemented in undominated strategies if, for every agent i there is a set of strategies D_i so that playing within D_i is a kind of dominant strategy for every player i. This means that no matter what the other players' strategies are, there is some strategy in D_i that maximizes the utility of player i. In addition, obviously P must be satisfied for all possible input strategies in the product space $\prod_{i=1}^n D_i$. For a more formal description we refer to [7, Sec. 1.4] as well as [18] that have utilized this solution concept before. In our specific model, the idea is that as long as the agents stay in strategies that are close enough to truth-telling they are safe. Deviating from the set of strategies D_i will be an absurd choice for agent i, a dominated strategy.

4. Protocol Composability

In this section we explore the bigger picture behind our proposed notion and constructions of Sec. 3. We discuss the underlying schema that achieves resistance towards externalities while still approximating the mechanism designer's objective (e.g. social welfare, Thm. 6).

4.1. Boosting Truthfulness

When we look at Thm. 5, we see that the key property of rVCG (Def. 4) is that the following two values are approximately equal for all agents:

- the utility they end up with in the model *with* externalities after running rVCG, and
- the utility they would have ended up with in an ideal *no*-externalities model after running the traditional VCG mechanism.

In other words, while all agents still bid so as to maximize their new, complex *externality-modified* utilities, they end up with a base utility that is approximately what it would have been if all agents bid so as to maximize their *base* utility. Thus, these externally-resistant mechanisms try to simulate the agents' behavior in an externalities-free utopia and, as a result, they manage to approximate optimal social welfare and revenue.

Above all, what these mechanisms achieve is boosting truthfulness by enforcing incentive-compatibility in this challenging externalities-modified utility model. The key design feature is that of *composability*. If we look at rVCG (Def. 4) we will see that it *randomizes* over *strongly truthful* mechanisms. In particular, it uses the advantage of a strongly truthful mechanism that punishes agents for misbehaving in order to forcefully extract truthful reporting by the agents. It does so by running with some (small) probability such a punishing protocol on a random agent. With the remaining probability we run a standard truthful mechanism that performs optimally with respect to base utilities. In

other words, we enrich mechanisms that perform well in the traditional externalities-free model by *composing* them with small, powerful, strongly truthful subroutines.

Such a composable design paradigm, where different mechanisms are combined to boost truthfulness, has been used before in MD settings, e.g. utilizing differential privacy [19], and more subtly in the scoring rules [20,21] and responsive lotteries [22]. However, in [7] the first step is made towards the systematic study of this scheme and the quantification of the performance of the composable mechanisms. Also, this is the first time when this design is used to achieve externality-resistance. Furthermore, our construction has the advantage that it is readily applicable to multidimensional MD settings, such as multi item auctions and scheduling jobs to machines.

4.2. *Extensions and Multidimensional Domains*

The externality-resistant mechanism rVCG, presented in Sec. 3.3, was applied in a simple, single-dimensional auction setting with only a single item for sale. We want to extend this powerful externality-resistant idea and the composable scheme described in Sec. 4.1 to incorporate more involved *multidimensional* settings with many items, or the scheduling problem.

It turns out that our construction is generic enough to achieve this in a very straightforward way. Consider, for example, the MD scheduling problem of minimizing the makespan of unrelated parallel machines (Sec. 1.2.3). We show how to take advantage of strongly truthful mechanisms to give an unexpected solution to this problem. We will give a mechanism for the problem under the following assumptions:

- The execution times are bounded, in particular we assume that $t_{i,j} \in [L, H]$.
- As in the classical version of the problem, each task must be executed at least once, but in our version it may be executed more than once, even by the same machine [5]. When a machine executes the same task many times, we assume that it pays the same cost t_{ij} for every execution.
- The solution concept for the truthfulness of our mechanism is not dominant strategies but *undominated strategies* (Sec. 3.4).

The mechanism is defined by two parameters $\delta \in [0, 1]$ and r.

- The players declare their values $\tilde{t}_{ij}$ that can differ from the real values t_{ij}.
- With the probability $1 - \delta$, using the declared values, assign the tasks optimally to the players[6].
- With the remaining probability δ, for every player i and every task, run the truth-extracting LM mechanism (Def. 3), like in the case of the externality-resistant VCG (Def. 4), r times using the declared values $\tilde{t}_{ij}$ from the first step. In fact, we need only to simulate LM once, pretending that the execution time of every task has been scaled up by a factor of r.

---

[5]The proofs of Nisan and Ronen that give a lower bound of 2 and an upper bound of *n* for the approximation ratio can be easily extended to this variant of the scheduling problem. The same holds for the lower bound of truthful-in-expectation mechanisms.

[6]Finding or even approximating the optimal allocation with a factor of 1.5 is an NP-hard problem [23], but this is not a concern here, as we focus on the game-theoretic difficulties of the problem. We can replace this part with an approximation algorithm to obtain a polynomial-time approximation mechanism.

Theorem 7. *For every $\delta > 0$ and $\varepsilon > 0$, we can choose the parameter r so that with probability $1 - \delta$ the mechanism has the approximation ratio $1 + \varepsilon$ and makes no payments; the result holds as long as the players do not play a dominated strategy. This, for example, is achieved for every*

$$r \geq 8n^2 m \frac{H^2}{L^2} \frac{1}{\delta \cdot \varepsilon^2} \quad .$$

Proof. The main idea is that if a machine lies even for one task by more than $\varepsilon_0 = \frac{L}{2n}\varepsilon$, the expected cost of the lie in the truth extraction part of the mechanism will exceed any possible gain. Therefore, the truth-telling strategy dominates any strategy that lies by more than ε_0.

We now proceed with the calculations. If a machine lies about one of its tasks by at least an additive term ε_0, it will pay an expected cost of at least $r\delta\frac{1}{2}\frac{\varepsilon_0^2}{H-L}$. The maximum gain from such a lie is to decrease (with the probability $1 - \delta$) its load from mH (the maximum possible makespan) to 0. So the expected gain is at most $(1-\delta)mH \leq mH$, while the loss is at least $r\delta\frac{1}{2}\frac{\varepsilon_0^2}{H-L}$. If we select the parameters so that

$$r\delta\frac{1}{2}\frac{\varepsilon_0^2}{H-L} \geq mH \quad , \tag{10}$$

no machine will have an incentive to lie by more than ε_0, i.e., $|\tilde{t}_{i,j} - t_{i,j}| \leq \varepsilon_0$. But then the makespan computed by the mechanism cannot be more than $m\varepsilon_0$ longer than the optimal makespan: $\text{Makespan}(\tilde{t}) \leq \text{Makespan}(t) + m\varepsilon_0$. We can use the trivial lower bound $\text{Makespan}(t) \geq mL/n$ (or equivalently $n\,\text{Makespan}(t)/(mL) \geq 1$) to bound the makespan of $\tilde{t}$:

$$\text{Makespan}(\tilde{t}) \leq \text{Makespan}(t) + m\varepsilon_0$$

$$\leq \text{Makespan}(t) + m\varepsilon_0\frac{n\,\text{Makespan}(t)}{mL}$$

$$= \left(1 + \frac{n\varepsilon_0}{L}\right)\text{Makespan}(t)$$

$$= (1+\varepsilon)\,\text{Makespan}(t) \quad ,$$

where $\varepsilon = \frac{n\varepsilon_0}{L}$. We can make the value of ε as close to 0 as we want by choosing an appropriately high value for r. Constraint (10) shows that $r = \Theta(\delta\varepsilon^{-2})$ is enough. Therefore, with the probability $1 - \delta$, the makespan of the declared values is $(1+\varepsilon)$-approximate, for every fixed $\varepsilon > 1$. $\qquad\square$

References

[1] John von Neumann and Oskar Morgenstern. *Theory of games and economic behavior*. Princeton University Press, 3rd edition, 1953.

[2] Gillat Kol and Moni Naor. Cryptography and game theory: designing protocols for exchanging information. In *Proceedings of the 5th conference on Theory of cryptography*, TCC'08, pages 320–339, Berlin, Heidelberg, 2008. Springer-Verlag.

[3]   Yevgeniy Dodis and Tal Rabin. Cryptography and game theory. In Noam Nisan, Tim Roughgarden, Éva Tardos, and Vijay Vazirani, editors, *Algorithmic Game Theory*, chapter 8, pages 181–206. Cambridge University Press, 2007.

[4]   Ittai Abraham, Danny Dolev, Rica Gonen, and Joe Halpern. Distributed computing meets game theory: robust mechanisms for rational secret sharing and multiparty computation. In *Proceedings of the twenty-fifth annual ACM symposium on Principles of distributed computing*, PODC '06, pages 53–62, New York, NY, USA, 2006. ACM.

[5]   S. Izmalkov, S. Micali, and M. Lepinski. Rational secure computation and ideal mechanism design. In *Foundations of Computer Science, 2005. FOCS 2005. 46th Annual IEEE Symposium on*, pages 585–594. IEEE, 2005.

[6]   Y. Dodis, S. Halevi, and T. Rabin. A cryptographic solution to a game theoretic problem. In *Advances in Cryptology—CRYPTO 2000*, pages 112–130. Springer, 2000.

[7]   Amos Fiat, Anna R. Karlin, Elias Koutsoupias, and Angelina Vidali. Approaching utopia: Strong truthfulness and externality-resistant mechanisms. In *Innovations in Theoretical Computer Science (ITCS)*, August 2013. http://arxiv.org/abs/1208.3939v1.

[8]   N. Nisan and A. Ronen. Algorithmic mechanism design. *Games and Economic Behavior*, 35(1/2):166–196, 2001.

[9]   Noam Nisan. Introduction to mechanism design (for computer scientists). In Noam Nisan, Tim Roughgarden, Éva Tardos, and Vijay Vazirani, editors, *Algorithmic Game Theory*, chapter 9. Cambridge University Press, 2007.

[10] M.J. Osborne and A. Rubinstein. *A Course in Game Theory*. MIT Press, 1994.

[11] Roger B. Myerson. Optimal auction design. *Mathematics of Operations Research*, 6(1):58–73, 1981.

[12] Jean-Charles Rochet. The taxation principle and multi-time hamilton-jacobi equations. *Journal of Mathematical Economics*, 14(2):113 – 128, 1985.

[13] William Vickrey. Counterspeculation, auctions and competitive sealed tenders. *The Journal of Finance*, 16(1):8–37, March 1961.

[14] E.H. Clarke. Multipart pricing of public goods. *Public Choice*, 11(1):17–33, 1971.

[15] T. Groves. Incentives in Teams. *Econometrica*, 41(4):617–631, 1973.

[16] George Christodoulou and Elias Koutsoupias. Mechanism design for scheduling. *Bulletin of the EATCS*, 97:40–59, 2009.

[17] George Christodoulou, Elias Koutsoupias, and Annamária Kovács. Mechanism design for fractional scheduling on unrelated machines. *ACM Trans. Algorithms*, 6(2):38:1–38:18, April 2010.

[18] M. Babaioff, R. Lavi, and E. Pavlov. Single-value combinatorial auctions and implementation in undominated strategies. In *Proceedings of the seventeenth annual ACM-SIAM symposium on Discrete algorithm*, pages 1054–1063. ACM, 2006.

[19] K. Nissim, R. Smorodinsky, and M. Tennenholtz. Approximately optimal mechanism design via differential privacy. In *Proceedings of the 3rd Innovations in Theoretical Computer Science Conference*, pages 203–213. ACM, 2012.

[20] G.W. Brier. Verification of forecasts expressed in terms of probability. *Monthly weather review*, 78(1):1–3, 1950.

[21] J. Eric Bickel. Some comparisons among quadratic, spherical, and logarithmic scoring rules. *Decision Analysis*, 4(2):49–65, June 2007.

[22] U. Feige and M. Tennenholtz. Responsive lotteries. *Algorithmic Game Theory*, pages 150–161, 2010.

[23] J.K. Lenstra, D.B. Shmoys, and É. Tardos. Approximation algorithms for scheduling unrelated parallel machines. *Mathematical Programming*, 46(1):259–271, 1990.

Chapter 9
Verifiable Computation in Multiparty Protocols with Honest Majority

Alisa PANKOVA [a] and Peeter LAUD [a]

[a] *Cybernetica AS, Estonia*

Abstract. We present a generic method for turning passively secure protocols into protocols secure against covert attacks. This method adds to the protocol a post-execution verification phase that allows a misbehaving party to escape detection only with negligible probability. The execution phase, after which the computed protocol result is already available to the parties, has only negligible overhead added by our method.

The method uses shared verification based on linear probabilistically checkable proofs. The checks are done in zero-knowledge, thereby preserving the privacy guarantees of the original protocol. This method is inspired by recent results in verifiable computation, adapting them to the multiparty setting and significantly lowering their computational costs for the provers. The verification is straightforward to apply to protocols over finite fields.

A longer preprocessing phase can be introduced to shorten the verification phase even more. Beaver triples can be used to make it possible to verify the entire protocol execution locally on shares, leaving for verification just some linear combinations that do not need complex zero-knowledge proofs. Using preprocessing provides a natural way of verifying computation over rings of the size of 2^n.

Introduction

Any multiparty computation can be performed so that the participants only learn their own outputs and nothing else [1]. While the generic construction is expensive in computation and communication, the result has sparked research activities in secure multiparty computation (SMC), with results that are impressive both performance-wise [2,3,4,5] as well as in the variety of concrete problems that have been tackled [6,7,8,9]. From the start, two kinds of adversaries — passive and active — have been considered in the construction of SMC protocols, with the highest performance and the greatest variety achieved for protocol sets secure against passive adversaries.

Verifiable computation (VC) [10] allows a weak client to outsource a computation to a more powerful server that accompanies the computed result with proof of correct computation, the verification of which by the client is cheaper than performing the computation itself. VC can be used to strengthen protocols secure against passive adversaries — after executing the protocol, the parties can prove to each other that they have correctly followed the protocol. If the majority of the parties are honest (an assumption which is made also by the most efficient SMC protocol sets secure against passive adversaries),

then the resulting protocol will satisfy a strong version of *covert security* [11], where any deviations from the protocol are guaranteed to be discovered and reported. Unfortunately, existing approaches to VC have a large computational overhead for the server/prover. Typically, if the computation is represented as an arithmetic circuit C, the prover has to perform $\Omega(|C|)$ public-key operations in order to ensure its good behavior, as well as to protect its privacy.

In this work we show that in the multiparty context with an honest majority, these public-key operations are not necessary. Instead, verifications can be done in a distributed manner, in a way that provides the same security properties. For this, we apply the ideas of existing VC approaches based on linear probabilistically checkable proofs (PCPs) [12], and combine them with linear secret sharing, which we use also for commitments. We end up with a protocol transformation that makes the executions of any protocol (and not just SMC protocols) verifiable afterwards. Our transformation commits the randomness (this takes place offline), inputs, and the communication of the participants. The commitments are cheap, as they are based on digital signatures, and do not add a significant overhead to the execution phase. The results of the protocol are available after the execution. The verification can take place at any time after the execution. Dedicated high-bandwidth high-latency communication channels can be potentially used for it. The verification itself is succinct. The proof is generated in $O(|C|\log|C|)$ field operations, but the computation is local. The generation of challenges costs $O(1)$ in communication and $O(|C|)$ in local computation.

We present our protocol transformation as a functionality in the universal composability (UC) framework, which is described more precisely in Chapter 1. After reviewing related work in Sec. 1, we describe the ideal functionality in Sec. 2 and its implementation in Sec. 4. Before the latter, we give an overview of the existing building blocks we use in Sec. 3. We estimate the computational overhead of our transformation in Sec. 5.

Apart from increasing the security of SMC protocols, our transformation can be used to add verifiability to other protocols. In Sec. 6 we demonstrate how a verifiable secret sharing (VSS) scheme can be constructed. We compare it with state-of-the-art VSS schemes and find that despite much higher generality, our construction enjoys similar complexity.

In order to make the verification phase even more efficient, in Sec. 7 we propose to push more computation into the preprocessing phase. This allows to simplify the final zero-knowledge proofs significantly.

1. Related Work

The property brought by our protocol transformation is similar to security against *covert adversaries* [11] that are prevented from deviating from the prescribed protocol by a non-negligible chance of getting caught.

A similar transformation, applicable to protocols of a certain structure, was introduced by Damgård et al. [13]. They run several instances of the initial protocol, where only one instance is run on real inputs, and the other on randomly generated shares. No party should be able to distinguish the protocol executed on real inputs from the protocol executed on random inputs. In the end, the committed traces of the random executions are revealed by each party, and everyone may check if a party acted honestly in the

random executions. This way, in the beginning all the inputs must be reshared, and the computation must leak no information about the inputs, so that no party can guess which inputs are real and which are random. Hence, the transformation does not allow using the advantage gained from specific sharings of inputs between the parties (where a party can recognize its input), or deliberated leakage of the information that will be published in the end anyway. The probability of cheating decreases linearly with the number of dummy executions.

Compared to the transformation of [13], ours is more general, has lower overhead in the execution phase, and is guaranteed to catch the deviating parties. Our transformation can handle protocols where some of the results are made available to the computing parties already before the end of the protocol. This may significantly decrease the complexity of the protocol [8]. A good property of their construction is its black box nature, which our transformation does not have. Hence, different transformations may be preferable in different situations.

Many works have been dedicated to short verifications of solutions to NP-complete problems. Probabilistically checkable proofs [14] allow verifying a possibly long proof by querying a small number of its bits. Micali [15] has presented *computationally sound proofs* where the verification is not perfect, and the proof can be forged, but it is computationally hard to do. Kilian [16] proposed interactive probabilistically checkable proofs using bit commitments. A certain class of *linear* probabilistically checkable proofs [12] allows making argument systems much simpler and more general.

In computation verification, the prover has to prove that given the valuations of certain wires of a circuit, there is a correct valuation of all the other wires so that the computation is correct with respect to the given circuit. Verifiable computation can in general be based not only on the PCP theorem. In [10], Yao's garbled circuits [17] are executed using fully homomorphic encryption (see Chapter 1 for details). Quadratic span programs for Boolean circuits and quadratic arithmetic programs for arithmetic circuits without PCP have first been proposed in [18], later extended to PCP by [19], and further optimized and improved in [20,21,22]. Particular implementations of verifiable computations have been done, for example, in [21,22,23].

The goal of our transformation is to provide security against a certain form of active attackers. SMC protocols secure against active attackers have been known for a long time [1,24]. SPDZ [4,25] is currently the SMC protocol set secure against active adversaries with the best online performance achieved through extensive offline precomputations (see Chapter 1 for details of SPDZ protocols). Similarly to several other protocol sets, SPDZ provides only a minimum amount of protocols to cooperatively evaluate an arithmetic circuit. We note that very recently, a form of post-execution verifiability has been proposed for SPDZ [26].

2. Ideal Functionality

We use the universal composability (UC) framework [27] to specify our verifiable execution functionality. We have n parties (indexed by $[n] = \{1, \ldots, n\}$), where $\mathcal{C} \subseteq [n]$ are corrupted for $|\mathcal{C}| = t < n/2$ (we denote $\mathcal{H} = [n] \backslash \mathcal{C}$). 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, either over a finite field $\mathbb{F}$ or over rings $\mathbb{Z}_{n_1}, \ldots, \mathbb{Z}_{n_K}$. We define the following gate operations for such a circuit:

In the beginning, $\mathcal{F}_{vmpc}$ gets from $\mathcal{Z}$ for each party P_i the message $(\text{circuits}, i, (C_{ij}^\ell)_{i,j,\ell=1,1,1}^{n,n,r})$ and forwards them all to $\mathcal{A}_S$. For each $i \in \mathcal{H}$ [resp $i \in \mathcal{C}$], $\mathcal{F}_{vmpc}$ gets $(\text{input}, \mathbf{x}_i)$ from $\mathcal{Z}$ [resp. $\mathcal{A}_S$]. For each $i \in [n]$, $\mathcal{F}_{vmpc}$ randomly generates $\mathbf{r}_i$. For each $i \in \mathcal{C}$, it sends $(\text{randomness}, i, \mathbf{r}_i)$ to $\mathcal{A}_S$.
For each round $\ell \in [r]$, $i \in \mathcal{H}$ and $j \in [n]$, $\mathcal{F}_{vmpc}$ uses C_{ij}^ℓ to compute the message $\mathbf{m}_{ij}^\ell$. For all $j \in \mathcal{C}$, it sends $\mathbf{m}_{ij}^\ell$ to $\mathcal{A}_S$. For each $j \in \mathcal{C}$ and $i \in \mathcal{H}$, it receives $\mathbf{m}_{ji}^\ell$ from $\mathcal{A}_S$.
After r rounds, $\mathcal{F}_{vmpc}$ sends $(\text{output}, \mathbf{m}_{1i}^r, \ldots, \mathbf{m}_{ni}^r)$ to each party P_i with $i \in \mathcal{H}$. Let $r' = r$ and $\mathcal{B}_0 = \emptyset$.
Alternatively, **at any time** before outputs are delivered to parties, $\mathcal{A}_S$ may send $(\text{stop}, \mathcal{B}_0)$ to $\mathcal{F}_{vmpc}$, with $\mathcal{B}_0 \subseteq \mathcal{C}$. In this case the outputs are not sent. Let $r' \in \{0, \ldots, r-1\}$ be the last completed round.
After r' rounds, $\mathcal{A}_S$ sends to $\mathcal{F}_{vmpc}$ the messages $\mathbf{m}_{ij}^\ell$ for $\ell \in [r']$ and $i, j \in \mathcal{C}$.
$\mathcal{F}_{vmpc}$ defines $\mathcal{M} = \mathcal{B}_0 \cup \{i \in \mathcal{C} \,|\, \exists j \in [n], \ell \in [r'] : \mathbf{m}_{ij}^\ell \neq C_{ij}^\ell(\mathbf{x}_i, \mathbf{r}_i, \mathbf{m}_{1i}^1, \ldots, \mathbf{m}_{ni}^{\ell-1})\}$.
Finally, for each $i \in \mathcal{H}$, $\mathcal{A}_S$ sends $(\text{blame}, i, \mathcal{B}_i)$ to $\mathcal{F}_{vmpc}$, with $\mathcal{M} \subseteq \mathcal{B}_i \subseteq \mathcal{C}$. $\mathcal{F}_{vmpc}$ forwards this message to P_i.

Figure 1. The ideal functionality for verifiable computations

- The operations $+$ (addition) and $*$ (multiplication) are in $\mathbb{F}$ or in rings $\mathbb{Z}_{n_1}, \ldots, \mathbb{Z}_{n_K}$ ($n_i < n_j$ for $i < j$).
- The operations trunc and zext are between rings. Let $x \in \mathbb{Z}_{n_x}$, $y \in \mathbb{Z}_{n_y}$, $n_x < n_y$.
  * $x = \text{trunc}(y)$ computes $x = y \mod n_x$, going from a larger ring to a smaller ring.
  * $y = \text{zext}(x)$ takes x and uses the same value in n_y. It can be treated as taking the bits of x and extending them with zero bits.
- The operation bits are from an arbitrary ring $\mathbb{Z}_n$ to $(\mathbb{Z}_2)^{\log n}$. This operation 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. Although the current set of gates is sufficient to represent any other operation, the verifications designed for special gates may be more efficient. For example, introducing the division gate $c = a/b$ explicitly would allow to verify it as $a = b * c$ instead of expressing the division through addition and multiplication. In this work, we do not define any other gates, as the verification of most standard operations is fairly straightforward, assuming that bit decomposition is available.

The circuit C_{ij}^ℓ computes the ℓ-th round messages $\mathbf{m}_{ij}^\ell$ to all parties $j \in [n]$ from the input $\mathbf{x}_i$, randomness $\mathbf{r}_i$ and the messages P_i has received before (all values $\mathbf{x}_i, \mathbf{r}_i, \mathbf{m}_{ij}^\ell$ are vectors over rings $\mathbb{Z}_n$ or a finite field $\mathbb{F}$). We define that the messages received during the r-th round comprise the *output of the protocol*. The ideal functionality $\mathcal{F}_{vmpc}$, running in parallel with the environment $\mathcal{Z}$ and the adversary $\mathcal{A}_S$, is given in Fig. 1.

We see that $\mathcal{M}$ is the set of all parties that deviate from the protocol. Our verifiability property is very strong as *all* of them will be reported to *all* honest parties. Even if only *some* rounds of the protocol are computed, all the parties that deviated from the protocol in completed rounds will be detected. Also, no honest parties (in $\mathcal{H}$) can be falsely blamed. We also note that if $\mathcal{M} = \emptyset$, then $\mathcal{A}_S$ does not learn anything that a semi-honest adversary could not learn.

$\mathcal{F}_{transmit}$ works with unique message identifiers *mid*, encoding a sender $s(mid) \in [n]$, a receiver $r(mid) \in [n]$, and a party $f(mid) \in [n]$ to whom the message should be forwarded by the receiver (if no forwarding is foreseen then $f(mid) = r(mid)$).

Secure transmit: Receiving (transmit, *mid*, *m*) from $P_{s(mid)}$ and (transmit, *mid*) from all (other) honest parties, store $(mid, m, r(mid))$, mark it as undelivered, and output $(mid, |m|)$ to the adversary. If the input of $P_{s(mid)}$ is invalid (or there is no input), and $P_{r(mid)}$ is honest, then output (corrupt, $s(mid)$) to all parties.

Secure broadcast: Receiving (broadcast, *mid*, *m*) from $P_{s(mid)}$ and (broadcast, *mid*) from all honest parties, store (mid, m, bc), mark it as undelivered, output $(mid, |m|)$ to the adversary. If the input of $P_{s(mid)}$ is invalid, output (corrupt, $s(mid)$) to all parties.

Synchronous delivery: At the end of each round, for each undelivered (mid, m, r) send (mid, m) to P_r; mark (mid, m, r) as delivered. For each undelivered (mid, m, bc), send (mid, m) to each party and the adversary; mark (mid, m, bc) as delivered.

Forward received message: On input (forward, *mid*) from $P_{r(mid)}$ after (mid, m) has been delivered to $P_{r(mid)}$, and receiving (forward, *mid*) from all honest parties, store $(mid, m, f(mid))$, mark as undelivered, output $(mid, |m|)$ to the adversary. If the input of $P_{r(mid)}$ is invalid, and $P_{f(mid)}$ is honest, output (corrupt, $r(mid)$) to all parties.

Publish received message: On input (publish, *mid*) from the party $P_{f(mid)}$, which at any point received (mid, m), output (mid, m) to each party, and also to the adversary.

Do not commit corrupt to corrupt: If for some *mid* both $P_{s(mid)}, P_{r(mid)}$ are corrupt, then on input (forward, *mid*) the adversary can ask $\mathcal{F}_{transmit}$ to output (mid, m') to $P_{f(mid)}$ for any m'. If, additionally, $P_{f(mid)}$ is corrupt, then the adversary can ask $\mathcal{F}_{transmit}$ to output (mid, m') to all honest parties.

Figure 2. Ideal functionality $\mathcal{F}_{transmit}$

3. Building Blocks

Throughout this work, bold letters $\mathbf{x}$ denote vectors, where x_i denotes the *i*-th coordinate of $\mathbf{x}$. Concatenation of $\mathbf{x}$ and $\mathbf{y}$ is denoted by $(\mathbf{x} \| \mathbf{y})$, and their scalar product by $\langle \mathbf{x}, \mathbf{y} \rangle$, which is defined (only if $|\mathbf{x}| = |\mathbf{y}|$) as $\langle \mathbf{x}, \mathbf{y} \rangle = \sum_{i=1}^{|\mathbf{x}|} x_i y_i$. Our implementation uses a number of previously defined subprotocols and algorithm sets.

Message transmission. For message transmission between parties, we use functionality $\mathcal{F}_{transmit}$ [13], which allows one to provably reveal to third parties the messages that one received during the protocol, and to further transfer such revealed messages. Our definition of $\mathcal{F}_{transmit}$ differs from that of Damgård et al. [13] $\mathcal{F}_{transmit}$ by supporting the forwarding of received messages as well as broadcasting as a part of the outer protocol. The definition of the ideal functionality of $\mathcal{F}_{transmit}$ is shown in Fig. 2. The real implementation of the transmission functionality is built on top of signatures. This makes the implementation very efficient, as hash trees allow several messages (sent in the same round) to be signed with almost the same computation effort as a single one [28], and signatures can be verified in batches [29]. An implementation of $\mathcal{F}_{transmit}$ is given in [30].

Shamir's secret sharing. For commitments, we use (n,t) Shamir's secret sharing [31], where any *t* parties are able to recover the secret, but fewer than *t* are not (see Chapter 1 for details). By sharing a *vector* $\mathbf{x}$ over $\mathbb{F}$ into vectors $\mathbf{x}^1, \ldots, \mathbf{x}^n$ we mean that each *i*-th entry $x_i \in \mathbb{F}$ of $\mathbf{x}$ is shared into the *i*-th entries $x_i^1 \in \mathbb{F}, \ldots, x_i^n \in \mathbb{F}$ of $\mathbf{x}^1, \ldots, \mathbf{x}^n$. In this way, for each $T = \{i_1, \ldots, i_t\} \subseteq [n]$, the entries can be restored as $x_i = \sum_{j=1}^{t} b_{Tj} x_i^{i_j}$ for certain

constants b_{Tj}, and hence, $\mathbf{x} = \sum_{j=1}^{t} b_{Tj} \mathbf{x}^{i_j}$. The linearity extends to scalar products: if a vector π is shared to $\pi^1, \ldots, \pi^n$, then for any vector $\mathbf{q}$ and $T = \{i_1, \ldots, i_t\}$, we have $\sum_{j=1}^{t} b_{Tj} \langle \pi^{i_j}, \mathbf{q} \rangle = \langle \pi, \mathbf{q} \rangle$.

We note that sharing a value x as $x^1 = \cdots = x^k = x$ is valid, i.e. x can be restored from $x^{i_1}, \ldots, x^{i_t}$ by forming the same linear combination. In our implementation of the verifiable computation functionality, we use such sharing for values that become public due to the adversary's actions.

Beaver triples.   This primitive will be used in cases where we want to make the verification phase more efficient by pushing some computation into the preprocessing phase. Beaver triples [32] are triples of values (a, b, c) in a ring $\mathbb{Z}_n$, so that $a, b \xleftarrow{\$} R$, and $c = a \cdot b$. Precomputing such triples can be used to linearize multiplications. For example, if we want to multiply $x \cdot y$, 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 $x \cdot y = (x' + r_x)(y' + r_y) = x'y' + r_x y' + x'r_y + r_x r_y = x'y' + r_x y' + x'r_y + r_{xy}$. Differently from standard usage (like in SPDZ), we do not use these triples in the original protocol, but instead use them to simplify the verification phase. See Chapter 1 for details on the generation and usage of such triples.

Linear PCP.   This primitive forms the basis of our verification. Before giving its definition, let us formally state when a protocol is statistically privacy-preserving.

Definition 1 (δ-private protocol [33]) *Let Π be a multiparty protocol that takes input $\mathbf{x}$ from honest parties and $\mathbf{y}$ from adversarially controlled parties. The protocol Π is δ-private against a class of adversaries $\mathcal{A}$ if there exists a simulator Sim, so that for all adversaries $A \in \mathcal{A}$ and inputs $\mathbf{x}, \mathbf{y}$, $\left| \mathbf{Pr}\left[A^{\Pi(\mathbf{x},\mathbf{y})}(\mathbf{y}) = 1 \right] - \mathbf{Pr}\left[A^{\mathsf{Sim}(\mathbf{y})}(\mathbf{y}) = 1 \right] \right| \leq \delta$.*

Definition 2 (Linear probabilistically checkable proof (LPCP) [19]) *Let $\mathbb{F}$ be a finite field, $\upsilon, \omega \in \mathbb{N}$, $R \subseteq \mathbb{F}^{\upsilon} \times \mathbb{F}^{\omega}$. Let P and Q be probabilistic algorithms, and D a deterministic algorithm. The pair (P, V), where $V = (Q, D)$ is a d-query δ-statistical honest verifier zero-knowledge (HVZK) linear PCP for R with the knowledge error ε and the query length m, if the following holds.*

Syntax *On input $\mathbf{v} \in \mathbb{F}^{\upsilon}$ and $\mathbf{w} \in \mathbb{F}^{\omega}$, algorithm P computes $\pi \in \mathbb{F}^m$. The algorithm Q randomly generates d vectors $\mathbf{q}_1, \ldots \mathbf{q}_d \in \mathbb{F}^m$ and some state information $\mathbf{u}$. On input $\mathbf{v}, \mathbf{u}$, as well as $a_1, \ldots, a_d \in \mathbb{F}$, the algorithm D accepts or rejects. Let $V^{\pi}(\mathbf{v})$ denote the execution of Q followed by the execution of V on $\mathbf{v}$, the output $\mathbf{u}$ of Q, and $a_1, \ldots, a_d$, where $a_i = \langle \pi, \mathbf{q}_i \rangle$.*

Completeness *For every $(\mathbf{v}, \mathbf{w}) \in R$, the output of $P(\mathbf{v}, \mathbf{w})$ is a vector $\pi \in \mathbb{F}^m$ so that $V^{\pi}(\mathbf{v})$ accepts with probability 1.*

Knowledge *There is a knowledge extractor E so that for every vector $\pi^* \in \mathbb{F}^m$, if $\Pr\left[V^{\pi^*}(\mathbf{v}) \text{ accepts} \right] \geq \varepsilon$ then $E(\pi^*, \mathbf{v})$ outputs $\mathbf{w}$ so that $(\mathbf{v}, \mathbf{w}) \in R$.*

Honest Verifier Zero-Knowledge *The protocol between an honest prover executing $\pi \leftarrow P(\mathbf{v}, \mathbf{w})$ and an adversarial verifier executing $V^{\pi}(\mathbf{v})$ with common input $\mathbf{v}$ and the prover's input $\mathbf{w}$ is δ-private for the class of passive adversaries.*

Similarly to different approaches to verifiable computation [18,19,20,21,22], in our work we let the relation R correspond to the circuit C executed by the party whose observance of the protocol is being verified. In this correspondence, $\mathbf{v}$ is the tuple of all inputs, outputs, and the used random values of that party. The vector $\mathbf{w}$ extends $\mathbf{v}$ with the results of all intermediate computations by that party. Differently from existing approaches, $\mathbf{v}$ itself is private. Hence, it is unclear how the decision algorithm D can be executed on it. Therefore, we do not use D as a black box, but build our solution on top of a particular LPCP [21].

The LPCP algorithms used by Ben-Sasson et al. [21] are statistical HVZK. Namely, the values $\langle \pi, \mathbf{q}_i \rangle$ do not reveal any private information about π, unless the random seed $\tau \in \mathbb{F}$ for Q is chosen in badly, which happens with negligible probability for a sufficiently large field. In [21], Q generates 5 challenges $\mathbf{q}_1, \ldots, \mathbf{q}_5$ and the state information $\mathbf{u}$ with the length $|\mathbf{v}| + 2$. Given the query results $a_i = \langle \pi, \mathbf{q}_i \rangle$ for $i \in \{1, \ldots, 5\}$ and the state information $\mathbf{u} = (u_0, u_1, \ldots, u_{|\mathbf{v}|+1})$, the following two checks have to pass:

$$a_1 a_2 - a_3 - a_4 u_{|\mathbf{v}|+1} = 0, \tag{$*$}$$
$$a_5 - \langle (1\|\mathbf{v}), (u_0, u_1, \ldots, u_{|\mathbf{v}|}) \rangle = 0. \tag{$**$}$$

Here $(*)$ is used to show the existence of $\mathbf{w}$, and $(**)$ shows that a certain segment of π equals $(1\|\mathbf{v})$ [21]. Throughout this work, we reorder the entries of π compared to [21] and write $\pi = (\mathbf{p}\|1\|\mathbf{v})$, where $\mathbf{p}$ represents all the other entries of π, as defined in [21]. The challenges $\mathbf{q}_1, \ldots, \mathbf{q}_5$ are reordered in the same way.

This linear interactive proof can be converted into a zero-knowledge succinct non-interactive argument of knowledge [19]. Unfortunately, it requires homomorphic encryption, and the number of encryptions is linear in the size of the circuit. We show that the availability of honest majority allows the proof to be completed without public-key encryptions.

The multiparty setting introduces a further difference from [21]: the vector $\mathbf{v}$ can no longer be considered public, as it contains a party's private values. Thus, we have to strengthen the HVZK requirement in Def. 2, making $\mathbf{v}$ private to the prover. The LPCP constructions of [21] do not satisfy this strengthened HVZK requirement, but their authors show that this requirement would be satisfied if a_5 were not present. In the following, we propose a construction where just the first check $(*)$ is sufficient, so only $a_1, \ldots, a_4$ have to be published. We prove that the second check $(**)$ will be passed implicitly. We show the following.

Theorem 1 *Given a δ-statistical HVZK instance of the LPCP of Ben-Sasson et al. [21] with the knowledge error ε, any n-party r-round protocol Π can be transformed into an n-party $(r+8)$-round protocol Ξ in the $\mathcal{F}_{transmit}$-hybrid model, which computes the same functionality as Π and achieves covert security against adversaries statically corrupting at most $t < n/2$ parties, where the cheating of any party is detected with probability of at least $(1-\varepsilon)$. If Π is δ'-private against passive adversaries statically corrupting at most t parties, then Ξ is $(\delta' + \delta)$-private against cover adversaries. Under active attacks by at most t parties, the number of rounds of the protocol may at most double.*

Thm. 1 is proved by the construction of the real functionality Sec. 4, as well as the simulator presented in [30]. In the construction, we use the following algorithms implicitly defined by Ben-Sasson et al. [21]:

Circuits: M_i gets from $\mathcal{Z}$ the message $(\text{circuits}, i, (C_{ij}^{\ell})_{i,j,\ell=1,1,1}^{n,n,r})$ and sends it to $\mathcal{A}$.

Randomness generation and commitment: Let $\mathcal{R} = [t+1]$. For all $i \in \mathcal{R}$, $j \in [n]$, M_i generates $\mathbf{r}_{ij}$ for M_j. M_i shares $\mathbf{r}_{ij}$ to n vectors $\mathbf{r}_{ij}^1, \ldots, \mathbf{r}_{ij}^n$ according to $(n, t+1)$ Shamir's scheme. For $j \in [n]$, M_i sends $(\text{transmit}, (\text{r\_share}, i, j, k), \mathbf{r}_{ij}^k)$ to $\mathcal{F}_{transmit}$ for M_k.

Randomness approval: For each $j \in [n] \setminus \{k\}$, $i \in \mathcal{R}$, M_k sends $(\text{forward}, (\text{r\_share}, i, j, k))$ to $\mathcal{F}_{transmit}$ for M_j. Upon receiving $((\text{r\_share}, i, j, k), \mathbf{r}_{ij}^k)$ for all $k \in [n]$, $i \in \mathcal{R}$, M_j checks if the shares comprise a valid $(n, t+1)$ Shamir's sharing. M_j sets $\mathbf{r}_i = \sum_{i \in \mathcal{R}} \mathbf{r}_{ij}$.

Input commitments: M_i with $i \in \mathcal{H}$ [resp. $i \in \mathcal{C}$] gets from $\mathcal{Z}$ [resp. $\mathcal{A}$] the input $\mathbf{x}_i$ and shares it to n vectors $\mathbf{x}_i^1, \ldots, \mathbf{x}_i^n$ according to $(n, t+1)$ Shamir's scheme. For each $k \in [n] \setminus \{i\}$, M_i sends to $\mathcal{F}_{transmit}$ $(\text{transmit}, (\text{x\_share}, i, k), \mathbf{x}_i^k)$ for M_k.

At any time: If $(\text{corrupt}, j)$ comes from $\mathcal{F}_{transmit}$, M_i writes $mlc_i[j] := 1$ and goes to the accusation phase.

