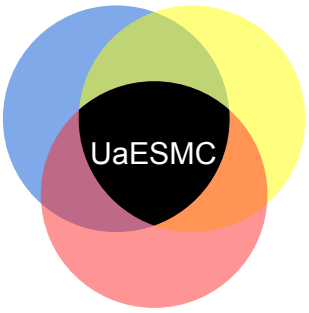

Project N°: **FP7-284731**

Project Acronym: **UaESMC**

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

Instrument: **Specific Targeted Research Project**

Scheme: **Information & Communication Technologies**

Future and Emerging Technologies (FET-Open)

Deliverable D2.1
Review of the state of the art in secure multiparty computation

Due date of deliverable: 31st May 2012

Actual submission date: 31st May 2012

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

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

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

Executive Summary:

Review of the state of the art in secure multiparty computation

This document summarizes deliverable D2.1 of project FP7-284731 (UaESMC), a Specific Targeted Research Project supported by the 7th Framework Programme of the EC within the FET-Open (Future and Emerging Technologies) scheme. Full information on this project, including the contents of this deliverable, is available online at `http://www.usable-security.eu`.

This report contains a review of the current state of the art of cryptographic techniques for secure multiparty computation. It focuses on the variety of techniques, not the problems that can be solved with it. For each technique, we characterize its costs, and the security properties achievable with it. As such, the report provides a basis for the further work in the UaESMC project. As the work in the project is at the point where these techniques are just about to start being applied, no conclusions are provided in this report.

This report is the initial version of the working document *Challenges, bottlenecks, and breakthroughs*, maintained by the scientific coordinator of the project and summarizing the state of the art in secure multiparty computation, achieved in- and outside of UaESMC so far. Further versions of this document will be used to prioritize and direct the research effort towards the various approaches investigated by UaESMC.

List of Authors

Peeter Laud (CYB)
Riivo Talviste (CYB)

Contents

Chapter 1

Introduction

The goal of the UaESMC project is to push the state of the art in the secure multiparty computation (SMC) to a level above the threshold for applicability in many different sectors of the society. It achieves that goal through the identification of common problems in practical applications, and through solving them using cryptographic, game-theoretic and engineering means.

This deliverable gives an overview of the existing cryptographic techniques of SMC. It covers different data representations in secure computations and different SMC frameworks (different sets of protocols working on the same data representation). This deliverable also reviews specific techniques for certain tasks that are more efficient than the application of general techniques. As such, it will serve as a reference of existing methods and techniques for the subsequent work in this project.

The deliverable attempts to characterize different techniques in terms of the security guarantees they provide, as well as in terms of their complexity (computation, communication bandwidth and rounds). The deliverable also attempts to give a high-level overview of the technical details, and provide concrete pointers for further study.

In Chapter 2 of this deliverable we outline the different security definitions for multiparty computation. We consider the difference between stand-alone and composable protocols, as well as between different attacker models (semi-honest, consistent and malicious). Chapter 3 reviews the main frameworks for standalone and Chapter 4 for composable SMC protocols (although the distinction is not complete). In Chapters 5 and 6 we review specific efficient techniques for particular tasks. Chapter 5 in particular concentrates on the technique of *server-assisted* secure computation; we have singled out this technique because we believe in its wide applicability in securely solving practical problems.

Being targeted towards the performers of UaESMC, this report does not aim to be completely self-contained. We assume certain familiarity with common building blocks of cryptographic protocols, including homomorphic encryption and oblivious transfer.

This deliverable does not provide any conclusions, because at this point in the UaESMC project, there cannot be any. Instead, it will serve as one of the knowledge bases for the work that is going to be performed in this project.

This document will continue its life after its submission to the EC. It will be updated with new results in SMC, cryptographic or otherwise, achieved in the UaESMC project and elsewhere. As such, it will become the working document *Challenges, bottlenecks, and breakthroughs*, used as the basis for the direction of the research effort towards the various approaches investigated by UaESMC. It shows where the most gains can be made with the least effort, and which problems must be overcome or worked around in order to achieve the project's objectives. To keep the document updated, we will monitor top-tier conferences in security, cryptography and game theory, as well as preprint archives for new results in this area.