Figure 3. Initialization phase of the real functionality

- *witness*$(C, \mathbf{v})$: if $\mathbf{v}$ corresponds to a valid computation of C, it returns a witness $\mathbf{w}$ so that $(\mathbf{v}, \mathbf{w}) \in \mathcal{R}_C$.
- *proof*$(C, \mathbf{v}, \mathbf{w})$ if $(\mathbf{v}, \mathbf{w}) \in \mathcal{R}_C$, it constructs a corresponding proof $\mathbf{p}$.
- *challenge*(C, τ): returns $\mathbf{q}_1, \ldots, \mathbf{q}_5, \mathbf{u}$ that correspond to τ, so that:
    * for any valid proof $\pi = (\mathbf{p} \| 1 \| \mathbf{v})$, where $\mathbf{p}$ is generated by *proof*$(C, \mathbf{v}, \mathbf{w})$ for $(\mathbf{v}, \mathbf{w}) \in \mathcal{R}_C$, the checks $(*)$ and $(**)$ succeed with probability 1;
    * for any proof π^* generated without knowing τ, or such $\mathbf{w}$ that $(\mathbf{v}, \mathbf{w}) \in \mathcal{R}_C$, either $(*)$ or $(**)$ fails, except with negligible probability ε.

4. Real Functionality

Our initial construction works for arithmetic circuits over a finite field $\mathbb{F}$, assuming that the only gate types are $+$ and $*$. In Sec. 7, we show how it can be extended to a circuit over multiple rings, so that the gates trunc, zext, and bits can be added.

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 functionality $\mathcal{F}_{transmit}$. The internal state of each M_i contains a bit-vector mlc_i of the length n, where M_i marks which other parties are acting maliciously. The protocol Π_{vmpc} runs in five phases: initialization, execution, message commitment, verification, and accusation.

In the initialization phase, the inputs $\mathbf{x}_i$ and the randomness $\mathbf{r}_i$ are committed. It is ensured that the randomness indeed comes from uniform distribution. This phase is given in Fig.3. If at any time $(\text{corrupt}, j)$ comes from $\mathcal{F}_{transmit}$, each (uncorrupted) M_i writes $mlc_i[j] := 1$ (for each message $(\text{corrupt}, j)$) and goes to the accusation phase.

In the execution phase, the parties run the original protocol as before, just using $\mathcal{F}_{transmit}$ to exchange the messages. This phase is given in Fig.4. If at any time in some round ℓ the message $(\text{corrupt}, j)$ comes from $\mathcal{F}_{transmit}$ (all uncorrupted machines receive it at the same time), the execution is cut short, no outputs are produced and the protocol continues with the commitment phase.

In the message commitment phase, all the n parties finally commit their sent messages $\mathbf{c}_{ij}^{\ell}$ for each round $\ell \in [r']$ by sharing them to $\mathbf{c}_{ij}^{\ell 1}, \ldots, \mathbf{c}_{ij}^{\ell n}$ according to $(n, t+1)$ Shamir's scheme. This phase is given in Fig. 5. Let $\mathbf{v}_{ij}^{\ell} = (\mathbf{x}_i \| \mathbf{r}_i \| \mathbf{c}_{1i}^{1} \| \cdots \| \mathbf{c}_{ni}^{\ell-1} \| \mathbf{c}_{ij}^{\ell})$ be the

---

For each round ℓ the machine M_i computes $\mathbf{c}_{ij}^{\ell} = C_{ij}^{\ell}(\mathbf{x}_i, \mathbf{r}_i, \mathbf{c}_{1i}^1, \ldots, \mathbf{c}_{ni}^{\ell-1})$ for each $j \in [n]$ and sends to $\mathcal{F}_{transmit}$ the message (transmit, (message, ℓ, i, j), $\mathbf{c}_{ij}^{\ell}$) for M_j.
After r rounds, uncorrupted M_i sends (output, $\mathbf{c}_{1i}^r, \ldots, \mathbf{c}_{ni}^r$) to $\mathcal{Z}$ and sets $r' := r$.
At any time: If (corrupt, j) comes from $\mathcal{F}_{transmit}$, each (uncorrupted) M_i writes $mlc_i[j] := 1$, sets $r' := \ell - 1$ and goes to the message commitment phase.

Figure 4. Execution phase of the real functionality

Message sharing: As a sender, M_i shares $\mathbf{c}_{ij}^{\ell}$ to $\mathbf{c}_{ij}^{\ell 1}, \ldots, \mathbf{c}_{ij}^{\ell n}$ according to $(n, t + 1)$ Shamir's scheme. For each $k \in [n] \setminus \{i\}$, M_i sends to $\mathcal{F}_{transmit}$ the messages (transmit, (c_share, ℓ, i, j, k), $\mathbf{c}_{ij}^{\ell k}$) for M_j.
Message commitment: Upon receiving ((c_share, ℓ, i, j, k), $\mathbf{c}_{ij}^{\ell k}$) from $\mathcal{F}_{transmit}$ for all $k \in [n]$, the machine M_j checks if the shares correspond to the $\mathbf{c}_{ij}^{\ell}$ it has already received. If they do not, M_j sends (publish, (message, ℓ, i, j)) to $\mathcal{F}_{transmit}$, so now everyone sees the values that it has actually received from M_i, and each (uncorrupted) M_k should now use $\mathbf{c}_{ij}^{\ell k} := \mathbf{c}_{ij}^{\ell}$. If the check succeeds, then M_i sends to $\mathcal{F}_{transmit}$ (forward, (c_share, ℓ, i, j, k)) for M_k for all $k \in [n] \setminus \{i\}$.

Figure 5. Message commitment phase of the real functionality

---

vector of inputs and outputs to the circuit C_{ij}^{ℓ} that M_i uses to compute the ℓ-th message to M_j. If the check performed by M_j fails, then M_j has received from M_i enough messages to prove its corruptness to others (but Fig. 5 presents an alternative, by publicly agreeing on $\mathbf{c}_{ij}^{\ell}$). After this phase, M_i has shared $\mathbf{v}_{ij}^{\ell}$ among all n parties. Let $\mathbf{v}_{ij}^{\ell k}$ be the share of $\mathbf{v}_{ij}^{\ell}$ given to machine M_k.

Each M_i generates a witness $\mathbf{w}_{ij}^{\ell} = witness(C_{ij}^{\ell}, \mathbf{v}_{ij}^{\ell})$, a proof $\mathbf{p}_{ij}^{\ell} = proof(C_{ij}^{\ell}, \mathbf{v}_{ij}^{\ell}, \mathbf{w}_{ij}^{\ell})$, and $\pi_{ij}^{\ell} = (\mathbf{p}_{ij}^{\ell} \| 1 \| \mathbf{v}_{ij}^{\ell})$ in the verification phase, as explained in Sec. 3. The vector $\mathbf{p}_{ij}^{\ell}$ is shared to $\mathbf{p}_{ij}^{\ell 1}, \ldots, \mathbf{p}_{ij}^{\ell n}$ according to $(n, t + 1)$ Shamir's scheme.

All parties agree on a random τ, with M_i broadcasting τ_i and τ being their sum. A party refusing to participate is ignored. The communication must be synchronous, with no party P_i learning the values τ_j from others before he has sent his own τ_i. Note that $\mathcal{F}_{transmit}$ already provides this synchronicity. If it were not available, then standard tools (commitments) could be used to achieve fairness.

All (honest) parties generate $\mathbf{q}_{1ij}^{\ell}, \ldots, \mathbf{q}_{4ij}^{\ell}, \mathbf{q}_{5ij}^{\ell}, \mathbf{u}_{ij}^{\ell} = challenge(C_{ij}^{\ell}, \tau)$ for $\ell \in [r']$, $i \in [n]$, $j \in [n]$. In the rest of the protocol, only $\mathbf{q}_{1ij}^{\ell}, \ldots, \mathbf{q}_{4ij}^{\ell}$, and $(u_{ij}^{\ell})_{|\mathbf{v}|+1}$ will be used.

Each M_k computes $\pi_{ij}^{\ell k} = (\mathbf{p}_{ij}^{\ell k} \| 1 \| \mathbf{v}_{ij}^{\ell k}) = (\mathbf{p}_{ij}^{\ell k} \| 1 \| \mathbf{x}_i^k \| \sum_{j \in \mathcal{R}} \mathbf{r}_{ji}^k \| \mathbf{c}_{1i}^{1k} \| \cdots \| \mathbf{c}_{ni}^{\ell-1,k} \| \mathbf{c}_{ij}^{\ell k})$ for verification, and then computes and publishes the values $\langle \pi_{ij}^{\ell k}, \mathbf{q}_{1ij}^{\ell} \rangle, \ldots, \langle \pi_{ij}^{\ell k}, \mathbf{q}_{4ij}^{\ell} \rangle$. M_i checks these values and complains about M_k that has computed them incorrectly. An uncorrupted M_k may disprove the complaint by publishing the proof and message shares that it received. Due to the linearity of scalar product and the fact that all the vectors have been shared according to the same $(n, t + 1)$ Shamir's sharing, if the n scalar product shares correspond to a valid $(n, t + 1)$ Shamir's sharing, the shared value is uniquely defined by any $t + 1$ shares, and hence, by the shares of some $t + 1$ parties that are all from $\mathcal{H}$. Hence, M_i is obliged to use the values it has committed before. The verification phase for C_{ij}^{ℓ} for fixed $\ell \in [r']$, $i \in [n]$, $j \in [n]$ is given in Fig.6. For different C_{ij}^{ℓ}, all the verifications can be done in parallel.

Remaining proof commitment: As the prover, M_i obtains $\mathbf{w}_{ij}^{\ell}$ and $\pi_{ij}^{\ell} = (\mathbf{p}_{ij}^{\ell} \| 1 \| \mathbf{v}_{ij}^{\ell})$ using the algorithms *witness* and *proof*. M_i shares $\mathbf{p}_{ij}^{\ell}$ to $\mathbf{p}_{ij}^{\ell 1}, \ldots, \mathbf{p}_{ij}^{\ell n}$ according to $(n, t+1)$ Shamir's scheme. For each $k \in [n] \setminus \{i\}$, it sends to $\mathcal{F}_{transmit}$ (transmit, $(\mathsf{p\_share}, \ell, i, j, k), \mathbf{p}_{ij}^{\ell k})$ for M_k.

Challenge generation: Each M_k generates random $\tau_k \leftarrow \mathbb{F}$ and sends to $\mathcal{F}_{transmit}$ the message (broadcast, $(\mathsf{challenge\_share}, \ell, i, j, k), \tau_k)$. If some party refuses to participate, its share will be omitted. The challenge randomness is $\tau = \tau_1 + \ldots + \tau_n$. Machine M_k generates $\mathbf{q}_{1ij}^{\ell}, \ldots, \mathbf{q}_{4ij}^{\ell}, \mathbf{q}_{5ij}^{\ell}, \mathbf{u}_{ij}^{\ell} = challenge(C_{ij}^{\ell}, \tau)$, then computes $\pi_{ij}^{\ell k} = (\mathbf{p}_{ij}^{\ell k} \| 1 \| \mathbf{v}_{ij}^{\ell k}) = (\mathbf{p}_{ij}^{\ell k} \| 1 \| \mathbf{x}_i^k \| \sum_{j \in \mathcal{R}} \mathbf{r}_{ji}^k \| \mathbf{c}_{1i}^{1k} \| \cdots \| \mathbf{c}_{ni}^{\ell-1,k} \| \mathbf{c}_{ij}^{\ell k})$, and finally computes and broadcasts $\langle \pi_{ij}^{\ell k}, \mathbf{q}_{1ij}^{\ell} \rangle, \ldots, \langle \pi_{ij}^{\ell k}, \mathbf{q}_{4ij}^{\ell} \rangle$.

Scalar product verification: Each M_i verifies the published $\langle \pi_{ij}^{\ell k}, \mathbf{q}_{sij}^{\ell} \rangle$ for $s \in \{1, \ldots, 4\}$. If M_i finds that M_k has computed the scalar products correctly, it sends to $\mathcal{F}_{transmit}$ the message (broadcast, $(\mathsf{complain}, \ell, i, j, k), 0)$. If some M_k has provided a wrong value, M_i sends to $\mathcal{F}_{transmit}$ (broadcast, $(\mathsf{complain}, \ell, i, j, k), (1, sh_{sij}^{\ell k}))$, where $sh_{sij}^{\ell k}$ is M_i's own version of $\langle \pi_{ij}^{\ell k}, \mathbf{q}_{sij}^{\ell} \rangle$. Everyone waits for a disproof from M_k. An uncorrupted M_k sends to $\mathcal{F}_{transmit}$ the messages (publish, mid) for $mid \in \{(\mathsf{x\_share}, i, k),$ $(\mathsf{r\_share}, 1, i, k), \ldots, (\mathsf{r\_share}, |\mathcal{R}|, i, k), (\mathsf{p\_share}, \ell, i, j, k), (\mathsf{c\_share}, 1, 1, i, k), \ldots,$ $(\mathsf{c\_share}, r', n, i, k), (\mathsf{c\_share}, \ell, i, j, k)\}$. Now everyone may construct $\pi_{ij}^{\ell k}$ and verify whether the version provided by M_i or M_k is correct.

Final verification: Given $\langle \pi_{ij}^{\ell k}, \mathbf{q}_{sij}^{\ell} \rangle$ for all $k \in [n]$, $s \in \{1, \ldots, 4\}$, each machine M_v checks if they indeed correspond to valid $(n, t+1)$ Shamir's sharing, and then locally restores $a_{sij}^{\ell} = \langle \pi_{ij}^{\ell}, \mathbf{q}_{sij}^{\ell} \rangle$ for $s \in \{1, \ldots, 4\}$, and checks $(*)$. If the check succeeds, then M_v accepts the proof of M_i for C_{ij}^{ℓ}. Otherwise it immediately sets $mlc_v[i] := 1$.

Figure 6. Verification phase of the real functionality

Finally, each party M_i sends to $\mathcal{Z}$ the message (blame, $i, \{j \mid mlc_i[j] = 1\}$).

Figure 7. Accusation phase of the real functionality

As described, the probability of cheating successfully in our scheme is proportional to $1/|\mathbb{F}|$. In order to exponentially decrease it, we may run s instances of the verification phase in parallel, as by that time $\mathbf{v}_{ij}^{\ell}$ are already committed. This will not break the HVZK assumption if fresh randomness is used in $\mathbf{p}_{ij}^{\ell}$.

During the message commitment and verification phases, if at any time (corrupt, j) comes from $\mathcal{F}_{transmit}$, the proof for P_j ends with failure, and all uncorrupted machines M_i write $mlc_i[j] := 1$.

Finally, each party outputs the set of parties that it considers malicious. This short phase is given in Fig. 7. The formal UC proof for the real functionality can be found in [30].

5. Efficiency

In this section we estimate the overheads caused by our protocol transformation. The numbers are based on the dominating complexities of the algorithms of linear PCP of [21]. We omit local addition and concatenation of vectors as these are cheap operations.

The preprocessing phase of [21] is done offline and can be re-used, so we will not estimate its complexity here. It can be done with practical overhead [21].

Let n be the number of parties, $t < n/2$ the number of corrupt parties, r the number of rounds, N_g the number of gates, N_w the number of wires, N_x the number of inputs (elements of $\mathbb{F}$), N_r the number of random elements of $\mathbb{F}$, N_c the number of communicated elements of $\mathbb{F}$, and $N_i = N_w - N_x - N_r - N_c$ the number of intermediate wires in the circuit. Then $|\mathbf{v}| = N_x + N_r + N_c$.

Let $\overline{S}(n,k)$ denote the number of field operations used in sharing one field element according to Shamir's scheme with threshold k which is at most nk multiplications. We use $\overline{S^{-1}}(n,k)$ to denote the complexity of verifying if the shares comprise a valid sharing, and of recovering the secret which is also at most nk multiplications. Compared to the original protocol, for each M_i the proposed solution has the following computation/communication overheads.

Initialization: Do Shamir's sharing of one vector of the length N_x in $N_x \cdot \overline{S}(n, t+1)$ field operations. Transmit $t + 1$ vectors of the length N_r and one vector of the length N_x to each other party. Do $t + 1$ recoverings in $(t + 1) \cdot N_r \cdot \overline{S^{-1}}(n, t+1)$. The parties that generate randomness perform $n \cdot N_r \cdot \overline{S}(n, t+1)$ more field operations to compute n more sharings and transmit n more vectors of the length N_r to each other party.

Execution: No computation/communication overheads are present in this phase, except for those caused by the use of the message transmission functionality.

Message commitment: Share all the communication in $rn(n-1) \cdot N_c \cdot \overline{S}(n, t+1)$ operations. Send to each other party rn vectors of the length N_c. Do $r(n-1)$ recoverings in $r(n-1) \cdot N_c \cdot \overline{S^{-1}}(n, t+1)$ operations.

Verification: Compute the proof $\mathbf{p}$ of the length $(4 + N_g + N_i)$ in $18N_g + 3 \cdot \overline{FFT}(N_g) + \log N_g + 1$ field operations [21], where $\overline{FFT}(N)$ denotes the complexity of the Fast Fourier Transform, which is $c \cdot N \cdot \log N$ for a small constant c. Share $\mathbf{p}$ in $(4 + N_g + N_i) \cdot \overline{S}(n, t+1)$ operations. Send a vector of the length $(4 + N_g + N_i)$ to every other party. Broadcast one field element (the τ). Generate the 4 challenges and the state information in $14 \cdot N_g + \log(N_g)$ field operations [21]. Compute and broadcast 4 scalar products of vectors of the length $(5 + N_w + N_g)$ (the shares of $\langle (\mathbf{p} \| 1 \| \mathbf{v}), \mathbf{q_s} \rangle$). Compute 4 certain linear combinations of t scalar products and perform 2 multiplications in $\mathbb{F}$ (the products in $a_1 a_2 - a_3 - a_4 u$).

Assuming $N_w \approx 2 \cdot N_g$, for the whole verification phase, this adds up to $\approx rn(2 \cdot \overline{S}(n, t+1)N_g + 3\overline{FFT}(2N_g) + 26nN_g)$ field operations, the transmission of $\approx 4rn^2 N_g$ elements of $\mathbb{F}$, and the broadcast of $4rn^2$ elements of $\mathbb{F}$ per party.

If there are complaints, then at most rn vectors of the length N_c should be published in the message commitment phase, and at most rn vectors of length $(4 + N_g + N_i)$ ($\mathbf{p}$ shares), rn^2 vectors of the length N_c (communication shares), $n \cdot (t + 1)$ vectors of the length N_r (randomness shares) and n vectors of length N_x (input shares) in the verification phase (per complaining party).

As long as there are no complaints, the only overhead that $\mathcal{F}_{transmit}$ causes is that each message is signed, and each signature is verified.

The knowledge error of the linear PCP of [21] is $\varepsilon = 2N_g/\mathbb{F}$, and the zero-knowledge is δ-statistical for $\delta = N_g/\mathbb{F}$. Hence, the desired error and the circuit size define the field size. If we do not want to use fields that are too large, then the proof can be parallelized as proposed at the end of Sec. 4.

Preprocessing: Parties run the *Randomness generation and commitment* and *Randomness approval* steps of Fig. 3, causing the dealer to learn $r_1, \ldots, r_t$. Each r_i is shared as $r_{i1}, \ldots, r_{in}$ between $P_1, \ldots, P_n$.

Sharing: The dealer computes the shares $s_1, \ldots, s_n$ of the secret s, using the randomness $r_1, \ldots, r_t$ [31], and uses $\mathcal{F}_{transmit}$ to send them to parties $P_1, \ldots, P_n$.

Reconstruction: All parties use the publish-functionality of $\mathcal{F}_{transmit}$ to make their shares known to all parties. The parties reconstruct s as in [31].

Verification: The dealer shares each s_i, obtaining $s_{i1}, \ldots, s_{in}$. It transmits them all to P_i, which verifies that they are a valid sharing of s_i and then forwards each s_{ij} to P_j. *[Message commitment]*

The dealer computes $\mathbf{w} = witness(C, s, r_1, \ldots, r_t)$ and $\mathbf{p} = proof(C, (s, r_1, \ldots, r_t), \mathbf{w})$. It shares $\mathbf{p}$ as $\mathbf{p}_1, \ldots, \mathbf{p}_n$ and transmits $\mathbf{p}_j$ to P_j. *[Proof commitment]*

Each party P_i generates a random $\tau_i \in \mathbb{F}$ and broadcasts it. Let $\tau = \tau_1 + \cdots + \tau_n$. Each party constructs $\mathbf{q}_1, \ldots, \mathbf{q}_4, \mathbf{q}_5, \mathbf{u} = challenge(C, \tau)$. *[Challenge generation]*

Each party P_i computes $a_{ji} = \langle (\mathbf{p}_i \| 1 \| s_i \| r_{1i} \| \cdots \| r_{ti} \| s_{1i} \| \cdots \| s_{ni}), \mathbf{q}_j \rangle$ for $j \in \{1, 2, 3, 4\}$ and broadcasts them. The dealer may complain, in which case $\mathbf{p}_i, s_i, r_{1i}, \ldots, r_{ti}, s_{1i}, \ldots, s_{ni}$ are made public and all parties repeat the computation of a_{ji}. *[Scalar product verification]*

Each party reconstructs $a_1, \ldots, a_4$ and verifies the LPCP equation $(*)$.

Figure 8. LPCP-based verifiable secret sharing

6. Example: Verifiable Shamir's Secret Sharing

In this section we show how our solution can be applied to [31], yielding a verifiable secret sharing (VSS) protocol. Any secret sharing scheme has two phases — sharing and reconstruction — to which the construction presented in this work adds the verification phase.

To apply our construction, we have to define the arithmetic circuits used in [31]. For $i \in \{1, \ldots, n\}$ let C_i be a circuit taking $s, r_1, \ldots, r_t \in \mathbb{F}$ as inputs and returning $s + \sum_{j=1}^{t} r_j i^j$. If s is the secret to be shared, then C_i is the circuit used by the dealer (who is one of the parties $P_1, \ldots, P_n$) to generate the share for the i-th party using the randomness $(r_1, \ldots, r_t)$. It computes a linear function, and has no multiplication gates. According to the LPCP construction that we use, each circuit should end with a multiplication. Hence, we append a multiplication gate to it, the other argument of which is 1. Let C be the union of all C_i, it is a circuit with $1 + t$ inputs and n outputs.

In the reconstruction phase, the parties just send the shares they have received to each other. A circuit computing the messages of this phase is trivial: it just copies its input to output. We note that $\mathcal{F}_{transmit}$ already provides the necessary publishing functionality for that. Hence, we are not going to blindly follow our verifiable multiparty computation (VMPC) construction, but use this opportunity to optimize the protocol. In effect, this amounts to only verifying the sharing phase of the VSS protocol, and relying on $\mathcal{F}_{transmit}$ to guarantee the proper behavior of parties during the reconstruction. The whole protocol is depicted in Fig. 8.

A few points are noteworthy there. First, the reconstruction and verification phases can take place in any order. In particular, verification could be seen as a part of the sharing, resulting in a 3-round protocol (in the optimistic case). Second, the activities of the dealer in the sharing phase have a dual role in terms of the VMPC construction. They

	Rounds	Sharing	Reconstruction	Verification
Ours	7	$(n-1) \cdot tr$	$n \cdot bc$	$(3n+t+4)(n-1) \cdot tr + 5n \cdot bc$
[34]	4	$3n^2 \cdot tr$	$O(n^2) \cdot tr$	0
[35]	3	$2n \cdot tr + (n+1) \cdot bc$	$2n \cdot bc$	0
[36]	2	$4n^2 \cdot tr + 5n^2 \cdot bc$	$n^2 \cdot bc$	0

Table 1. Comparison of the efficiency of VSS protocols (*tr* transmissions, *bc* broadcasts)

form both the *input commitment* step in Fig. 3, as well as the execution step for actual sharing.

Ignoring the randomness generation phase (which takes place offline), the communication complexity of our VSS protocol is the following. In the sharing phase, $(n-1)$ values (elements of $\mathbb{F}$) are transmitted by the dealer, and in the reconstruction phase, each party broadcasts a value. These coincide with the complexity numbers for non-verified secret sharing. In the verification phase, in order to commit to the messages, the dealer transmits a total of $n(n-1)$ values to different parties. The same number of values are forwarded. According to Sec. 5, the proof **p** contains $t+n+4$ elements of $\mathbb{F}$. The proof is shared between the parties, causing $(n-1)(t+n+4)$ elements of $\mathbb{F}$ to be transmitted. The rest of the verification phase takes place over the broadcast channel. In the optimistic case, each party broadcasts a value in the challenge generation and four values in the challenge verification phase. Hence, the total cost of the verification phase is $(n-1)(3n+t+4)$ point-to-point transmissions and $5n$ broadcasts of $\mathbb{F}$ elements.

We have evaluated the communication costs in terms of $\mathcal{F}_{transmit}$ invocations, and have avoided estimating the cost of implementing $\mathcal{F}_{transmit}$. This allows us to have more meaningful comparisons with other VSS protocols. We will compare our solution to the 4-round statistical VSS of [34], the 3-round VSS of [35], and the 2-round VSS of [36] (see Table 1). These protocols have different security models and different optimization goals. Therefore, different methods are also selected to secure communication between the parties. Thus, the number of field elements communicated is likely the best indicator of complexity.

The 4-round statistical VSS of [34]. This information-theoretically secure protocol uses an *information checking protocol* (*ICP*) for transmission, which is a modified version of the *ICP* introduced in [37]. The broadcast channel is also used.

In the protocol, the dealer constructs a symmetric bivariate polynomial $F(x,y)$ with $F(0,0) = s$, and gives $f_i(x) = F(i,x)$ to party P_i. Conflicts are then resolved, leaving honest parties with a polynomial $F^H(x,y)$ that allows the reconstruction of s. The distribution takes $3n^2$ transmissions of field elements using the *ICP* functionality, while the conflict resolution requires $4n^2$ broadcasts (in the optimistic case). The reconstruction phase requires each honest party P_i to send its polynomial f_i to all other parties using the *ICP* functionality, which again takes $O(n^2)$ transmissions.

The 3-round VSS of [35]. Pedersen's VSS is an example of a computationally secure VSS. The transmission functionality of this protocol is based on homomorphic commitments. Although the goal of commitments is also to ensure message delivery and make further revealing possible, they are much more powerful than $\mathcal{F}_{transmit}$ and *ICP*, so direct comparison is impossible. In the following, let $Comm(m,d)$ denote the commitment of the message m with the witness d. We note that the existence of a suitable *Comm*

is a much stronger computational assumption than the existence of a signature scheme sufficient to implement $\mathcal{F}_{transmit}$.

To share s, the dealer broadcasts a commitment $Comm(s,r)$ for a random r. It shares both s and r, using Shamir's secret sharing with polynomials f and g, respectively. It also broadcasts commitments to the coefficients of f, using the coefficients of g as witnesses. This takes $2n$ transmissions of field elements, and $(n+1)$ broadcasts (in the optimistic case). Due to the homomorphic properties of $Comm$, the correctness of any share can be verified without further communication. The reconstruction requires the shares of s and r to be broadcast, i.e. there are 2 broadcasts from each party.

The 2-round VSS of [36].   This protocol also uses commitments that do not have to be homomorphic. This is still different from $\mathcal{F}_{transmit}$ and *ICP*: commitments can ensure that the same message has been transmitted to distinct parties.

The protocol is again based on the use of a symmetric bivariate polynomial $F(x,y)$ with $F(0,0)=s$ by the dealer. The dealer commits to all values $F(x,y)$, where $1 \leq x, y \leq n$ and opens the polynomial $F(i,x)$ for the i-th party. The reduction in rounds has been achieved through extra messages committed and sent to the dealer by the receiving parties. These messages can help in conflict resolution. In the optimistic case, the sharing protocol requires $4n^2$ transmissions of field elements and $5n^2$ broadcasts. The reconstruction protocol is similar to [34], with each value of $F(x,y)$ having to be broadcast by one of the parties.

We see that the LPCP-based approach performs reasonably well in verifiable Shamir's sharing. The protocols from the related works have fewer rounds, and the 3-round protocol of [35] clearly also has less communication. However, for a full comparison we have to take into account local computation, as operations on homomorphic commitments are more expensive. Also, the commitments may be based on more stringent computational assumptions than the signature-based communication primitives we are using. We have shown that the LPCP-based approach is at least comparable to similar VSS schemes. Its low usage of the broadcast functionality is definitely of interest.

7. From Finite Fields to Rings

Generalizing a finite field $\mathbb{F}$ to a set of rings (or even to one ring) in a straightforward manner does not work, as we are using Shamir's secret sharing and the LPCP based on finite fields. However, a circuit over rings can be still represented by a circuit over a finite field. We need to add a trunc gate (as defined in Sec. 2) after each gate whose output may become larger than the ring size. The size of $\mathbb{F}$ should be large enough, so that before applying trunc, the output of any gate (assuming that its inputs are truncated to the ring size) would fit into the field. For example, if we want to get a ring of the size 2^n, and we have a multiplication operation, then the field size should be at least 2^{2n}. This, in general, is not the most efficient approach, and we will not explain it in this chapter. The verification of the operations trunc, zext, and bits is similar to the one for rings that we will present.

In this section, we assume that the computation takes place over several rings $\mathbb{Z}_{2^{n_1}}, \ldots, \mathbb{Z}_{2^{n_K}}$. Taking a ring of a size that is not a power of 2 is possible, but less efficient. Instead of Shamir's secret sharing, we now have to use *additive secret sharing* (see Chapter 1 for details). Each value is shared in the corresponding ring in which it is used.

As additive secret sharing does not support a threshold, the prover has to repeat the proof with each subset of t verifiers separately (excluding the sets containing the prover itself). The proof succeeds if and only if the outcomes of all the verifier sets are satisfiable. The number of verification sets is exponential in the number of parties, but it can be reasonable for a small number of parties.

7.1. Additional Operations for Rings

We can now define the verification for the remaining gate operations defined in Sec. 2 that we could not verify straightforwardly in $\mathbb{F}$. If we need to compute $z := \mathsf{trunc}(x)$, we *locally* convert the shares over the larger ring to shares over the smaller ring, which is correct as the sizes of the rings are powers of 2, and so the size of the smaller ring divides the size of the larger ring. However, if we need to compute $z := \mathsf{zext}(x)$, then we cannot just covert the shares of committed z locally, as zext is not an inverse of trunc, and we need to ensure that all the excessive bits of z are 0.

Formally, the gate operations of Sec. 2 are verified as follows:

1. The bit decomposition operation $(z_0, \ldots, z_{n-1}) := \mathsf{bits}(z)$:
   check $z = z_0 + z_1 \cdot 2 + \cdots + z_{n-1} 2^{n-1}$;
   check $\forall j : z_j \in \{0, 1\}$.
2. The transition from $\mathbb{Z}_{2^m}$ to a smaller ring $\mathbb{Z}_{2^n}$: $z := \mathsf{trunc}(x)$:
   compute locally the shares of z from x, do not perform any checks.
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 $\forall j : z_j \in \{0, 1\}$.

As the computation takes place over a ring, we can no longer apply the LPCP used in Sec. 3. In Sec. 7.2, we propose some other means for making the number of rounds in the verification phase constant.

7.2. Pushing More into the Preprocessing Phase

A significant drawback of the construction presented in Sec. 4 is that the local computation of the prover is superlinear in the size of the circuit ($|C| \log |C|$). Now we introduce a slightly different setting that requires a more expensive offline precomputation phase, but makes the verification more efficient. The main idea is that if the circuit does not contain any multiplication gates, then linear secret sharing allows the verifiers to repeat the entire computation of the prover locally, getting the shares of all the outputs in the end. For an arbitrary circuit, we may get rid of the multiplications using Beaver triples.

Consider a circuit C_{ij}^{ℓ} being verified. For each multiplication gate, a Beaver triple is generated in the corresponding ring $\mathbb{Z}_{2^n}$. The triple is known by the prover, and it is used only in the verification, but not in the computation itself. The triple generation is performed using an ideal functionality $\mathcal{F}_{Bt}$ (see Fig. 9) that generates Beaver triples and shares them amongst the parties. Additionally, this functionality generates and shares random bits, which will be used similarly to Beaver triples: at some moment, b' is published, so that $b = (b' + r_b) \mod 2$. These random bits are not used in multiplication,

---

$\mathcal{F}_{Bt}$ works with unique wire identifiers *id*, encoding a ring size $n(id)$ of the value of this wire. It stores an array *mult* of the shares of Beaver triples for multiplication gates, referenced by unique identifiers *id*, where *id* corresponds to the output wire of the corresponding multiplication gate. It also stores an independent array *bit*, referenced by *id*, that stores the shares of random bit vectors that will be used in the bit decomposition of the wire identified by *id*.

Initialization: On input (init) from the environment, set $mult := []$, $bit := []$.

Beaver triple distribution: On input (beaver, j, id) from M_i, check if $mult[id]$ exists. If it does, take $(r_x^1, \ldots, r_x^n, r_y^1, \ldots, r_y^n, r_{xy}^1, \ldots, r_{xy}^n) := mult[id]$. Otherwise, generate $r_x \xleftarrow{\$} \mathbb{Z}_{n(id)}$ and $r_y \xleftarrow{\$} \mathbb{Z}_{n(id)}$. Compute $r_{xy} = r_x \cdot r_y$. Share r_x to r_x^k, r_y to r_y^k, r_{xy} to r_{xy}^k. Assign $mult[id] := (r_x^1, \ldots, r_x^n, r_y^1, \ldots, r_y^n, r_{xy}^1, \ldots, r_{xy}^n)$. If $j \neq i$, send r_x^i, r_y^i, r_{xy}^i to M_i. Otherwise, send $(r_x^1, \ldots, r_x^n, r_y^1, \ldots, r_y^n, r_{xy}^1, \ldots, r_{xy}^n)$ to M_i.

Random bit distribution: On input (bit, j, id) from M_i, check if $bit[id]$ exists. If it does, take $(\mathbf{b}^1, \ldots, \mathbf{b}^n) := bit[id]$. Otherwise, generate a bit vector $\mathbf{b} \xleftarrow{\$} (\mathbb{Z}_2)^{n(id)}$ and share it to $\mathbf{b}^k$. Assign $bit[id] := (\mathbf{b}^1, \ldots, \mathbf{b}^n)$. If $j \neq i$, send $\mathbf{b}^i$ to M_i. Otherwise, send $(\mathbf{b}^1, \ldots, \mathbf{b}^n)$ to M_i.

Figure 9. Ideal functionality $\mathcal{F}_{Bt}$

and they are used to ensure that b is a bit. Namely, if $b' = 0$, then $b = r_b$, and $b = 1 - r_b$ otherwise. If r_b is indeed a bit (which can be proved in the preprocessing phase), then b is also a bit.

7.3. Modified Real Functionality

Due to additional preprocessing, the real functionality becomes somewhat different from the real functionality of Sec. 4.

Preprocessing.   This is a completely offline preprocessing phase that can be performed before any inputs are known. The following values are precomputed for the prover M_i:

- Let *id* be the identifier of a circuit wire that needs a proof of correctness of its bit decomposition (proving that $z_j \in \{0, 1\}$ and $z = z_0 + z_1 \cdot 2 + \cdots + z_{n(id)-1} \cdot 2^{n(id)-1}$). Each party M_k sends query (bit, i, id) to $\mathcal{F}_{Bt}$. The prover M_i receives all the shares $(\mathbf{b}^1, \ldots, \mathbf{b}^n)$, and each verifier just the share $\mathbf{b}^k$. Let $\bar{\mathbf{b}}_i^k$ be the vector of all such bit shares of the prover M_i issued to M_k.
- Let *id* be the identifier of a multiplication gate of M_i, where both inputs are private. Each party M_k sends a query (beaver, i, id) to $\mathcal{F}_{Bt}$. The prover M_i receives all the shares $(r_x^1, \ldots, r_x^n, r_y^1, \ldots, r_y^n, r_{xy}^1, \ldots, r_{xy}^n)$, and each verifier just the shares (r_x^k, r_y^k, r_{xy}^k).

Initialization.   The same as in Sec. 4. The inputs $\mathbf{x}_i$ and the circuit randomness $\mathbf{r}_i$ are shared.

Message commitment.   The first part of this phase is similar to Sec. 4. In the message commitment phase, all the n parties finally commit their sent messages $\mathbf{c}_{ij}^\ell$ for each round $\ell \in [r']$ by sharing them to $\mathbf{c}_{ij}^{\ell k}$ and sending these shares to the other parties. This phase is given in Fig. 10. Let $\mathbf{v}_{ij}^\ell = (\mathbf{x}_i \| \mathbf{r}_i \| \mathbf{c}_{1i}^1 \| \cdots \| \mathbf{c}_{ni}^{\ell-1} \| \mathbf{c}_{ij}^\ell)$ be the vector of inputs and outputs

Message sharing: As a sender, M_i shares $\mathbf{c}_{ij}^{\ell}$ to $\mathbf{c}_{ij}^{\ell k}$ according to Shamir's secret sharing scheme. For each $k \in [n]$, M_i sends to $\mathcal{F}_{transmit}$ the messages $(\mathsf{transmit}, (\mathsf{c\_share}, \ell, i, j, k), \mathbf{c}_{ij}^{\ell k})$ for M_j.

Public values: The prover M_i constructs the vector $\mathbf{b}_i^{\ell}$ which denotes which entries of communicated values of $\bar{\mathbf{b}}_i^{\ell}$ (related to communication values) should be flipped. Let $\mathbf{p}_i^{\ell}$ be the vector of the published values c' so that $c = (c' + r_c)$ is a masqued communication value. M_i sends to $\mathcal{F}_{transmit}$ a message $(\mathsf{broadcast}, (\mathsf{communication\_public}, \ell, i), (\mathbf{p}_i^{\ell}, \mathbf{b}_i^{\ell}))$.

Message commitment: Upon receiving $((\mathsf{c\_share}, \ell, i, j, k), \mathbf{c}_{ij}^{\ell k})$ and $(\mathsf{broadcast}, (\mathsf{communication\_public}, \ell, i), (\mathbf{p}_i^{\ell}, \mathbf{b}_i^{\ell}))$ from $\mathcal{F}_{transmit}$ for all $k \in [n]$, the machine M_j checks if the shares correspond to $\mathbf{c}_{ij}^{\ell}$ it has already received. If only $c_{ij}^{\ell s \prime}$ is published for some $c_{ij}^{\ell s}$, then it checks $c_{ij}^{\ell s} = c_{ij}^{\ell s \prime} + r_c$ for the corresponding preshared randomness r_c (related to the Beaver triple). If something is wrong, M_j sends $(\mathsf{publish}, (\mathsf{message}, \ell, i, j))$ to $\mathcal{F}_{transmit}$, so now everyone sees the values that it has actually received from M_i, and each (uncorrupted) M_k should now use $\mathbf{c}_{ij}^{\ell k} := \mathbf{c}_{ij}^{\ell}$. If the check succeeds, then M_i sends to $\mathcal{F}_{transmit}$ $(\mathsf{forward}, (\mathsf{c\_share}, \ell, i, j, k))$ for M_k for all $k \in [n] \setminus \{i\}$.

Figure 10. Message commitment phase of the real functionality

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 $\mathbf{v}_{ij}^{\ell}$ among all n parties. Let $\mathbf{v}_{ij}^{\ell k}$ be the share of $\mathbf{v}_{ij}^{\ell}$ given to machine M_k.

The proving party now also publishes all the public Beaver triple communication values: for each $c = (c' + r_c)$, it publishes c'. It also publishes a bit b_{ij}' for each communicated bit b_{ij} that requires proof of being a bit. For the communicated values of $\mathbf{c}_{ij}^{\ell}$, publishing only the value $\mathbf{c}_{ij}'^{\ell}$ is sufficient, and $\mathbf{c}_{ij}^{\ell}$ itself does not have to be reshared.

During the message commitment phase, if at any time $(\mathsf{corrupt}, j)$ comes from $\mathcal{F}_{transmit}$, the proof for P_j ends with failure, and all uncorrupted machines M_i write $mlc_i[j] := 1$.

Verification phase. The proving party publishes all the remaining public Beaver triple values, and all the remaining bits b_{ij}' for each bit b_{ij} that require proof of being a bit (see Fig. 11). For each operation where $z \in \mathbb{Z}_{2^{n_e}}$, the prover commits by sharing the value of z in the ring $\mathbb{Z}_{2^{n_e}}$.