Chapter 2

Definitions

In this chapter, we review the security definitions of secure multiparty computation. The security requirements are stated through the ideal/real system paradigm, stating that a protocol is secure if it cannot in some sense be distinguished from an ideal functionality.

2.1 Adversary model

In this work we will model all adversarial behavior in a given protocol as one adversary that can corrupt one or more parties, see their private inputs/outputs and control their behavior. A coalition of dishonest parties that collude and exchange data, is considered as a single adversary.

In a large scale, we can talk about two groups of adversarial behavior:

- A *passive* adversary is able to gain access to all private inputs and outputs of the corrupted party, but the party still follows the protocol. Such party is called *semi-honest* or *honest-but-curious*. A semi-honest party tries to use all available to information to break the confidentiality of the protocol, but it does not deviate from its specification.

- An *active* adversary takes full control of the corrupted party. Such *malicious* party does not follow the protocol but rather obeys the adversary.

In general, protocols against passive adversaries have simpler structure and require less computation and communication. Hence the "default" setting in this report is that of semi-honest parties. Protocols that are secure against active adversaries are much more complex, requiring commitment schemes, zero-knowledge proofs, etc.

This kind of active adversary may seem too strong and protocols secure against such adversarial behavior are often too complex. Laur and Lipmaa [44] introduced a new concept – *consistent computations* – which means that malicious parties cannot learn anything beyond their intended output and honest parties can detect malicious behavior that alters their outputs. Hence, a valid corruption complain reveals only a single bit of information about the inputs of honest parties. Using this model, it is possible to construct more efficient protocols than for the classical active adversary.

2.2 Protocol security

The security of multiparty computation can be formally defined through the ideal/real system paradigm. An ideal system contains a trusted third party that privately interacts with all parties, collects their inputs, does the necessary computations and distributes parties' outputs. This model can also incorporate adversarial behavior. The real model has no such party. A protocol (in the real model) is secure if whatever an adversary can accomplish in the real world, can be simulated in the ideal world by constructing a suitable ideal-world adversary. Thus, the security in the ideal setting is no stronger than the real protocol is able to provide.

This framework was introduced independently by Canetti [15, 16], and Pfitzmann and Waidner [62] and is largely used in protocol security proofs.

Let us have an ideal functionality $\mathcal{F}$ of a task that specifies the actions of the trusted third party and the security requirements. Let ϕ be an ideal protocol that realized this task using $\mathcal{F}$ in the ideal world. The protocol ϕ has a very simple structure: the n parties of the protocol receive their inputs from the outside (from the environment $\mathcal{Z}$ described below), hand them over to $\mathcal{F}$, receive $\mathcal{F}$'s output and forward them to the outside. In the real world we have a protocol π that emulates ϕ for $\mathcal{F}$. We also have an adversary $\mathcal{A}$ that is able to control the communication of parties of π and can corrupt some of them. The amount of control $\mathcal{A}$ has over corrupted parties models whether the adversary is passive or active. On the other hand, in the ideal setting ϕ is coupled with a simulator $\mathcal{S}$ that gets the state of the corrupted parties and other leaked information from $\mathcal{F}$ and tries to emulate the behavior of $\mathcal{A}$. The simulator $\mathcal{S}$ may corrupt some of the parties of ϕ; the corrupted parties no longer forward messages between the outside environment $\mathcal{Z}$ and the ideal functionality $\mathcal{F}$, but yield to the complete control of $\mathcal{S}$. In both worlds there is also an environment $\mathcal{Z}$ that communicates with either π or ϕ to provide protocol inputs and collect outputs. At the same time $\mathcal{Z}$ also talks to $\mathcal{A}$ or $\mathcal{S}$ as part of the attack against the protocol. Hence, in the real world, the adversarial entities $\mathcal{Z}$ and $\mathcal{A}$ can cooperate and coordinate their activities.

We say that the real protocol π is at least as secure as the ideal protocol ϕ if for all adversaries $\mathcal{A}$ there exists a simulator $\mathcal{S}$, such that no environment $\mathcal{Z}$ can distinguish whether it is running together with π and $\mathcal{A}$ (real world) or with ϕ and $\mathcal{S}$ (ideal world). From this definition we see that in order to prove some π at least as secure as ϕ, it is necessary to show the existence of $\mathcal{S}$, given $\mathcal{A}$. Most commonly, the proof consists of the construction of a machine *Sim*, such that for any $\mathcal{A}$, the composition of $\mathcal{A}$ and *Sim* is a suitable $\mathcal{S}$.

2.2.1 Basic SMC functionality

In the most simple case let us consider secure function evaluation (SFE), where each party P_i ($i \in \{1, \ldots, n\}$) holds an input value x_i and receives an output value y_i. For a function f we can write this as

$$(y_1, \ldots, y_n) \leftarrow f(x_i, \ldots, x_n).$$

We can now consider an ideal functionality $\mathcal{F}_f$ that has private point-to-point communication channels to all of the n parties and to the simulator. The functionality $\mathcal{F}_f$ receives the inputs x_i from the n parties. Note that if a party is uncorrupted, then its input originates from $\mathcal{Z}$, while for a corrupted party, the input originates from $\mathcal{S}$. After receiving the inputs, $\mathcal{F}_f$ expects the command "compute!" from $\mathcal{S}$. After receiving that, it computes $y_1, \ldots, y_n$ and hands them over to respective parties. Again, note that if the i-th party is uncorrupted then y_i is forwarded to $\mathcal{Z}$. If i-th party is corrupted then y_i is given to $\mathcal{S}$, which also picks the value that is given to $\mathcal{Z}$.

2.2.2 An arithmetic black box

Damgård and Nielsen [23] propose a slightly different ideal functionality for characterizing the security of multiparty computation. Their construction does not just provide secure function evaluation, but is equivalent to something that they call the *arithmetic black box* (ABB). An ABB can be thought as a secure general-purpose computer. Any party can give private inputs to the ABB and any majority of parties can ask it to perform any feasible computation. The result of that computation is stored in the internal state of the ABB and can be used as an input to subsequent computations. Any value stored in the ABB is made public only if a majority of parties explicitly ask for it.

In their construction the execution of the protocol π happens in the presence of environment $\mathcal{Z}$ that also models the adversarial behavior. The environment $\mathcal{Z}$ also sees the messages exchanged between parties. The corresponding ideal functionality $\mathcal{F}$ has private communication channel with all of the parties and special channel to $\mathcal{S}$ that is used to receive information about corrupted parties and leak data.

The arithmetic black box is an ideal functionality $\mathcal{F}_{\mathsf{ABB}}$. In each activation, $\mathcal{F}_{\mathsf{ABB}}$ expects a command from all honest parties and executes it iff all honest parties agree. The command can be one of the following.

- Input private data. One of the honest parties performs the command $(x \leftarrow v)$, where x is a variable and v is a value. Other honest parties are expected to perform the command $(x \leftarrow ?)$. As the result, the variable x is added to the internal state of ABB and its value will be v. If x was already contained in the state then its value is updated to v.

- Compute. All honest parties have to perform the command $(x \leftarrow \otimes(x_1, \ldots, x_k))$, where $x, x_1, \ldots, x_k$ are variables and $\otimes$ is an arithmetic operation. The ABB is parameterized with the set of arithmetic operations it is capable to perform. Typically, these are addition, multiplication with a constant, multiplication, comparison (returning either 0 or 1), etc. In practice, the set of available operations depends on the efficient protocols that are or are not available for various computations. The variables $x_1, \ldots, x_k$ must be defined. The variable x is added or updated to the internal state of ABB, receiving the result of the computation.

- Output. All honest parties have to perform the command $(\leftarrow x)$, where x is a variable that has been defined in the internal state of ABB. As the result, the value of x is made available to all parties.

If two honest parties disagree, then $\mathcal{F}_{\mathsf{ABB}}$ gets corrupted, meaning that it will output the entire current state and all future inputs on the special channel to leak data. Also, it lets the environment $\mathcal{Z}$ decide all the future inputs via the special channel.

The availability of a protocol securely realizing $\mathcal{F}_{\mathsf{ABB}}$ makes the implementation of any secure computation protocol conceptually simple — just perform all computations inside $\mathcal{F}_{\mathsf{ABB}}$. However, the efficiency of such solution may be unacceptable. It is important to come up with methods to combine $\mathcal{F}_{\mathsf{ABB}}$ with specific protocols for particular subproblems, in order to obtain a more efficient solution.