After all the values are committed and published, each verifier M_k does the following locally:

- Let $\bar{\mathbf{b}}_i^k$ be the vector of precomputed random bit shares for the prover M_i, and $\mathbf{b}_i$ the vector of published bits. For each entry $\bar{b}_{ij}^k$ of $\bar{\mathbf{b}}_i^k$, if $b_{ij} = 1$, then the verifier takes $1 - \bar{b}_{ij}^k$, and if $b_{ij} = 1$, then it takes $\bar{b}_{ij}^k$ straightforwardly. These values will now be used in place of all shares of corresponding bits.
- For all Beaver triple shares (r_x^k, r_y^k, r_{xy}^k) of M_i, the products $x' r_y^k$, $y' r_x^k$, and $x' y'$ are computed locally.

As a verifier, each M_k computes each circuit of the prover on its local shares. Due to preshared Beaver triples, the computation of $+$ and $*$ gates is linear, 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. 7.1. The condition $\forall j : z_j \in \{0, 1\}$ can be ensured as follows:

Remaining proof commitment: The prover M_i constructs the vector $\mathbf{b}_i^\ell$ that denotes which entries of non-communicated values $\bar{\mathbf{b}}_i^\ell$ should be flipped. Let $\mathbf{p}_i^\ell$ be the vector of the published values z' so that $z = (z' + r_z)$ is a masqued non-communicated value. M_i sends to $\mathcal{F}_{transmit}$ a message $(\mathsf{broadcast}, (\mathsf{remaining\_public}, \ell, i), (\mathbf{p}_i^\ell, \mathbf{b}_i^\ell))$.

For each operation $z = \mathsf{zext}(x)$, $z = \mathsf{trunc}(x)$, M_i shares z to z^k. Let $\mathbf{z}_i^{\ell k}$ be the vector of all such shares in all the circuits of M_i. It sends $((\mathsf{z\_share}, \ell, i, k), \mathbf{z}_i^{\ell k})$ to $\mathcal{F}_{transmit}$.

Local computation: After receiving all the messages $(\mathsf{broadcast}, (\mathsf{remaining\_public}, \ell, i), (\mathbf{p}_i^\ell, \mathbf{b}_i^\ell))$ and $((\mathsf{z\_share}, \ell, i, k), \mathbf{z}_i^{\ell k})$, each verifying party M_k locally computes the circuits of the proving party M_i on its local shares, collecting the necessary linear equality check shares. In the end, it obtains a set of shares $A_1\mathbf{x}_1^k, \ldots, A_K\mathbf{x}_K^k$. M_i computes and publishes $\mathbf{d}_{ij}^{\ell k} = (A_1\mathbf{x}_1^k \| \ldots \| A_K\mathbf{x}_K^k)$.

Complaints and final verification: The prover M_i knows how a correct verification should proceed and hence, it may compute the values $\mathbf{d}_{ij}^{\ell k}$ itself. If the published $\mathbf{d}_{ij}^{\ell k}$ is wrong, then the prover accuses M_k and publishes all the shares sent to M_k using $\mathcal{F}_{transmit}$. All the honest parties may now repeat the computation on these shares and compare the result. If the shares $\mathbf{d}_{ij}^{\ell k}$ correspond to $\mathbf{0}$, then the proof of M_i for C_{ij}^ℓ is accepted. Otherwise, each honest party now immediately sets $mlc_v[i] := 1$.

Figure 11. Verification phase of the real functionality

using the bit r_{z_j} shared in the preprocessing phase, and z_j' published in the commitment phase, each party locally computes the share of $z_j \in \{0, 1\}$ as r_{z_j} if $z_j' = 0$, and $1 - r_{z_j}$ if $z_j' = 1$. In the case of zext, the verifiers compute the shares of y locally, and take the shares of z that are committed by the prover in the commitment phase. Now, the 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. As the parties cannot verify locally if the shared value is 0, they postpone these checks to the last round.

For each multiplication input, the verifiers need to check $x = (x' + r_x)$, where x is either the initial commitment of x, or the value whose share the verifier has computed locally. The shares of $x' + r_x$ can be different from the shares of x, and that is why an online check is not sufficient. As $z = x * y = (x' + r_x)(y' + r_y) = x'y' + x'r_y + y'r_x + r_{xy}$, the verifiers compute locally $z^k = x'y' + x'r_y^k + y'r_x^k + r_{xy}^k$ and proceed with z^k. The checks $x = (x' + r_x)$ and $y = (y' + r_y)$ are delayed.

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 $c_{ij}^\ell = \bar{c}_{ij}^\ell$, but the shares $c_{ij}^{\ell k}$ and $\bar{c}_{ij}^{\ell k}$ can be different. Again, an online linear equality check is needed for each $c_{ij}^{\ell k}$.

In the end, the verifiers get the linear combination systems $A_1\mathbf{x}_1 = 0, \ldots, A_K\mathbf{x}_K = 0$, where $A_i\mathbf{x}_i = 0$ has to be checked in $\mathbb{Z}_{2^{n_i}}$. They compute the shares of $\mathbf{d}_i^k := A_i\mathbf{x}_i^k$ locally. If the prover is honest, then the vectors $\mathbf{d}_i^k$ are just shares of a zero vector and hence, can be revealed without leaking any information.

Unfortunately, in a ring we cannot merge the checks $\mathbf{d}_i = \mathbf{0}$ into one $\langle \mathbf{d}_i, \mathbf{s}_i \rangle = 0$ due to a large number of zero divisors (the probability of cheating becomes too high). However, if the total number of parties is 3, then there are 2 verifiers in a verifying set. They want to check if $\mathbf{0} = \mathbf{d}_i = \mathbf{d}_i^1 + \mathbf{d}_i^2$, which is equivalent to checking whether $\mathbf{d}_i^1 = -\mathbf{d}_i^2$. For this, take a collision-resistant hash function and publish $h_{ij}^{\ell 1} := h((\mathbf{d}_1^1 \| \ldots \| \mathbf{d}_K^1))$ and $h_{ij}^{\ell 2} := h(-(\mathbf{d}_1^2 \| \ldots \| \mathbf{d}_K^2))$. Check $h_{ij}^{\ell 1} = h_{ij}^{\ell 2}$.

Accusation.   The accusation phase is the same as in Sec. 4.

The formal security proofs can be found in [38].

8. Conclusions and Further Work

We have proposed a scheme transforming passively secure protocols to covertly secure ones, where a malicious party can skip detection only with negligible probability. The protocol transformation proposed here is particularly attractive to be implemented on top of some existing, highly efficient, passively secure SMC framework. The framework will retain its efficiency, as the time from starting a computation to obtaining the result at the end of the execution phase will not increase. Also, the overheads of verification, which are proportional to the number of parties, will be rather small due to the small number of *computing parties* in all typical SMC deployments (the number of *input* and *result parties* (see Chapter 1 for details) may be large, but they can be handled separately).

The implementation would allow us to study certain trade-offs. Sec. 5 shows that the proof generation is still slightly superlinear in the size of circuits, due to the complexity of FFT. Shamir's secret sharing would allow the parties to commit to some intermediate values in their circuits, thereby replacing a single circuit with several smaller ones, and decreasing the computation time at the expense of communication. The usefulness of such modifications and the best choice of intermediate values to be committed would probably largely depend on the actual circuits.

Note that the verifications could be done after each round. This would give us security against active adversaries quite cheaply, but would incur the overhead of the verification phase during the runtime of the actual protocol.

References

[1] 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.

[2] 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.

[3] 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.

[4] 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.

[5] 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.

[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] Octavian Catrina and Sebastiaan de Hoogh. Secure multiparty linear programming using fixed-point arithmetic. In Dimitris Gritzalis, Bart Preneel, and Marianthi Theoharidou, editors, *ESORICS*, volume 6345 of *Lecture Notes in Computer Science*, pages 134–150. Springer, 2010.

[8] Justin Brickell and Vitaly Shmatikov. Privacy-preserving graph algorithms in the semi-honest model. In Bimal K. Roy, editor, *ASIACRYPT*, volume 3788 of *Lecture Notes in Computer Science*, pages 236–252. Springer, 2005.

[9]   Matthew K. Franklin, Mark Gondree, and Payman Mohassel. Communication-efficient private protocols for longest common subsequence. In Marc Fischlin, editor, *CT-RSA*, volume 5473 of *Lecture Notes in Computer Science*, pages 265–278. Springer, 2009.

[10] Rosario Gennaro, Craig Gentry, and Bryan Parno. Non-interactive verifiable computing: Outsourcing computation to untrusted workers. In Tal Rabin, editor, *CRYPTO*, volume 6223 of *Lecture Notes in Computer Science*, pages 465–482. Springer, 2010.

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

[12] Yuval Ishai, Eyal Kushilevitz, and Rafail Ostrovsky. Efficient arguments without short PCPs. In *Twenty-Second Annual IEEE Conference on Computational Complexity, CCC'07.*, pages 278–291. IEEE, 2007.

[13] 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.

[14] Sanjeev Arora and Shmuel Safra. Probabilistic Checking of Proofs: A New Characterization of NP. *J. ACM*, 45(1):70–122, 1998.

[15] Silvio Micali. CS Proofs (Extended Abstract). In *FOCS*, pages 436–453. IEEE Computer Society, 1994.

[16] Joe Kilian. A note on efficient zero-knowledge proofs and arguments (extended abstract). In *Proceedings of the Twenty-fourth Annual ACM Symposium on Theory of Computing*, STOC '92, pages 723–732, New York, NY, USA, 1992. ACM.

[17] A.C. Yao. Protocols for secure computations. In *Proceedings of the 23rd Annual Symposium on Foundations of Computer Science*, pages 160–164, 1982.

[18] 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.

[19] 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.

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

[21] 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.

[22] Bryan Parno, Jon Howell, Craig Gentry, and Mariana Raykova. Pinocchio: Nearly practical verifiable computation. In *IEEE Symposium on Security and Privacy*, pages 238–252. IEEE Computer Society, 2013.

[23] Srinath T. V. Setty, Victor Vu, Nikhil Panpalia, Benjamin Braun, Andrew J. Blumberg, and Michael Walfish. Taking proof-based verified computation a few steps closer to practicality. In *USENIX Security Symposium*, 2012.

[24] Ronald Cramer, Ivan Damgård, and Jesper Buus Nielsen. Multiparty computation from threshold homomorphic encryption. In Birgit Pfitzmann, editor, *EUROCRYPT*, volume 2045 of *Lecture Notes in Computer Science*, pages 280–299. Springer, 2001.

[25] 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.

[26] Carsten Baum, Ivan Damgård, and Claudio Orlandi. Publicly auditable secure multi-party computation. In Michel Abdalla and Roberto De Prisco, editors, *Security and Cryptography for Networks - 9th International Conference, SCN 2014, Amalfi, Italy, September 3-5, 2014. Proceedings*, volume 8642 of *Lecture Notes in Computer Science*, pages 175–196. Springer, 2014.

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

[28] Ralph C. Merkle. *Secrecy, authentication, and public key systems*. PhD thesis, Stanford University, 1979.

[29] Jan Camenisch, Susan Hohenberger, and Michael Østergaard Pedersen. Batch verification of short signatures. In Moni Naor, editor, *EUROCRYPT*, volume 4515 of *Lecture Notes in Computer Science*, pages 246–263. Springer, 2007.

[30] Peeter Laud and Alisa Pankova. Verifiable Computation in Multiparty Protocols with Honest Majority. Cryptology ePrint Archive, Report 2014/060, 2014. `http://eprint.iacr.org/`.

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

[32] Donald Beaver. Efficient multiparty protocols using circuit randomization. In Feigenbaum [39], pages 420–432.

[33] Dan Bogdanov, Peeter Laud, Sven Laur, and Pille Pullonen. From Input Private to Universally Composable Secure Multi-party Computation Primitives. In *Proceedings of the 27th IEEE Computer Security Foundations Symposium*. IEEE, 2014.

[34] Ranjit Kumaresan, Arpita Patra, and C. Pandu Rangan. The round complexity of verifiable secret sharing: The statistical case. In Masayuki Abe, editor, *ASIACRYPT*, volume 6477 of *Lecture Notes in Computer Science*, pages 431–447. Springer, 2010.

[35] Torben P. Pedersen. Non-interactive and information-theoretic secure verifiable secret sharing. In Feigenbaum [39], pages 129–140.

[36] Michael Backes, Aniket Kate, and Arpita Patra. Computational verifiable secret sharing revisited. In Dong Hoon Lee and Xiaoyun Wang, editors, *ASIACRYPT*, volume 7073 of *Lecture Notes in Computer Science*, pages 590–609. Springer, 2011.

[37] 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.

[38] Peeter Laud and Alisa Pankova. Precomputed verification of multiparty protocols with honest majority. In preparation, 2015.

[39] Joan Feigenbaum, editor. *Advances in Cryptology - CRYPTO '91, 11th Annual International Cryptology Conference, Santa Barbara, California, USA, August 11-15, 1991, Proceedings*, volume 576 of *Lecture Notes in Computer Science*. Springer, 1992.

Chapter 10
Universally Verifiable Outsourcing and Application to Linear Programming

Sebastiaan DE HOOGH [a,1], Berry SCHOENMAKERS [b] and Meilof VEENINGEN [b]

[a] *Philips Research Europe, Netherlands*
[b] *Eindhoven University of Technology, Netherlands*

Abstract In this chapter, we show how to guarantee correctness when applying multiparty computation in outsourcing scenarios. Specifically, we consider how to guarantee the correctness of the result when neither the parties supplying the input nor the parties performing the computation can be trusted. Generic techniques to achieve this are too slow to be of practical use. However, we show that it is possible to achieve practical performance for specific problems by exploiting the existence of certificates proving that a computation result is correct.

Introduction

In several previous chapters, multiparty computation has been applied in the traditional model in which the same set of parties provide the inputs, perform the computation, and obtain the outputs. We have seen that in this setting, multiparty computation guarantees privacy, robustness, and correctness, as long as certain assumptions on the type and number of corruptions are satisfied.

In this chapter, we consider the application of multiparty computation in outsourcing scenarios, where the parties providing the inputs (*input parties*) and obtaining the outputs (*result parties*) do not have to be participants in the computation (*computation parties*). This is useful if it saves time for the input or result parties. Moreover, it is necessary if there are many of them, in which case performing a protocol involving all of them can become very costly. In an outsourcing scenario, it is desirable for the computation to be *verifiable*, i.e. the correctness with respect to the result party to be satisfied, even when none of the computation parties can be trusted (i.e. may be actively corrupted). An even stronger property that is desirable in many use cases is *universal verifiability* [1], meaning that *anybody* can check if the computation was performed correctly. A classical example of such a use case is e-voting [2,3]: indeed, this is a computation with a public result (the result of the election) that anybody should be able to check. Another example is when a secure computation needs to be auditable by an external watchdog. In this case, the watchdog does not necessarily need to learn the inputs or outputs of the computation, it just wants to be sure that the computation was performed correctly.

---

[1] Work carried out when the author was working at the Eindhoven University of Technology.

Existing techniques can be used in outsourcing scenarios. For instance, in secret sharing based protocols, the input parties can provide shares of their input to each computation party, and the computation parties can provide shares of the result to the result party. Unfortunately, existing techniques typically do not provide verifiability. In particular, while protocols such as SPDZ (see Sec. 7 of Chapter 1) ensure correctness for each protocol participant even if all other participants are corrupted, this is only true for participants and not for a non-participating result party. Two recent proposals show that it is possible to make a secure computation universally verifiable [4,5], but unfortunately, these proposals are not really practical; they put an especially heavy workload on the verifier.

In this chapter, we present practical techniques for universally verifiable computation, taking secure linear programming as an example. Our core idea is what we call *verifiability by certificate validation*: instead of computing the result in a verifiable way, we use normal techniques to compute the result and a certificate of its correctness, and then perform a verifiable check to see whether the computed result is correct with respect to this certificate. In particular, for linear programming, algorithms for finding the solution turn out to naturally give a certificate that proves correctness, namely, the solution of the so-called *dual linear program*. We present verifiability by certificate validation, analyze its security, and show that it indeed allows practical universally verifiable linear programming.

This chapter is an extended and generalized version of [6]. In [6], only two-party ElGamal-based verification (Sec. 3) is discussed. Here, we also give a more general Paillier-based construction (Sec. 2). Moreover, we extend the protocols from [6] to the *universally* verifiable setting, i.e. where external parties can check correctness without learning the result. Finally, we provide a more extensive discussion of linear programming, and in particular, of the simplex algorithm used to find solutions.

First, in Sec. 1 we outline the UVCDN protocol, a recent proposal for universally verifiable multiparty computation based on a threshold homomorphic cryptosystem such as Paillier. This protocol itself is too slow to be applied to practical computations, but we show how it can be combined with fast passively secure protocols to obtain verifiability by certificate validation (Sec. 2). Moreover, for a class of simple validation functions, we propose a more efficient approach based on ElGamal encryption (Sec. 3). We then apply the approach to linear programming by introducing the problem (Sec. 4), recalling how it can be solved using passively secure techniques, and showing how the solution can be verified using the solution to the dual linear program as certificate. We present experiments that show that, particularly for larger linear programs, the correctness of a solution can be verified more quickly than it is computed, even when compared to passively secure computations (Sec. 5).

Figure 1 summarizes the notation used in this chapter.

1. Universally Verifiable Multiparty Computation from Threshold Cryptosystems

We start our discussion of universal verifiability with a general but inefficient technique. Later, we show that this technique can be optimized and combined with our *verifiability by certificate validation* approach to obtain practical universal verifiability. Recall that, at an abstract level, we want to perform secure computations whose correctness anybody could check. More specifically, let us assume that, at the beginning of a computation, all

parties P / **party** $\mathcal{P}$ **do** S	Let parties P/party $\mathcal{P}$ (simultaneously) perform S
$\mathcal{I}/\mathcal{P}/\mathcal{R}/\mathcal{V}$	Input/computation/result/verifier parties
$F \subset \mathcal{I} \cup \mathcal{P} \cup \{\mathcal{R}, \mathcal{V}\}$	Global variable: set of parties found to misbehave
$\mathsf{recv}(\mathcal{P}), \mathsf{send}(v; \mathcal{P})$	Receive/send v from/to $\mathcal{P}$ over a secure channel
$\mathsf{bcast}(v)$	Exchange v over the broadcast channel
$a \in_R S$	Sample a uniformly at random from S
$\mathcal{H}$	Cryptographic hash function
$\mathsf{Enc}_{\mathsf{pk}}(x, r)$	Encryption under pk of x with randomness r
$\mathfrak{R}$	Randomness space of the encryption scheme
$\oplus$	Homomorphic addition of ciphertexts
$\Sigma_{\mathrm{PK}}, \Sigma_{\mathrm{CM}}, \Sigma_{\mathrm{CD}}, \Phi_{\mathrm{PK}}, \Phi_{\mathrm{CM}}, \Phi_{\mathrm{CD}}$	Σ-protocols (p. 5) and homomorphisms (p. 6)
$\mathrm{NIZKVER}(\Sigma, v, (a, c, r), aux)$	Verification of Fiat-Shamir Σ-proof (p. 6):
	$\quad \mathcal{H}(v\|a\|aux) = c \wedge \Sigma.\mathsf{ver}(v, a, c, r)$
$[x], [\![x]\!]$	Additive/Shamir's secret sharing of x

Figure 1. Notation in algorithms and protocols

input parties provide encryptions of their respective inputs. The computation should give an encryption of the result, of which a result party (or, if the result is public, anybody) knows the plaintext, and a proof that this encryption encrypts the output of the function to be computed.

A natural way of achieving the above goal is to perform the computation step by step on encryptions, using non-interactive zero-knowledge proofs (NIZKs) to show that every step was performed correctly. This can be done based on the CDN protocol [7]. As discussed in Sec. 6 of Chapter 1, this protocol performs computations on encryptions using a threshold homomorphic encryption scheme (e.g. a threshold variant of Paillier encryption [8]). Zero-knowledge proofs are used to ensure that cheating parties cannot break the correctness of the computation. In [7], these proofs are interactive, so they only convince parties involved in the computation. However, as shown in [1,5], the proofs can be made non-interactive so that they would also be convincing to external verifiers. We will now review this non-interactive variant of the CDN protocol called UVCDN [5].

1.1. Computation Using a Threshold Homomorphic Cryptosystem

We describe the UVCDN protocol in the verifiable outsourcing setting. That is, we have m input parties $i \in \mathcal{I}$, n computation parties $j \in \mathcal{P}$, a result party $\mathcal{R}$, and a verifier $\mathcal{V}$. The aim of the protocol is to compute a function $f(x_1, \ldots, x_m)$ (seen as an arithmetic circuit) on private inputs x_i of the input parties. In the end, the result party obtains the result, the verifier obtains an encryption of the result, and both outputs are guaranteed to be correct regardless of which computation parties are corrupted.

The UVCDN protocol follows largely the same steps as the CDN protocol described in Sec. 6 of Chapter 1. In an outsourcing setting, the inputs are provided by the input parties. This happens in two rounds. In the second round, each input party $i \in \mathcal{I}$ provides an encryption X_i of its input x_i along with a non-interactive zero-knowledge proof of knowledge of x_i. Before this, in the first round, the parties commit to these encryptions and proofs. This prevents input parties from choosing their inputs depending on the inputs of others. The inputs are broadcast in encrypted form to all other parties, including the verifier $\mathcal{V}$. In practice, some kind of bulletin board can be used. Computation and

output to the result party are performed as in the CDN protocol, except that the proofs are non-interactive. We discuss these non-interactive proofs in more detail in Sec. 1.3.

Finally, the result party provides an encryption of the result to the verifier $\mathcal{V}$, and extracts from the messages exchanged so far proof that this encryption is the correct result of the computation. Namely, recall from Sec. 6 of Chapter 1 that a multiplication of encryptions X and Y in the CDN protocol involves exchanging encryptions D_i and E_i such that E_i encrypts the product of the plaintexts of Y and D_i and a decryption s of $X \oplus D_1 \oplus \ldots \oplus D_n$; and proving that E_i and s are correct. In UVCDN, these proofs are NIZKs. From the transcript of the multiplication of X and Y, the result party computes $D = D_1 \oplus \ldots \oplus D_n$ and $E = E_1 \oplus \ldots \oplus E_n$. Moreover, it combines the NIZKs from the individual computation parties into one NIZK that E encrypts the product of y and d (see Sec. 1.3). Hence, for each multiplication, the result party provides D, E, s, and the two NIZKs to the verifier, who can use these values to check the multiplication. For his output, the result party provides the random encryption D, the decryption y of $V \oplus D$, and the NIZK that this decryption is correct. Overall, this convinces the verifier that $Y \ominus D$ encrypts the computation result, where Y is a fixed encryption of y. Note that the multiplication proofs already result in an encryption of the computation result, but the result party does not know the randomness used in this encryption. It is desirable that the result party *does* know this randomness so that he would later have the possibility to prove (e.g. in zero-knowledge) which result was encrypted.

1.2. Security of the UVCDN Protocol

Overall, the UVCDN protocol achieves the following security guarantees. Whenever the computation succeeds, the results (i.e. the plaintext result of $\mathcal{R}$ and the encrypted result of $\mathcal{V}$) are correct. Whether or not privacy (i.e. only $\mathcal{R}$ can learn the result) and robustness (i.e. the computation does not fail) are guaranteed depends on which parties are corrupted. Privacy is guaranteed as long as fewer than t (the decryption threshold of the cryptosystem) computation parties are corrupted. Similarly, robustness with respect to $\mathcal{R}$ is guaranteed as long as there are at least t honest computation parties, i.e. as long as at most $n - t$ computation parties are corrupted. In particular, if t is chosen to be $\lceil n/2 \rceil$, then privacy and robustness both hold with fewer than t corruptions, cf. [5]. For robustness with respect to $\mathcal{V}$, moreover, the result party $\mathcal{R}$ needs to be honest, otherwise, $\mathcal{R}$ can simply not provide the proof to the verifier. For an ideal-world formalization of these guarantees, see [5].

Note that the above security guarantees do not capture the *universality* aspect of universal verifiability. In fact, as argued in [5], ideal world formalizations capture only the input/output behavior of protocols, not the fact that the verifier can be anybody. In practice, universally verifiable protocols can be designed by proving that they are verifiable as above, and then arguing based on the characteristics of the protocol (e.g. the verifier does not have any secret values) that this verifiability is universal. In the case of the above protocol, the input parties publicly announce their encrypted inputs, and anybody can verify that the proof sent by the result party is correct. Because the verifier does not have to be present at the time of the computation, and it does not hold any secrets, the above protocol can be considered universally verifiable.

Finally, we note that, due to the non-interactive proofs used, the above protocol is only secure in the random oracle model [9,10], an idealised model of hash functions. In this model, evaluations of the hash function $\mathcal{H}$ are modelled as queries to a random oracle

$\mathcal{O}$ that evaluates a perfectly random function. When simulating an adversary, a simulator can intercept these oracle queries and answer them at will, as long as the answers look random to the adversary. In particular, the model used for the UVCDN security proof assumes that the random oracle has not been used before the protocol starts. In practice, the random oracle can be instantiated with a keyed hash function, with every computation using a fresh random key.

1.3. Proving the Correctness of Results

The techniques for proving correctness used in the UVCDN protocol are based on Σ-protocols. Recall that a Σ-protocol for a binary relation R is a three-move protocol in which a potentially malicious prover convinces an honest verifier that he knows a *witness* w for *statement* v so that $(v, w) \in R$. First, the prover sends an *announcement* (computed using algorithm $\Sigma.\text{ann}$) to the verifier, the verifier responds with a uniformly random *challenge*, and the prover sends his *response* (computed using algorithm $\Sigma.\text{res}$), which the verifier verifies (using predicate $\Sigma.\text{ver}$). Σ-protocols are defined as follows:

Definition 1. Let $R \subset V \times W$ be a binary relation and $L_R = \{v \in V \mid \exists w \in W : (v, w) \in R\}$ its language. Let Σ be a collection of PPT algorithms $\Sigma.\text{ann}$, $\Sigma.\text{res}$, $\Sigma.\text{sim}$, $\Sigma.\text{ext}$, and polynomial time predicate $\Sigma.\text{ver}$. Let C be a finite set called the *challenge space*. Then Σ is a *Σ-protocol* for relation R if it satisfies the following properties:

Completeness If $(a, s) \leftarrow \Sigma.\text{ann}(v, w)$, $c \in C$, and $r \leftarrow \Sigma.\text{res}(v, w, a, s, c)$, then $\Sigma.\text{ver}(v, a, c, r)$.

Special soundness If $v \in V$, $c \neq c'$, and $\Sigma.\text{ver}(v, a, c, r)$ and $\Sigma.\text{ver}(v, a, c', r')$ both hold, then $w \leftarrow \Sigma.\text{ext}(v, a, c, c', r, r')$ satisfies $(v, w) \in R$.

Special honest-verifier zero-knowledge If $v \in L_R$, $c \in C$, then $(a, r) \leftarrow \Sigma.\text{sim}(v, c)$ has the same probability distribution as (a, r) obtained by $(a, s) \leftarrow \Sigma.\text{ann}(v, w)$, $r \leftarrow \Sigma.\text{res}(v, w, a, s, c)$. If $v \in V \setminus L_R$, then $(a, r) \leftarrow \Sigma.\text{sim}(v, c)$ satisfies $\Sigma.\text{ver}(v, a, c, r)$.

Completeness states that a protocol between an honest prover and verifier succeeds. Special soundness states that there exists an extractor $\Sigma.\text{ext}$ that can extract a witness from two conversations with the same announcement. Special honest-verifier zero-knowledge states that there exists a simulator $\Sigma.\text{sim}$ that can generate transcripts with the same distribution as full protocol runs without knowing the witness. We also require *non-triviality*, meaning that two announcements for the same statement are different with overwhelming probability. This property essentially holds whenever the witness is hard to derive from the statement (see [5] for a discussion).

In [5], Σ-protocols are given for the proofs needed in the UVCDN protocol. Proof Σ_{PK} for statement Y proves knowledge of the plaintext and randomness of Y. Proof Σ_{CM} for statement (X, Y, Z) proves that Z encrypts the product of the plaintexts of X and Y, and proves knowledge of the plaintext and randomness of Y. Proof Σ_{CD} for statement (Z, d, v, v_i) proves that d is a valid threshold decryption share of Z with respect to a publicly known verification key v_i of computation party i.

Σ-protocols can be used to obtain non-interactive zero-knowledge proofs (NIZKs) using the well-known *Fiat-Shamir heuristic* [11,12]. Namely, party $i \in P$ proves knowledge of a witness for statement v_i by generating an announcement a_i using $\Sigma.\text{ann}$, setting a challenge $c_i = \mathcal{H}(v_i \| a_i \| i)$, and computing a response r_i using $\Sigma.\text{ver}$. A ver-

ifier accepts the proof (a_i, c_i, r_i) if $\text{NIZKVER}(\Sigma, v_i, (a_i, c_i, r_i),) := \mathcal{H}(v_i || a_i || i) = c_i \wedge \Sigma.\text{ver}(v_i, a_i, c_i, r_i)$ holds. Indeed, the UVCDN protocol uses the Fiat-Shamir heuristic for the proofs of knowledge by the input and result parties.

For the proofs by the computation parties, an optimisation is possible so that the verifier does not need to verify each proof individually. This is done by exploiting homomorphisms of the announcements and responses of certain Σ-protocols. For instance, let a_i be valid announcements for proving that E_i is a correct multiplication of X and D_i. Let c be a challenge, and let r_i be the responses to this challenge for the respective announcements a_i. Hence, $\Sigma.\text{ver}((X, D_i, E_i), a_i, c, r_i)$. Then, it is possible to homomorphically combine a_i into a and r_i into r in such a way that (a, c, r) is a proof that $\oplus_i E_i$ is a correct multiplication of X and $\oplus_i D_i$, that is, that $\Sigma.\text{ver}((X, \oplus_i D_i, \oplus_i E_i), a, c, r)$. The result party can present this combined proof to the verifier. This way, the verifier no longer needs to verify a proof for each individual computation party.

For such combining to be possible in a secure way, these homomorphisms of announcements and responses need to satisfy two properties. First, when enough transcripts, say T, are combined, the result is a valid transcript. Note that for the multiplication proof, $T = 1$, and for the decryption proof, $T = t$. Second, when fewer than T parties are corrupted, the combination of different honest announcements with the same corrupted announcements is likely to lead to a different combined announcement.

Definition 2. Let Σ be a Σ-protocol for relation $R \subset V \times W$. Let Φ be a collection of partial functions $\Phi.\text{stmt}$, $\Phi.\text{ann}$, and $\Phi.\text{resp}$. We call Φ a *homomorphism* of Σ with threshold T if:

Combination Let c be a challenge, I a set of parties so that $|I| \geq T$, and $\{(v_i, a_i, r_i)\}_{i \in I}$ a collection of statements, announcements, and responses. If $\Phi.\text{stmt}(\{v_i\}_{i \in I})$ is defined, and for all i $\Sigma.\text{ver}(v_i, a_i, c, r_i)$ holds, then also
$\Sigma.\text{ver}(\Phi.\text{stmt}(\{v_i\}_{i \in I}), \Phi.\text{ann}(\{a_i\}_{i \in I}), c, \Phi.\text{resp}(\{r_i\}_{i \in I}))$.

Randomness Let c be a challenge, $C \subset I$ sets of parties so that $|C| < T \leq |I|$, $\{v_i\}_{i \in I}$ statements so that $\Phi.\text{stmt}(\{v_i\}_{i \in I})$ is defined, and $\{a_i\}_{i \in I \cap C}$ announcements. If $(a_i, \_), (a_i', \_) \leftarrow \Sigma.\text{sim}(v_i, c) \; \forall i \in I \setminus C$, then with overwhelming probability, $\Phi.\text{ann}(\{a_i\}_{i \in I}) \neq \Phi.\text{ann}(\{a_i\}_{i \in I \cap C} \cup \{a_i'\}_{i \in I \setminus C})$.

Homomorphisms for Σ_{PK}, Σ_{CM}, and Σ_{CD} are given in [5]. Homomorphism Φ_{PK} for Σ_{PK} combines proofs of knowledge of encryptions D_i into a proof of knowledge of $\oplus_i D_i$. As discussed above, homomorphism Φ_{CM} for Σ_{CM} combines multiplication proofs (X, D_i, E_i) into a multiplication proof $(X, \oplus_i D_i, \oplus_i E_i)$. Homomorphism Φ_{CD} for Σ_{CD} combines proofs of correct decryption shares into an overall decryption proof.

Now, combined non-interactive proofs can be obtained by applying a multiparty variant of the Fiat-Shamir heuristic. Namely, all parties provide an announcement a_i for their own witness, compute $v = \Phi.\text{stmt}(\{v_i\})$, $a = \Phi.\text{ann}(\{a_i\})$ and $\mathcal{H}(v || a || aux)$, and provide a response r_i. Taking $r = \Phi.\text{resp}(\{r_i\})$, the combination property from Def. 2 guarantees that we indeed get a validating proof. For security reasons, we add an initial round in which each party commits to its announcement. The full protocol is shown in Protocol 1. It includes error handling, keeping track of a set F of parties that have failed to follow the protocol, and continuing until either a valid combined transcript is produced, or not enough honest parties are left. For a security proof of this protocol, see [5].

Protocol 1 $M\Sigma$: The multiparty Fiat-Shamir heuristic [5]

Require: Σ is a Σ-protocol, Φ a homomorphism of Σ with threshold T, P a set of non-failed parties $(P \cap F = \emptyset)$, $v_P = \{v_i\}_{i \in P}$ statements with witnesses $w_P = \{w_i\}_{i \in P}$

Ensure: If $|P \setminus F| \geq T$, then $v_{P \setminus F}$ is the combined statement $\Phi.\mathsf{stmt}(\{v_i\}_{i \in P \setminus F})$, and $\pi_{P \setminus F}$ is a corresponding Fiat-Shamir proof

Invariant: $F \subset C$: set of failed parties only includes corrupted parties

1: **protocol** $(v_{P \setminus F}, \pi_{P \setminus F}) \leftarrow M\Sigma(\Sigma, \Phi, P, w_P, aux)$
2:      **repeat**
3:          **parties** $i \in P \setminus F$ **do**
4:              $(a_i, s_i) \leftarrow \Sigma.\mathsf{ann}(v_i, w_i)$; $h_i \leftarrow \mathcal{H}(a_i || i)$; $\mathsf{bcast}(h_i)$
5:          **parties** $i \in P \setminus F$ **do** $\mathsf{bcast}(a_i)$
6:          $F' \leftarrow F$; $F \leftarrow F \cup \{i \in P \setminus F \mid h_i \neq \mathcal{H}(a_i || i)\}$
7:          **if** $F = F'$ **then**   ▷ all parties left provided correct hashes
8:              $c \leftarrow \mathcal{H}(\Phi.\mathsf{stmt}(\{v_i\}_{i \in P \setminus F}) || \Phi.\mathsf{ann}(\{a_i\}_{i \in P \setminus F}) || aux)$
9:          **parties** $i \in P \setminus F$ **do** $r_i := \Sigma.\mathsf{res}(v_i, w_i, a_i, s_i, c)$; $\mathsf{bcast}(r_i)$
10:         $F \leftarrow F \cup \{i \in P \setminus F \mid \neg\Sigma.\mathsf{ver}(v_i, a_i, c, r_i)\}$
11:         **if** $F = F'$ **then**   ▷ all parties left provided correct responses
12:             **return** $(\Phi.\mathsf{stmt}(\{v_i\}_{i \in P \setminus F}), (\Phi.\mathsf{ann}(\{a_i\}_{i \in P \setminus F}), c, \Phi.\mathsf{resp}(\{r_i\}_{i \in P \setminus F})))$
13:     **until** $|P \setminus F| < T$ ▷ until not enough parties left
14:     **return** $(\bot, \bot)$

---

2. Universal Verifiability by Certificate Validation

Performing a full computation using a universally verifiable protocol is prohibitively expensive for real applications like linear programming. For instance, the protocol from the preceding section uses threshold homomorphic encryption, which makes it computationally much heavier than protocols based on, for example, secret sharing.

In this section, we present our solution to this problem, based on the concept of *verifiability by certificate validation*. Namely, we combine fast computation of a solution and certificate with verifiable validation that the solution is correct with respect to the certificate. Consider the example of computing an eigenvalue λ of a matrix M. It is not easy to compute λ, but it is easy to verify its correctness using an eigenvector v as a certificate by checking that $Mv = \lambda v$. Hence, in our approach, we compute the output λ and the certificate v with VIFF [13] that uses SMC based on secret sharing, and then check $Mv = \lambda v$ using UVCDN. More generally, suppose we have four kinds of parties: m input parties $i \in \mathcal{I}$, n computation parties $i \in \mathcal{P}$, a result party $\mathcal{R}$, and a verifier $\mathcal{V}$. The computation parties use VIFF to compute $(\vec{a}, \vec{r}) = f(\vec{x})$, where $\vec{x}$ is the input (of the length m), $\vec{a}$ is the certificate (of the length k), and $\vec{r}$ is the output (of the length l). Let ϕ be a certificate so that, if $(\vec{a}, \vec{r}) = f(\vec{x})$, then $\phi(\vec{x}, \vec{a}, \vec{r})$ holds.[2] The computation parties then use UVCDN to compute a proof that $\phi(\vec{x}, \vec{a}, \vec{r})$ is indeed the case, and deliver the result to the result party, and an encryption to the verifier. The idea is that the combined protocol inherits its privacy and robustness properties from the protocols for computing f, but guarantees correctness regardless of corruptions.

---

[2]Note that we do not demand the converse, i.e. if $\phi(\vec{x}, \vec{a}, \vec{r})$, then it is not necessarily true that $(\vec{a}, \vec{r}) = f(\vec{x})$. For instance, the solution $\vec{r}$ might not be unique. Then f is an algorithm to find some solution, and ϕ merely checks if the solution is valid. Indeed, this will be the case for linear programming.
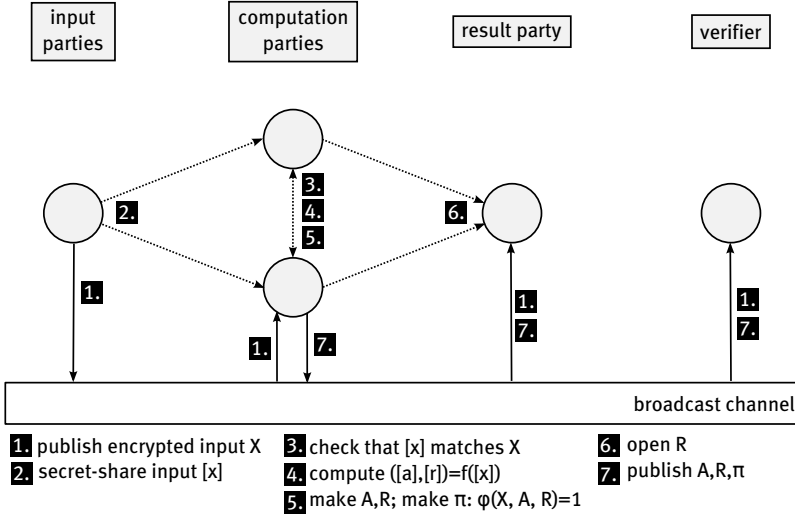
Figure 2. The VERMPC protocol for verifiability by certificate validation

One technical problem of combining VIFF and UVCDN is that they represent values in different ways. VIFF represents values as (t, n)-Shamir's secret sharings with $t < n/2 + 1$ over a prime order field. More precisely, suppose that all values needed to compute $(\vec{a}, \vec{r}) = f(\vec{x})$ lie in $\mathbb{Z}_{\langle 2^k \rangle} = \{-2^{k-1} + 1, \ldots, 2^{k-1}\}$ for some k. VIFF computes over $\mathbb{Z}_{\langle 2^k \rangle}$ using Shamir's secret sharings over a field $\mathbb{F}_p$ with p being a prime much bigger than k, say $p > 2^{k+\kappa}$ where κ is a statistical security parameter, but still relatively small (e.g. 128 bits). The smaller p is, the faster the computation, but p needs to be large enough for the computation at hand. UVCDN, on the other hand, represents values using encryptions using a threshold homomorphic cryptosystem (with the same threshold t) with a different plaintext space. For instance, for the Paillier cryptosystem, the plaintext space is $\mathbb{Z}_N$, with N an RSA modulus of, say, 2048 bits. This means that a conversion between the shares modulo p of VIFF and the plaintexts modulo N of UVCDN is needed. Strictly speaking, the above function f and predicate ϕ are functions on $\mathbb{Z}_N$. If potential reductions modulo N are undesirable for a particular application, then they should be prevented in an application-dependent way, e.g. by letting input parties prove that their inputs are bounded.

2.1. The VERMPC Protocol

Working out the details of the above idea, the main challenge is to perform conversions between different share representations while minimizing the cheating opportunity for different parties. The general structure of our design is shown in Fig. 2, the full protocol in Protocol 2. In the figure and protocol, lower-case variables (e.g. r) denote plaintext values, upper-case variables (e.g. R) denote encrypted values, values $[r]$ with brackets denote additive secret sharings over $\mathbb{Z}$, and values $[\![r]\!]$ with double brackets denote Shamir's secret sharings over $\mathbb{F}_p$. Communication between parties is performed over private, authenticated channels (using the command send). Apart from that, a broadcast channel is used to announce the inputs and result of the computation (command bcast).

Protocol 2 The VERMPC protocol for verifiability by certificate validation

---

1: **protocol** VERMPC(pk, vk, $\{x_i\}_{i \in \mathcal{I}}, \{s_i\}_{i \in \mathcal{P}}$)
2:     **parties** $i \in \mathcal{I}$ **do**   ▷ step 1
3:         $r_{x,i} \in_R \mathfrak{R}$; $X_i \leftarrow \mathsf{Enc}_{\mathsf{pk}}(x_i, r_{x,i})$
4:         $(a_i, s_i) \leftarrow \Sigma_{\mathsf{PK}}.\mathsf{ann}(X_i, (x_i, r_{x,i}))$; $c_i \leftarrow \mathcal{H}(X_i || a_i || i)$
5:         $r_i \leftarrow \Sigma_{\mathsf{PK}}.\mathsf{res}(X_i, (x_i, r_{x,i}), a_i, s_i, c_i)$; $\pi_{x,i} \leftarrow (a_i, c_i, r_i)$
6:         $h_i \leftarrow \mathcal{H}(i || X_i || \pi_{x,i})$; $\mathsf{bcast}(h_i)$; $\mathsf{bcast}((X_i, \pi_{x,i}))$
7:         **if** $\exists j : h_j \neq \mathcal{H}(j || X_j || \pi_{x,j}) \vee \neg \mathsf{NIZKVER}(\Sigma_{\mathsf{PK}}, X_j, \pi_{x,j}, j)$ **then return** $\bot$
8:         **for all** $1 \leq j < n$ **do**   ▷ step 2
9:             $[x_i]^{(j)} \in_R \mathbb{Z}_{\langle 2^{\kappa - \log n} \rangle}$; $[r_{x,i}]^{(j)} \in_R \mathfrak{R}$; $\mathsf{send}([x_i]^{(j)}, [r_{x,i}]^{(j)}; \mathcal{P}_j)$
10:         $[x_i]^{(n)} \leftarrow x_i - \sum_j [x_i]^{(j)}$; $[r_{x,i}]^{(n)} \leftarrow r_{x,i} - \sum_j [r_{x,i}]^{(j)}$
11:         $\mathsf{send}([x_i]^{(n)}, [r_{x,i}]^{(n)}; \mathcal{P}_n)$
12:     **parties** $j \in \mathcal{P}$ **do**
13:         **for all** $1 \leq i \leq m$ **do**
14:             **if** $[x_i]^{(j)} \notin \mathbb{Z}_{\langle 2^\kappa \rangle}$ **then return** $\bot$
15:             $[X_i]^{(j)} \leftarrow \mathsf{Enc}_{\mathsf{pk}}([x_i]^{(j)}, r_{x,i}^{(j)})$; $\mathsf{send}([X_i]^{(j)}; \{\mathcal{P}_{j'}\}_{j' \neq j})$ ▷ step 3
16:             **if** $X_i \neq \oplus_j [X_i]^{(j)}$ **then return** $\bot$
17:         $(\llbracket x_1 \rrbracket, \ldots, \llbracket x_m \rrbracket) \leftarrow \mathsf{ADD}\mathbb{Z}\mathsf{TOSHAMIR}p([x_1], \ldots, [x_m])$ ▷ step 4
18:         $(\llbracket a_1 \rrbracket, \ldots, \llbracket a_k \rrbracket, \llbracket r_1 \rrbracket, \ldots, \llbracket r_l \rrbracket) \leftarrow f(\llbracket x_1 \rrbracket, \ldots, \llbracket x_m \rrbracket)$
19:         $([a_1], \ldots, [r_l]) \leftarrow \mathsf{SHAMIR}p\mathsf{TOADD}\mathbb{Z}(\llbracket a_1 \rrbracket, \ldots, \llbracket r_l \rrbracket)$
20:         **for all** $1 \leq i \leq k$ **do**   ▷ step 5
21:             $[r_{a,i}]^{(j)} \in_R \mathfrak{R}$; $[A_i]^{(j)} \leftarrow \mathsf{Enc}_{\mathsf{pk}}([a_i]^{(j)}, [r_{a,i}]^{(j)})$
22:             $\mathsf{send}([A_i]^{(j)}; \{\mathcal{P}_{j'}\}_{j' \neq j})$
23:             $A_i, \pi_{a,i} \leftarrow M\Sigma(\Sigma_{\mathsf{PK}}, \Phi_{\mathsf{PK}}, [A_i]^{(j)}, ([a_i]^{(j)}, [r_{a,i}]^{(j)}))$
24:         **for all** $1 \leq i \leq l$ **do**
25:             $[r_{r,i}]^{(j)} \in_R \mathfrak{R}$; $[R_i]^{(j)} \leftarrow \mathsf{Enc}_{\mathsf{pk}}([r_i]^{(j)}, [r_{r,i}]^{(j)})$
26:             $h_{i,j} \leftarrow \mathcal{H}(i || j || [R_i]^{(j)})$; $\mathsf{send}(h_{i,j}; \{\mathcal{P}_{j'}\}_{j' \neq j})$; $\mathsf{send}([R_i]^{(j)}; \{\mathcal{P}_{j'}\}_{j' \neq j})$
27:             **if** $\exists i, j : h_{i,j} \neq \mathcal{H}(i || j || [R_i]^{(j)})$ **then return** $\bot$
28:             $R_i, \pi_{r,i} \leftarrow M\Sigma(\Sigma_{\mathsf{PK}}, \Phi_{\mathsf{PK}}, [R_i]^{(j)}, ([r_i]^{(j)}, [r_{r,i}]^{(j)}))$
29:         $r, \pi \leftarrow \mathsf{UVCDNPROVE}(\phi; X_1, \ldots, R_l)$
30:         **for all** $1 \leq i \leq l$ **do** $\mathsf{send}([r_i]^{(j)}, [r_{r,i}]^{(j)}; \mathcal{R})$ ▷ step 6
31:     **party** $\mathcal{P}_1$ **do** $\mathsf{bcast}(A_1, \ldots, R_l, \pi, \pi_{a,1}, \ldots, \pi_{r,l})$ ▷ step 7
32:     **parties** $\mathcal{R}, \mathcal{V}$ **do**
33:         **for all** $1 \leq i \leq m$ **do if** $\neg \mathsf{NIZKVER}(\Sigma_{\mathsf{PK}}, X_i, \pi_{x,i})$ **then return** $\bot$
34:         **for all** $1 \leq i \leq k$ **do if** $\neg \mathsf{NIZKVER}(\Sigma_{\mathsf{PK}}, A_i, \pi_{a,i})$ **then return** $\bot$
35:         **for all** $1 \leq i \leq l$ **do if** $\neg \mathsf{NIZKVER}(\Sigma_{\mathsf{PK}}, R_i, \pi_{r,i})$ **then return** $\bot$
36:         $r \leftarrow \mathsf{UVCDNVER}(\phi; X_1, \ldots, R_l; \pi)$; **if** $r \neq 1$ **then return** $\bot$
37:     **party** $\mathcal{R}$ **do**
38:         **for all** $1 \leq i \leq l$ **do if** $R_i \neq \mathsf{Enc}_{\mathsf{pk}}(\sum_j [r_i]^{(j)}, \sum_j [r_{r,i}]^{(j)})$ **then return** $\bot$
39:         **return** $((r_1, r_{r,1}), \ldots, (r_l, r_{r,l}))$
40:     **party** $\mathcal{V}$ **do return** $(R_1, \ldots, R_l)$

Step 1.   First, the input parties provide the inputs of the computation. The input parties encrypt their inputs x (line 3), and make a proof of knowledge of the plaintext for the encryption (lines 4–5). These encryptions and proofs are put on a broadcast channel, but

to prevent corrupted parties from adaptively choosing their input based on the inputs of others, this happens in two rounds. First, the parties provide hashes as a commitment to their inputs, then they open the commitments (line 6). If any party provides an incorrect input, the protocol is terminated (line 7).

Step 2. Next, the parties provide the plaintext x and randomness s of the encryption to the computation parties in secret-shared form (lines 8–11). We should ensure that the input parties provide consistent sharings. To guarantee correctness of the secret sharing-based MPC, we may also need to ensure that the inputs are bounded, say, that they lie in $\mathbb{Z}_{\langle 2^{\kappa - \log n} \rangle}$. We achieve both at the same time by using statistically secure bounded additive shares $[x_i]^{(j)}, r_{x,i}^{(j)}$: they cannot be inconsistent, and the computation parties can check that the shares they receive respect the bound (line 14). Statistical security holds if the inputs of honest parties are smaller than $\mathbb{Z}_{\langle 2^{\kappa - \log n} \rangle}$ by a statistical security parameter.

Step 3. The computation parties now check if the provided sharing of the input is consistent with the encryptions that were broadcast in step 1. Without this check, corrupted input parties could learn information about both their encrypted and their secret-shared inputs, which should not be possible. They do this by simply encrypting their shares of the inputs using their shares of the randomness, encrypting the result, and checking correctness using the homomorphic property of the cryptosystem (lines 15–16).[3]

Step 4. Then, the actual computation takes place. The computation parties have additive shares $[x_1], \ldots, [x_m]$, which they convert into Shamir's shares $[\![x_1]\!], \ldots, [\![x_m]\!]$ by simply Shamir-sharing each separate additive share, denoted as ADD$\mathbb{Z}$TOSHAMIRp (line 17). The computation is then performed on Shamir's shares using VIFF, giving shares $[\![a_1]\!], \ldots, [\![a_k]\!], [\![r_1]\!], \ldots, [\![r_l]\!]$ of the certificate and result. These Shamir's shares computed with VIFF are finally converted back into additive shares $[a_1], \ldots, [a_k], [r_1], \ldots, [r_l]$ over $\mathbb{Z}$ (line 19). Namely, Shamir's shares are converted into additive shares (a local computation using Lagrange interpolation coefficients), then, these additive shares over $\mathbb{F}_p$ are converted into additive shares over $\mathbb{Z}$ (and hence, $\mathbb{Z}_N$) using a statistically secure protocol due to Algesheimer et al. [14]. Note that this protocol requires $a_1, \ldots, r_l$ to be smaller than p by a statistical security parameter, so p should be chosen to be large enough for this to hold. These operations together are denoted SHAMIRpTOADD$\mathbb{Z}$.

Step 5. Computation parties produce the encrypted result and prove its correctness. First, the parties produce encryptions of the certificate values $a_1, \ldots, a_k$: they exchange encryptions of their respective additive shares (line 22), and use the multiparty Fiat-Shamir heuristic (Sec. 1.3) to prove knowledge of the corresponding plaintexts (line 23).[4] Encryptions of the result $r_1, \ldots, r_l$ are produced in the same way (line 24–28), except that the parties first commit to their encrypted results and then open them (line 26). This seems to be necessary to obtain security in our model.

Finally, the parties run the UVCDN protocol to obtain a proof that $\phi(X, A, R) = 1$ holds (line 29). To be precise, they evaluate ϕ as an arithmetic circuit as described in Sec. 1. Instead of outputting the result privately to a result party by threshold decrypting

---

[3] Alternatively, the computation parties could derive $[x]$ from X. However, this alternative method does not combine well with the optimisations described in Sec. 3, so we will not consider it further.

[4] The homomorphism Φ_{PK} needed is not given in [5], but easily follows from Φ_{CM}. The proof is needed because our security model demands that the adversary know the certificate for which a proof is produced. With the optimizations from Sec. 3, this knowledge usually already follows from the use of UVCDNPROVE, e.g. if the certificate values occur as right-hand sides of a multiplication. The proof can then be left out.

the blinded encryption $V \oplus D$ of the result v, they simply threshold decrypt V so that anybody can observe that it equals one. This call of the UVCDN protocol is denoted UVCDNPROVE. It returns the decrypted result v of computing $\phi(X, A, R)$, and a proof π that v is correct, i.e. the (combined) encryptions D, E, decryptions s and non-interactive zero-knowledge proofs produced during the protocol.

Step 6.   The computation parties send their additive shares of the result, and the randomness used to produce their encryption shares R_i, to the result party (line 30).

Step 7.   One of the computation parties publishes the encryptions of the certificate and computation result and their proof of correctness. The proof of correctness consists of the proof π proving $\phi(X, A, R) = 1$ produced by UVCDNEVAL and the proofs of knowledge $\pi_{a,1}, \ldots, \pi_{r,l}$ for the encryptions (line 31). The result party and verifier check the correctness of these proofs. Namely, both verify the proofs of knowledge (line 33–35), and the proof π of ϕ (line 36). In more detail, this latter check UVCDNVER means going through the circuit for ϕ gate by gate, verifying the multiplication and decryption proofs for each multiplication gate, and verifying the final decryption of the computation of ϕ to the value 1. After checking the proof for ϕ, the verifier is convinced that $R_1, \ldots, R_l$ are valid, and it returns them (line 40). The result party also checks that the shares of the result it received on line 30 were correct (line 38) and then returns the plaintext and randomness for $R_1, \ldots, R_l$ (line 39). Note that the verifier only reads publicly announced values, and does not contribute anything to the computation. Anybody can perform this task, which is why we consider the protocol universally verifiable.

2.2. Security Guarantees

We formalize the security guarantees of our approach by adapting the definitions of Chapter 1 to the outsourcing setting. Recall from Sec. 1 of Chapter 1 that the security of a multiparty protocol is defined by comparing the real-model outcome of executing the protocol to the ideal-model outcome of a simple interaction with an incorruptible ideal component $\mathcal{F}^f_{\mathsf{SMC}}$ that receives the inputs $x_1, \ldots, x_m$ and distributes the output $r = f(x_1, \ldots, x_m)$. Security means that under some assumptions on the number and type of corruptions, the real-model and ideal-model outcomes have similar distributions (e.g. they are computationally indistinguishable).

To express the security guarantees of universal verifiability by certificate validation, we define an ideal component $\mathcal{F}^{f,\phi}_{\mathsf{VerSMC}}$ that differs from $\mathcal{F}^f_{\mathsf{SMC}}$ in several ways. First, verifiability requires us to consider different attacker models for the different security properties: robustness, privacy, and correctness. Namely, correctness (in the sense that the solution satisfies ϕ) should hold regardless of who is corrupted, robustness and privacy depend on who is corrupted. To capture this, our ideal component acts differently depending on the corruptions at hand, and the real-model and ideal-model outcomes have similar distributions for all possible corruptions. Second, universal verifiability requires capturing that the verifier receives the result of the computation in encrypted form. Hence, we change the ideal model of Def. 4 in Chapter 1 so that it first performs a trusted set-up of the threshold homomorphic cryptosystem and then gives the public key pk to $\mathcal{F}^{f,\phi}_{\mathsf{VerSMC}}$; $\mathcal{F}^{f,\phi}_{\mathsf{VerSMC}}$ uses pk to produce encryptions.[5] Third, verifiability by certificate validation guarantees that $\phi(x, a, r)$ holds rather than that $r = f(x)$, which is captured by

---

[5]This is similar to defining security in a hybrid model [15,7], except that we also let the ideal component use the result of the trusted set-up as we need this to model that the verifier's outputs are encryptions.

Algorithm 1 Ideal component for verifiability by certificate validation

---

Require: Parties C corrupted, $A \subset C$ actively corrupted, pk is a public key

 1: **function** $\mathcal{F}_{\mathsf{VerSMC}}^{f,\phi}(C, A, \mathsf{pk})$
 2:     ▷ input phase
 3:     **for all** $\mathcal{I}_i \in \mathcal{I} \setminus C$ **do** $x_i \leftarrow \mathsf{recv}(\mathcal{I}_i)$
 4:     $\{x_i\}_{i \in \mathcal{I} \cap C} := \mathsf{recv}(\mathcal{S})$
 5:     **if** $\mathcal{P} \cap A \neq \emptyset \vee |\mathcal{P} \cap C| \geq t$ **then** $\mathsf{send}(\{x_i\}_{i \in \mathcal{I} \setminus C}; \mathcal{S})$
 6:     ▷ computation phase
 7:     **if** $\mathcal{P} \cap A = \emptyset$ **then**
 8:         $a_1, \ldots, r_l \leftarrow f(x_1, \ldots, x_m)$
 9:     **else**
10:         $a_1, \ldots, r_l \leftarrow \mathsf{recv}(\mathcal{S})$; **if** $\neg\phi(x_1, \ldots, r_l)$ **then** $r_1, \ldots, r_l \leftarrow \bot$
11:     **if** $\mathcal{P} \cap A \neq \emptyset \vee |\mathcal{P} \cap C| \geq t$ **then** $s_1, \ldots, s_l \leftarrow \mathsf{recv}(\mathcal{S})$ **else** $s_1, \ldots, s_l \in_R \mathfrak{R}$
12:     ▷ result phase
13:     **if** $r_1, \ldots, r_l \neq \bot$ **then** $R_1, \ldots, R_l \leftarrow \mathsf{Enc}_{\mathsf{pk}}(r_1, s_1), \ldots, \mathsf{Enc}_{\mathsf{pk}}(r_l, s_l)$
14:         **else** $R_1, \ldots, R_l \leftarrow \bot$
15:     $\mathsf{send}(R_1, \ldots, R_l; \mathcal{S})$; $\mathsf{send}(R_1, \ldots, R_l; \mathcal{V})$
16:     **if** $\mathcal{P} \cap A = \emptyset \vee \mathsf{recv}(\mathcal{S}) = \top$ **then** $\mathsf{send}((r_1, s_1), \ldots, (r_l, s_l); \mathcal{R})$

---

allowing an active attacker to choose the output r, and letting $\mathcal{F}_{\mathsf{VerSMC}}^{f,\phi}$ check if ϕ holds for this r.

Our ideal component $\mathcal{F}_{\mathsf{VerSMC}}^{f,\phi}$ for universal verifiability by certificate validation is shown in Alg. 1. As inputs it receives a set C of corrupted parties, of which those in A are actively corrupted, and public key pk used to encrypt the result for the verifier. First, the ideal component receives the inputs of the honest and corrupted parties (lines 3–4). If $|\mathcal{P} \cap C| \geq t$ or $\mathcal{P} \cap A \neq \emptyset$, we do not guarantee privacy, so $\mathcal{F}_{\mathsf{VerSMC}}^{f,\phi}$ sends the inputs to the adversary (line 5). Note that this happens *after* the adversary supplies its inputs, so it cannot choose the corrupted parties' inputs adaptively. The computation differs depending on whether $\mathcal{P} \cap A \neq \emptyset$, i.e. on whether there are any actively corrupted computation parties. If not, then the computation is performed according to function f (line 8). If there are corrupted computation parties, the adversary might arbitrarily interfere with the passively secure protocol used to evaluate f, which we capture by letting the adversary choose outputs a and r. However, the adversary can only choose outcomes for which ϕ holds, so $\mathcal{F}_{\mathsf{VerSMC}}^{f,\phi}$ checks this and otherwise sets r to $\bot$ (line 10). Finally, the ideal component produces encryptions which it provides to the verifier (line 15), and it provides the corresponding plaintext and randomness to the result party (line 16). This step can be blocked by the adversary. It is understood that, if one of the inputs to f or ϕ is $\bot$, the outcome is also $\bot$. Moreover, if one of the x_i input to f is not in $\mathbb{Z}_{\langle 2^\kappa \rangle}$, then f returns $\bot$.

We will now state our security definition and theorem. The details are given in Sec. 2.3.

Definition 3. An n-party protocol Π for a functionality f and validation function ϕ is a *secure multiparty protocol with universal verifiability by certificate validation* if, for all adversaries $\mathcal{A}$ corrupting set C of parties and actively corrupting $A \subset C$, there exists an adversary $\mathcal{S}$ so that for all possible inputs $\vec{x}$ to f,

$$\mathrm{REAL}_{\Pi,\mathcal{A}}^{A,C}(\vec{x}) \stackrel{d}{=} \mathrm{IDEAL}_{f,\phi,\mathcal{S}}^{A,C}(\vec{x}) \ .$$

(Here, $\overset{d}{=}$ denotes computational indistinguishability.)

Theorem 4. *The* VERMPC *protocol as shown in Protocol 2 is a secure multiparty protocol with universal verifiability by certificate validation.*

Several remarks about our security guarantees are in place. The presented security guarantees are specific to our using VIFF for the secure evaluation of f. In particular, universal verifiability by certificate validation always guarantees correctness, but privacy and robustness depend on the techniques used. We provide security against corrupted input parties in the sense that they cannot learn the inputs of other parties or let their own input depend on them. On the other hand, any single corrupted input party can block the full computation. This is not the case in [5], in which incorrect inputs are simply set to zero. As noted before, we cannot achieve this because input parties may provide inconsistent encrypted and shared inputs. Alternative solutions without this limitation are possible, but probably less efficient. We do not model data leakage (e.g. the number of iterations used) from the secret sharing computation as this can be done in standard ways. We also note that the trusted party takes the public key pk as input. In particular, we assume a secure set-up of the encryption scheme, and secure distribution of the threshold decryption keys to the computation parties. This ideal-world trusted party should be interpreted in the standalone execution environment in the random oracle model as defined in [5]. Their remarks on composability and the use of random oracles also apply here.

2.3. Security Proof of the VERMPC Protocol

We prove Thm. 4 in two steps. We need to show that, for all adversaries in the real-model execution, there is an adversary in the ideal-model execution that gives rise to the same output distribution. We explicitly construct two simulators. Simulator $\mathcal{S}_{\text{VERMPC}}^{\text{PRIVATE}}$ (Alg. 2) simulates an adversary that passively corrupts fewer than t computation parties. In this case, the protocol guarantees privacy. We prove this in Lemma 5 by reducing the indistinguishability of the ideal-model and real-model executions to semantic security of the threshold homomorphic cryptosystem. Simulator $\mathcal{S}_{\text{VERMPC}}^{\text{CORRECT}}$ (Alg. 3) simulates an adversary that actively corrupts computation parties, or passively corrupts at least t computation parties. In this case, privacy is not guaranteed, and we prove correctness in Lemma 6 by showing that the real-model and ideal-model outcomes are statistically indistinguishable. The two lemmas together imply Thm. 4.

We briefly discuss some notation we use in our simulators. We work in the random oracle model, in which the hash function $\mathcal{H}$ is modelled by an oracle $\mathcal{O}$ to which all parties send queries. The simulator can program this oracle. We model it as a partial function (i.e. a set of name/value pairs) that the simulator can change at will (cf. [5]). When simulating an adversary $\mathcal{A}$, denoted $\mathcal{A}^{\mathcal{O}}()$, the simulator handles all oracle queries and responds to new queries by sampling elements from the codomain of the hash function. Finally, as we discuss below, we use the witness-extended simulation techniques from Groth [16]. The witness-extended emulator of an adversary $\mathcal{B}$ is denoted $\mathcal{E}_{\mathcal{B}}$.

First, we deal with the case $\mathcal{P} \cap A = \emptyset \land |\mathcal{P} \cap C| < t$. The simulator $\mathcal{S}_{\text{VERMPC}}^{\text{PRIVATE}}$ (Alg. 2) for this case needs to simulate the protocol with respect to the adversary without knowing the inputs of the honest parties. However, because the adversary only passively corrupts few parties, the simulator can in fact perform the computation with zero inputs for the honest parties without the adversary noticing. Hence, the simulator performs step 1 of the protocol by making zero encryptions on behalf of the honest parties (line 3) and

simulating their proofs of knowledge (line 4).[6] It programs the random oracle so that the simulated proof $\pi_{x,i}$ validates, and h_i is a commitment to the encryption and proof (line 5). Then, it needs to extract the inputs of the corrupted parties. For this, we use the technique of witness-extended emulation in the random oracle model due to Groth [16]. In [16], it is proven that for every adversary $\mathcal{B}$ that produces non-interactive zero-knowledge proofs, we can construct an extractor $\mathcal{E}_{\mathcal{B}}$ that, along with these proofs, returns the witness to the simulator. The extractor lets the adversary produce a proof, rewinds to the random oracle query for the challenge to obtain a new proof with a different challenge, and extracts the witness by special soundness. Hence, we define an adversary $\mathcal{B}$ that exchanges inputs with the simulated adversary $\mathcal{A}$ and returns the proofs of knowledge of $\mathcal{A}$ (lines 7–10). From this adversary, the simulator can extract the plaintexts x_i for any encryptions that will be accepted in the real protocol (line 11). Then, the simulator simply runs steps 2 and 3 with respect to the adversary (line 12). If the corrupted input parties provided consistent sharings, then the simulator provides the corrupted inputs to the trusted party (line 14).

The simulator can then continue to simulate the protocol using the zero inputs (line 15); it only needs to make sure that the resulting encryptions are those it receives from the trusted party in line 16. This is the reason for the additional round of hashes on the result encryption in the protocol (line 26), as this way, the simulator can extract the corrupted parties' contributions from their hashes, and make its own contributions match them (lines 18–24). If the result party is corrupted, the simulator receives the output of the computation (line 26) and matches its contributions of the shares for the result party with those outputs. Otherwise, it continues to simulate the protocol on the zero inputs.

We will now show that this simulator is correct.

Lemma 5. *For all inputs* $\vec{x}, C, A, a$ *with* $\mathcal{P} \cap A = \emptyset \wedge |\mathcal{P} \cap C| < t$,

$$\text{REAL}^{A,C}_{\text{VerMPC},\mathcal{A}}(\vec{x}) \stackrel{c}{=} \text{IDEAL}^{A,C}_{f,\phi,\mathcal{S}^{\text{PRIVATE}}_{\text{VerMPC}}}(\vec{x}) \ ,$$

where $\stackrel{c}{=}$ *denotes computational indistinguishability.*

Proof. Suppose that the two distributions are not computationally indistinguishable. That is, there are inputs $x'_1, \ldots, x'_m, C, A, a, \mathcal{A}$ with $\mathcal{P} \cap A = \emptyset \wedge |\mathcal{P} \cap C| < t$ so that there is a distinguisher $\mathcal{D}$ between the REAL and IDEAL distributions. We use $\mathcal{D}$ to build a distinguisher $\mathcal{D}'$ for the threshold homomorphic encryption scheme used.

Intuitively, the distinguisher $\mathcal{D}'$ works as follows. As input, it receives the public key of the threshold homomorphic cryptosystem, the threshold decryption keys of the corrupted computation parties $\mathcal{P} \cap C$, auxiliary information a, and an encryption B of $b = 0$ or $b = 1$. We construct $\mathcal{D}'$ so that if $b = 0$, it produces an output that is statistically indistinguishable from the IDEAL distribution on input $x'_1, \ldots, a$, and if $b = 1$, it produces an output that is statistically indistinguishable from the REAL distribution on input $x'_1, \ldots, a$. Then, it runs $\mathcal{D}$ on the outputs of IDEAL and REAL. Hence, if $\mathcal{D}$ can distinguish IDEAL and REAL, then $\mathcal{D}'$ can distinguish between an encryption of 0 and 1, contradicting the semantic security of the threshold homomorphic encryption scheme.

Let us look at how $\mathcal{D}'$ performs the simulation depending on B. Let $\bar{B} = 1 \ominus B$ denote an encryption of $1 - b$. Broadly speaking, $\mathcal{D}'$ runs simulator $\mathcal{S}^{\text{PRIVATE}}_{\text{VerMPC}}$. However,

---

[6]The simulator could *make* the proof because he knows the plaintext and randomness, but the security proof depends on being able to perform the simulation without knowing the plaintext.

Algorithm 2 Simulator for VerMPC (private case)

1: **function** $\mathcal{S}_{\text{VerMPC}}^{\text{PRIVATE}}(\text{pk}, C, A, \{x_i\}_{i \in C}, \{s_i\}_{i \in \mathcal{P} \cap C}, a)$
2:     **for all** $i \in \mathcal{I} \setminus C$ **do**
3:         $x_i \leftarrow 0$; $r_{\text{x},i} \in_R \mathfrak{R}$; $X_i \leftarrow \text{Enc}_{\text{pk}}(x_i, r_{\text{x},i})$;
4:         $h_i', h_i \in_R \text{codom}(\mathcal{O})$; $(a_i, h_i', r_i) \leftarrow \Sigma.\text{sim}(X_i, h_i')$; $\pi_{\text{x},i} \leftarrow (a_i, h_i', r_i)$
5:         $\mathcal{O} \leftarrow \mathcal{O} \cup \{(X_i || a_i || i, h_i'), (i || X_i || \pi_{\text{x},i}, h_i)\}$
6:     **function** $\mathcal{B}$
7:         $\{h_i\}_{i \in \mathcal{I} \cap C} \leftarrow \mathcal{A}^{\mathcal{O}}(\{h_i\}_{i \in \mathcal{I} \setminus C})$
8:         $\{(X_i, \pi_{\text{x},i})\}_{i \in \mathcal{I} \cap C} \leftarrow \mathcal{A}^{\mathcal{O}}(\{(X_i, \pi_{\text{x},i})\}_{i \in \mathcal{I} \setminus C})$
9:         $S \leftarrow \{i \in \mathcal{I} \cap C \mid (i || X_i || \pi_{\text{x},i}, h_i) \in \mathcal{O} \land \text{NizkVer}(\Sigma_{\text{PK}}, X_i, \pi_{\text{x},i}, i)\}$
10:         **return** $(S, \{(X_i, \pi_{\text{x},i})\}_{i \in S})$
11:     $(S, \{(X_i, \pi_{\text{x},i}, x_i, r_{\text{x},i})\}_{i \in S}) \leftarrow \mathcal{E}_{\mathcal{B}}()$
12:     $\langle$perform protocol lines 8–16 w.r.t. $\mathcal{A}\rangle$
13:     **if** $S \neq \mathcal{I} \cap C$ **or** some $\mathcal{P}_j$ returned $\perp$ **then** $\text{send}(\perp, \ldots, \perp; \mathcal{F}_{\text{VerSMC}}^{f,\phi})$; **return**
14:     $\text{send}(\{x_i\}_{i \in \mathcal{I} \cap C}; \mathcal{F}_{\text{VerSMC}}^{f,\phi})$
15:     $\langle$perform protocol lines 17–23 w.r.t. $\mathcal{A}\rangle$
16:     $R_1, \ldots, R_l \leftarrow \text{recv}(\mathcal{F}_{\text{VerSMC}}^{f,\phi})$
17:     **for all** $1 \leq i \leq l$ **do**
18:         **for all** $j \in \mathcal{P} \setminus C$ **do** $h_{i,j} \in_R \text{codom}(\mathcal{O})$
19:         $\{h_{i,j}\}_{j \in \mathcal{P} \cap C} \leftarrow \mathcal{A}^{\mathcal{O}}(\{h_{i,j}\}_{j \in \mathcal{P} \setminus C})$
20:         **for all** $j \in \mathcal{P} \cap C$ **do** $R_i^{(j)} \leftarrow (x \text{ s.t. } (i || j || x, h_{i,j}) \in \mathcal{O} \text{ else } \perp)$
21:         $\{R_i^{(j)}\}_{j \in \mathcal{P} \setminus C} \leftarrow \langle$random values s.t. $\oplus_j R_i^{(j)} = R_i\rangle$
22:         **for all** $j \in \mathcal{P} \setminus C$ **do** $\mathcal{O} \leftarrow \mathcal{O} \cup \{(i || j || R_i^{(j)}, h_{i,j})\}$
23:         $\{R_i'^{(j)}\}_{j \in \mathcal{P} \cap C} \leftarrow \mathcal{A}^{\mathcal{O}}(\{R_i^{(j)}\}_{j \in \mathcal{P} \setminus C})$; **if** $\exists j : R_i'^{(j)} \neq R_i^{(j)}$ **then return** $\perp$
24:         $(\_, \pi_{\text{r},i}, ([r_i]^{(j)}, r_{\text{r},i,j})_{j \in \mathcal{P} \cap C}) \leftarrow \mathcal{S}_{\text{M}\Sigma}(\Sigma_{\text{PK}}, \Phi_{\text{PK}}, R_i^{(j)})$
25:     **if** $\mathcal{R} \in C$ **then**
26:         $((r_1, s_1), \ldots, (r_l, s_l)) \leftarrow \text{recv}(\mathcal{F}_{\text{VerSMC}}^{f,\phi})$
27:         **for all** $1 \leq i \leq l$ **do** $\{([r_i]^{(j)}, r_{\text{r},i,j})\}_{j \in \mathcal{P} \setminus C} \leftarrow \langle$random summing to $r_1, s_1\rangle$
28:     $\mathcal{S}_{\text{UvCdnProve}_\phi}^{\text{PRIVATE}}(X_1, \ldots, R_l)$
29:     $\langle$perform protocol lines 30–40 w.r.t. $\mathcal{A}\rangle$

---

on line 3, it computes X_i as $B^{x_i'} \oplus \text{Enc}_{\text{pk}}(0, r_{\text{x},i})$. If B encrypts zero, this is a random encryption of 0; if B encrypts one, this is a random encryption of the actual input of the honest party. Then, when simulating the protocol with respect to the adversary, on line 15 of the protocol, it computes contributions of $X_i^{(j)}$ so that $\oplus_j X_i^{(j)} = X_i$. It simulates the secret sharing computation (line 18) as in the simulator, i.e. with the shares x_i of zero. Because of the security of the secret sharing computation, this is statistically indistinguishable from the computation in the real world with the values x_i'. Similarly to what it did for the encrypted inputs X_i, it computes the encrypted certificate values A_i' as $B^{a_i'} \oplus \bar{B}^{a_i}$, where a_i' is output by f on input $(x_1', \ldots, x_m')$ and a_i is output by f on input $(x_1, \ldots, x_m)$. Then, it chooses the honest parties' $A_i^{(j)}$ at random so that $(\oplus_{j \in \mathcal{P} \setminus C} A_i^{(j)}) \oplus (\oplus_{j \in \mathcal{P} \cap C} \text{Enc}([a_i]^{(j)}, 0)) = A_i'$. Although the value A_i resulting from the simulation will not be the same as A_i', it will encrypt the same value and have the same distribution. To perform the proof of knowledge on line 23, it uses the simulator for $M\Sigma$. For the encryptions $R_1, \ldots, R_l$ of the result on line 16, it computes actual

---

Algorithm 3 Simulator for VERMPC (correct case)

1: **function** $\mathcal{S}^{\text{CORRECT}}_{\text{VERMPC}}(\text{pk}, C, A, \{x_i\}_{i \in C}, \{s_i\}_{i \in \mathcal{P} \cap C}, a)$
2:     **for all** $i \in \mathcal{I} \setminus C$ **do** $h_i \in_R \text{codom}(\mathcal{O})$
3:     **function** $\mathcal{B}_1$
4:         $\{h_i\}_{i \in \mathcal{I} \cap C} \leftarrow \mathcal{A}^{\mathcal{O}}(\{h_i\}_{i \in \mathcal{I} \setminus C})$
5:         **for all** $i \in \mathcal{I} \cap C$ **do** $(X_i, \pi_{\text{x},i}) \leftarrow ((x, \pi) \text{ s.t. } (i||x||\pi, h_i) \in \mathcal{O} \text{ else } \bot)$
6:         $S \leftarrow \{i \in \mathcal{I} \cap C \mid X_i \neq \bot \wedge \text{NIZKVER}(\Sigma_{\text{PK}}, X_i, \pi_{\text{x},i})\}$
7:         **return** $(S, \{(X_i, \pi_{\text{x},i})\}_{i \in S})$
8:     $(S, \{(X_i, \pi_{\text{x},i}, x_i, r_{\text{x},i})\}_{i \in S}) \leftarrow \mathcal{E}_{\mathcal{B}_1}()$
9:     $\text{send}(\{x_i\}_{i \in \mathcal{I} \cap C}; \mathcal{F}^{f,\phi}_{\text{VerSMC}})$
10:     $\{x_i\}_{i \in \mathcal{I} \setminus C} \leftarrow \text{recv}(\mathcal{F}^{f,\phi}_{\text{VerSMC}})$
11:     **for all** $i \in \mathcal{I} \setminus C$ **do**
12:         $r_{\text{x},i} \in_R \mathfrak{R}; X_i \leftarrow \text{Enc}_{\text{pk}}(x_i, r_{\text{x},i}); (a_i, s_i) \leftarrow \Sigma_{\text{PK}}.\text{ann}(X_i, (x_i, r_{\text{x}_i}))$
13:         $c_i \in_R \text{codom}(\mathcal{O}); r_i \leftarrow \Sigma_{\text{PK}}.\text{res}(X_i, (x_i, r_{\text{x}_i}), a_i, s_i, c_i); \pi_{\text{x},i} \leftarrow (a_i, c_i, r_i)$
14:         $\mathcal{O} \leftarrow \mathcal{O} \cup \{(c_i, X_i||a_i||i), (i||X_i||\pi_{\text{x},i}, h_i)\}$
15:     $\{(X'_i, \pi'_{\text{x},i})\}_{i \in \mathcal{I} \cap C} \leftarrow \mathcal{A}^{\mathcal{O}}((X_i, \pi_{\text{x},i})_{i \in \mathcal{I} \setminus C})$
16:     **if** $\exists j : X_j \neq X'_j \vee \pi_{\text{x},j} \neq \pi'_{\text{x},j} \vee \neg\text{NIZKVER}(\Sigma_{\text{PK}}, X_j, \pi_{\text{x},j}, j)$ **then return** $\bot$
17:     **function** $\mathcal{B}_2$
18:         ⟨perform protocol lines 8–40⟩
19:         **return** $(A_i, \pi_{\text{a},i}, R_i, \pi_{\text{r},i})$
20:     $(A_i, \pi_{\text{a},i}, a_i, r_{\text{a},i}, R_i, \pi_{\text{r},i}, r_i, r_{\text{r},i}) \leftarrow \mathcal{E}_{\mathcal{B}_2}()$
21:     **if** $\mathcal{P} \cap A \neq \emptyset$ **then**
22:         **if** ⟨$\mathcal{V}$ accepts⟩ **then** $\text{send}(a_1, \ldots, r_l; \mathcal{F}^{f,\phi}_{\text{VerSMC}})$ **else** $\text{send}(\bot, \ldots; \mathcal{F}^{f,\phi}_{\text{VerSMC}})$
23:     $\text{send}(r_{\text{r},1}, \ldots, r_{\text{r},l}; \mathcal{F}^{f,\phi}_{\text{VerSMC}})$
24:     **if** $\mathcal{P} \cap A \neq \emptyset$ **then**
25:         **if** ⟨$\mathcal{R}$ accepts proof/shares⟩ **then** $\text{send}(\top; \mathcal{F}^{f,\phi}_{\text{VerSMC}})$ **else** $\text{send}(\bot; \mathcal{F}^{f,\phi}_{\text{VerSMC}})$

---

random encryptions of $f(x'_1, \ldots, x'_m)$. For the values $((r_1, s_1), \ldots, (r_l, s_l))$ on line 26, if applicable, it uses the x'_i and randomness used for R_i. The simulation of the UVCDN protocol on line 28 can also be done conditionally on the encrypted bit B (see [5]). After this, the remainder of the protocol (lines 30–40) simply consists of sending around already constructed data, which is now easy.