Chapter 3

Generic Constructions of Protocols

There exist several generic constructions for SMC protocols that receive as input a description of some algorithm and the distribution of the inputs to that algorithm among the data owners, and output the description of a secure protocol that executes that algorithm in a privacy-preserving manner. The goal of this chapter is to give an overview of these constructions.

In this chapter, the considered computational task, given by the algorithm description, is *non-interactive*, i.e. all inputs are received at the beginning of the computation and the parties do not decide on the next steps during the computation; based on what has already been computed. Secure methods for such computations are reviewed in Chapter 4. In terms of ideal functionalities and their realizations, this chapter considers various constructions that, on input of the description of a function f with n inputs and outputs, construct a protocol π_f that securely realizes the ideal functionality $\mathcal{F}_f$. Some constructions put extra requirements on the function (e.g. all n outputs must be the same), but these are usually easy to overcome with standard techniques (masking).

3.1 Yao's garbled circuits

This construction is inherently a technique for two-party computation (although generalizations to multiple parties exist, too [8]). Let us assume that we have two parties P_1 and P_2 who want to compute an arbitrary functionality $f(x_1, x_2) = (y_1, y_2)$ so that P_i holds the input x_i and at the end, receives the output y_i ($i \in \{1, 2\}$). We assume that the functionality f is a boolean circuit and inputs x_1 and x_2 are bitstrings. In Yao's construction [75], P_1 encrypts ("garbles") the circuit to preserve the privacy of the input values. For each wire in the circuit, two random values are chosen, one representing 0 and another representing 1. For a gate g with inputs $b_1 \in \{0, 1\}$ and $b_2 \in \{0, 1\}$, the random values of input wires of g corresponding to each pair of the values of b_1 and b_2 are used as keys to encrypt the value corresponding to $g(b_1, b_2)$ of the output wire of g. Each gate's computation table is also randomly permuted, so that it maps random inputs to the corresponding random output. P_1 constructs the garbled circuit and sends it to P_2 together with its encrypted inputs. While evaluating the garbled circuit, P_2 uses 1-out-of-2 Oblivious Transfer (OT) for each of its input bits to get the corresponding encrypted input value from P_1.

An in depth description of this method and its security proof is given in [46].

3.1.1 Evaluation of Yao's circuits

Schröpfer and Kerschbaum have put together a detailed analysis of the running time of creating and evaluation Yao's garbled circuits. They do so by combining time complexities of necessary primitives. In the following, we give an overview of their results, but for a more in-depth description, we refer the reader to the original paper [67].

Table 3.1 lists the parameters that are used in the performance equations.

In [67], Yao's circuit evaluation is split into five steps and time-complexity is given separately for each step. The total evaluation complexity is a sum $\sum_{i=1}^{5} t_i$.

Parameter	Description
Input/output:	
l	bit-length of an input or output value
ρ	party, $\rho \in \{P_1, P_2\}$
α^ρ	number of private inputs for ρ
β^ρ	number of private outputs for ρ
n_g	number of gates in the binary circuit
Secure computation:	
k_{OT}	key-length of Oblivious Transfer
k_{GC}	key-length of garbled circuit
System:	
$t^\rho_{RND}(n)$	time in [ms] to select n random bits
$t^\rho_{MUL}(n)$	time in [ms] to multiply two n-bit numbers
$t^\rho_{POW}(n)$	time in [ms] for one modular exponentiation of two n-bit numbers
$t^\rho_{INV}(n)$	time in [ms] to compute one modular inverse in a field of n bits
$t^\rho_{OWH}(n)$	time in [ms] for hash function of one n-bit number
t_{LAT}	network latency in [ms]
b	network bandwidth [Mbit/s]
$r_{t_{LAT},b}(n)$	transfer rate in [Mbit/s] for n bits (depends on latency and bandwidth)
MTU	maximum number of bits of payload per network packet

Table 3.1: Parameters for evaluating the performance of garbled circuits

1. Garbling the circuit:
$$t_1 = 2n_w t^{P_1}_{RND}(k_{GC}),$$
where $n_w = \alpha^{P_1} + \alpha^{P_2} + n_g$ is the number of wires in the circuit.