The distinguisher $\mathcal{D}'$ has now performed an ideal world protocol execution (if B encrypts zero) or a real world protocol execution (if B encrypts one). Indeed, in both cases, one verifies, step by step, that the messages seen by the adversary and the outputs of the honest parties are statistically indistinguishable from the IDEAL and REAL distributions, respectively. As discussed above, $\mathcal{D}'$ can now ask $\mathcal{D}$ to distinguish the two situations and hence, distinguish encryptions of zero and one. However, this is a contradiction, proving that IDEAL and REAL must in fact be computationally indistinguishable. □

Now let us look at the second case where $\mathcal{P} \cap A \neq \emptyset \vee |\mathcal{P} \cap C| \geq t$. Here the simulator $\mathcal{S}^{\text{CORRECT}}_{\text{VERMPC}}$ (Alg. 3) can use the honest parties' inputs, but it only learns these encryptions after it provides the corrupted parties' inputs. We extract these inputs as before using witness-extended emulation, except that we now already extract them from the hash provided in line 6 of the protocol. Namely, we define an adversary $\mathcal{B}$ that receives the hash from the adversary, reads the proof from the random oracle, and returns this proof (lines 3–7). Again, due to witness-extended emulation, there is an extractor

$\mathcal{E}_\mathcal{B}$ that can extract the plaintext of X_i from this proof (line 8). Now, the simulator sends these extracted inputs to $\mathcal{F}_{\mathsf{VerSMC}}^{f,\phi}$ (line 9). It receives the honest parties' inputs in return (line 10), and then simply provides actual encryptions of these inputs to the adversary (lines 11–15).

Now that the simulator knows the honest parties' inputs, it can run the actual protocol with respect to the adversary (line 18). If $|\mathcal{P} \cap C| \geq t$, then by interpolation from the corrupted decryption keys $\{s_i\}_{i \in \mathcal{P} \cap C}$, it can also determine the honest decryption keys (cf. [5]). Hence, in this case, it knows everything so it is easy to run the protocol together with the adversary. However, also in case $|\mathcal{P} \cap C| < t$, the simulator has enough information to run the remainder of the protocol. Namely, it knows the plaintexts for all encryptions $A_i^{(j)}, R_i^{(j)}$ of both honest and corrupted parties: for the honest parties, it has generated them, and for the corrupted parties, it can extract them by simulating the $M\Sigma$ protocols from lines 23 and 28. In short, for all encryptions encountered in the protocol, including those in UvCDNEval, it knows the corresponding plaintext. Hence, it can simulate decryption to the right values, even if it does not know the decryption keys of the honest parties. Finally, by applying witness-extended emulation to the protocol run (line 20), the simulator learns the plaintext certificate and result produced by the computation, which it forwards to $\mathcal{F}_{\mathsf{VerSMC}}^{f,\phi}$ depending on whether $\mathcal{V}$ and $\mathcal{R}$ receive a valid proof (lines 21–25).

We will now prove the correctness of this simulator.

Lemma 6. *For all inputs $\vec{x}, C, A, a$ with $\mathcal{P} \cap A \neq \emptyset \vee |\mathcal{P} \cap C| \geq t$,*

$$\mathrm{REAL}_{\mathsf{VerMPC},\mathcal{A}}^{A,C}(\vec{x}) \stackrel{s}{=} \mathrm{IDEAL}_{f,\phi,\mathcal{S}_{\mathsf{VerMPC}}^{\mathrm{PRIVATE}}}^{A,C}(\vec{x}) \ ,$$

where $\stackrel{s}{=}$ denotes statistical indistinguishability.

Proof. For the two distributions to be statistically indistinguishable, the simulator has to simulate the protocol with respect to the adversary in a statistically indistinguishable way, and the outputs of the result party and the verifier in the two cases need to be the same. First, consider how the protocol is simulated with respect to the adversary. The simulator consists of two parts: first, on lines 2–16, it simulates lines 2–7 of the protocol, then it runs the remainder of the protocol.

For the first part, the real-world and ideal-world executions are statistically indistinguishable with respect to the adversary by inspection. Indeed, on behalf of the honest input parties, the simulator provides a hash h_i, followed by a pre-image $i||X_i||\pi_i$ containing the actual encrypted input X_i and a proof of knowledge π_i. This is the same as in the real protocol. The only difference is that by controlling the random oracle, the simulator can extract the inputs of the corrupted parties from the hashes they provide (line 8) *before* determining its own pre-image to the hash (line 14), which it needs to do because the trusted party needs the corrupted inputs before giving the honest inputs.

For the second part, as argued previously, the simulator can simply run the remainder of the protocol. From this, it is clear that the outputs of the adversary in the ideal and real world are statistically indistinguishable. For the outputs of the result party and the verifier, the simulator simply checks if the adversary supplies a correct proof (lines 22, 25) and uses that to decide if the trusted party should pass the computation result to the ideal-world result party and verifier. Note that the trusted party does not directly supply these to the result party and the verifier, it first checks if ϕ actually holds on them. However, if

ϕ does not hold and the proofs presented to $\mathcal{R}$ and $\mathcal{V}$ nonetheless verify, then this means that the adversary has managed to produce non-interactive zero-knowledge proofs for incorrect statements. The chance of the adversary succeeding in this is only negligible, so with overwhelming probability, ϕ holds and hence, the ideal-world and real-world outputs of the result party and the verifier are the same. This concludes the proof. $\square$

3. More Efficient Validation of Polynomial Equations

In this section we show that, if verification consists of checking a number of polynomial equations in $\vec{x}, \vec{a}, \vec{r}$, then the efficiency of UVCDN in our setting can be considerably improved. As we will see later, the optimality of a solution for a linear program can indeed be phrased in this way. This improvement comes from two observations.

First, note that we require $(\vec{a}, \vec{r}) = f(\vec{x})$ to imply $\phi(\vec{x}, \vec{a}, \vec{r})$. In particular, if $\phi = \phi_1 \wedge \ldots \wedge \phi_k$, then all equations need to hold. Hence, we do not need to securely compute the and-function; instead, we produce separate proofs for each of the k equations.

Second, the verification of polynomial equations can be done using the ElGamal threshold homomorphic cryptosystem. Recall that in the additively homomorphic ElGamal cryptosystem, the public key is a pair (g, h) of generators of a discrete logarithm group of the size p (where p is a prime) so that $s = \log_g h$ is unknown, the private key is s, the encryption of $m \in \mathbb{Z}_p$ with randomness $r \in \mathbb{Z}_p$ is $(g^r, g^m h^r)$, and decryption of (a, b) is $g^m = ba^{-s}$. Because ElGamal decrypts to g^m and not to m, it is only possible to decrypt small values for which the discrete logarithm problem with respect to g is tractable. ElGamal is turned into a threshold cryptosystem by applying Shamir's secret sharing to the private key s, resulting in shares s_i [17]. Parties publish their decryption shares a^{-s_i}, recombine to a^{-s}, and compute g^m from this. Proofs of knowledge, correct multiplication, and correct decryption and homomorphisms are similar to the Paillier case [6]. ElGamal has two main advantages over Paillier: encryptions at the same security level are smaller[7], and it is easier to perform verifiable key generation. However, we cannot directly use ElGamal for UVCDN, because multiplication in UVCDN requires the decryption of random (i.e. large) ciphertexts.

However, for the verification of a polynomial equation, it is not actually necessary to use the full UVCDN multiplication protocol. Namely, let $\psi = 0$ be a polynomial equation, i.e. $\psi \in \mathbb{Z}_N[\vec{x}, \vec{a}, \vec{r}]$ is a polynomial in $\vec{x}, \vec{a}, \vec{r}$. We can prove that $\psi = 0$ by producing an encryption of the left-hand side, and proving that it encrypts to 0. We can do the latter using ElGamal encryption by decrypting to $g^0 = 1$. For the former, we only need to repeatedly multiply some encryption Z with an encryption $R \in \{X_1, \ldots, R_l\}$. However, recall that we have additive shares of the encryptions X, A, R as a result of the computation of f. Hence, we can perform this multiplication by letting each party multiply Z by its additive share $[R]^{(j)}$ in a provably correct way, and combining these proofs into a proof of multiplication of Z by $R = [R]^{(1)} \oplus \ldots \oplus [R]^{(n)}$ using the techniques from Sec. 1.3. This way, we can prove that an equation $\phi = 0$ holds without decrypting random ciphertexts.

Finally, we note that because the message space of ElGamal encryption has the prime order p, it is possible to perform the VIFF multiparty computation directly modulo this prime. In this case, the call to the SHAMIRpTOADD$\mathbb{Z}$ protocol due to [14] in line 19

---

[7]For example, 224 bits instead of 2,048 for Paillier, using elliptic curve-based discrete logarithm groups instead.

of the VERMPC protocol is no longer needed. On the other hand, increasing the modulus for VIFF to p makes the VIFF computation slower. We discuss this trade-off in our experiments section (Sec. 5.4).

4. Linear Programming

We will now introduce linear programming, our application for verifiability by certificate validation. Linear programming is a broad class of optimization problems occurring in many applications. For instance, it was used to compute the optimal price in the Danish sugar beet auctions that were performed using MPC [18]. Specifically, the problem is to minimize the output of a linear function, subject to linear constraints on its variables. One instance of this problem is called a linear program, and it is given by a matrix $\mathbf{A}$ and vectors $\mathbf{b}$ and $\mathbf{c}$. The vector $\mathbf{c} = (c_1, \ldots, c_n)$ gives the linear function $\mathbf{c}^T \cdot \mathbf{x} = c_1 \cdot x_1 + \ldots + c_n \cdot x_n$ in variables $\mathbf{x} = (x_1, \ldots, x_n)$ that needs to be minimized. The matrix $\mathbf{A}$ and vector $\mathbf{b}$ give the constraints $\mathbf{A} \cdot \mathbf{x} \leq \mathbf{b}$ that need to be satisfied. $\mathbf{A}$ has n columns, and $\mathbf{A}$ and $\mathbf{b}$ have the same number of rows (say, m), where each row is a constraint. In addition to these constraints, we require $x_i \geq 0$. For instance, the linear program

$$\mathbf{A} = \begin{pmatrix} 1 & 2 & 1 \\ 1 & -1 & 2 \end{pmatrix}, \mathbf{b} = \begin{pmatrix} 2 \\ 1 \end{pmatrix}, \mathbf{c} = \begin{pmatrix} -10 \\ 3 \\ -4 \end{pmatrix} \tag{1}$$

represents the problem to find x_1, x_2, x_3 satisfying $x_1 + 2x_2 + x_3 \leq 2$, $x_1 - x_2 + 2x_3 \leq 1$, and $x_1, x_2, x_3 \geq 0$, so that $-10x_1 + 3x_2 - 4x_3$ is minimal.

In this chapter, we only consider linear programs for which $\mathbf{b} \geq 0$. This means that the simplex algorithm discussed below does not need an initialization step [1]. We make this assumption to simplify our discussion and because it does not have an impact on the verifiability aspects on which we focus in this chapter. However, a full implementation of linear programming should not make this assumption.

4.1. Small Tableau Simplex

Small tableau simplex (a variant of the simplex algorithm) is a well-known algorithm for solving linear programs. The algorithm is best understood by thinking of the inequalities $\mathbf{A} \cdot \mathbf{x} \leq \mathbf{b}$ as equalities $\mathbf{A}' \cdot \mathbf{x}' = \mathbf{b}$ involving so-called slack variables $x_{n+1}, \ldots, x_{n+m}$. Solving a linear program then means minimizing $\mathbf{c}'^T \cdot \mathbf{x}'$ subject to $\mathbf{A}' \cdot \mathbf{x}' = \mathbf{b}$ and $x_i \geq 0$. For instance, the above linear program can be re-written as the following system:

$$\begin{aligned} \text{given} \quad & 1 \cdot x_1 + 2 \cdot x_2 + 1 \cdot x_3 + 1 \cdot x_4 + 0 \cdot x_5 = 2 \\ & 1 \cdot x_1 + -1 \cdot x_2 + 2 \cdot x_3 + 0 \cdot x_4 + 1 \cdot x_5 = 1, \\ \text{minimize} \quad & -10 \cdot x_1 + 3 \cdot x_2 + -4 \cdot x_3 + 0 \cdot x_4 + 0 \cdot x_5 \ . \end{aligned} \tag{2}$$

This formulation of the problem directly shows an assignment to the x_i that satisfies the constraints (but does not necessarily minimize the value of $\mathbf{c}'^T \cdot \mathbf{x}'$): $(x_1, \ldots, x_n) = (0, \ldots, 0)$ and $(x_{n+1}, \ldots, x_{n+m}) = \mathbf{b}$. This works because $\mathbf{b} \geq 0$. We call the first n variables, which are set to 0, the *co-basis*, and we call the last m variables, which are set to $\mathbf{b}$, the *basis*. The value of $\mathbf{c}'^T \cdot \mathbf{x}'$ for this assignment is zero, and the above formulation

of the linear program directly suggests a way of decreasing it. Namely, observe that the value of $\mathbf{c}'^T \cdot \mathbf{x}'$ depends negatively on variables x_1 and x_3 in the co-basis, and it does not depend on variables x_4 and x_5 in the basis. Hence, we can decrease the value of $\mathbf{c}'^T \cdot \mathbf{x}'$ by choosing a co-basis column $i \in \{1, 3\}$ and a basis column $j \in \{4, 5\}$, and rearranging the equations so that $x_j = 0$ and $x_i \neq 0$, i.e. swapping i and j.

The simplex algorithm performs this swap, and then rewrites the linear program to the form of (2). This way, the solution can be iteratively improved. For instance, swapping variables x_1 and x_5 in (2) gives:

$$
\begin{aligned}
\text{given } 0 \cdot x_5 + \quad 2 \cdot x_2 + \quad 1 \cdot x_3 + 1 \cdot x_4 + \quad 1 \cdot x_1 &= 2 \\
1 \cdot x_5 + -1 \cdot x_2 + \quad 2 \cdot x_3 + 0 \cdot x_4 + \quad 1 \cdot x_1 &= 1, \\
\text{minimize } 0 \cdot x_5 + \quad 3 \cdot x_2 + -4 \cdot x_3 + 0 \cdot x_4 + -10 \cdot x_1 &\ .
\end{aligned}
$$

To rewrite this problem to form (2) with the right half of the system consisting of an identity matrix and a zero vector, we subtract the middle equation from the top equation once, and add ten times the middle equation to the bottom equation. This gives:

$$
\begin{aligned}
\text{given } -1 \cdot x_5 + \quad 3 \cdot x_2 + -1 \cdot x_3 + 1 \cdot x_4 + 0 \cdot x_1 &= 1 \\
1 \cdot x_5 + -1 \cdot x_2 + \quad 2 \cdot x_3 + 0 \cdot x_4 + 1 \cdot x_1 &= 1, \\
\text{minimize } 10 \cdot x_5 + -7 \cdot x_2 + 16 \cdot x_3 + 0 \cdot x_4 + 0 \cdot x_1 + 10 &\ .
\end{aligned}
\tag{3}
$$

This system represents the solution $(x_1, x_2, x_3, x_4, x_5) = (1, 0, 0, 1, 0)$ with $\mathbf{c}'^T \cdot \mathbf{x}' = -10$.

In this example, the value of $\mathbf{c}' \cdot \mathbf{x}'$ is further reduced by swapping variables x_2 and x_4, giving the following system:

$$
\begin{aligned}
\text{given } \tfrac{-1}{3} \cdot x_5 + \tfrac{1}{3} \cdot x_4 + \tfrac{-1}{3} \cdot x_3 + 1 \cdot x_2 + 0 \cdot x_1 &= \tfrac{1}{3} \\
\tfrac{2}{3} \cdot x_5 + \tfrac{1}{3} \cdot x_4 + \tfrac{5}{3} \cdot x_3 + 0 \cdot x_2 + 1 \cdot x_1 &= \tfrac{4}{3}, \\
\text{minimize } \tfrac{23}{3} \cdot x_5 + \tfrac{7}{3} \cdot x_4 + \tfrac{41}{3} \cdot x_3 + 0 \cdot x_2 + 0 \cdot x_1 + \tfrac{37}{3} &\ .
\end{aligned}
\tag{4}
$$

This system represents the solution $(x_1, x_2, x_3, x_4, x_5) = (\tfrac{4}{3}, \tfrac{1}{3}, 0, 0, 0)$ with $\mathbf{c}'^T \cdot \mathbf{x}' = -\tfrac{37}{3}$. The bottom equation only has non-negative coefficients, i.e. the value of $\mathbf{c}'^T \cdot \mathbf{x}'$ depends positively on each variable. This means that the solution is globally optimal.

The simplex algorithm thus repeatedly swaps basis and co-basis variables until doing so no longer decreases the value of $\mathbf{c}'^T \cdot \mathbf{x}'$, at which point it has found the solution to the linear program. However, we have not yet discussed *which* variables from the basis and co-basis the algorithm should choose. From the co-basis, it can choose any variable on which $\mathbf{c}'^T \cdot \mathbf{x}'$ depends negatively, i.e. with a coefficient smaller than zero in the bottom row. We pick the variable with the smallest coefficient.[8] From the basis, it needs to choose a variable so that the resulting system would satisfy $\mathbf{b} \geq 0$ (and hence, the represented solution satisfy $x_i \geq 0$). This can be guaranteed by selecting the variable corresponding to the value $\mathbf{b}_k$ for which $\mathbf{b}_k/\mathbf{s}_k$ is minimal but positive, where $\mathbf{s}$ is the vector of coefficients corresponding to the co-basis variable to be swapped. For instance, in linear program (2), $\mathbf{b}_2/\mathbf{s}_2 = 1/1 < \mathbf{b}_1/\mathbf{s}_1$. Hence, we choose x_5, the variable corresponding to $\mathbf{b}_2$ in the solution represented by (2). In (3), we have fractions

---

[8]This choice is known as *Dantzig's pivoting rule*. This rule is relatively easy to compute, but can cause the algorithm to cycle forever between different basis/co-basis sets.

$\mathbf{b}_1/\mathbf{s}_1 = 1/3$ and $\mathbf{b}_2/\mathbf{s}_2 = -1$; the second one is not positive, so we choose variable x_4 corresponding to $\mathbf{b}_1$.[9]

4.2. Simplex Implementation with Small Tableau

The above algorithm can be implemented by representing systems like (2)–(4) as *small tableaux*.[10] A tableau holds the order of the basis and co-basis variables, and the coefficients for the co-basis variables as a $(m+1) \times (n+1)$ matrix $\mathbf{T}$. Note that it is not necessary to store the coefficients for basis variables because they are always the same. The coefficients may be fractions. We represent these fractions as integral values over one common denominator q. This way, it is possible to perform the simplex algorithm using integer calculations only, which makes an implementation using multiparty computation easier. For instance, systems (2)–(4) above can be represented by the following integer tableaux (basis variables are shown to the right of $\mathbf{T}$, co-basis variables are shown on the bottom):

$$
\begin{array}{c}
\left(\begin{array}{ccc|c}
1 & 2 & 1 & 2 \\
1 & -1 & 2 & 1 \\
-10 & 3 & -4 & 0
\end{array}\right)
\begin{array}{c} x_4 \\ x_5 \\ \\ \end{array}
\\
\begin{array}{ccc|c} x_1 & x_2 & x_3 & q=1 \end{array}
\end{array}
\rightarrow
\begin{array}{c}
\left(\begin{array}{ccc|c}
-1 & 3 & -1 & 1 \\
1 & -1 & 2 & 1 \\
10 & -7 & 16 & 10
\end{array}\right)
\begin{array}{c} x_4 \\ x_1 \\ \\ \end{array}
\\
\begin{array}{ccc|c} x_5 & x_2 & x_3 & q=1 \end{array}
\end{array}
$$

$$
\rightarrow
\begin{array}{c}
\left(\begin{array}{ccc|c}
-1 & 1 & -1 & 1 \\
2 & 1 & 5 & 4 \\
23 & 7 & 41 & 37
\end{array}\right)
\begin{array}{c} x_2 \\ x_1 \\ \\ \end{array}
\\
\begin{array}{ccc|c} x_5 & x_4 & x_3 & q=3 \end{array}
\end{array}
. \tag{5}
$$

The whole simplex algorithm can be described as a series of systematic transformations of these integer tableaux. The first step of the simplex algorithm is to decide which variables to swap. As mentioned, our implementation picks the co-basis variable with the smallest coefficient. In terms of the integer tableau, we find the index $1 \le l \le n$ so that $\mathbf{T}_{m+1,l}$ is minimal. If $\mathbf{T}_{m+1,l} \ge 0$, then the current tableau represents the optimal solution. Moreover, we choose the k-th basis variable so that $\mathbf{T}_{k,n+1}/\mathbf{T}_{k,l}$ is minimal but positive. As shown in [1], we can equivalently choose the k-th basis variable so that $\frac{\mathbf{T}_{k,n+1}+\delta_{\le 0}(\mathbf{T}_{k,l})}{\mathbf{T}_{k,l}}$ is minimal (where $\delta_{\le 0}(x)$ is 1 if $x \le 0$, and 0 otherwise). Rewriting the system of equations by swapping basis variable k and co-basis variable l translates to updating the tableau $\mathbf{T}'$ and denominator q' as follows:

---

[9]It may happen that all $\mathbf{b}_i/\mathbf{s}_i$ are non-positive. In this case, the value of the co-basis variable represented by column $\mathbf{s}$ can be increased indefinitely under the given constraints, making the value of $\mathbf{c}' \cdot \mathbf{x}'$ decrease indefinitely. Hence, there is no solution that would minimize $\mathbf{c}' \cdot \mathbf{x}'$. In this case, the linear program is called *unbounded*.

[10]Other representations are also possible, with efficiency depending on the application, cf. [1].

$$
\mathbf{T'}_{i,j} = \begin{cases} (\mathbf{T}_{i,j}\mathbf{T}_{k,l} - \mathbf{T}_{i,l}\mathbf{T}_{k,j})/q & \text{if } i \neq k, j \neq l \\ \mathbf{T}_{i,j} & \text{if } i = k, j \neq l \\ -\mathbf{T}_{i,j} & \text{if } i \neq k, j = l \\ q & \text{if } i = k, j = l \end{cases}, \tag{6}
$$

$$q' = \mathbf{T}_{k,l} \ .$$

In [1], it is shown that the division by q in the $i \neq k, j \neq l$ case is proper. Finally, if the optimal solution has been found, it can be obtained from the tableau by assigning the values $\mathbf{T}_{i,n+1}$ to the respective basis variables, and dividing by the denominator q.

4.3. Dual Solution and Proof of Optimality

To prove that a solution to a linear program is optimal, [1] suggests using the fact that the simplex algorithm not only solves the linear program, but also its so-called *dual linear program*. Given a linear program *minimize* $\mathbf{c}^T \cdot \mathbf{x}$ *so that* $\mathbf{A} \cdot \mathbf{x} \leq \mathbf{b}, \mathbf{x} \geq 0$, its dual program is defined as *maximize* $\mathbf{b}^T \cdot \mathbf{p}$ *so that* $\mathbf{A}^T \cdot \mathbf{p} \leq \mathbf{c}, \mathbf{p} \leq 0$. It appears that a linear program has an optimal solution $\mathbf{x}$ if and only if its dual has an optimal solution $\mathbf{p}$, and that, in this case, $\mathbf{p}^T \cdot \mathbf{b} = \mathbf{c}^T \cdot \mathbf{x}$. On the other hand, if $\mathbf{p}^T \cdot \mathbf{b} = \mathbf{c}^T \cdot \mathbf{x}$ for some $\mathbf{x}$ and $\mathbf{p}$ satisfying the constraints $\mathbf{A} \cdot \mathbf{x} \leq \mathbf{b}, \mathbf{x} \geq 0, \mathbf{A}^T\mathbf{p} \leq \mathbf{c}$, and $\mathbf{p} \leq 0$, then both $\mathbf{x}$ and $\mathbf{p}$ must be optimal. Indeed, any other $\mathbf{x}'$ satisfies $\mathbf{c}^T \cdot \mathbf{x}' \geq (\mathbf{A}^T \cdot \mathbf{p}) \cdot \mathbf{x}' = \mathbf{p}^T \cdot (\mathbf{A} \cdot \mathbf{x}') \geq \mathbf{p}^T \cdot \mathbf{b} = \mathbf{c}^T \cdot \mathbf{x}$, and similarly for any other $\mathbf{p}'$. Hence, one can prove the optimality of $\mathbf{x}$ by finding the solution $\mathbf{p}$ of the dual linear program, and proving that

$$\mathbf{p}^T \cdot \mathbf{b} = \mathbf{c}^T \cdot \mathbf{x}, \quad \mathbf{A} \cdot \mathbf{x} \leq \mathbf{b}, \quad \mathbf{x} \geq 0, \quad \mathbf{A}^T \cdot \mathbf{p} \leq \mathbf{c}, \quad \mathbf{p} \leq 0.$$

Equivalently, if $\mathbf{x}$ and $\mathbf{p}$ are integers and q is the common denominator, one proves that

$$q > 0, \quad \mathbf{p}^T \cdot \mathbf{b} = \mathbf{c}^T \cdot \mathbf{x}, \quad \mathbf{A} \cdot \mathbf{x} \leq q \cdot \mathbf{b}, \quad \mathbf{x} \geq 0, \quad \mathbf{A}^T \cdot \mathbf{p} \leq q \cdot \mathbf{c}, \quad \mathbf{p} \leq 0. \tag{7}$$

Apart from solving the original linear program, the simplex algorithm also solves its dual. Consider the transpose of a tableau $\mathbf{T}$ for the original linear problem. The transpose of $\mathbf{T}$ is a tableau for the dual problem when minus signs are put in front of the variables in the basis. For instance, the transpose of the first tableau from (5) corresponds to

$$
\begin{array}{c}
\left(\begin{array}{cc|c} 1 & 1 & -10 \\ 2 & -1 & 3 \\ 1 & 2 & -4 \\ \hline 2 & 1 & 0 \end{array} \right) \begin{array}{c} -p_3 \\ -p_4 \\ -p_5 \\ \\ \end{array} \\
\begin{array}{cc} p_1 & p_2 \end{array} \quad q = 1 \ .
\end{array}
\tag{8}
$$

This tableau represents solution $(p_1, p_2, p_3, p_4, p_5) = (0, 0, 10, -3, 4)$ to the dual linear problem (that, however, does not satisfy $\mathbf{A}^T \cdot \mathbf{p} \leq q \cdot \mathbf{c}$). Now, consider how the update formula (6) acts on this transposed tableau. Recall that in the original linear program, this formula swaps the k-th basis variable and the l-th co-basis variable. Because of the minus signs in the $i = k, j \neq l$ and $i \neq k, j = l$ cases, this formula can be interpreted as swapping the l-th basis variable with the k-th co-basis variable of the dual problem, *and*

inverting their signs. This is the reason for the minus signs in the basis. For instance, the two updates (5) turn dual tableau (8) into:

$$
\begin{pmatrix}
-1 & 1 & 10 \\
3 & -1 & -7 \\
-1 & 2 & 16 \\
\hline
1 & 1 & 10
\end{pmatrix}
\begin{matrix}
-p_2 \\
-p_4 \\
-p_5 \\
\\
\end{matrix}
\quad \rightarrow \quad
\begin{pmatrix}
-1 & 2 & 23 \\
1 & 1 & 7 \\
-1 & 5 & 41 \\
\hline
1 & 4 & 37
\end{pmatrix}
\begin{matrix}
-p_2 \\
-p_1 \\
-p_5 \\
\\
\end{matrix}
\tag{9}
$$

$$
\begin{matrix}
p_1 & p_3 & \quad q = 1
\end{matrix}
\qquad\qquad
\begin{matrix}
p_4 & p_3 & \quad q = 3 \ ,
\end{matrix}
$$

representing solutions $(p_1, p_2, p_3, p_4, p_5) = (0, -10, 0, 7, -16)$ and $(\frac{-7}{3}, \frac{-23}{3}, 0, 0, \frac{-41}{3})$.

Recall that in terms of the original linear program, the simplex algorithm iterates through solutions that satisfy the constraints, meaning that the rightmost column of the tableau contains only non-negative entries, until it finds that an optimal solution, meaning that the bottommost row of the tableau contains only non-negative entries. In contrast, in terms of the dual linear program, the simplex algorithm iterates through solutions that are optimal, in the sense that any swap decreases the value of $\mathbf{p} \cdot \mathbf{b}$ (in the example, from 0 to -10 to $-\frac{37}{3}$), until it finds one that satisfies the constraints, meaning that the rightmost column contains only non-negative entries, so $-p_i \geq 0$ for all i. In both cases, the final tableau has non-negative entries in both the bottommost row and the rightmost column, meaning that the solution found is optimal. In the end, the solution $\mathbf{p}$ to the dual linear program can be read off the transpose tableau in the usual way. Translated into the original tableau $\mathbf{T}$, this means assigning values $\mathbf{T}_{m+1,j}$ to the respective co-basis variables. For instance, in the final tableau of (5), p_1 corresponds to $-x_4$ and p_2 corresponds to $-x_5$, so the solution is $(p_1, p_2) = (\frac{-7}{3}, \frac{-23}{3})$.

5. Secure Linear Programming and Verifiability by Certificate Validation

We will now show how to obtain universally verifiable linear programming using our verifiability by certificate validation approach. We will show how to securely compute a solution $([r], [a]) \leftarrow f([x])$ to a linear program, where f is the simplex algorithm, and how to validate the solution using predicate $\phi(r, a, x)$, where ϕ is a number of polynomial equations representing the optimality criterion (7). By linking these components, we get verifiability by certificate validation using the techniques from Sec. 2, combined with ElGamal-based validation from Sec. 3. Solving linear programs using SMC implementations of the simplex algorithm is established in the literature (see, e.g. [18,19,1]). We will briefly discuss our implementation and present some optimizations. In [1], it was shown how to validate solutions in general, but we will show how to phrase this validation as a number of polynomial equations.

5.1. Small Tableau Simplex Algorithm

Alg. 4 shows the small tableau simplex algorithm, formulated in terms of primitive operations that can be implemented with SMC. Normal variables denote public values (e.g. m and n) and variables between brackets denote private values (e.g. $[\![q]\!]$). Boldface variables denote vectors or arrays.[11]

---

[11]Note that a single value is sometimes implemented internally using multiple secret shares, e.g. $[\![l]\!]$ represents a secret array index that is implemented as an array of zeros and ones to facilitate indexing.

Algorithm 4 Small tableau simplex algorithm for computing the optimal solution and the dual

---

Require: $[\![\mathbf{T}]\!]$ is the tableau of a linear problem of the size n by m that has an optimal solution, and of which the initial solution satisfies $[\![\mathbf{T}]\!]_{m+1,i} \geq 0$ for all i

Ensure: $[\![\mathbf{X}]\!], [\![q]\!]$ optimal solution to the linear program, $[\![\mathbf{P}]\!], [\![q]\!]$ optimal solution to the dual

1: **function** SOLVELP$(n, m, [\![\mathbf{T}]\!])$
2:      $[\![\textbf{Co-basis}]\!] \leftarrow [1, \ldots, n]; [\![\textbf{Basis}]\!] \leftarrow [n+1, \ldots, n+m]$
3:      $[\![q]\!] \leftarrow 1$
4:      $([\![\textit{coeff}]\!], [\![l]\!]) \leftarrow$ FINDMIN$([\![\mathbf{T}]\!]_{m+1,1}, \ldots, [\![\mathbf{T}]\!]_{m+1,n})$
5:      **while** $[\![\textit{coeff}]\!] < 0$ **do**
6:          $([\![\textbf{Col}]\!]_1, \ldots, [\![\textbf{Col}]\!]_{m+1}) \leftarrow ([\![\mathbf{T}]\!]_{1,[\![l]\!]}, \ldots, [\![\mathbf{T}]\!]_{m+1,[\![l]\!]})$ ▷ cobasis column
7:          $(\frac{*}{[\![q']\!]}, [\![k]\!]) \leftarrow$ FINDMIN$\left(\frac{[\![\mathbf{T}]\!]_{1,n+1}+\delta_{\leq 0}([\![\textbf{Col}]\!]_1)}{[\![\textbf{Col}]\!]_1}, \ldots, \frac{[\![\mathbf{T}]\!]_{m,n+1}+\delta_{\leq 0}([\![\textbf{Col}]\!]_m)}{[\![\textbf{Col}]\!]_m}\right)$
8:          $([\![\textbf{Col}]\!]_1, \ldots, [\![\textbf{Col}]\!]_{m+1}) \leftarrow ([\![\textbf{Col}]\!]_1 [\![q]\!]^{-1}, \ldots, [\![\textbf{Col}]\!]_{m+1} [\![q]\!]^{-1})$
9:          $[\![\textbf{Col}]\!]_{[\![k]\!]} \leftarrow [\![\textbf{Col}]\!]_{[\![k]\!]} - 1$
10:         $([\![\textbf{Row}]\!]_1, \ldots, [\![\textbf{Row}]\!]_{n+1}) \leftarrow ([\![\mathbf{T}]\!]_{[\![k]\!],1}, \ldots, [\![\mathbf{T}]\!]_{[\![k]\!],n+1})$ ▷ basis row
11:         $[\![\textbf{Row}]\!]_{[\![l]\!]} \leftarrow [\![\textbf{Row}]\!]_{[\![l]\!]} + [\![q]\!]$
12:         **for** $i \leftarrow 1, .., m+1; j \leftarrow 1, .., n+1$ **do** $[\![\mathbf{T}]\!]_{i,j} \leftarrow \frac{[\![q']\!]}{[\![q]\!]}[\![\mathbf{T}]\!]_{i,j} - [\![\textbf{Col}]\!]_i \cdot [\![\textbf{Row}]\!]_j$
13:         $[\![q]\!] \leftarrow [\![q']\!]$
14:         SWAP$([\![\textbf{Basis}]\!]_{[\![k]\!]}, [\![\textbf{Co-basis}]\!]_{[\![l]\!]})$
15:         $([\![\textit{coeff}]\!], [\![l]\!]) \leftarrow$ FINDMIN$([\![\mathbf{T}]\!]_{m+1,1}, \ldots, [\![\mathbf{T}]\!]_{m+1,n})$
16:      **for** $i \leftarrow 1, \ldots, m$ **do if** $[\![\textbf{Basis}]\!]_i \leq n$ **then** $[\![\mathbf{X}]\!]_{[\![\textbf{Basis}]\!]_i} \leftarrow [\![\mathbf{T}]\!]_{i,n+1}$
17:      **for** $j \leftarrow 1, \ldots, n$ **do if** $n < [\![\textbf{Co-basis}]\!]_j$ **then** $[\![\mathbf{P}]\!]_{[\![\textbf{Co-basis}]\!]_j - n} \leftarrow -[\![\mathbf{T}]\!]_{m+1,j}$
18:      **return** $([\![\mathbf{X}]\!], [\![\mathbf{P}]\!], [\![q]\!])$

The algorithm takes as input an integer tableau $[\![\mathbf{T}]\!]$ with n variables and m equations. It starts by initializing the tableau: the common denominator $[\![q]\!]$ is set to 1, the co-basis is set to numbers 1 to n (representing the first n variables), and the basis is set to numbers $n+1$ to $n+m$ (line 2).

Then, the algorithm repeatedly performs simplex iterations. On line 4, it tries to find a column $[\![l]\!]$ corresponding to a co-basis vector to swap. For this, it uses a FINDMIN function (which we assume to be given) that, given a list of numbers, returns the minimal value in that list and the index at which it occurs. It checks if the resulting minimal value $[\![\textit{coeff}]\!]$ is smaller than zero, If not, then the simplex algorithm is completed. Note that while the minimal value $[\![\textit{coeff}]\!]$ remains secret, the result of the comparison $[\![\textit{coeff}]\!] < 0$ is opened. Hence, this algorithm leaks the number of iterations.[12]

When column index $[\![l]\!]$ has been determined, the algorithm retrieves the co-basis column $[\![\textbf{Col}]\!]$ (line 6) and computes the row index $[\![k]\!]$ corresponding to the basis variable with witch to swap (line 7). As suggested in the previous section, it does this by computing the minimum of the values $\frac{[\![\mathbf{T}]\!]_{i,n+1}+\delta_{\leq 0}([\![\textbf{Col}]\!]_i)}{[\![\textbf{Col}]\!]_i}$. In this case, the FINDMIN algorithm returns the numerator and denominator of the minimal value. We store the denominator as $[\![q']\!]$. Finally, the algorithm performs the swap between the co-basis column $[\![l]\!]$ and the basis row $[\![k]\!]$. For this, it evaluates the tableau update formula (6). In [1] it is proven

---

[12] This is necessary in order to obtain an efficient solution. Otherwise, we would have to run the maximum possible number of iterations, which is exponential in the size of the linear program (cf. [19,1]).

that the computations shown on lines 8–13 achieve this. Finally, the basis and co-basis variables are swapped (line 14), and the iteration procedure is repeated.

After the simplex iterations have been performed, the solution to the original linear program is computed by taking the entries from the last column of the tableau according to the indices in $[\![\mathbf{Basis}]\!]$ (line 16). Similarly, the solution to the dual linear program is given by the entries from the last row according to the indices in $[\![\mathbf{Co\text{-}basis}]\!]$ (line 17). Note that this algorithm does not check for unboundedness of the linear program. Thus, if the linear program does not have an optimal solution, the algorithm will not terminate.

5.2. Implementing Small Tableau Simplex with VIFF

Our implementation of small tableau simplex is in the VIFF[13] framework for secure computation based on Shamir's secret sharing. We will now briefly discuss how some noteworthy primitive operations from Alg. 4 are implemented.

The FINDMIN algorithm, taken from [1], returns the indices $[\![k]\!]$, $[\![l]\!]$ in unary form, i.e. as a vector with zeros except for a one at the index of the minimum. This means that indexing with $[\![k]\!]$ and $[\![l]\!]$ can be performed with an inner product, e.g. $[\![\mathbf{T}]\!]_{1,[\![l]\!]}$ is computed as the inner product of the first row of $[\![\mathbf{T}]\!]$ with the vector of indices of $[\![l]\!]$. Comparison is due to [20].

The final step of the simplex algorithm is to index $[\![\mathbf{T}]\!]$ using the indices from the basis and co-basis lists $[\![\mathbf{Basis}]\!]$, $[\![\mathbf{Co\text{-}basis}]\!]$ (lines 16–17). These indices are initialized (line 2), then, in every simplex iteration, one index from the basis is swapped with one index from the co-basis (line 14). One way to implement this is by representing each index $[\![\mathbf{Basis}]\!]_i$, $[\![\mathbf{Co\text{-}basis}]\!]_j$ in unary form like $[\![k]\!]$, $[\![l]\!]$ above, but this makes the swapping operation expensive. An alternative is to represent each index as a number, and then convert it it into a unary vector for indexing at the end of the algorithm. This makes the swapping on line 14 cheap, but the conversion on lines 16–17 expensive.

We propose an alternative way of representing the indices that allows for both efficient swapping and efficient indexing. Namely, we represent index $1 \leq j \leq n + m$ as $[\![\omega^{-j}]\!]$, where ω is an N-th root of unity for some $N \geq n + m$. Now, note that $\left\langle (\omega^{-j}, \ldots, (\omega^{-j})^N), (\omega^i, \ldots, (\omega^i)^N) \right\rangle = \delta_{i,j} N$. Hence, given array $[\![\mathbf{y}]\!]$ of the length n and value $[\![\omega^{-j}]\!]$, we can obtain $[\![\mathbf{y}]\!]_j$ as

$$\sum_{i=1}^{n} \frac{1}{N} \left\langle ([\![\omega^{-j}]\!], \ldots, ([\![\omega^{-j}]\!])^N), (\omega^i [\![\mathbf{y}]\!]_i, \ldots, (\omega^i)^N [\![\mathbf{y}]\!]_i) \right\rangle = \sum_{i=1}^{n} \delta_{i,j} [\![\mathbf{y}]\!]_i = [\![\mathbf{y}]\!]_j \ .$$

Hence, we can store an index j as a single value ω^{-j}, so swapping is cheap. We can index at j using N multiplications to determine the powers of $[\![\omega^{-j}]\!]$, and n inner products of these powers with vectors $(\omega^i [\![\mathbf{y}]\!]_i, \ldots, (\omega^i)^N [\![\mathbf{y}]\!]_i)$ that can be computed locally.

In our linear programming application, the root-of-unity indexing method turns out to give better performance than indexing using numbers or unary vectors. However, note that this method requires an N-th root of unity, where N should be as small as possible but at least $n + m$. In fact, if computation is over $\mathbb{F}_p$, then such a root exists if and only if $N | (p - 1)$. In the case of the NIST P-224 curve we used for our ElGamal encryptions, $p - 1$ has many small factors (e.g. $3^4 | p - 1$), so a favourable N is easy to find.

---

[13] The VIFF version used is available at http://www.win.tue.nl/~berry/TUeVIFF/.

5.3. A Polynomial Predicate for Optimality

The small tableau simplex algorithm above gives us optimal solutions $\mathbf{x}$ and $\mathbf{p}$ to the original and dual linear programs, with the common denominator q. As shown in Sec. 5, proving that $\mathbf{x}$ is optimal with respect to inputs $\mathbf{A}$, $\mathbf{b}$, $\mathbf{c}$ means showing that

$$q > 0, \quad \mathbf{p}^T \cdot \mathbf{b} = \mathbf{c}^T \cdot \mathbf{x}, \quad \mathbf{A} \cdot \mathbf{x} \leq q \cdot \mathbf{b}, \quad \mathbf{x} \geq 0, \quad \mathbf{A}^T \cdot \mathbf{p} \leq q \cdot \mathbf{c}, \quad \mathbf{p} \leq 0 \ .$$

To obtain efficient proofs in terms of ElGamal encryptions (see Sec. 3), we need to write this validation as a set of polynomial equalities.

Clearly, $\mathbf{p}^T \cdot \mathbf{b} = \mathbf{c}^T \cdot \mathbf{x}$ is already in the correct form and hence, we are left with checking inequalities between different values. To do this, we write all inequalities in the form $y \geq 0$, compute bit decompositions $y = \sum_{i=0}^{k} y_i 2^i$ for a sufficiently large k in VIFF, and include these in the certificate passed on to UVCDN. Bit decompositions can be computed efficiently with a protocol due to [21]. Then, $y \geq 0$ is equivalent to

$$y = \sum_{i=0}^{k} y_i 2^i \ \wedge \ y_0 \cdot (1 - y_0) = 0 \ \wedge \ \ldots \wedge y_k \cdot (1 - y_k) = 0 \ .$$

Rewriting the optimality criterion for linear programs using the above equation for comparisons, we get a validation predicate that is suitable for use with ElGamal.

5.4. Implementation of Universally Verifiable Linear Programming

To study the practical feasibility of our approach to universally verifiable linear programming, we have made a prototype implementation of the described protocols. We used the simplex implementation from the TUeVIFF distribution of VIFF as a starting point, adding code to produce the certificate of correctness, i.e. the dual solution and the required bit decompositions. We implemented ElGamal-based UVCDNPROVE using SCAPI [22]. SCAPI is a high-level cryptographic library that supports homomorphic encryption (Paillier and ElGamal), several Σ-protocols, and the Fiat-Shamir heuristic. We implemented Σ-protocols and homomorphisms for ElGamal [6], a two-party multiparty Fiat-Shamir heuristic [5], and threshold decryption.

Fig. 3 shows performance results for the different components of the system. We measured performance by running all components on one single desktop PC[14]. For the ElGamal encryptions, we used the NIST P-224 [23] elliptic curve, which is supported in SCAPI through the MIRACL library. We ran the implementation on LPs of several sizes: two randomly generated small LPs and several larger LPs based on Netlib test programs[15]. We show performance numbers for the VIFF computation, and the ElGamal-based proof production and verification. For VIFF and the ElGamal prover, the computation times are per party participating in the computation. The VIFF computation was performed with three parties and the ElGamal computation with two: the minimum needed for privacy against one corrupted computation party. We took the minimal bit lengths needed for correctness. In practice, these are not known in advance: for VIFF, one takes a safe margin, but for the proof, one can reveal the maximal bit length of all bit decompositions in the certificate.

---

[14]An Intel Core i7-3770 CPU @ 3.40GHz x8.

[15]See http://www.netlib.org/lp/data/, coefficients were rounded for performance.
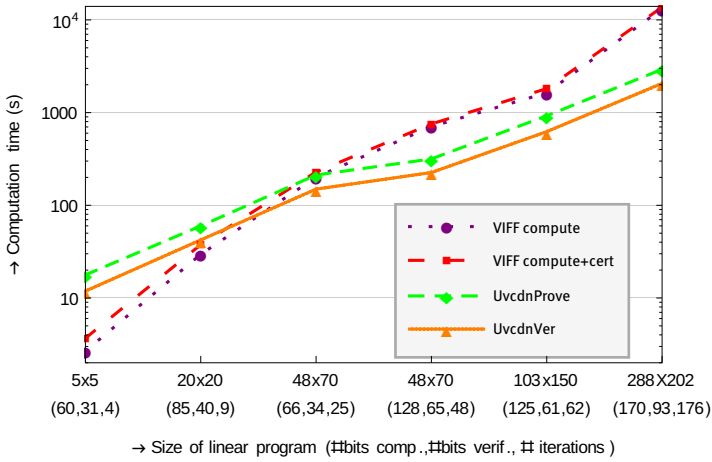
Figure 3. Performance figures for Universally Verifiable Linear Programming: absolute computation time per party for several linear programs [6]

For VIFF, we performed experiments both using a small prime big enough for the values in the respective linear programs, and using the prime order of the NIST P-224 discrete logarithm group. As discussed, in the former case, the computation results need to be converted into the bigger modulus using the [14] conversion protocol. We found that in all but the smallest example, using the bigger NIST P-224 prime did not have a noticeable effect on performance. Hence, when using VIFF together with ElGamal-based verification, computing directly with the prime used for verifiability is generally the best option. On the other hand, when using Paillier, conversion is needed: the plaintext space of Paillier is $\mathbb{Z}_N$ with N a RSA modulus, and VIFF cannot compute over such a ring.

Our experiments show that as the size of the linear program increases, producing and verifying proofs becomes relatively more efficient. Indeed, both the computation of the solution and the verification of its correctness scale in the size of the LP, but computation additionally scales in the number of iterations needed to reach the optimal solution. This number of iterations typically grows with the LP size. For the smaller linear programs, verification of the solution is slower than its computation in the the three-party case. Hence, in settings with three or fewer input/result parties, outsourcing does not save computational effort compared to a local multiparty computation, but it does additionally offer correctness. As the computational effort per party scales linearly in the number of input/result parties, outsourcing saves computational effort if there are more parties, even for small LPs. For larger LPs, verifying is already faster for three parties, so outsourcing always saves computation effort in addition to guaranteeing correctness.

6. Discussion

In this chapter, we have shown that outsourcing a multiparty computation using verifiability by certificate validation is feasible. Instead of performing a multiparty computation among themselves, parties can decide to outsource their computation to several untrusted cloud parties. They receive unconditional guarantees that the result of the computation is correct, and that each party honestly supplied inputs without seeing those of others. Privacy is guaranteed against passive corruption if not too many computation parties col-

lude. In the case of linear programming, for smaller problems, verification of the solution is slower than computation with a passively secure protocol, but it gives the advantage of guaranteeing correctness. With more parties or larger problems, verification quickly becomes faster than computation.

The best previous solution combining unconditional correctness and privacy is $(n-1)$-out-of-n actively secure multiparty computation protocols. The best such protocols (see Sec. 7 of Chapter 1) have an online phase with performance comparable to that of passively secure protocols, but combine this with (expensive) preprocessing. Hence, compared to their online phase, verification by certification offers the same performance trade-offs as in the passive case. When we also take into account the expensive preprocessing phase, experiments from [6] suggest that validating a linear programming solution using our techniques is about two orders of magnitude faster than computing the optimal solution with active security. On the other hand, our approach offers privacy against one passive attacker, while actively secure protocols guarantee this unconditionally.

We only protect against a semi-honest attacker. In our model, an active attacker learns all honest parties' inputs. In fact, this is probably an overestimation of the amount of information that an active attacker can actually obtain.

There are several ways in which certificate validation can be made more efficient. First, we note that our present implementation is not optimized. We think that a more careful and efficient implementation could easily be twice as fast. With some changes, the current construction can be made to work with Pedersen commitments instead of ElGamal encryptions which would probably decrease proof size and verification time.

However, especially in the case of linear programming, the biggest performance saving would come from using alternative techniques for proving that a certain value is positive. Namely, our current solution is based on bit decompositions, so it scales both in the size of the linear program and in the maximal bit length of the solution (which in turn grows with the size of the linear program). Alternative proof techniques, such as the range proofs of Boudot [24], depend much less on the size of the values at hand. The work of Keller et al. [25] suggests ways of distributing these proofs that could be adapted to our setting. Alternatively, recent techniques for verifiable computation [26] provide proofs whose size and verification time are very small and independent from the size of the verification circuit, which would be very attractive to verifiers. However, it remains to be seen how hard it would be for computation parties to produce such proofs in a privacy-friendly way.

References

[1] Sebastiaan de Hoogh. *Design of large scale applications of secure multiparty computation: secure linear programming*. PhD thesis, Eindhoven University of Technology, 2012.

[2] J. Cohen and M. Fischer. A Robust and Verifiable Cryptographically Secure Election Scheme. In *Proceedings of FOCS '85*, pages 372–382. IEEE, 1985.

[3]   K. Sako and J. Kilian. Receipt-Free Mix-Type Voting Scheme—A Practical Solution to the Implementation of a Voting Booth. In *Proceedings of EUROCRYPT '95*, volume 921 of *Lecture Notes in Computer Science*, pages 393–403. Springer, 1995.

[4]   Carsten Baum, Ivan Damgård, and Claudio Orlandi. Publicly Auditable Secure Multi-Party Computation. In Michel Abdalla and Roberto De Prisco, editors, *Security and Cryptography for Networks - 9th International Conference, SCN 2014, Amalfi, Italy, September 3-5, 2014. Proceedings*, volume 8642 of *Lecture Notes in Computer Science*, pages 175–196. Springer, 2014.

[5]   Berry Schoenmakers and Meilof Veeningen. Universally Verifiable Multiparty Computation from Threshold Homomorphic Cryptosystems. Accepted at ACNS, 2015.

[6]   Berry Schoenmakers and Meilof Veeningen. Guaranteeing Correctness in Privacy-Friendly Outsourcing by Certificate Validation. Submitted, 2015.

[7]   Ronald Cramer, Ivan Damgård, and Jesper B. Nielsen. Multiparty Computation from Threshold Homomorphic Encryption. In Birgit Pfitzmann, editor, *Advances in Cryptology – EUROCRYPT 2001*, volume 2045 of *Lecture Notes in Computer Science*, pages 280–300. Springer Berlin Heidelberg, 2001.

[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, 4th International Workshop on Practice and Theory in Public Key Cryptography, PKC 2001, Cheju Island, Korea, February 13-15, 2001, Proceedings*, volume 1992 of *Lecture Notes in Computer Science*, pages 119–136. Springer, 2001.

[9]   Mihir Bellare and Phillip Rogaway. Random Oracles are Practical: A Paradigm for Designing Efficient Protocols. In Dorothy E. Denning, Raymond Pyle, Ravi Ganesan, Ravi S. Sandhu, and Victoria Ashby, editors, *CCS '93, Proceedings of the 1st ACM Conference on Computer and Communications Security, Fairfax, Virginia, USA, November 3-5, 1993.*, pages 62–73. ACM, 1993.

[10] Hoeteck Wee. Zero Knowledge in the Random Oracle Model, Revisited. In Mitsuru Matsui, editor, *Advances in Cryptology - ASIACRYPT 2009, 15th International Conference on the Theory and Application of Cryptology and Information Security, Tokyo, Japan, December 6-10, 2009. Proceedings*, volume 5912 of *Lecture Notes in Computer Science*, pages 417–434. Springer, 2009.

[11] Amos Fiat and Adi Shamir. How to Prove Yourself: Practical Solutions to Identification and Signature Problems. In Andrew M. Odlyzko, editor, *CRYPTO*, volume 263 of *Lecture Notes in Computer Science*, pages 186–194. Springer, 1986.

[12] Michel Abdalla, Jee Hea An, Mihir Bellare, and Chanathip Namprempre. From Identification to Signatures Via the Fiat-Shamir Transform: Necessary and Sufficient Conditions for Security and Forward-Security. *IEEE Transactions on Information Theory*, 54(8):3631–3646, 2008.

[13] Martin Geisler. *Cryptographic Protocols: Theory and Implementation*. PhD thesis, Aarhus University, February 2010.

[14] Joy Algesheimer, Jan Camenisch, and Victor Shoup. Efficient Computation Modulo a Shared Secret with Application to the Generation of Shared Safe-Prime Products. In Moti Yung, editor, *CRYPTO*, volume 2442 of *Lecture Notes in Computer Science*, pages 417–432. Springer, 2002.

[15] Ran Canetti. Security and Composition of Multi-party Cryptographic Protocols. *Journal of Cryptology*, 13:2000, 1998.

[16] Jens Groth. Evaluating Security of Voting Schemes in the Universal Composability Framework. In Markus Jakobsson, Moti Yung, and Jianying Zhou, editors, *Applied Cryptography and Network Security, Second International Conference, ACNS 2004, Yellow Mountain, China, June 8-11, 2004, Proceedings*, volume 3089 of *Lecture Notes in Computer Science*, pages 46–60. Springer, 2004.

[17] Torben P. Pedersen. A Threshold Cryptosystem without a Trusted Party (Extended Abstract). In Donald W. Davies, editor, *EUROCRYPT*, volume 547 of *Lecture Notes in Computer Science*, pages 522–526. Springer, 1991.

[18] 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 Dingledine and Golle [27], pages 325–343.

[19] Tomas Toft. Solving Linear Programs Using Multiparty Computation. In Dingledine and Golle [27], pages 90–107.

[20] Tord Reistad and Tomas Toft. Linear, Constant-Rounds Bit-Decomposition. In Donghoon Lee and Seokhie Hong, editors, *Information, Security and Cryptology - ICISC 2009, 12th International Conference, Seoul, Korea, December 2-4, 2009, Revised Selected Papers*, volume 5984 of *Lecture Notes in Computer Science*, pages 245–257. Springer, 2009.

[21] Berry Schoenmakers and Pim Tuyls. Efficient Binary Conversion for Paillier Encrypted Values. In Serge

Vaudenay, editor, *EUROCRYPT*, volume 4004 of *Lecture Notes in Computer Science*, pages 522–537. Springer, 2006.

[22] Yael Ejgenberg, Moriya Farbstein, Meital Levy, and Yehuda Lindell. SCAPI: The Secure Computation Application Programming Interface. *IACR Cryptology ePrint Archive*, 2012:629, 2012.

[23] NIST. Recommended elliptic curves for federal government use, 1999. Available at http://csrc.nist.gov/encryption.

[24] Fabrice Boudot. Efficient Proofs that a Committed Number Lies in an Interval. In Bart Preneel, editor, *Advances in Cryptology - EUROCRYPT 2000, International Conference on the Theory and Application of Cryptographic Techniques, Bruges, Belgium, May 14-18, 2000, Proceeding*, volume 1807 of *Lecture Notes in Computer Science*, pages 431–444. Springer, 2000.

[25] Marcel Keller, Gert Læssøe Mikkelsen, and Andy Rupp. Efficient Threshold Zero-Knowledge with Applications to User-Centric Protocols. In *Proceedings of ICITS 2012*, volume 7412 of *Lecture Notes in Computer Science*, pages 147–166. Springer, 2012.

[26] Rosario Gennaro, Craig Gentry, Bryan Parno, and Mariana Raykova. Quadratic Span Programs and Succinct NIZKs without PCPs. *IACR Cryptology ePrint Archive*, 2012:215, 2012. informal publication.

[27] Roger Dingledine and Philippe Golle, editors. *Financial Cryptography and Data Security, 13th International Conference, FC 2009, Accra Beach, Barbados, February 23-26, 2009. Revised Selected Papers*, volume 5628 of *Lecture Notes in Computer Science*. Springer, 2009.

Chapter 11
Transformation-based Computation and Impossibility Results

Alisa PANKOVA [a] and Peeter LAUD [a]

[a] *Cybernetica AS, Estonia*

Abstract. In this chapter we study transformation-based approaches for outsourcing some particular tasks, based on publishing and solving a transformed version of the problem instance. First, we demonstrate a number of attacks against existing transformations for privacy-preserving linear programming. Our attacks show the deficiencies of existing security definitions; 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 be transformations that are information-theoretically or even computationally secure. Finally, we study the possibility of achieving IND-CPA security in privately outsourcing linear equation systems over real numbers, and see that it is not as easy as for linear equations over finite fields for which it is known that efficient information-theoretically secure transformations do exist.

Introduction

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 he would like to solve, but it 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 the server engage in a protocol that results in the client learning the solution to T, but that is computationally less demanding for the client than solving T himself. Such outsourcing protocols have been proposed for different tasks, e.g. sequence matching [1], database operations [2], cryptographic operations [3,4], and some linear algebra tasks [5] including linear programming [6,7]. 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.

In some cases, the transformation-based approach can be provably secure. A simple example is outsourcing the computation of matrix inverse over a finite field. Matrix multiplication is easier than matrix inverse. Given an $n \times n$ invertible matrix $\mathbf{A}$ over $\mathbb{F}$, the client generates a random invertible matrix $\mathbf{R} \xleftarrow{\$} \mathbb{F}^{n \times n}$. Sampling each entry uniformly from $\mathbb{F}$ gives a uniformly distributed matrix over $\mathbb{F}^{n \times n}$ which is invertible with the probability $1 - \varepsilon$ for a negligible ε. As the invertible $n \times n$ matrices over $\mathbb{F}$ form a multi-

plicative group $GL(n, \mathbb{F})$, the product $\mathbf{RA}$ is distributed uniformly in $GL(n, \mathbb{F})$ and hence does not depend on $\mathbf{A}$, so $\mathbf{RA}$ leaks no information about $\mathbf{A}$. The client sends $\mathbf{RA}$ to the server. The server computes $(\mathbf{RA})^{-1} = \mathbf{A}^{-1}\mathbf{R}^{-1}$, sends it back, and the client computes $\mathbf{A}^{-1}\mathbf{R}^{-1} \cdot \mathbf{R} = \mathbf{A}^{-1}$. However, the same security argument does not pass for matrices over $\mathbb{R}$ as we cannot define a uniform distribution on $GL(n, \mathbb{R})$.

Outsourcing a computation can be used to make *secure multiparty computation* (SMC) easier by delegating some hard computational subtask to one of the parties instead of devising a complex multiparty algorithm for solving this task. In order to do that, one must verify that using such an outsourcing does not break the security of the entire protocol. In general, the security properties of SMC protocols 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. Many transformation-based methods have so far lacked the understanding of their privacy properties at a comparable level. This chapter demonstrates that such unavailability of security definitions is dangerous. In particular, we study the case of affine transformations for outsourcing linear programming tasks and matrix inverse over real numbers. This work is based on our impossibility results from [8,9,10].

In Sec. 1, we define a linear programming task and transformation methods that have been used in previous works. We present several attacks against these methods in Sec. 2. In Sec. 3, we give a desired security definition of a transformation that would exclude the possibility of attacks of Sec. 2, and that would be standard enough to make the transformation composable with larger protocols. In Sec. 4, we show that achieving this security definition for linear programming tasks using an *efficient* transformation is impossible. Finally, in Sec. 5, we study the possibility of achieving this security definition for transformations of linear equation systems. We provide some possibility and impossibility results, concluding that finding a good transformation is not easy in general, but is nevertheless possible for some certain classes of matrices.

Throughout this chapter, the upper case letters $\mathbf{A}$ denote matrices, and the bold lower case letters $\mathbf{b}$ denote column vectors. Writing two matrices/vectors together without an operator $\mathbf{Ab}$ denotes multiplication, while separating them with a whitespace and enclosing with parentheses $(\mathbf{A}\ \mathbf{b})$ denotes augmentation. By augmentation we mean attaching a column $\mathbf{b}$ to the matrix $\mathbf{A}$ from the right. This can be generalized to matrices: $(\mathbf{A}\ \mathbf{B})$ denotes a matrix that contains all the columns of $\mathbf{A}$ followed by all the columns of $\mathbf{B}$. Row augmentation is defined analogously. For clarity the augmentation is sometimes also denoted $(\mathbf{A}|\mathbf{b})$, whereas the multiplication operation has higher priority. The determinant of a matrix is denoted $|\mathbf{A}|$, and the matrix ℓ_p operator norm (for $p > 1$) is $\|\mathbf{A}\|_p := \sup\{\|\mathbf{A}\mathbf{x}\|_p \mid \mathbf{x} \in \mathbb{R}^n, \|\mathbf{x}\|_p = 1\}$, where $\|\mathbf{x}\|_p = \sqrt[p]{x_1^p + \ldots + x_n^p}$ is the vector ℓ_p-norm. We use calligraphic letters for sets $(\mathcal{G})$. A distribution over a set $\mathcal{G}$ is denoted $\mathcal{D}_{\mathcal{G}}$.

1. Privacy-preserving Linear Programming

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 are 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 to compute the optimal solution for a linear program that is obtained by combining the data of different owners. It is likely that such a protocol would directly give us efficient privacy-preserving protocols for many other optimization tasks.

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

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

Here $\mathbf{A}$ is an $m \times n$ matrix, $\mathbf{b}$ is a vector of the length m and $\mathbf{c}$ is a vector of the 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 a privacy-preserving solution commonly expect the task to be in the *standard form*:

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

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

A *feasible solution* of a linear program is any vector $\mathbf{x}_0 \in \mathbb{R}^n$ 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 a 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 $\mathbf{A}$ and the vectors $\mathbf{b}, \mathbf{c}$ are 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 $\mathbf{Ax} \leq \mathbf{b}$ are horizontal partitioning (each party contributes some of the constraints) and vertical partitioning (each party knows certain columns of the matrix $\mathbf{A}$). More general ways of data partitioning are possible, but these are not considered by the transformation methods that we are attacking.

The parties apply a transformation to the shared linear programming task. The resulting task is published, so that any of the parties can solve it. Without lessening the generality, we assume that there are two parties called Alice and Bob.

1.1. Transformation Methods

Transformation-based methods for linear programming have been proposed in [11,6,12, 13,14,7,15,16,17,18]. 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 $\mathbf{A}$ and $\mathbf{b}$ in (2) by a random $m \times m$ invertible matrix $\mathbf{P}$ from the left was first introduced by Du [11]. This transformation modifies $\mathbf{A}$ and $\mathbf{b}$, but the feasible region remains unchanged.

Multiplying from the right (scaling and permutation). The idea of multiplying $\mathbf{A}$ and $\mathbf{b}$ in (2) by a random invertible matrix $\mathbf{Q}$ from the right was also proposed by Du [11]. Bednarz et al. [16] have shown that, in order to preserve the inequality $\mathbf{x} \geq \mathbf{0}$, the most general type of $\mathbf{Q}$ 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 $\mathbf{A}$. This transformation may also be applied to a problem in the canonical form (1).

Shifting. The shifting of variables was first proposed in [6], and it was also used in [7]. This transformation is achieved by replacing the constraints $\mathbf{Ax} \leq \mathbf{b}$ with $\mathbf{Ay} \leq \mathbf{b} + \mathbf{Ar}$, where $\mathbf{r}$ is a random non-negative vector of the 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 $\mathbf{r}$.

1.2. Security Definition

There are no formal security definitions for existing transformation-based approaches to privacy-preserving linear programming. The definition that has been used in previous works is *acceptable security*. This notion was first used in [19].

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 to the application.*

Some works provide a more detailed analysis [20,6] that estimates the probability that the adversary guesses some secret value. The leakage quantification analysis [6] 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. Such informal definitions allow the existence of some attacks that may be yet unknown, but may turn out to be efficient. Some particular attacks against transformation-based privacy-preserving linear programming will be described in Sec. 2.

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 provide such a definition in Sec. 3.

1.3. 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 [20] 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 $\mathbf{A}$ 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? 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 into equalities is not allowed.

Allowable Variable Types. May the variables be assumed to be non-negative? Or may they be assumed to be 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 in our attacks.

The attacks described in this chapter 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 major role in the main attacks, although some proposed transformation methods leave into it information that makes the attacks easier. In our treatment, we assume all variables to be non-negative.

1.4. Overview of the Proposed Methods

For exactly the setting described in the previous paragraph, Bednarz [20, Chapter 6] has proposed the following transformation. The set of constraints in (1) is transformed into

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

where $\hat{\mathbf{A}} = \mathbf{P}(\mathbf{A}\,\mathbf{I})\mathbf{Q}$, $\hat{\mathbf{b}} = \mathbf{P}\mathbf{b}$, $\mathbf{I}$ is the $m \times m$ identity matrix, $\mathbf{P}$ is a random invertible $m \times m$ matrix and $\mathbf{Q}$ 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 $\mathbf{x}_s$ by the equation $\left(\begin{smallmatrix} \mathbf{x} \\ \mathbf{x}_s \end{smallmatrix}\right) = \mathbf{Q}\mathbf{y}$. The objective function is disguised as $\hat{\mathbf{c}}^{\mathrm{T}} = (\mathbf{c}^{\mathrm{T}}\,\mathbf{0}^{\mathrm{T}})\mathbf{Q}$, where $\mathbf{0}$ is a vector of m zeroes.

Other proposed transformations for horizontally partitioned constraints can be easily compared with Bednarz's. Du [11] applied the multiplication with both $\mathbf{P}$ and $\mathbf{Q}$ (where $\mathbf{Q}$ 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. [16]. Vaidya [12] uses only the matrix $\mathbf{Q}$, with similar correctness problems. Mangasarian [14] uses only the multiplication with $\mathbf{P}$ for a system

with only equality constraints (2). Hong et al. [15] propose a complex set of protocols for certain kinds 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. [17] 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 [18] and shown to provide no security (their attack has slight similarities with the one we present in Sec. 2.2). 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. 2.1. The transformation by Dreier and Kerschbaum [6], when applied to (1), basically shifts the variables (as shown in Sec. 1.1), followed by Bednarz's transformation.

2. Attacks against Linear Programming Transformations

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. The transformed linear program is public and hence, it is fully available to Alice. As a target for our attack, we choose Bednarz's transformation (3), as it is a good generalization of most other transformations.

The outline of the attacks is following. 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. 2.1. When she has located the slack variables, she can remove these, turning the equalities back into inequalities of the form $\mathbf{A}'\mathbf{x}' \leq \mathbf{b}'$. These constraints are related to (1) by $\mathbf{A}' = \mathbf{P}'\mathbf{A}\mathbf{Q}'$, $\mathbf{b}' = \mathbf{P}'\mathbf{b}$, where both $\mathbf{P}'$ and $\mathbf{Q}'$ are generalized permutation matrices (of the size $m \times m$ and $n \times n$, respectively; $\mathbf{Q}'$ is also positive). Multiplication with $\mathbf{P}'$ from the left does not actually change the constraints, so the goal of Alice is to find $\mathbf{Q}'$. 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. This is described in Sec. 2.2.

2.1. Identifying the Slack Variables

For our attacks, we will need to change the *equalities* resulting from Bednarz's transformation (3) to *inequalities*. For that, we first need to locate the slack variables.

2.1.1. Looking at the Objective Function

When we add the slack variables to the system of inequalities in order to turn them to equations, the coefficients of these slack variables in the cost vector **c** will be 0. In the existing transformation methods, the cost vector **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. This way, the zero entries in **c** are not changed. If all original variables had nonzero coefficients in the objective function, then the location of zeroes in the transformed vector **c** would tell us the location of the slack variables.

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

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

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 maximize it.

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

2.1.2. Looking at the 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 $\mathbf{A}$. Namely, after the slack variables are introduced, they form an identity matrix that is augmented to $\mathbf{A}$ from the right. Thus, each slack column contains exactly one nonzero entry. The columns of $\mathbf{A}$ are very unlikely to contain just one nonzero entry. We have found that the columns of $\mathbf{P} \begin{pmatrix} \mathbf{A} & \mathbf{I} \end{pmatrix}$ can be distinguished by performing statistical analysis on the sizes of their entries. Even if using both positive and negative entries in $\mathbf{A}$ 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 $\mathbf{Q}$ does not provide any more protection.

We discovered this problem accidentally, 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 $\mathbf{P}$ (and $\mathbf{Q}$) should be sampled. We have conducted experiments where we have sampled these entries independently of each other, according to uniform distribution, or normal distribution (the parameters of the distribution do not matter here as they only affect the scale of the resulting matrix, or the variance of its entries relative to each other). It turns out that selecting the entries of $\mathbf{P}$ randomly according to either one of these distributions keeps the variables distinguishable. The results of the experiments can be seen in [8].

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 sizes of column entries approximately the same, 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 variances of the distributions of the sizes of column entries.

Another option is to extend the initial linear programming task with more inequalities, the entries of which are large enough to provide noise for all the variables after the transformation is applied. The problem is that introducing more constraints requires introducing more slack variables for correctness (otherwise, these new inequalities may conflict with the initial feasible region). 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 describe this attack in more detail in Sec. 2.3.

2.1.3. 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 $\hat{\mathbf{A}}$ 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 a manner that slack variables will be 0 more often than the original variables.

Our experimental results, which can be found in [8], show that, for many linear programs in canonical form (1), it is indeed possible to identify the slack variables after Bednarz's transformation. The validity of our hypothesis has been verified.

2.1.4. Several Slack Variables per Inequality

The authors of [18] proposed introducing multiple slack variables for the same inequality. We have found experimentally that in this case there is even a 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 $\hat{\mathbf{A}}$, corresponding to slack variables added to the same inequality, are multiples of each other. This makes them easily locatable.

2.1.5. Removing the Slack Variables

Once we have located the slack variables, we will reorder the variables in the constraints $\hat{\mathbf{A}}\mathbf{y} = \hat{\mathbf{b}}$ 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 $\hat{\mathbf{A}}$ 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 $(\mathbf{A}' \ \mathbf{I})\,\mathbf{y} = \mathbf{b}'$, where $\mathbf{I}$ is an $m \times m$ identity matrix. This system, together with the non-negativity constraints, is equivalent to the system of inequalities $\mathbf{A}'\mathbf{x}' \leq \mathbf{b}'$, where $\mathbf{x}'$ are the first n elements of $\mathbf{y}$.

2.2. 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 slack variables has been proposed in [6]. The attack works as follows. If the system contains only inequalities, then they completely reveal a scaled permutation of $\mathbf{P}$ that may be used afterwards to recover a scaled permutation of $\mathbf{A}$ whose scaling may be removed afterwards by searching for common factors. The factoring attack can be avoided by using real entries in $\mathbf{Q}$. 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 $j \in \{1, \dots, r\}$) 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 the permutation 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 $1 \le k, l \le r$) and two inequalities from the scaled and permuted system (e.g. k'-th and l'-th, where $1 \le k', l' \le m$). 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 $\sum_{i=1}^{n} a_{ji} x_i \le b_j$ in the original system let H_j be the corresponding hyperplane where $\le$ 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 $\{z_{ki}/z_{li} \mid 1 \le i \le n\}$ and $\{z'_{k'i}/z'_{l'i} \mid 1 \le i \le n\}$. 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 for discovering 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 to 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 [8], and verified that the attack works in practice.

2.3. Affine Relationships in Small Sets of Variables

Some works propose adding more variables to the system. For example, Dreier and Kerschbaum [6] propose a transformation where the variables are first shifted by a positive vector (as described in Sec. 1.1), and then Bednarz's transformation is applied to the resulting system (in [6], the construction is described somewhat differently, and the resulting positive generalized permutation matrix $\mathbf{Q}$ used to scale and permute the columns of the constraint system is not the most general matrix possible). This results in pairs of affinely related variables in the system. We present an attack that detects which variables form pairs, and that can be generalized also to larger sets of affinely related variables.

Suppose that we have a linear equation system $\mathbf{Ax} = \mathbf{b}$. Consider the solution space of this system. If the space contains small sets of t variables that are in an affine rela-

tionship $\alpha_1 x_{i_1} + \ldots + \alpha_t x_{i_t} = \beta$ for some $\alpha_i, \beta \in \mathbb{R}$ (that may not be obvious from the outer appearance of $\mathbf{A}$), then these equations may be recovered by looking through all the sets of variables of the 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 is the following.

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 a nonzero coefficient. Through this equation, express the variable x_j in terms of the other variables. Substitute it in all the other equations. Remove the equation and the variable x_j. If there are no equations where x_j has nonzero 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 $\alpha_i = 0$) so that $\alpha_1 x_{i_1} + \ldots + \alpha_t x_{i_t} = \beta$.

The way to apply this attack against shifting from Sec. 1.1 can be seen in [8].

3. Desired Security Definition

We have presented attacks against transformation-based methods for solving LP tasks in a privacy-preserving manner. The attacks are not merely theoretical constructions, but work with reasonable likelihood on problems of practical size.

The attacks work 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 [14,7], 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. 2.2. 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 into one another and the actual nature of the constraints may be hidden in the specified LP task.

Now we will 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 distinguished by it after the transformation.

We 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

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

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: for a transformed linear program to be 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 investigated recently by Bellare et al. [21] in the context of garbling circuits [22]. 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 *side information function* $\Phi : \mathfrak{T} \to \{0,1\}^*$ that, again, must be computable in polynomial time. 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 2. *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* $\mathbf{Exp}_{\mathcal{A}}^{\mathfrak{T}, \Phi}$:

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

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

Let $\mathfrak{T}$ be the set of all linear programming tasks "maximize $\mathbf{c}^T\mathbf{x}$, subject to $\mathbf{Ax} = \mathbf{b}$, $\mathbf{x} \geq \mathbf{0}$", determined by $\mathbf{A} \in \mathbb{R}^{m \times n}$, $\mathbf{b} \in \mathbb{R}^m$ and $\mathbf{c} \in \mathbb{R}^n$. In the following, we will take $\Phi(\mathbf{A}, \mathbf{b}, \mathbf{c}) = (m, n, \mathbf{x}_{opt}, bb(\mathbf{A}, \mathbf{b}))$, where $\mathbf{x}_{opt}$ is the unique optimal solution for the linear programming task and $bb(\mathbf{A}, \mathbf{b})$ is the bounding box for the polyhedron $\mathbf{Ax} = \mathbf{b}$, $\mathbf{x} \geq \mathbf{0}$. If the task is infeasible, unbounded, or has several different optimal solutions, then we consider the task unsuitable for transformation and take $\Phi(\mathbf{A}, \mathbf{b}, \mathbf{c}) = (\mathbf{A}, \mathbf{b}, \mathbf{c})$. 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). This is the reason why we do not attempt to hide $\mathbf{x}_{opt}$, although it might be important for some tasks.

The transformations described in Sec. 1.1 do not satisfy this definition. Let $\mathcal{F}$ be any transformation of Sec. 1.1, excluding shifting, as it is easy to undo by Gaussian elimination (Sec. 2.3). The remaining transformations just scale the feasible region and permute its coordinates. The adversary picks two LP tasks T_0 and T_1 with n variables, m equality constraints and the same optimal solution, so 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 the 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 the coordinates $(0, \ldots, 0, 1, 0, \ldots, 0)$. This count is not changed by scaling and permuting the coordinates.

4. Impossibility Results for Transformation-based Linear Programming

We show that it is impossible to information-theoretically achieve the security level of Def. 2, 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. 4.2 we mention a number of candidate secure transformations we have studied in [9]. 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. 4.3 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 multiparty LP solving are unusable, and the only currently known feasible method is to use privacy-preserving building blocks in implementing existing LP solving algorithms [23,24].

4.1. 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. 2 is zero. It is 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.

Without loss of generality we assume that both the inputs and the outputs of $\mathcal{F}$ are linear programs in the canonical form (1). We prove that any perfectly secure transformation with the following properties cannot be computationally much simpler than solving the LP task.

1. The optimal solution $\mathbf{y}_{\text{opt}}$ to the transformed linear program $\mathcal{F}(\mathbf{A}, \mathbf{b}, \mathbf{c}; r)$ only depends on r and $\Phi(\mathbf{A}, \mathbf{b}, \mathbf{c})$.
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 $\delta > 0$, so that if $(\mathbf{A}^\circ, \mathbf{b}^\circ, \mathbf{c}^\circ)$ and $(\mathbf{A}^\bullet, \mathbf{b}^\bullet, \mathbf{c}^\bullet)$ are two LP tasks with n variables, m con-

straints and the optimal solutions satisfying $\|\mathbf{x}^{\circ}{}_{\text{opt}} - \mathbf{x}^{\bullet}{}_{\text{opt}}\| \leq \delta$, then the optimal solutions of the transformed tasks $\mathcal{F}(\mathbf{A}^{\circ}, \mathbf{b}^{\circ}, \mathbf{c}^{\circ}; r)$ and $\mathcal{F}(\mathbf{A}^{\bullet}, \mathbf{b}^{\bullet}, \mathbf{c}^{\bullet}; r)$ satisfy $\|\mathbf{y}^{\circ}{}_{\text{opt}} - \mathbf{y}^{\bullet}{}_{\text{opt}}\| \leq \varepsilon$.

We find the properties natural: they are satisfied by all transformations proposed in the literature so far 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).

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 a number of bounding hyperplanes m. These allow us to afterwards solve an arbitrarily large proportion of interesting LP tasks of the dimension $n-1$ with $m-2$ facets with an effort that is only marginally greater than applying $\mathcal{F}$ and $\mathcal{G}$.

For the precomputation, we first fix the randomness r. We construct an 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 $\mathbf{x}^{\text{pre}}_{\text{opt}} = (1, \ldots, 1)^{\text{T}}$. 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 $\mathbf{y}^{\text{pre}}_{\text{opt}}$. All this has to be done only once for a fixed n and m.

Let T be an LP task with $n-1$ variables and $m-2$ constraints. Let P be the polyhedron defined by these constraints, and let the bounding box of P be an $(n-1)$-dimensional hypercube. We want to find the solution $\mathbf{y}'_{opt}$ that is at the distance at most ε from $\mathbf{y}^{\text{pre}}_{\text{opt}}$ (we show that, for a small enough ε, this allows to find $\mathbf{y}'_{\text{opt}}$ precisely). Let δ be the corresponding distance between $\mathbf{x}^{\text{pre}}_{\text{opt}}$ and $\mathbf{x}'_{\text{opt}}$ (from the continuity of the transformation). 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 (an n-dimensional hypercube). This modifies the system of constraints in the following way.

   - 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.

Note that the intersection of the constraints $\sum_{i=1}^{n-1} a_i(x_i + (1 - \delta)x_n) \leq b$ with $x = 0$ forms P_0. We denote the intersection of these constraints with $x = 1$ as P_1, which is a shifted copy of P_0. An example of 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 $\mathbf{c} = (c_1, \ldots, c_{n-1})$ is the objective function vector of T and C is a large constant. Hence, the optimal solution

Figure 1. Results of preparatory transformations to task T

$\mathbf{x}'_{\text{opt}}$ of T' is located on the base P_1 of the hyperprism, otherwise being at the same vertex as the optimal solution $\mathbf{x}_{\text{opt}}$ to T. In particular, $\mathbf{x}'_{\text{opt}}$ can easily be transformed back to $\mathbf{x}_{\text{opt}}$. Also note that $\|\mathbf{x}'_{\text{opt}} - \mathbf{x}^{\text{pre}}_{\text{opt}}\| \leq \delta$.