2. Encrypting the output values of garbled truth tables:
$$t_2 = 4n_g t^{P_1}_{OWH}(k_{GC}).$$

3. Encrypting and transmitting the inputs of P_2 using 1-out-of-2 OT:
$$t_3 = \alpha^{P_2} l t_{OT},$$
where t_{OT} is the time necessary for one 1-out-of-2 OT. For the protocol in [52] it can be estimated as
$$\begin{aligned} t_{OT} =& 2t^{P_1}_{MUL}(k_{OT}) + 5t^{P_1}_{POW}(k_{OT}) + t^{P_1}_{INV}(k_{OT}) + 4t^{P_1}_{OWH}(k_{OT}) + \\ & t^{P_2}_{MUL}(k_{OT}) + 2t^{P_2}_{POW}(k_{OT}) + t^{P_2}_{INV}(k_{OT}) + 2t^{P_2}_{OWH}(k_{OT}) + 4t_{LAT}. \end{aligned}$$

4. Sending the encrypted circuit, P_1's encrypted inputs and output keys to P_2:
$$t_4 = (n_i + n_t + n_o) \cdot r_{t_{LAT},b}(n_i + n_t + n_o),$$
where $n_i = \alpha^{P_1} l k_{GC}$ denotes the bits for P_1's encrypted input bits; $n_t = 4n_g k_{GC}$ denotes the bits for the encrypted circuit; and $n_o = 2\beta^{P_2} l k_{GC}$ is the bits for the output keys.

5. Circuit evaluation by P_2 and returning P_1's encrypted output:
$$t_5 = n_g t^{P_2}_{OWH}(k_{GC}) + \left\lceil \frac{\beta^{P_1} l k_{GC}}{MTU} \right\rceil t_{LAT}.$$

3.2 Private Branching Programs

In this Section we describe private branching programs and its prerequisite, computationally-private information retrieval. Both definitions are given by Lipmaa [48].

Computationally-Private Information Retrieval. In a 1-out-of-n computationally-private information retrieval protocol, $(n, 1)$-CPIR, for l-bit strings, the client has an index $x \in \{0, \ldots, n-1\}$ and the server has a database $f = (f_0, \ldots, f_{n-1})$ of l-bit strings. The client obtains f_x. In the following we need a CPIR protocol that has the next property. We say a CPIR protocol Γ is private branching program friendly if it satisfies the next three assumptions:

1. Γ has two messages, a query $Q(l, x)$ from the client and a reply $R(l, f, Q)$ from the server, such that the stateful client can recover f_x by computing $A(l, x, R(l, f, Q))$.

2. Γ is uniform in l, that is, it can be easily modified to work on other values of l.

3. There exists a compress function C that maps $Q(l', x)$ to $Q(l, x)$ for any $l' \leq l$ and x.

More formally, $\Gamma = (Q, R, A, C)$ is a quadruple of probabilistic polynomial-time algorithms that satisfy $A(l, x, R(l, f, Q(l, x))) = f_x$, and $C(l', l, Q(l', x)) = Q(l, x)$ for any $l' \leq l$ and x. If the existence of C is not required we also write $\Gamma = (Q, R, A)$ even if C exists.

Private Branching program. A *branching program* [74] (BP) is an acyclic graph where the internal nodes are labeled from some variable set $\{x_0, \ldots, x_{m-1}\}$, the sinks (leafs) are labeled by l-bit strings and the two output edges of each internal node are labeled by 0 and 1 respectively. Every source and every assignment of the variables corresponds to one path from this source to one of the sinks as follows. The path starts from the source. If the current version of the path does not end with a sink, test the variable at the end of the path. Select one of the outgoing edges according to the value of this variable and append its endpoint to the path. If the path ends with a sink, return the label of this sink as the value of the branching program. Hence, a branching program with σ sources computes a function $f : \{0,1\}^m \to \{0,1\}^{\sigma l}$; by default $\sigma = l = 1$.