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

If a hyperplane bounding $\bar{P}$ is at a distance of at most ε from the point $\mathbf{y}^{\text{pre}}_{\text{opt}}$, then this hyperplane contains $\mathbf{y}'_{\text{opt}}$ with a probability p that decreases with ε. Hence, to find $\mathbf{y}'_{\text{opt}}$, we measure how far each hyperplane bounding $\bar{P}$ is from the point $\mathbf{y}^{\text{pre}}_{\text{opt}}$, 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. Proof of the existence of a suitable ε can be found in [9].

We have shown that the existence of a perfectly secure transformation with properties (1) and (2) implies getting the optimal solution $\mathbf{x}_{\text{opt}}$ *without* needing to solve the transformed linear programming task.

4.2. Observing 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 private randomness in order to obtain the optimal solution for the LP task.

We have studied certain more general transformations:

- adding new variables with constraints;
- adding new variables without constraints;
- splitting the variables.

None of these satisfy our privacy requirement (the transformation details and the attacks can be seen in [9]). 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 *empirically* set an extra requirement that in the transformed task (in the polyhedron corresponding to this task), all variables essentially look the same. In addition to the variables, we want the same condition to hold for (small) sets of variables, to avoid attacks based on Gaussian elimination from Sec. 2.3. We need the following notions to formally define our requirement. Let $\mathbf{e}_i^k = (0, \ldots, 0, 1, 0, \ldots, 0)$ be the *i*-th *unit vector* in the *k*-dimensional space $\mathbb{R}^k$ (the length of $\mathbf{e}_i^k$ is *k* and the only 1 is on the *i*-th position). For $I \subseteq \mathbb{N}$ and $i \in I$ let $\mathrm{idx}_I\, i$ be the *index* of *i* in the ordered set *I*, meaning that *I* has exactly $\mathrm{idx}_I\, i$ 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 the dimensions in *I*. It is a linear mapping defined by $\pi_I^k(\mathbf{e}_i^k) = \mathbf{e}_{\mathrm{idx}_I\, i}^n$ if $i \in I$, and $\pi_I^k(\mathbf{e}_i^k) = 0$ 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 $\hat{\sigma}(\mathbf{e}_i^n) = \mathbf{e}_{\sigma(i)}^n$.

Definition 3. *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 4. *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:*

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

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 an LP task to be such that a possible set of the 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 (we can achieve it by assuming that the bounding box of the transformed task is a hypercube) and 2-symmetric. If we fail to achieve this property for $t = 2$, then we cannot achieve it for any larger *t*, as any projection to *t* dimensions can in turn be projected to 2 dimensions.

4.3. 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 $\mathbf{Ax} = \mathbf{b}$, $\mathbf{x} \geq \mathbf{0}$.

4.3.1. A Computable Property of Polyhedra

Consider the polyhedron *P* determined by $\mathbf{Ax} = \mathbf{b}$, $\mathbf{x} \geq \mathbf{0}$, with *m* equality constraints and *n* variables. Let $i, j \in \{1, \ldots, n\}$ be two coordinates, so that $x_i = x_j = 0$ would not
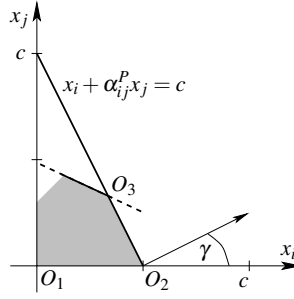
Figure 2. Finding the value of α_{ij}^P

contradict the constraints (such variables always exist if $m \leq n+2$), and consider the projection of P to the (x_i, x_j)-plane. Assume that the projection does not equal the whole first quadrant of the plane. Also assume that it is not a point or a line (segment). These assumptions are reasonable, as by 2-symmetry all the projections should be equal, and if all the projections were first quadrants, then the initial feasible region would have to be the first orthant. By similar reasoning, if the projection to at least one (x_i, x_j) plane is a line, then all the projections to other (x_k, x_ℓ) planes should be lines, and such a transformation would be able to hide only 2-dimensional polyhedrons.

Under these assumptions, 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, which 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 is a polynomial time algorithm that on input* **A**, **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 $\mathbf{x} \in P$, $x_j = 0$".

Let $D(\mathbf{x}, \gamma)$ denote the direction $(\cos \gamma) \cdot x_i + (\sin \gamma) \cdot x_j$. Using binary search, the algorithm will then find $\gamma \in [-\frac{\pi}{2}, \frac{\pi}{2}]$, so that the optimal solution for the LP task "maximize $D(\mathbf{x}, \gamma - \varepsilon)$ subject to $\mathbf{x} \in P$" is in a point $\mathbf{x}'$ with $x_i' = c$ and $x_j' = 0$, while the optimal solution for the LP task "maximize $D(\mathbf{x}, \gamma + \varepsilon)$, subject to $\mathbf{x} \in P$" is in a point $\mathbf{x}''$ 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.

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

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.

4.3.2. 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

$$\{(x_1, \ldots, x_n) \mid x_1 + \ldots + x_n \leq c\} \tag{5}$$

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

This result implies that any computationally 2-symmetric transformation $(\mathcal{F}, \mathcal{G})$ easily can be turned into an LP solving algorithm. Given an LP task, the solver will first apply $\mathcal{F}$ to it, resulting in the polyhedron (5) 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 with $x_1, \ldots, x_{n-1}$, or by binary search).

Let the transformed polyhedron P be defined by $\mathbf{Ax} = \mathbf{b}$, $\mathbf{x} \geq \mathbf{0}$, where $\mathbf{A}$ is a nonsingular $m \times n$ matrix for $m \leq n - 1$. If $m = n - 1$, then P is one-dimensional and hence, it has at most 2 vertices that can be found easily and $(\mathcal{F}, \mathcal{G})$ can be turned again into an 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 $\mathbf{Ax} = \mathbf{b}$ 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 same property, otherwise the (computational) 2-symmetry would be broken. For an arbitrary i, j, consider the projection of P to the dimensions x_i and x_j. As 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. 4.3.1. 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, $\alpha \leq 1$, 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.

As 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 $x_i + \alpha x_j \leq c$ 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, as the inequality $x_i + \alpha x_j \leq c$ implies the existence of the inequality $\alpha x_i + x_j \leq c$ due to the symmetry requirement, as shown in Fig. 3. Due to the convexity of the projection, any valuation (v_i, v_j) of x_i and x_j so that $v_i + v_j \leq c$ must be possible.

Given n variables, any inequality $x_i + \alpha x_j \leq c$ 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 $a_k \geq 0$ 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:

$$\begin{cases} \alpha x_1 + x_2 + a_{123} x_3 + \ldots + a_{12(n-1)} x_{n-1} + a_{12n} x_n = c \\ x_1 + \alpha x_2 + a_{213} x_3 + \ldots + a_{21(n-1)} x_{n-1} + a_{21n} x_n = c \\ a_{231} x_1 + \alpha x_2 + x_3 + \ldots + a_{23(n-1)} x_{n-1} + a_{23n} x_n = c \\ a_{321} x_1 + x_2 + \alpha x_3 + \ldots + a_{32(n-1)} x_{n-1} + a_{32n} x_n = c \quad , \\ \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \\ \quad a_{n(n-1)1} x_1 + a_{n(n-1)2} x_2 + \ldots + \alpha x_{n-1} + x_n = c \\ \quad a_{(n-1)n1} x_1 + a_{(n-1)n2} x_2 + \ldots + x_{n-1} + \alpha x_n = c \end{cases} \tag{6}$$

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

where $a_{ijk} \geq 0$ for all $i, j, k \in \{1, \ldots, n\}$, and each equation contains at least one strictly positive a_{ijk}. These equations imply $x_1 + \ldots + x_n = c$. This can be done by case distinction on $\alpha < 0$, $\alpha = 0$, and $\alpha > 0$. The detailed proof can be found in [9].

Finally, we show 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:

$$x_1 + \ldots + x_n = c \ , \tag{7}$$

$$a_1 x_1 + \ldots + a_n x_n = b \ , \tag{8}$$

where $a_i, b \in \mathbb{R}$. One can show that the equation (8) can be at most a multiple of the equation (7), representing the same constraint. Proof details can be found in [9].

We see that the only way to achieve 2-symmetry is by defining the feasible region of the transformed linear program $\mathcal{F}(\mathbf{A}, \mathbf{b}, \mathbf{c}; r)$ as $\{(x_1, \ldots, x_n) \mid x_1 + \ldots + x_n \leq c\}$. This is not enough to encode anything reasonable. All the uniquely identifiable optimal solutions of such a feasible region are the points $(0, \ldots, 0)$ and $c \cdot \mathbf{e}_i$ for a unit vector $\mathbf{e}_i$. Hence, there are at most as many solutions as coordinates in the transformed program, unless the solution is already encoded in c, which would imply that we have already solved the initial LP. In order to encode at least all the vertices of Q_n, we would need 2^n coordinates, and that would make the transformed LP task too large.

4.4. Summary

We have shown that the current approaches towards privacy-preserving outsourcing of 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 a privacy-preserving manner, cryptographic methods for securely implementing Simplex or some other linear programming solving algorithm are the only viable approach.

5. Transformation-based Solving of Linear Equation Systems

After obtaining negative results for transformation-based privacy-preserving *linear programming*, we study the transformation-based privacy-preserving solving of *systems of*

linear equations. This seems simpler, as instead of hiding the entire feasible region, it is sufficient to hide just one point. This can be later used to develop a privacy-preserving method for linear programming. One way to solve linear programming tasks is the interior point method, the privacy-preserving version of which has been proposed in [20]. On each iteration, the algorithm solves a linear equation system over $\mathbb{R}$. Outsourcing could give us an efficient way of solving such systems.

The task is to transform $(\mathbf{A}, \mathbf{b})$ into another system $(\mathbf{B}, \mathbf{d})$ in such a way that $(\mathbf{B}, \mathbf{d})$ would leak no information about $(\mathbf{A}, \mathbf{b})$, and it would be easy to recover the solution $\mathbf{x}_0$ to $(\mathbf{A}, \mathbf{b})$ from the solution $\mathbf{y}_0$ to $(\mathbf{B}, \mathbf{d})$. Since the transformation itself has to be computed in a privacy-preserving way, it should be more efficient than computing $\mathbf{x}_0$ directly. Again, we constrain this work to affine transformations, as they are the most natural approach for hiding a system of linear equations. Let the initial system be $n \times n$ full-rank.

5.1. Structure of Affine Transformations

Using affine transformations, the solution to the initial problem is computed from the solution to the transformed problem as $\mathbf{x}_0 = \mathbf{F}\mathbf{y}_0 + \mathbf{r}_0$ for some $\mathbf{F} \in \mathbb{R}^{n \times m}$ and $\mathbf{r}_0 \in \mathbb{R}^n$ that are generated by the client together with $\mathbf{B}$ and $\mathbf{d}$. We may assume that $\mathbf{F}$ is full-rank, otherwise there will have to be certain constant relationships between the variables of $\mathbf{x}_0$, and in general, we cannot make any assumptions about the solution.

The entire transformed problem is $(\mathbf{B}, \mathbf{d}, \mathbf{F}, \mathbf{r}_0)$. In our settings, we only require that $\mathbf{B}$ be published, as the most complex part of solving the system of equations is finding the inverse of $\mathbf{B}$. Hence, the affine transformation $\mathbf{F}\mathbf{y}_0 + \mathbf{r}_0$ does not have to be hiding, and $(\mathbf{d}, \mathbf{F}, \mathbf{r}_0, \mathbf{y}_0)$ are allowed leak information about $(\mathbf{A}, \mathbf{b})$ (for example, they leak $\mathbf{x}_0$ in any case). The main questions are under which conditions $\mathbf{d}$, $\mathbf{F}$, $\mathbf{r}_0$, $\mathbf{B}$ can be generated efficiently, and whether satisfying these conditions allows keeping $\mathbf{B}$ secure according to Def. 2 in Sec. 3.

We consider only affine transformations for the purpose of generating $\mathbf{B}$, i.e. $\mathbf{B} = \mathbf{P}\mathbf{A}\mathbf{Q} + \mathbf{R}$, where $\mathbf{P} \in \mathbb{R}^{m \times n}$, $\mathbf{Q} \in \mathbb{R}^{n \times m}$, and $\mathbf{R} \in \mathbb{R}^{m \times m}$ are random matrices over real numbers sampled independently from $\mathbf{A}$ according to some distribution. Here $m \geq n$, and $\mathbf{P}$, $\mathbf{Q}$ are both full-rank, as otherwise $\mathbf{y}_0$ would not contain enough information to reconstruct $\mathbf{x}_0$.

5.2. Unavoidable Restrictions on $\mathbf{B} = \mathbf{P}\mathbf{A}\mathbf{Q} + \mathbf{R}$

We show that, if the transformation has to work for all $n \times n$ systems of linear equations, then the form $\mathbf{B} = \mathbf{P}\mathbf{A}\mathbf{Q} + \mathbf{R}$ is not more general than the apparently more restricted form $\mathbf{B} = \mathbf{P} \begin{pmatrix} \mathbf{A} & \mathbf{0} \\ \mathbf{0} & \mathbf{I} \end{pmatrix}$ for invertible $m \times m$ matrices $\mathbf{P}$ and $\mathbf{Q}$. First we show that $\mathbf{R}$ cannot be full-rank if we do not make any assumptions on $\mathbf{x}_0$.

As $\mathbf{x}_0 = \mathbf{F}\mathbf{y}_0 + \mathbf{r}_0$, we may write, without loss of generality, $\mathbf{y}_0 = \begin{pmatrix} \mathbf{y}_1 \\ \mathbf{y}_2 \end{pmatrix}$ and $\mathbf{F} = (\mathbf{F}_1 \mid \mathbf{F}_2)$, where $\mathbf{F}_1$ is invertible. If $\mathbf{F}_1$ is not invertible, we may always permute the columns by defining a new matrix $\mathbf{F}' := \mathbf{F}\mathbf{T}$ for an $m \times m$ permutation matrix $\mathbf{T}$, and such an $\mathbf{F}_1$ exists as $\mathbf{F}$ is full-rank. Then $\mathbf{y}_1 = \mathbf{F}_1^{-1}(\mathbf{x}_0 - \mathbf{r}_0) - \mathbf{F}_1^{-1}\mathbf{F}_2\mathbf{y}_2$. We may now express the equation $(\mathbf{P}\mathbf{A}\mathbf{Q} + \mathbf{R})\mathbf{y}_0 = \mathbf{d}$ through $\mathbf{y}_2$ only. Let $\mathbf{B} = \mathbf{P}\mathbf{A}\mathbf{Q} + \mathbf{R}$ be divided into four blocks, indexed $\mathbf{B}_{11}$, $\mathbf{B}_{12}$, $\mathbf{B}_{21}$ and $\mathbf{B}_{22}$, where the left blocks ($\mathbf{B}_{11}$ and $\mathbf{B}_{21}$) correspond to the variables $\mathbf{y}_1$, and the right blocks ($\mathbf{B}_{12}$ and $\mathbf{B}_{22}$) to the variables $\mathbf{y}_2$. Let

$\mathbf{B}_{ij} = (\mathbf{PAQ} + \mathbf{R})_{ij} = (\mathbf{PAQ})_{ij} + \mathbf{R}_{ij}$, where $(\mathbf{PAQ})_{ij}$ and $\mathbf{R}_{ij}$ are analogous blocks. We get the following:

$$\begin{pmatrix} \mathbf{B}_{11}\mathbf{F}_1^{-1}\mathbf{F}_2 + \mathbf{B}_{12} \\ \mathbf{B}_{21}\mathbf{F}_1^{-1}\mathbf{F}_2 + \mathbf{B}_{22} \end{pmatrix} \mathbf{y}_2 = \mathbf{d} - \begin{pmatrix} \mathbf{B}_{11} \\ \mathbf{B}_{21} \end{pmatrix} \mathbf{F}_1^{-1}(\mathbf{x}_0 - \mathbf{r}_0) \ . \tag{9}$$

As the number of rows is $m > m - n$, some n rows of the matrix on the left-hand side are a certain linear combination of the other $m - n$ rows. Here again, without loss of generality, we may permute the rows of $\mathbf{B}$ in such a way that the *upper n rows are linear combinations of the lower $m - n$ rows* by taking $\mathbf{P}' := \mathbf{TP}$ and $\mathbf{R}' := \mathbf{TR}$ for an $m \times m$ permutation matrix $\mathbf{T}$. Let this linear combination be represented by a matrix $\mathbf{X}$. Formally, $\mathbf{X}$ is an $(n \times (m - n))$ matrix so that

$$\mathbf{X}\mathbf{B}_{21}\mathbf{F}_1^{-1}\mathbf{F}_2 + \mathbf{X}\mathbf{B}_{22} = \mathbf{B}_{11}\mathbf{F}_1^{-1}\mathbf{F}_2 + \mathbf{B}_{12} \ . \tag{10}$$

For (9) to be satisfied, the upper n entries of the right-hand side must be the same linear combination of the lower $m - n$ entries. If we denote $\mathbf{d} = \begin{pmatrix} \mathbf{d}_1 \\ \mathbf{d}_2 \end{pmatrix}$, where $\mathbf{d}_1$ are the n upper entries of $\mathbf{d}$, and $\mathbf{d}_2$ are the $m - n$ lower entries, we get

$$\mathbf{X}\mathbf{d_2} - \mathbf{X}\mathbf{B}_{21}\mathbf{F}_1^{-1}(\mathbf{x}_0 - \mathbf{r}_0) = \mathbf{d_1} - \mathbf{B}_{11}\mathbf{F}_1^{-1}(\mathbf{x}_0 - \mathbf{r}_0) \ . \tag{11}$$

By assumption, $\mathbf{d}$, $\mathbf{r}_0$, $\mathbf{F}$ do not depend on $\mathbf{x}_0$. Also, the value $\mathbf{x}_0$ is completely independent of $\mathbf{A}$ alone, if $\mathbf{b}$ is not taken into account, as for a fixed $\mathbf{A}$, for any $\mathbf{x}_0$ there exists a $\mathbf{b}$ so that $\mathbf{A}\mathbf{x}_0 = \mathbf{b}$. Hence, (11) must hold for any $\mathbf{x}_0$, and we may treat the entries of $\mathbf{x}_0$ as formal variables. We get the following polynomial vector equality:

$$\mathbf{X}\mathbf{d}_2 - \mathbf{d}_1 + (\mathbf{B}_{11} - \mathbf{X}\mathbf{B}_{21})\mathbf{F}_1^{-1}\mathbf{x}_0 - (\mathbf{B}_{11} - \mathbf{X}\mathbf{B}_{21})\mathbf{F}_1^{-1}\mathbf{r}_0 = \mathbf{0} \ .$$

For the left-hand side to be a zero polynomial with respect to $\mathbf{x}_0$, the equality $X\mathbf{d}_2 - \mathbf{d}_1 - (\mathbf{B}_{11} - \mathbf{X}\mathbf{B}_{21})\mathbf{F}_1^{-1}\mathbf{r}_0 = \mathbf{0}$ must hold, as one possible solution is $\mathbf{x}_0 = \mathbf{0}$. Also, $(\mathbf{B}_{11} - \mathbf{X}\mathbf{B}_{21})\mathbf{F}_1^{-1} = \mathbf{0}$ must hold to satisfy any $\mathbf{x}_0$ that does not nullify it. As $\mathbf{F}_1^{-1}$ is invertible, this reduces to $\mathbf{B}_{11} - \mathbf{X}\mathbf{B}_{21} = \mathbf{0}$, or equivalently $(\mathbf{PAQ})_{11} - \mathbf{X}(\mathbf{PAQ})_{21} + \mathbf{R}_{11} - \mathbf{X}\mathbf{R}_{21} = \mathbf{0}$. We have got that $\mathbf{R}_{11} = \mathbf{X}(\mathbf{PAQ})_{21} - (\mathbf{PAQ})_{11} + \mathbf{X}\mathbf{R}_{21}$. Substituting $\mathbf{X}\mathbf{B}_{21}$ with $\mathbf{B}_{11}$ in $\mathbf{X}\mathbf{B}_{21}\mathbf{F}_1^{-1}\mathbf{F}_2 + \mathbf{X}\mathbf{B}_{22} = \mathbf{B}_{11}\mathbf{F}_1^{-1}\mathbf{F}_2 + \mathbf{B}_{12}$, we get $\mathbf{X}\mathbf{B}_{22} = \mathbf{B}_{12}$.

We can write this as $\mathbf{R}_{12} = \mathbf{X}(\mathbf{PAQ})_{22} - (\mathbf{PAQ})_{12} + \mathbf{X}\mathbf{R}_{22}$. Now $(\mathbf{R}_{11} \mid \mathbf{R}_{12}) = \mathbf{X}((\mathbf{PAQ})_{21} \mid (\mathbf{PAQ})_{22}) - ((\mathbf{PAQ})_{11} \mid (\mathbf{PAQ})_{12}) + \mathbf{X}(\mathbf{R}_{21} \mid \mathbf{R}_{22})$.

For any $\mathbf{P} = \begin{pmatrix} \mathbf{P}_1 \\ \mathbf{P}_2 \end{pmatrix}$ and $\mathbf{Q} = (\mathbf{Q}_1 \mid \mathbf{Q}_2)$, the quantity $\mathbf{PAQ}$ looks as follows:

$$\begin{pmatrix} \mathbf{P}_1 \\ \mathbf{P}_2 \end{pmatrix} \mathbf{A} \ (\mathbf{Q}_1 \mid \mathbf{Q}_2) = \begin{pmatrix} \mathbf{P}_1\mathbf{A}\mathbf{Q}_1 & \mathbf{P}_1\mathbf{A}\mathbf{Q}_2 \\ \mathbf{P}_2\mathbf{A}\mathbf{Q}_1 & \mathbf{P}_2\mathbf{A}\mathbf{Q}_2 \end{pmatrix} \ .$$

Hence, we have $(\mathbf{PAQ})_{ij} = \mathbf{P}_i\mathbf{A}\mathbf{Q}_j$. We may always take $\mathbf{P}'$ so that $\mathbf{P}'_2 := \mathbf{P}_2$ and $\mathbf{P}'_1 := \mathbf{X}\mathbf{P}_2$. Such a choice of $\mathbf{P}'$ does not depend on $\mathbf{A}$: if we have achieved hiding for $\mathbf{B} = \mathbf{PAQ} + \mathbf{R}$, then $\mathbf{X}$ is independent of $\mathbf{A}$, as it is computable from $\mathbf{B}_{1j} = \mathbf{X}\mathbf{B}_{2j}$, and so would leak information about $\mathbf{A}$. We get $(\mathbf{PAQ} + \mathbf{R}) = (\mathbf{P}'\mathbf{AQ} + \mathbf{R}')$ where $\mathbf{R}'$ is such

that $(\mathbf{R}'_{11} \mid \mathbf{R}'_{12}) = \mathbf{X}(\mathbf{R}'_{21} \mid \mathbf{R}'_{22})$ and hence, $\mathbf{R}$ is of a rank of at most $(m-n)$. It is actually exactly of the rank $m-n$, as otherwise $(\mathbf{PAQ}+\mathbf{R})$ would not be full-rank. We get that hiding with an arbitrary $\mathbf{R}$ is not better than hiding with an $\mathbf{R}$ of the rank $(m-n)$.

If $\mathbf{R}$ is of the rank $(m-n)$, then it is of the form $\begin{pmatrix} \mathbf{SR'T} & \mathbf{SR'} \\ \mathbf{R'T} & \mathbf{R'} \end{pmatrix}$, where $\mathbf{R}'$ is an $(m-n) \times (m-n)$ invertible submatrix, and $\mathbf{S}$ and $\mathbf{T}$ are arbitrary (here without loss of generality we assume that the $(m-n) \times (m-n)$ invertible submatrix is located in the right lower part of $\mathbf{R}$, as again we may take $\mathbf{R}' = \mathbf{T}_1\mathbf{R}\mathbf{T}_2$ for permutation matrices $\mathbf{T}_1$ and $\mathbf{T}_2$. Now $\mathbf{B} = (\mathbf{PAQ}+\mathbf{R})$ can be rewritten as follows:

$$\mathbf{B} = \begin{pmatrix} \mathbf{P}_1\mathbf{AQ}_1 & \mathbf{P}_1\mathbf{AQ}_2 \\ \mathbf{P}_2\mathbf{AQ}_1 & \mathbf{P}_2\mathbf{AQ}_2 \end{pmatrix} + \begin{pmatrix} \mathbf{SR'T} & \mathbf{SR'} \\ \mathbf{R'T} & \mathbf{R'} \end{pmatrix} = \begin{pmatrix} \mathbf{P}_1 & \mathbf{S} \\ \mathbf{P}_2 & \mathbf{I} \end{pmatrix} \cdot \begin{pmatrix} \mathbf{A} & \mathbf{0} \\ \mathbf{0} & \mathbf{I} \end{pmatrix} \cdot \begin{pmatrix} \mathbf{Q}_1 & \mathbf{Q}_2 \\ \mathbf{R'T} & \mathbf{R'} \end{pmatrix} .$$

In order for $\mathbf{B}$ to be invertible, both $\mathbf{P}' = \begin{pmatrix} \mathbf{P}_1 & \mathbf{S} \\ \mathbf{P}_2 & \mathbf{I} \end{pmatrix}$ and $\mathbf{Q}' = \begin{pmatrix} \mathbf{Q}_1 & \mathbf{Q}_2 \\ \mathbf{R'T} & \mathbf{R'} \end{pmatrix}$ have to be invertible. Let $\mathbf{A}' = \begin{pmatrix} \mathbf{A} & \mathbf{0} \\ \mathbf{0} & \mathbf{I} \end{pmatrix}$. The most general form of an affine transformation is thus $\mathbf{B} = \mathbf{P}'\mathbf{A}'\mathbf{Q}'$, where the distributions of $\mathbf{P}'$ and $\mathbf{Q}'$ do not depend on $\mathbf{A}'$, and $\mathbf{A}'$ uniquely defines $\mathbf{A}$ (for the fixed n and m). Hence, hiding $\mathbf{A}'$ is equivalent to hiding $\mathbf{A}$, and increasing the size of $\mathbf{I}$ may be used to increase the security parameter if it turns out to be dependent on the size of $\mathbf{A}'$. In the next sections, we study the hiding properties of the transformation $\mathbf{B} = \mathbf{PAQ}$.

5.3. On the Entries of Hiding Matrices

Let $\mathcal{A}$ denote the set of matrices that we want to make indistinguishable after applying the transformation. We are looking for necessary and sufficient properties that $\mathcal{A}$ must satisfy in order to make hiding possible. Before starting to investigate the sets of matrices $\mathcal{A}$, we characterize the distributions of matrices $\mathbf{P}$ and $\mathbf{Q}$ on the basis of their entries. We consider that there must be an efficient procedure that samples $\mathbf{P}$ and $\mathbf{Q}$; hence, the entries of $\mathbf{P}$ and $\mathbf{Q}$ cannot be too large.

There may be a distribution of matrices $\mathbf{P}$ and $\mathbf{Q}$ where the entries of $\mathbf{P}$ and $\mathbf{Q}$ are much larger than the entries of the matrices in $\mathcal{A}$, so that the multiplication $\mathbf{B} = \mathbf{PAQ}$ gives statistical security. In practice, we can do it only if the *sizes* of the entries of $\mathbf{P}$ and $\mathbf{Q}$ are reasonably small. By *size* we mean the number of bits required to represent an entry, which is logarithmic in the actual entry.

For $a \in \mathbb{R}$, it is actually less clear what its *size* is, and it is something that we do not want to define in a detailed manner. It is clear, however, that we do not want the transformation mechanism to introduce numbers that are too large. We argue that $\log a$ is still a reasonable measure for the information content and thus, the size of a. Indeed, we are using larger entries of $\mathbf{P}$ and $\mathbf{Q}$ to statistically hide the matrices in $\mathcal{A}$, and the entropy of these entries is proportional to our definition of size.

We say that the sizes of the entries $\mathbf{P}$ and $\mathbf{Q}$ are *bounded*, if they are polynomial in the security parameter η. The bound may depend on the actual set of matrices $\mathcal{A}$.

If the entries of a matrix are bounded, then some matrix properties that grow with the sizes of the entries, such as *determinant* or *norms*, are also bounded. This in particular implies that the ℓ_2 operator norms of $\mathbf{P}$ and $\mathbf{Q}$ (the *spectral norms*) are bounded. This

fact can be used to apply some known theorems from group representation theory. The relation between the ℓ_2-norm and the sizes of the entries is $\|\mathbf{A}\|_2 \leq \sqrt{\|\mathbf{A}\|_1 \|\mathbf{A}\|_\infty}$, and hence $\|\mathbf{A}\|_2 \geq 1/\|\mathbf{A}^{-1}\|_2 \geq 1/\sqrt{\|\mathbf{A}^{-1}\|_1 \|\mathbf{A}^{-1}\|_\infty}$, where $\|\mathbf{A}\|_1 = \max_{1 \leq k \leq n} \sum_{j=1}^{n} a_{jk}$ and $\|\mathbf{A}\|_\infty = \max_{1 \leq j \leq n} \sum_{k=1}^{n} a_{jk}$ [25]. If the entries of $\mathbf{A}$ are at most c, then a rough bound on the absolute value of the determinant is $|n! \cdot c^n|$, according to the definition of the determinant.

5.4. Tolerable Side Information

In this section we look for the side information that multiplicative hiding $\mathbf{B} = \mathbf{PAQ}$ leaks. This allows us to define a suitable side information function Φ from Sec. 3.

5.4.1. Determinant

The determinant of a square matrix is multiplicative, so we have $|\mathbf{PAQ}| = |\mathbf{P}| \cdot |\mathbf{A}| \cdot |\mathbf{Q}|$. We show that, regardless of the distribution $\mathcal{D}_{(\mathbf{P},\mathbf{Q})}$ of $\mathbf{P}$ and $\mathbf{Q}$ (from which sampling is feasible), the absolute value of the determinant is leaked. Perfect hiding is impossible. The problem is similar to hiding $a \in \mathbb{R}$ by multiplying it with a random $r \in \mathbb{R}$. For a fixed $\mathbf{A}$ we get $|\mathbf{PAQ}| = |\mathbf{P}| \cdot |\mathbf{A}| \cdot |\mathbf{Q}| \leftarrow |\mathbf{A}| \cdot \mathcal{D}_{|\mathbf{P}| \cdot |\mathbf{Q}|}$ where $\mathcal{D}_{|\mathbf{P}| \cdot |\mathbf{Q}|} = \{|\mathbf{P}| \cdot |\mathbf{Q}| \mid (\mathbf{P},\mathbf{Q}) \leftarrow \mathcal{D}_{(\mathbf{P},\mathbf{Q})}|\}$. Hence, if $|\mathbf{A}_1| \neq \pm|\mathbf{A}_2|$, the distributions $|\mathbf{A}_1| \cdot \mathcal{D}_{|\mathbf{P}| \cdot |\mathbf{Q}|}$ and $|\mathbf{A}_2| \cdot \mathcal{D}_{|\mathbf{P}| \cdot |\mathbf{Q}|}$ are different: one is a scaled version of another. We can rewrite it as $\frac{|\mathbf{A}_1|}{|\mathbf{A}_2|} \cdot \mathcal{D}_{|\mathbf{P}| \cdot |\mathbf{Q}|} = \mathcal{D}_{|\mathbf{P}| \cdot |\mathbf{Q}|}$, so $\mathcal{D}_{|\mathbf{P}| \cdot |\mathbf{Q}|}$ has to be a scaled version of itself. Even if we find a method to hide it statistically, the sizes of the entries of $\mathbf{P}$ and $\mathbf{Q}$ become too large.

Claim 1. *Let η be a security parameter. If there exist $\mathbf{A}_1, \mathbf{A}_2 \in \mathcal{A}$, $|\mathbf{A}_1| \neq \pm|\mathbf{A}_2|$ so that $SD(\mathcal{D}_{|\mathbf{PA}_1\mathbf{Q}|}, \mathcal{D}_{|\mathbf{PA}_2\mathbf{Q}|}) < 2^{-\eta}$, then the sizes of the entries of $\mathbf{P}$ and $\mathbf{Q}$ are of the order $2^{\eta/n}$.*

The proof of Claim 1 can be found in [10]. We conclude that $\mathcal{D}_{(\mathbf{P},\mathbf{Q})} \approx k \cdot \mathcal{D}_{(\mathbf{P},\mathbf{Q})}$ is not acceptable for $k \neq \pm 1$. Hence, we add the determinant of the matrix to the side information function. Equivalently, we may demand that $\mathcal{A}$ contains only matrices with the same absolute value of the determinant.

5.4.2. Submultiplicative Norms

Another way to distinguish the distributions of $\mathbf{PA}_1\mathbf{Q}$ and $\mathbf{PA}_2\mathbf{Q}$ is computing any of their submultiplicative properties (a property p is called *submultiplicative* if $p(\mathbf{AB}) \leq p(\mathbf{A})p(\mathbf{B})$). For example, any submultiplicative matrix norm satisfies the inequality $\|\mathbf{PAQ}\| \leq \|\mathbf{P}\| \|\mathbf{A}\| \|\mathbf{Q}\|$. On the other hand, as $\mathbf{P}$ and $\mathbf{Q}$ are invertible, due to the same inequality, we have $\|\mathbf{P}^{-1}\|^{-1} \|\mathbf{A}\| \|\mathbf{Q}^{-1}\|^{-1} = \|\mathbf{P}^{-1}\|^{-1} \|\mathbf{P}^{-1}\mathbf{PAQQ}^{-1}\| \|\mathbf{Q}^{-1}\|^{-1} \leq \|\mathbf{PAQ}\|$. Hence, $\|\mathbf{PAQ}\|$ is bounded from below by $\|\mathbf{A}\| \cdot m$, and from above by $\|\mathbf{A}\| \cdot M$, where $m = \min_{\mathbf{P},\mathbf{Q}} (\|\mathbf{P}^{-1}\|^{-1} \|\mathbf{Q}^{-1}\|^{-1})$ and $M = \max_{\mathbf{P},\mathbf{Q}} (\|\mathbf{P}\| \|\mathbf{Q}\|)$ depend on $\mathbf{P}$ and $\mathbf{Q}$ only, but not on $\mathbf{A}$. However, different bounds do not imply different distributions, and requiring that all the submultiplicative norms of matrices of $\mathcal{A}$ be the same would be an excessively strong requirement that does not cover all the possible cases. We do not put any additional constraints on the norms of $\mathcal{A}$.

5.4.3. Hiding the Identity Matrix

Suppose that we have come up with a distribution $\mathcal{D}_{(\mathbf{P},\mathbf{Q})}$ that hides $\mathcal{A}$. Define a new distribution $\widetilde{\mathcal{D}}_{(\mathbf{P},\mathbf{Q})} = \{|(\mathbf{PA},\mathbf{Q}) \mid (\mathbf{P},\mathbf{Q}) \leftarrow \mathcal{D}_{(\mathbf{P},\mathbf{Q})}|\}$ for some $\mathbf{A} \in \mathcal{A}$. Now $\widetilde{\mathcal{D}}_{(\mathbf{P},\mathbf{Q})}$ hides the set $\mathbf{A}^{-1}\mathcal{A}$ as well as $\mathcal{D}_{(\mathbf{P},\mathbf{Q})}$ hides $\mathcal{A}$. Note that $\mathbf{I} \in \mathbf{A}^{-1}\mathcal{A}$. Hence, without loss of generality we are looking for (im)possibility results for $\mathcal{A}$ so that $\mathbf{I} \in \mathcal{A}$.

5.4.4. Summary

We have shown that the following additional assumptions on $\mathcal{A}$ are either required, or do not lessen generality:

- $\forall \mathbf{A}_1, \mathbf{A}_2 \in \mathcal{A} : |\mathbf{A}_1| = \pm|\mathbf{A}_2|$;
- $\mathbf{I} \in \mathcal{A}$ (and hence, $\forall \mathbf{A} \in \mathcal{A} : |\mathbf{A}| = \pm 1$).

In particular, we define the side information of outsourcing the inverse of $\mathbf{A}$ as $\Phi(\mathbf{A}) = \text{abs}(|\mathbf{A}|)$. The conditions $\mathbf{I} \in \mathcal{A}$ and $|\mathbf{A}| = \pm 1$ are assumed just without loss of generality, and they do not imply the impossibility of hiding $\mathcal{A}$ of some other fixed determinant, or so that $\mathbf{I} \notin \mathcal{A}$.

5.5. Perfect Secrecy

In this section we look for (im)possibilities of finding a perfectly hiding transformation for $\mathcal{A}$. As our transformation results in sampling a random element from some set, and then applying it to $\mathbf{A} \in \mathcal{A}$, we define more precisely what it means for a distribution (or a set) hide $\mathcal{A}$ perfectly.

Definition 5. *A distribution $\mathcal{D}_{\mathcal{S}}$ on a set $\mathcal{S}$ perfectly hides $\mathcal{A}$ if there is an operation $\oplus : \mathcal{S} \times \mathcal{A} \mapsto \{0,1\}^*$ so that $\mathcal{D}_{\mathcal{S} \oplus \mathbf{A}_i} = \mathcal{D}_{\mathcal{S} \oplus \mathbf{A}_j}$ for all $\mathbf{A}_i, \mathbf{A}_j \in \mathcal{A}$, where $\mathcal{D}_{\mathcal{S} \oplus \mathbf{A}} = \{|S \oplus \mathbf{A} \mid S \leftarrow \mathcal{D}_{\mathcal{S}}|\}$.*

Definition 6. *A set $\mathcal{S}$ perfectly hides $\mathcal{A}$ if there is a distribution $\mathcal{D}_{\mathcal{S}}$ on $\mathcal{S}$ that perfectly hides $\mathcal{A}$.*

5.5.1. Some Preliminary Results from Group Theory

Let $GL(n,\mathbb{R})$ denote the group of all invertible $(n \times n)$ matrices over $\mathbb{R}$. Let $O(n,\mathbb{R})$ denote the subgroup of $GL(n,\mathbb{R})$ that contains all the orthogonal matrices ($\mathbf{U}$ so that $\mathbf{UU}^T = \mathbf{I}$).

Theorem 2. *[26, Exercise 2.15]. Any finite subgroup of $GL(n,\mathbb{R})$ is conjugate to $O(n,\mathbb{R})$, i.e for any finite subgroup $\mathcal{G}$ of $GL(n,\mathbb{R})$ there is $\mathbf{C} \in GL(n,\mathbb{R})$ and a subgroup $\mathcal{U}$ of $O(n,\mathbb{R})$ so that $\mathcal{G} = \mathbf{C}\mathcal{U}\mathbf{C}^{-1}$.*

This is not the most general version of this theorem. A notion of *amenability* defines the groups on which it is possible to define a probability measure that is invariant under multiplication.

Definition 7. *[27] A group G is* amenable *if it admits a right-invariant, finitely additive probability measure μ: $\forall \mathcal{R} \subseteq \mathcal{G}, \forall A \in \mathcal{G}, \mu(\mathcal{R}A) = \mu(\mathcal{R})$.*

Now we show that Thm. 2 can be extended to amenable groups.

Definition 8. *[28] Let $\mathbb{B}(\mathcal{H})$ denote the space of bounded linear operators on some Hilbert space $\mathcal{H}$. A representation $\pi : \mathcal{G} \to \mathbb{B}(\mathcal{H})$ is called* uniformly bounded *if $\sup_{G \in \mathcal{G}}(\|\pi(G)\|_2) < \infty$.*

Theorem 3. *[28] Every uniformly bounded representation $\pi : \mathcal{G} \to \mathbb{B}(\mathcal{H})$ of an amenable group $\mathcal{G}$ is* unitarizable, *i.e there is $S \in \mathbb{B}(\mathcal{H})$ so that $G \mapsto S \circ \pi(G) \circ S^{-1}$ is a unitary operator.*

Corollary 1. *Any amenable subgroup of $GL(n, \mathbb{R})$ with a bounded ℓ_2 operator norm is conjugate to $O(n, \mathbb{R})$ and is hence easily invertible.*

Proof. We can take π to be an identity mapping $GL(n, \mathbb{R}) \mapsto GL(n, \mathbb{R})$, as matrices can be treated as linear operators on vectors over $\mathbb{R}$. Any subgroup $\mathcal{G}$ of $GL(n, \mathbb{R})$ is its own representation in $GL(n, \mathbb{R})$, where a bounded ℓ_2 operator norm implies that it is uniformly bounded.

Note that the elements of a group $\mathcal{G}$ that is conjugate to $O(n, \mathbb{R})$ are easily invertible. Any $\mathbf{A} \in \mathcal{G}$ is of the form $\mathbf{C}^{-1}\mathbf{U}\mathbf{C}$ for an orthogonal matrix $\mathbf{U}$. As $\mathbf{C}$ is the same for the entire group, the matrix $\mathbf{C}^{-1}$ has to be found only once for the entire group, and we can compute $\mathbf{A}^{-1} = \mathbf{C}^{-1}(\mathbf{C}\mathbf{A}\mathbf{C}^{-1})^T\mathbf{C}$. $\qquad\square$

5.5.2. Security of $\mathbf{B} = \mathbf{PA}$

First of all, we discuss the transformation $\mathbf{B} = \mathbf{PA}$, where the matrix $\mathbf{A} \in \mathcal{A}$ is only multiplied from the left with a randomly generated matrix $\mathbf{P}$, as in the case of finite fields it is sufficient for perfect security.

Suppose that there is a distribution $\mathcal{D}_{\mathbf{P}}$ of matrices $\mathbf{P}$ that perfectly hides $\mathcal{A}$. Let $\mathbf{A}_i \in \mathcal{A}$. As $\mathbf{I} \in \mathcal{A}$, there exist $\mathbf{P}_i, \mathbf{P}_j \in \mathcal{P} = supp(\mathcal{D}_{\mathbf{P}})$ so that $\mathbf{P}_i\mathbf{A}_i = \mathbf{P}_j\mathbf{I} = \mathbf{P}_j$, or $\mathbf{P}_j^{-1}\mathbf{P}_i = \mathbf{A}_i$. Let $\mathcal{G} = \langle \mathcal{A} \rangle$ be the subgroup of $GL(n, \mathbb{R})$ generated by $\mathcal{A}$. As $\mathbf{P}_j^{-1}\mathbf{P}_i \in \mathcal{G}$, $\mathbf{P}_i$ and $\mathbf{P}_j$ belong to the same left coset $\mathbf{H}\mathcal{G}$ of $\mathcal{G}$ for some $\mathbf{H} \in GL(n, \mathbb{R})$. The actual value of $\mathbf{H}$ does not even matter, as changing $\mathbf{H}_1$ to $\mathbf{H}_2$ just multiplies $\mathcal{D}_{\mathbf{P}}$ by the same group element $\mathbf{H}_2\mathbf{H}_1^{-1}$ from the left.

We now study the properties of $\mathcal{G} = \langle \mathcal{A} \rangle$ for a set $\mathcal{A}$ for which there is a perfectly hiding distribution $\mathcal{D}_{\mathbf{P}}$ over $\mathbf{H}\mathcal{G}$. We show that there exists a distribution $\mathcal{D}_{\mathcal{G}}$ such that $supp(\mathcal{D}_{\mathcal{G}}) = \mathcal{G}$ that hides $\mathcal{A}$ as well.

Claim 2. *There exists a distribution $\mathcal{D}_{\mathcal{G}}$ on $\mathcal{G}$ so that $\mathcal{D}_{\mathcal{G}}\mathbf{A} = \mathcal{D}_{\mathcal{G}}$ for any $\mathbf{A} \in \mathcal{A}$.*

Proof. Let $\mathcal{D}_{\mathcal{F}} = \{|\mathbf{P}_j^{-1}\mathbf{P}_i \mid \mathbf{P}_i \leftarrow \mathcal{D}_{\mathbf{P}}, \mathbf{P}_j \leftarrow \mathcal{D}_{\mathbf{P}}|\}$, and $\mathcal{F} = supp(\mathcal{D}_{\mathcal{F}})$. $\mathcal{D}_{\mathcal{F}}$ is an example of a distribution over $\mathcal{F}$ that is hiding if $\mathcal{D}_{\mathbf{P}}$ is hiding, as further multiplication by an independently sampled $\mathbf{P}_j^{-1}$ cannot harm the hiding achieved by the multiplication with $\mathbf{P}_i \leftarrow \mathcal{D}_{\mathbf{P}}$. Multiplying $\mathcal{D}_{\mathcal{F}}$ by *any* distribution over $\mathcal{G}$ from the left gives a perfectly hiding distribution $\mathcal{D}_{\mathcal{G}}$. The definition of perfect hiding and $\mathbf{I} \in \mathcal{A}$ give $\mathcal{D}_{\mathcal{G}}\mathbf{A} = \mathcal{D}_{\mathcal{G}}\mathbf{I} = \mathcal{D}_{\mathcal{G}}$ for any $\mathbf{A} \in \mathcal{A}$. $\qquad\square$

The claim proves the existence of *some* hiding $\mathcal{D}_{\mathcal{G}}$ so that $supp(\mathcal{D}_{\mathcal{G}}) = \mathcal{G}$. However, choosing such $\mathcal{D}_{\mathcal{G}}$ for hiding is sufficient, but not necessary (we may actually sample from an arbitrary coset of $\mathcal{G}$ that is not a group). We now state a necessary condition for $supp(\mathcal{D}_{\mathbf{P}})$.

Claim 3. *If $supp(\mathcal{D}_\mathbf{P})$ contains any element of a coset $\mathbf{H}\mathcal{G}$ of $\mathcal{G}$, then it contains that coset entirely.*

Proof: We prove the claim by induction on the length of the generating sequence of an element of $\mathcal{G}$.

- **Base:** We have shown that all $\mathbf{P}_i \in supp(\mathcal{D}_\mathbf{P})$ belong to the same coset $\mathbf{H}\mathcal{G}$. Taking $\mathbf{H}' := \mathbf{P}_i$ for any $\mathbf{P}_i \in supp(\mathcal{D}_\mathbf{P})$, we get $\mathbf{H}\mathcal{G} = \mathbf{H}'\mathcal{G}$, so for the base we may take *any* $\mathbf{P}_i \in supp(\mathcal{D}_\mathbf{P})$.
- **Step:** Let $\mathbf{G} \in \mathcal{G}$, then $\mathbf{G} = \mathbf{A}_1 \ldots \mathbf{A}_n$ for some $\mathbf{A}_1, \ldots, \mathbf{A}_n \in \mathcal{A}$ (the representation does not have to be unique). Consider the quantity $\mathbf{P}_i \mathbf{A}_1 \ldots \mathbf{A}_{n-1} \mathbf{A}_n$. By induction hypothesis, $\mathbf{P}_i \mathbf{A}_1 \ldots \mathbf{A}_{n-1} \in supp(\mathcal{D}_\mathbf{P})$. As $\mathcal{D}_\mathbf{P}$ is perfectly hiding, there exists a matrix from $supp(\mathcal{D}_\mathbf{P})$ that makes $\mathbf{I}$ indistinguishable from $\mathbf{A}_n$, and after $\mathbf{P}_i \mathbf{A}_1 \ldots \mathbf{A}_{n-1}$ is fixed, it is uniquely defined due to invertibility. At the same time, $\mathbf{P}_i \mathbf{A}_1 \ldots \mathbf{A}_n$ is such a matrix. Hence, $\mathbf{P}_i \mathbf{A}_1 \ldots \mathbf{A}_n \in supp(\mathcal{D}_\mathbf{P})$. □

As $\mathcal{D}_\mathcal{G}$ is perfectly hiding, and $\mathbf{I} \in \mathcal{A}$, we have $\mathcal{D}_\mathcal{G}\mathbf{A}_i = \mathcal{D}_\mathcal{G}$ for all $\mathbf{A}_i \in \mathcal{A}$. This can be extended to any $\mathbf{A} \in \mathcal{G}$.

Claim 4. $\mathcal{D}_\mathcal{G}\mathbf{A} = \mathcal{D}_\mathcal{G}$ *for all* $\mathbf{A} \in \mathcal{G}$.

Proof. For all $\mathbf{A} \in \mathcal{G}$, we have $\mathbf{A} = \mathbf{A}_1 \cdot \ldots \cdot \mathbf{A}_k$, where $\mathbf{A}_i \in \mathcal{A}$ are generators of $\mathcal{G}$. Applying $\mathcal{D}_\mathcal{G}\mathbf{A}_i = \mathcal{D}_\mathcal{G}$ k times, we get $\mathcal{D}_\mathcal{G}\mathbf{A} = \mathcal{D}_\mathcal{G}\mathbf{A}_1 \ldots \mathbf{A}_k = \mathcal{D}_\mathcal{G}\mathbf{A}_2 \ldots \mathbf{A}_k = \ldots = \mathcal{D}_\mathcal{G}$. □

We are interested in the distribution $\mathcal{D}_\mathcal{G}\mathbf{A} = \mathcal{D}_\mathcal{G}$ for $\mathbf{A} \in \mathcal{G}$. If $\mathcal{G}$ is a finite group, then $\mathcal{D}_\mathcal{G}$ must be a uniform distribution on $\mathcal{G}$, as in a group each product is a distinct element. By Thm. 2, a finite group is conjugate to $O(n, \mathbb{R})$. We are more interested in the cases where $\mathcal{G}$ is infinite.

Claim 5. *Let* $\mathcal{D}_\mathcal{G}\mathbf{A} = \mathcal{D}_\mathcal{G}$ *for all* $\mathbf{A} \in \mathcal{G}$. *Then* $\mathcal{G}$ *is amenable.*

Proof. We take the probability measure $\mu_{\mathcal{D}_\mathcal{G}}$ of $\mathcal{D}_\mathcal{G}$. Take any $\mathcal{R} \subseteq \mathcal{G}$. Then $\forall \mathbf{A} \in \mathcal{G}$: $\mu_{\mathcal{D}_\mathcal{G}}(\mathcal{R}) = \mu_{\mathcal{D}_\mathcal{G}\mathbf{A}}(\mathcal{R}) = \mu_{\mathcal{D}_\mathcal{G}}(\mathcal{R}\mathbf{A}^{-1})$, which is equivalent to $\forall \mathbf{A} \in \mathcal{G}$: $\mu_{\mathcal{D}_\mathcal{G}}(\mathcal{R}) = \mu_{\mathcal{D}_\mathcal{G}}(\mathcal{R}\mathbf{A})$. The measure $\mu_{\mathcal{D}_\mathcal{G}}$ is suitable for the amenability definition. □

The definition of amenability is not too interesting on its own, but it tells something more due to the ℓ_2-norm boundedness requirement.

Claim 6. *The elements of* $\mathcal{A}$ *are conjugate to* $O(n, \mathbb{R})$ *and are hence easily invertible.*

Proof. By Claim 3, we need to sample from a distribution $\mathcal{D}_\mathbf{P}$ so that $supp(\mathcal{D}_\mathbf{P}) = \mathbf{H}\mathcal{G}$. We have agreed in Sec. 5.4 that we deal with matrices $\mathbf{P}$ with bounded ℓ_2-norms, so the ℓ_2 norms of $\mathbf{H}\mathcal{G}$ should be bounded. Hence, the norms of $\mathcal{G}$ are also bounded: $\forall \mathbf{G} \in \mathcal{G}$: $\|\mathbf{G}\|_2 = \|\mathbf{H}^{-1}\mathbf{H}\mathbf{G}\|_2 \leq \|\mathbf{H}^{-1}\|_2 \|\mathbf{H}\mathbf{G}\|_2$.

By Claim 5, $\mathcal{G}$ is amenable. By Cor. 1, the elements of $\mathcal{G}$ are conjugate to $O(n, \mathbb{R})$ and are easily invertible. We have $\mathcal{A} \subseteq \mathcal{G}$. □

We conclude that it is indeed possible to find a perfect hiding of the form $\mathbf{B} = \mathbf{PA}$ for certain sets $\mathcal{A}$ for which the group $\langle \mathcal{A} \rangle$ is amenable. But knowing that $\mathbf{A} \in \mathcal{A}$ would already give enough information about the inverse of $\mathbf{A}^{-1}$ so that it could be computed easily without the transformation.

5.5.3. Security of $\mathbf{B} = \mathbf{PAQ}$

From Claim 6, we conclude that the transformation $\mathbf{B} = \mathbf{PA}$ is not powerful enough for perfect secrecy. We proceed with $\mathbf{B} = \mathbf{PAQ}$. First, we give a small example that demonstrates additional hiding that the transformation $\mathbf{B} = \mathbf{PAQ}$ can give compared to just $\mathbf{B} = \mathbf{PA}$.

Example 4. *Let* $\mathbf{A}_1 = \begin{pmatrix} \lambda & 0 \\ 0 & 1/\lambda \end{pmatrix}$ *and* $\mathbf{A}_2 = \mathbf{I}$. *Let* $(\mathbf{P}_1, \mathbf{Q}_1) = \left(\begin{pmatrix} 1/\lambda & 0 \\ 0 & 1 \end{pmatrix}, \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} \right)$ *and*

$(\mathbf{P}_2, \mathbf{Q}_2) = \left(\begin{pmatrix} 0 & 1 \\ 1/\lambda & 0 \end{pmatrix}, \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} \right)$. *Let* $\mathcal{D}_{(\mathbf{P},\mathbf{Q})}$ *be a distribution over pairs of matrices that uniformly chooses between* $(\mathbf{P}_1, \mathbf{Q}_1)$ *and* $(\mathbf{P}_2, \mathbf{Q}_2)$. *Then* $\mathcal{D}_{(\mathbf{P},\mathbf{Q})}$ *perfectly hides* $\{\mathbf{A}_1, \mathbf{A}_2\}$. *This would be impossible to achieve with the* $\mathbf{B} = \mathbf{PA}$ *transformation, as* $\langle \{\mathbf{A}_1, \mathbf{A}_2\} \rangle$ *is isomorphic to* $\mathbb{Z}$ *and does not accept a well-defined uniform distribution* $\mathcal{D}$ *that satisfies* $\mathcal{D}\mathbf{A} = \mathcal{D}$.

5.5.4. Formalizing $\mathbf{B} = \mathbf{PAQ}$

Suppose that there exists a perfectly hiding distribution $\mathcal{D}_{(\mathbf{P},\mathbf{Q})}$ of $\mathbf{P}$ and $\mathbf{Q}$ for $\mathcal{A}$. Similarly to the $\mathbf{B} = \mathbf{PA}$ case, we may rewrite $\mathbf{P}_i \mathbf{A}_k \mathbf{Q}_i = \mathbf{P}_j \mathbf{A}_\ell \mathbf{Q}_j$ as $\mathbf{P}_j^{-1} \mathbf{P}_i \mathbf{A}_k \mathbf{Q}_i \mathbf{Q}_j^{-1} = \mathbf{A}_\ell$. If multiplying by $(\mathbf{P}_i, \mathbf{Q}_i)$ once provides sufficient hiding, then multiplying the result again with an independently sampled $(\mathbf{P}_j^{-1}, \mathbf{Q}_j^{-1})$ is hiding as well. Let $\mathcal{D}_\mathcal{F} = \{|(\mathbf{P}_j^{-1}\mathbf{P}_i, \mathbf{Q}_i \mathbf{Q}_j^{-1}) \mid (\mathbf{P}_i, \mathbf{Q}_i), (\mathbf{P}_j, \mathbf{Q}_j) \leftarrow \mathcal{D}_{(\mathbf{P},\mathbf{Q})}|\}$ and $\mathcal{F} = supp(\mathcal{D}_\mathcal{F})$. Define a group $\mathcal{G} := \langle \mathcal{F} \rangle$, where the inverse is defined as $(\mathbf{P}, \mathbf{Q})^{-1} = (\mathbf{P}^{-1}, \mathbf{Q}^{-1})$, and the multiplication as $(\mathbf{P}_1, \mathbf{Q}_1) * (\mathbf{P}_2, \mathbf{Q}_2) = (\mathbf{P}_1 \mathbf{P}_2, \mathbf{Q}_2 \mathbf{Q}_1)$. We may now consider the hiding as an *action* of the group $\mathcal{G}$ on the set $\mathcal{A}$, defined as $(\mathbf{P}, \mathbf{Q}).\mathbf{A} = \mathbf{PAQ}$ for $\mathbf{A} \in \mathcal{A}$, $(\mathbf{P}, \mathbf{Q}) \in \mathcal{G}$. Define the following sets: $\mathcal{X}_0 := \mathcal{A}$, $\mathcal{X}_i := \mathcal{G}.\mathcal{X}_{i-1}$, $\mathcal{X} := \bigcup_{i=0}^{\infty} \mathcal{X}_i$. By construction, $\mathcal{A} \subseteq \mathcal{X}$.

5.5.5. (Im)possibility Results for $\mathbf{B} = \mathbf{PAQ}$

In all the claims proven it this Sec. 5.5, we assume that there exists a distribution $\mathcal{D}_{(\mathbf{P},\mathbf{Q})}$ that hides $\mathcal{A}$ perfectly. We study the action of $\mathcal{G}$ on $\mathcal{X}$ and find some of its interesting properties.

Claim 7. *Let* $\mathcal{G}$ *and* $\mathcal{X}$ *be defined as in Sec. 5.5.4. The set* $\mathcal{X}$ *is isomorphic (as an action of the group* $\mathcal{G}$*) to the set of cosets of* $\mathcal{H}_{\mathbf{X}_0} := \{G | G \in \mathcal{G}, G.\mathbf{X}_0 = \mathbf{X}_0\}$ *(for any* $\mathbf{X}_0 \in \mathcal{X}$*). The isomorphism is* $G\mathcal{H}_{\mathbf{X}_0} \leftrightarrow G.\mathbf{X}_0$.

The proof of Claim 7, based on known results from group theory, can be found in [10]. The next claim is similar to Claim 2, which states that $\mathcal{G}$ itself perfectly hides $\mathcal{A}$.

Claim 8. *Let* $\mathcal{G}$ *be defined as in Sec. 5.5.4. There exists a distribution* $\mathcal{D}_\mathcal{G}$ *on* $\mathcal{G}$*, so that* $\mathcal{D}_{\mathcal{G}.\mathbf{A}_i} = \mathcal{D}_{\mathcal{G}.\mathbf{A}_j}$ *for all* $\mathbf{A}_i, \mathbf{A}_j \in \mathcal{A}$*, where* $\mathcal{D}_{\mathcal{G}.\mathbf{A}} := \{|G.\mathbf{A} \mid G \leftarrow \mathcal{D}_\mathcal{G}|\}$.

The proof of Claim 8 is analogous to the proof of Claim 2. It can be found in [10]. In the case of $\mathbf{B} = \mathbf{PA}$, we observed the properties of the group $\langle \mathcal{A} \rangle$. It would be interesting to find something similar in $\mathbf{B} = \mathbf{PAQ}$. For each $\mathbf{A}_i \in \mathcal{A}$ there exists $(\mathbf{P}_i, \mathbf{Q}_i) \in supp(\mathcal{D}_{(\mathbf{P},\mathbf{Q})})$ so that $\mathbf{A}_i = \mathbf{P}_i \mathbf{I} \mathbf{Q}_i$.

Claim 9. *It is possible to sample uniformly from* $\mathcal{A}$.

The proof is based on the isomorphism between $\mathcal{X}$ and $\mathcal{G}$ from Claim 7, and the fact that $\mathcal{G}$ hides $\mathcal{A}$ perfectly (Claim 8).

Similarly to the $\mathbf{B} = \mathbf{PA}$ case, being able to sample uniformly from $\mathcal{G}$ would be sufficient for hiding. Due to the isomorphism with the cosets of some $\mathcal{H}_{\mathbf{A}_0}$, each $\mathbf{A} \in \mathcal{A}$ can be written out as $\mathbf{A} = G_i.\mathbf{A}_0$. Given a uniform distribution $\mathcal{D}_{\mathcal{G}}$ on $\mathcal{G}$, we have $\mathcal{D}_{\mathcal{G}}(G_i.\mathbf{A}_0) = (\mathcal{D}_{\mathcal{G}}G_i).\mathbf{A}_0 = \mathcal{D}_{\mathcal{G}}.\mathbf{A}_0$. This hides the entire $\mathcal{X}$. However, it may happen that hiding the entire $\mathcal{X}$ is a too strong of a requirement. If we could sample uniformly from $\mathcal{G}$, then we could also sample from $\mathcal{X}$, as group elements are distributed uniformly amongst its cosets. The question is whether we can always sample uniformly from $\mathcal{X}$.

A simple counterexample is the construction of Example 4. It is easy to sample uniformly from $\mathcal{A}$, which is a finite set. We have $\mathbf{A}_2 = \mathbf{P}_2^{-1}\mathbf{P}_1\mathbf{A}_1\mathbf{Q}_1\mathbf{Q}_2$. Defining the group $\mathcal{G} = \langle(\mathbf{P}_2^{-1}\mathbf{P}_1, \mathbf{Q}_1\mathbf{Q}_2^{-1})\rangle$ as before, we get $\mathcal{G} = \left\langle \left(\begin{pmatrix} 0 & \lambda \\ 1/\lambda & 0 \end{pmatrix}, \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} \right) \right\rangle$, where the first component gets infinite powers of λ in $\mathcal{G}$, and multiplying it by the second component, which is either $\mathbf{I}$ or $\begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}$, does not help to decrease these powers.

As the result, we cannot conclude that $\mathcal{G}$ is necessarily amenable in general, as it was in the case of $\mathbf{B} = \mathbf{PA}$.

5.6. Statistical Hiding

The previous results are partially extensible to statistical hiding. Instead of choosing a distribution $\mathcal{D}$ that perfectly hides $\mathcal{A}$ we could remove from its support a part with negligible weight, and renormalize the rest. However, if we had negligible statistical distance instead of the equality of distributions in Claim 4, then the statistical distance between $\mathcal{D}_{\mathcal{G}}$ and $\mathcal{D}_{\mathcal{G}}\mathbf{A}$ might grow with k and hence, this argument does not pass straightforwardly for statistical security. Claim 3 also does not pass: it no longer makes sense to require that the ℓ_2-norms of matrices of $\mathcal{D}$ are bounded, as we do not sample all its elements anyway. Hence, the preconditions of Claims 5 and 6 are not satisfied, and we cannot say that if there is statistical hiding for $\mathcal{A}$, then there is an easier way to invert $\mathcal{A}$. Indeed, it turns out that there are more possibility results for statistical hiding, which we present in this section.

5.6.1. Hiding Triangular Matrices by $\mathbf{B} = \mathbf{PA}$

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 $\mathbf{P} = (\mathbf{p}_1 | \ldots | \mathbf{p}_n)$. Due to the triangularity of $\mathbf{A}$, we can write $\mathbf{PA} = (a_{11}\mathbf{p}_1 + \ldots + a_{n1}\mathbf{p}_n | \ldots | a_{n-1,n-1}\mathbf{p}_{n-1} + a_{n,n-1}\mathbf{p}_n, a_{nn}\mathbf{p}_n)$. As $a_{nn} = 1$, the last vector of $\mathbf{PA}$ does not depend on $\mathbf{A}$.

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

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;

it is sufficient to sample the matrix $\mathbf{P}$ according to a distribution described in the previous paragraph. The sizes of the entries of $\mathbf{P}$ are bounded by $(\log c)n\eta$. These entries, although not small, are still of acceptable size according to our discussion in Sec. 5.4. On the other hand, lower triangular matrices can be inverted much more easily than general matrices, and it does not make any practical sense to outsource it like that.

5.6.2. Hiding Based on Matrix Decomposition

Again, let $\mathcal{A}$ be a set of $n \times n$ matrices with bounded entries, and determinant ± 1. Almost any matrix $\mathbf{A}$ can be decomposed as $\mathbf{A} = \mathbf{L_A U_A}$, where $\mathbf{L_A}$ is a lower and $\mathbf{R_A}$ an upper triangular matrix [29, Chap. 3]. If the set $\mathcal{A}$ satisfies the condition that there must be $c \in \mathbb{R}$, so that each $\mathbf{A} \in \mathcal{A}$ has an LU decomposition, where

- the absolute values of the entries in $\mathbf{L_A}$ and $\mathbf{U_A}$ are upper-bounded by c;
- the entries on the main diagonal of $\mathbf{L_A}$ and $\mathbf{U_A}$ are ± 1;

then the following distribution $\mathcal{D}_{(\mathbf{P,Q})}$ provides statistical hiding for $\mathcal{A}$. Let $\mathcal{D}_{\mathbf{P}}$ be defined as in Sec. 5.6.1. Let $\mathcal{D}_{(\mathbf{P,Q})} = \{|(\mathbf{P}_1, \mathbf{P}_2^T) \mid \mathbf{P}_1, \mathbf{P}_2 \leftarrow \mathcal{D}_{\mathbf{P}}|\}$. Indeed, $\mathbf{P}_1 \mathbf{L_A}$ statistically hides $\mathbf{L_A}$ and $\mathbf{U_A} \mathbf{P}_2^T$ statistically hides $\mathbf{U_A}$ for any $\mathbf{A} \in \mathcal{A}$.

More decomposition-based hiding distributions can be found in [10].

5.7. Summary

We have reduced outsourcing a linear equation system $\mathbf{Ax} = \mathbf{b}$ to outsourcing just the inverse of $\mathbf{A}$. We have shown that the most general type of affine transformation for outsourcing the matrix inverse is of the form $\mathbf{A} \mapsto \mathbf{P} \begin{pmatrix} \mathbf{A} & \mathbf{0} \\ \mathbf{0} & \mathbf{I} \end{pmatrix} \mathbf{Q}$. Only matrices with the same determinant absolute values can be perfectly indistinguishable. If we want to achieve statistical indistinguishability for arbitrary matrices $\mathbf{A}_1$ and $\mathbf{A}_2$, the entries of $\mathbf{P}$ and $\mathbf{Q}$ grow in $\Omega(||\mathbf{A}_1||/||\mathbf{A}_2||^{2^\eta})$ for a security parameter η.

We have found that over reals, it is much more difficult to hide matrices by multiplying them with random matrices. If we try to limit hiding to multiplication by one matrix $\mathbf{P}$, which works well in matrices over finite fields, we get that there exists an easier way of inverting elements $\mathcal{A}$ that does not require outsourcing. We do not have as good (im)possibility results for multiplication by $\mathbf{P}$ from the left and $\mathbf{Q}$ from the right.

We can still achieve statistical security. We have given some possibility results that nevertheless leak the determinant, unless the entries grow in $\Omega(c^{2^\eta})$ where c is the maximal ratio of different determinants.

6. Conclusion

We have shown that using a transformation-based approach for privacy-preserving tasks defined over real numbers is complicated. Satisfying standard formal security definitions is difficult or even impossible. We have shown that, at least for linear programming, there is no information-theoretically secure affine transformation, and it is quite unlikely that there is a computational one. Information-theoretical security is not achievable in general even for $n \times n$ linear equation systems. It may still be possible to get more possibility results for outsourcing matrix inversion based on some computational assumptions (although some matrix properties such as the determinant will be leaked nevertheless). However, cryptography over real numbers has not received much attention so far. Extending finite field assumptions to real numbers is not possible in general.

References

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

[2]   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.

[3]   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.

[4]   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.

[5]   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.

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

[7]   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.

[8]   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.

[9]   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.

[10] 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/.

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

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

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

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

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

[16] 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.

[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] 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.

[19] 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.

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

[21] 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.

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

[23] 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.

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

[25] W. Hundsdorfer and J.G. Verwer. *Numerical Solution of Time-Dependent Advection-Diffusion-Reaction Equations*. Springer Series in Computational Mathematics. Springer, 2003.

[26] A. Szczepański. *Geometry of Crystallographic Groups*. Algebra and discrete mathematics. World Scientific, 2012.

[27] Tullio Ceccherini-Silberstein and Michel Coornaert. Amenable groups. In *Cellular Automata and Groups*, Springer Monographs in Mathematics, pages 77–110. Springer Berlin Heidelberg, 2010.

[28] N. Monod and N. Ozawa. The dixmier problem, lamplighters and burnside groups. *ArXiv e-prints*, February 2009.

[29] R.A. Horn and C.R. Johnson. *Matrix Analysis*. Cambridge University Press, 2012.

Chapter 12
Practical Applications of Secure Multiparty Computation

Riivo TALVISTE [a]

[a] *Cybernetica AS, Estonia*

Abstract. As secure multiparty computation technology becomes more mature, we see more and more practical applications where computation moves from a lab setting to an actual deployment where each computation node is hosted by a different party. In this chapter, we present some of the practical SMC applications that have worked on real data. A few of these have been developed by Cybernetica AS, while others have been found from public sources. All scientific prototypes or simulations that work on generated or public data have been left out of this chapter. For each chosen SMC application, we give a brief overview and refer the reader to the corresponding published sources for more details.

1. Danish Sugar Beet Auction

1.1. Overview

Secure multiparty computation was first used in a large-scale practical application in Denmark in 2008 when it was used to reallocate sugar beet production contracts between farmers and a sugar production company [1]. SMC technology was used to carry out a double auction that would find the optimal quantities and price for both the farmers and the sugar production company Danisco.

In a double auction, buyers specify the quantity of goods they are willing to buy at each price for a number of different potential prices. Similarly, for each price, sellers give the quantity of goods they are willing to sell. From these bids, an auctioneer computes the *market clearing price* where total demand equals total supply. In this application, the clearing price was computed using secure multiparty computation. This allowed to keep the' production capabilities of individual farmers secret so that the information could not be misused.

1.2. Scheme and Roles

The application was based on 3-party Shamir's secret sharing scheme over a field $\mathbb{Z}_p$, where p is a 64-bit prime number. It was designed for the semi-honest security setting. The three independent computation parties were represented by the sugar production company Danisco, the representative of the farmers (DKS, the sugar beet growers' association) and the SMC technology provider (SIMAP project).

1.3. Process

The whole process was divided into two phases: bidding and tallying. The latter involves secure multiparty computation.

As there were more than a thousand input parties, it was important to make the bidding application easily accessible to the farmers. This was accomplished by distributing the input application as a Java applet accessible from a web page. Each of the $1,229$ bidders had the option of submitting a bid for selling, buying or both for $4,000$ different prices. It was decided that the three computation parties would not be online during the bidding phase. Thus, instead of sending the input shares directly from the input application to the computation parties, they were stored in a central proxy database. To protect against the reconstruction of shares to the original input value at the proxy server, each individual share was encrypted with a public key of one of the computation parties. These public keys were bundled with the input application and the encryption was performed also in the input application so the original input value never left the application.

The second phase — secure multiparty computation — was carried out in a local setting. First, the representatives of the three computation parties downloaded the shares that were encrypted with their public key from the proxy database, and used their private key to decrypt them. Second, with their decrypted shares they initiated the secure multiparty computation protocol that used about 12 secure comparisons to calculate the market clearing price by binary search. The second phase was done over a 100 Mbit local network (LAN) and took about 30 minutes in total. However, most of it was spent on decrypting individual shares.

1.4. Conclusion

As a result of the application, about 25 tonnes-worth of production rights changed owners [1]. Since its first use in 2008, this application has been used again on subsequent years to reallocate the sugar beet growing contracts in Denmark [2].

2. Financial Data Analysis

2.1. Overview

In 2010, the Estonian Association of Information Technology and Telecommunications (officially abbreviated as ITL), a consortium of ICT companies, decided that it would start periodically collecting and publishing economic benchmarking data on its members to promote their business. The idea was to collect economic indicators such as total return, number of employees, percentage of export, labour costs, profit, and release statistics about these values to ITL member companies so they could quickly react to changes in the economic sector. As such indicators are confidential, it was decided that the data collection and computation would be done using SMC technology [3,4].

2.2. Scheme and Roles

The application was built on the SHAREMIND secure multiparty computation platform that used a three-party additive secret sharing scheme over 32-bit integers. The three

computing parties hosting the servers were ITL members: Zone Media, Microlink and Cybernetica. It must be noted that although the three computation servers were hosted by separate companies, due to the lack of resources, they were administered by the same person. This means that it was not an ideal deployment as this single administrator could potentially recombine the original input.

2.3. Process

As in the Danish secure auction application, the process was split into two phases: data collection and data analysis. An overview of the whole process is given in Fig. 1.
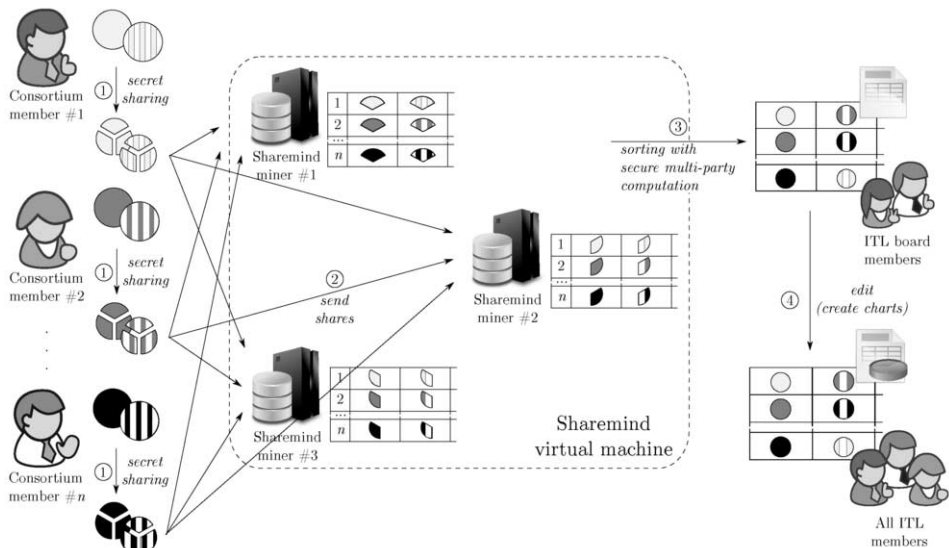
Figure 1. Data flow in the ITL application

Data was collected from ITL members using a a web-based form integrated with the member area of the official web page of ITL. Using the member area allowed ITL members to authenticate themselves to the application using the credentials they already had. In every secure multiparty application, the input values have to be secret shared at the source, in this case the user's web browser. However, instead of encrypting each share with a different public key and sending them all to the same storage server, as was the case with the Danish sugar beet auction, each share was sent directly to a different proxy server over a secure HTTPS channel. Using web-based forms instead of Java applets or other plug-ins is more accessible and transparent for the end user. The three proxy servers were hosted by the three computing parties.

The data analysis phase started after the first phase was completed and the input form was closed. First, each computing party started its SHAREMIND node and input shares were moved from the proxy to the corresponding secure multiparty computation server. For that, each of the computing parties ran a proxy application that loaded shares from the proxy server and contacted the other two proxy applications to coordinate the correct set intersection and order of input shares. After that, the SMC process was triggered and its results were made available to the ITL board, which reviewed the results and

published them to the rest of the ITL members. The secure multiparty computations performed in this applications were simple — the values for each collected economic indicator were sorted independently using the bubble sort algorithm so the ties between different indicators for one company were broken.

2.4. Conclusion

It was the first practical SMC application where computation was carried out over the public Internet. ITL used this application to collect economic health information from its members twice in 2011. After that, the ITL board did not feel the need for the enhanced privacy provided by the application and switched to a more open method of collecting financial data from its members.

3. Privacy-preserving Statistical Studies

3.1. Overview

In Estonia, there is a high drop-out rate among IT students. Universities believe that this is due to the fact that there is high demand for IT specialists on the market, and so a lot of the students are employed during their studies. Consequently, they cannot keep up with their studies and either postpone graduation or drop out altogether. The Estonian Association of Information Technology and Telecommunications (ITL) has put forth a question if and how much does working during one's studies affect graduation on time. The data to answer this question is already there — we could link students' educational records and tax records to see how one's income and work information affect the length of one's studies. However, linking such data poses a privacy risk as tax data is highly confidential. In the EU-funded project "Privacy-preserving statistical studies on linked databases"[1] (PRIST), this privacy risk was mitigated by linking and analyzing educational and tax records using secure multiparty computation technology.

3.2. Scheme and Roles

This study used a newer version of the SHAREMIND platform than the one used in the financial benchmarking application described in Sec. 2. Similarly to the financial data analysis application, it uses a three-party additive secret sharing scheme in the semi-honest security setting, but supports many different data types, including floating point numbers [5] required for implementig statistical analysis methods (see Chapter 4). The three computing parties were the Information Technology Center of the Ministry of Finance, the Estonian Information System Authority, and the technology provider Cyber-netica AS. The input data was collected from the Estonian Ministry of Education and Research that holds the education records, and the Estonian Tax and Customs Board that holds tax information. The statistical analysis was carried out by the Estonian Center for Applied Research (CentAR) acting as a result party.

---

[1]Privacy-preserving statistical studies on linked databases, funded by the European Regional Development Fund from the sub-measure of supporting the development of the R&D of information and communication technology through the Archimedes Foundation: `http://cyber.ee/en/research/research-projects/prist/`

3.3. Process

Each computing party deployed their SHAREMIND server instance on their premises and they were connected over the public Internet using mutually authenticated TLS connections.

In this deployment, there was no need for a widely accessible web interface as there were only two input parties. The input parties first exported a relevant data view from their database, and used a command-line application that secret-shared each individual value. The application also distributed the shares among the computing parties. In total, about 16 million tax records and 200,000 education records were imported into the system. After the data was imported, CentAR used a statistical analysis toolset RMIND [6] to process and query the data using secure multiparty computation.

3.4. Conclusion

The data was imported and the analysis conducted at the beginning of 2015. At the time of writing this book, the analyses were ongoing.

4. Energiauktion.dk — Automated Electricity Broker

Energiauktion.dk[2] is a practical SMC application for electricity procurement for small and medium sized companies in Denmark.

4.1. Deployment and Process

Energiauktion.dk provides electricity procurement as a secure software-as-a-service auction[3]. Thus, it works similarly to the sugar beet auction system described in Sec. 1. However, in the case of Energiauktion.dk, the computing parties are deployed in the Amazon cloud so the actual secure multiparty computation takes place over the public Internet and not over the local network. No more technical details are provided in public documentation.

5. Dyadic Security

5.1. Overview

All modern cryptographic operations are only as secure as the keys used in them. However, in many practical use cases, such as SSL/TLS, the secret key has to be kept online.

Dyadic [7] is a company that takes away this kind of single point of failure by secret sharing the secret key and distributing the shares among many parties. Thus, an attacker, even an insider, must attack all parties to get hold of the key. All operations that require this key (e.g. signing, creating an SSL/TLS connection or other type of encryption) are done using SMC between the parties holding the shares of the secret key. Similarly, this system can be used to check the correctness of the password on login so that the user password database itself is secret shared.

---

[2]Energiauktion.dk — `https://energiauktion.dk/`

[3]Auctions-as-a-Service — `http://www.partisia.dk/thepartisiaway/pages/aaas.aspx`

5.2. Technology

According to Dyadic's website [8], it uses both secure two-party (Yao garbled circuits) and secure multiparty (SPDZ) computation technologies, and the protocols are secure against an active adversary. Moreover, the system performs periodic resharing of the secrets so that an attacker has only a small timeframe to break into the databases of all the parties holding the shares.

References

[1] 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 *Financial Cryptography*, pages 325–343, 2009.

[2] Tomas Toft. The Danish Sugar Beet Auctions. Presented at Privacy Enhancing Cryptography (PEC'11), 2011.

[3] Riivo Talviste. Deploying secure multiparty computation for joint data analysis - a case study. Master's thesis, University of Tartu, 2011.

[4] Dan Bogdanov, Riivo Talviste, and Jan Willemson. Deploying secure multi-party computation for financial data analysis - (short paper). In Angelos D. Keromytis, editor, *Financial Cryptography*, volume 7397 of *Lecture Notes in Computer Science*, pages 57–64. Springer, 2012.

[5] Liina Kamm and Jan Willemson. Secure floating point arithmetic and private satellite collision analysis. *International Journal of Information Security*, pages 1–18, 2014.

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

[7] Dyadic. Dyadic Security White Paper. Technical report, 2014. Published online at `https://www.dyadicsec.com/`.

[8] Dyadic. Secure Computation: A Technical Primer. Technical report, 2014. Published online at `https://www.dyadicsec.com/`.

This page intentionally left blank

Author Index

This page intentionally left blank