One way to evaluate a branching program is to start from the sinks and end with the sources. Let v be some node (labeled x_i), such that the values R_{v_i} of the end nodes of the outgoing a_i-labeled edges are known but the value R_v is not yet known. Then one sets $R_v \leftarrow R_{v_i} = (R_{v_0}, R_{v_1})[i]$. Thus, every node can be seen as implementing a (2,1)-selector/branch operation. Ishai and Paskin [38] proposed a protocol for cryptocomputing (that is, computing of ciphertexts) any function f by first fixing the input length m and then designing an efficient branching program P_f for this length. Their protocol uses the private version of evaluating a branching program explained here. There, all computations are done on "ciphertexts", and the local selector/branch operation $R_v = (R_{v_0}, R_{v_1})[i]$ is implemented by using a communication-efficient two-message (2,1)-CPIR protocol. More precisely, for client's every input variable x_i to f, the client sends to the server the first message Q_i of the underlying (2,1)-CPIR protocol. After that, the server recursively computes the value R_v for every non-sink node of P_f as explained earlier. However, R_v is going to be equal to the second message of the (2,1)-CPIR protocol that uses Q_i and the database (R_{v_0}, R_{v_1}). For a sink v, R_v is just equal to its label. At the end, the server sends to the client R_v for every source v, and the client applies recursively the local decoding procedure to each such R_v to obtain the value of the branching program. We call this protocol PrivateBP. See [38] for an in-depth description about private branching programs. Their results are slightly improved by [48].

In the case of branching programs, the amount of communication and client's computation depend only on the length (or depth) of the BP and are independent of the total size of the BP. This makes the described approach useful for "wide" BP-s.

3.2.1 Evaluating BP-s

In the case on PrivateBP, server holds a function $f : \{0,1\}^m \to \{0,1\}^{\sigma\ell}$ from some set of functions $\mathcal{F}$, where all the functions can be computed by a branching programs of polynomial size. A client holds the m-bit input. Let σ be the number of ℓ-bit sinks of the corresponding branching program P_f. Also, let $len(P_f)$ be the length (longest path) and $size(P_f)$ be the size of this branching program. Then we define $len(\mathcal{F}) = \max_{f\in\mathcal{F}} len(P_f)$.

Using Lipmaa's $(2,1)$-CPIR protocol [47][1] we can cryptocompute $f(x)$ with a total complexity of $\Theta((m+\sigma)(\ell + len(\mathcal{F})k))$, where k is a security parameter (key size). In this case client sends its public key and m messages with size $\leq \ell(len(\mathcal{F})+2)k$, whereas the server responds with σ messages of the same size. The computational complexity of the client is equal to the total communication cost and the computational cost for the server is $\tilde{\Theta}(size(P_f))$ [48]. Lipmaa also proposes a couple of BP balancing techniques [48], that bring the total communication cost down to $\Theta((m/\log(m/\sigma))(\ell + len(\mathcal{F})k))$ or alternatively, $(1+o(1))\sigma\ell + \Theta(m \cdot len(\mathcal{F})) \cdot k$.

3.3 Private Linear Branching Programs

Although in the case of BP-s the amount of communication depends only on the depth of the BP, the evaluation protocol requires expensive homomorphic operations on ciphertexts. Furthermore, the server's computation still depends on the total size of the BP. To alleviate this, Barni et al. [7] introduces a concept of *linear branching programs* (LBP) — a generalization of BP-s that unifies and extends the work of [43, 13, 64]. Analogously to the Yao's garbled circuits, evaluating a LBP consists of three steps. First the server creates a garbled LBP from th efunction description and sends it to the client. Then the client uses a special protocol ObliviousLinearSelect to obtain garbled values which correspond to the outcome of the comparison of the linear combination of the attribute vector for the threshold for each garbled node. This step is similar to obtaining wire secrets osing OT in Yao's garbled circuits. Finally, the client is able to evaluate the LBP locally.

Barni et al. [7] give the size of the garbled LBP as well as the communication complexity of the ObliviousLinearSelect protocol using garbled circuits (GC), oblivious transfer (OT) and homomorphic encryption (HE) as building blocks. In both cases they also compare their results with the results of the work they extend. Parameters used in the following tables: z: #nodes, d: #decision nodes, n: #attributes, ℓ: bit length of attributes, ℓ': bit length of thresholds (for LBP-s), t: symmetric security parameter, T: asymmetric security parameter, κ: statistical correctness parameter.

Algorithm to create and evaluate garbled LBP	Garbled LBP size in bits
[43, 13]	$2z(\lceil\log(z)\rceil + t + \kappa)$
[7]	$2d(\lceil\log(d)\rceil + t + 1)$
[7], tiny garbled LBP[2]	$2^d\lceil\log(z-d)\rceil$

Oblivious selection protocol	Private function	Moves	Asymptotic communication complexity		
			GC	OT	HE
[13]	BP	$OT+2$	$12z\ell(t+\kappa)$	$OT_t^{z\ell}$	$(n+z)2T$
[13], extended by [7]	LBP		$12z\ell'(t+\kappa)$	$OT_t^{z\ell'}$	
[7], hybrid	BP	$OT+2$	$12d\ell t$	$OT_t^{d\ell}$	$(n+\frac{\ell}{T-\kappa}d)2T$
	LBP		$12d\ell' t$	$OT_t^{d\ell'}$	$(n+\frac{\ell'}{T-\kappa}d)2T$
[7], circuit	BP	OT	$4(n\log(d)+3d\log(d))\ell t$	$OT_t^{n\ell}$	
	LBP		$16nd(\ell^2+\ell')t$	$OT_t^{n\ell}$	

This construction of LBP-s is proven to be secure in semi-honest model, but can be also modified to

[1] Lipmaa's $(2,1)$-CPIR is currently the most efficient $(2,1)$-CPIR for the purpose of PrivateBP protocol.
[2] Tiny LBP-s ($d \leq 10$) result in substantially smaller communication complexity.

achieve full security against malicious adversaries by using committed OT, secure two-party computation on committed inputs and verifiable homomorphic encryption schemes (see [40] for details).

3.4 Private Boolean Circuits

Recently, Nielsen et al. [56] introduced a new OT-based two-party private boolean circuit evaluation technique that uses XOR-shared values. They build upon an OT extension by [37] that allows to turn κ seed OTs based on public key cryptography into $\ell = poly(\kappa)$ number of OTs using only $\mathcal{O}(\ell)$ invocations of cryptographic hash functions (e.g. $\kappa = 640$, $\ell = 2^{20}$). To achieve security against active adversary, they commit both parties to all of their shares and introduce the concept of *authenticated OT*. In the construction they use slightly leaky proofs and get leakage probability of $(2n)^{-B} = 2^{-\log_2(n)(B-1)}$, where n is the number of gates and B is the "bucket size" (the security parameter). For example, for $n = 2^{20}$ and $B = 6$, we get leakage probability 2^{-100}. The following table summarizes their results:

Gate	Cost (calls to hash func.)	Cost, optimized[3]
Input	59	8
AND	$856B + 118$	$142B + 16$
XOR	free	free

[3]These optimizations are not covered in [56] as the authors claim that they undermine the modularity of their constructions.

Chapter 4

Virtual machine based constructions

This chapter gives an overview of composable techniques for secure multiparty computation. The protocols described in this chapter work on private data that has been protected with a certain method; and produce outputs that are protected using the same method. In this way, the outputs can be used as inputs in the next protocol instance, and basic protocols for addition, multiplication, etc. can be composed to protocols that privately compute more complex functions. The protocols given here can be used to implement the ideal functionality $\mathcal{F}_{\mathsf{ABB}}$ for various sets of commands.

Invariably, the protocol sets reviewed here are based on secret sharing or threshold homomorphic encryption. A lot of material in this chapter is based on [1].

4.1 Multiparty computation based on Shamir secret sharing

In protocol sets based on Shamir's secret sharing [68], a value a from a field $\mathbb{Z}_q$ (q is a prime number) is represented as $[a] = ([a]_1, \ldots, [a]_n)$, where n is the number of parties and $[a]_i = f(i)$ for some $(t-1)$-degree polynomial f over $\mathbb{Z}_q$, such that $f(0) = a$. For any set of t positions $\mathbf{I} = \{i_1, \ldots, i_t\}$ there are the Lagrange interpolation coefficients $r^{\mathbf{I}}_{i_1}, \ldots, r^{\mathbf{I}}_{i_t}$, such that $\sum_{j \in \mathbf{I}} r^{\mathbf{I}}_j [a]_j = a$ for all a and f. If we want to represent m-bit numbers a, then $a, [a]_1, \ldots, [a]_n$ must belong to some $\mathbb{Z}_q$, where $\lfloor \log q \rfloor = m$. This will give us protocols that are secure against passive adversaries controlling up to $t-1$ parties. We assume $m \approx 100$, because this will make it easier to avoid overflows in computations, which are inconvenient to handle, because computing *modulo* a prime number q is usually not something we want to do in our actual algorithms that we want to implement in a privacy-preserving manner.

If we want security against active adversaries as well, then we have to commit to the coefficients of the polynomial f. Typically, the security of the commitments is based on the discrete logarithm problem and the commitments have to belong to a group where this problem is hard, e.g. $\mathbb{Z}_p$ for some p, where $\lfloor \log p \rfloor = k \approx 1000$. Alternatively, the group maybe over some elliptic curve, in which case the size of the commitments k is comparable to the size of the shares m.

The following table describes the complexity (per party) of local operations. Here the computationally secure commitments are those of Pedersen [61], while the unconditional commitments are again based on secret sharing [17]

Operation	Adversary/security	Computation	Comm. bw.	Rounds
Addition $(a+b)$	passive, uncond.	$\mathcal{O}(m)$ (1 addition in $\mathbb{Z}_q$)	0	0
	active, uncond.	$\mathcal{O}(mn)$ (t additions in $\mathbb{Z}_q$)	0	0
	active, comp.	$\mathcal{O}(k^2 n)$ (t multiplications in $\mathbb{Z}_p$)	0	0
Mult. w/ const. $(\lambda \cdot a)$	passive, uncond.	$\mathcal{O}(m^2)$ (1 multiplication in $\mathbb{Z}_q$)	0	0
	active, uncond.	$\mathcal{O}(m^2 n)$ (t multiplications in $\mathbb{Z}_q$)	0	0
	active, comp.	$\mathcal{O}(k^3 n)$ (t exponentiations in $\mathbb{Z}_p$)	0	0

Multiplication. Given two shared values $[a]$ and $[b]$, the goal of the multiplication protocol is to come up with a shared value $[c]$, such that $c = ab \bmod q$. With a semi-honest adversary, this is achieved by

- Each party P_i computing $[a]_i \cdot [b]_i$ and sharing it;
- The parties collectively picking a set $\mathbf{I} \subseteq \{1, \ldots, n\}$ of size $2t - 1$;
- The parties computing $[c] = \sum_{j \in \mathbf{I}} r_j^{\mathbf{I}} [[a]_j \cdot [b]_j]$, where $r_j^{\mathbf{I}}$ are the Lagrange interpolation coefficients for polynomials of degree $2t - 2$.

If the adversary is malicious then the first step requires the party P_i to prove that it is indeed sharing the value $[a]_i \cdot [b]_i$. In the second step, the set $\mathbf{I}$ is picked from the set of the indices of the parties that have provided a valid proof.

For semi-honest adversaries, the protocol requires one round, and the major part of computation for each party is the sharing of the value $[a]_i \cdot [b]_i$. This requires nt multiplications in $\mathbb{Z}_q$. The communication cost for each party P_i is the sending out of $n-1$ shares and receiving one share from each of the other $n-1$ parties. For malicious adversaries, the protocols are much more expensive. In this case, each party P_i has to commit to the value $[a]_i \cdot [b]_i$ (with a proof that it has been correctly computed; this is the most expensive part) and show that it is indeed sharing that value.

Below, we are describing more complex computations with shared values. The computational and communication complexity of these operations will be measured in "invocations". One invocation is either the multiplication of two shared values, or the generation of a shared random value (which is achieved by each party generating a random value, sharing it with others, and the addition of these shared values) and the subsequent opening of this or some other value.

Assume now that we are given an array of non-zero secret values $[a_1], \ldots, [a_k]$ and have to compute either their product or prefix product, i.e. $[p_j] = \prod_{i=1}^{j} [a_i]$, for $1 \leq j \leq k$. The two corresponding protocols are

$$[p] \leftarrow \mathsf{Mult}^*([a_1], \ldots, [a_k]), \text{ and}$$

$$([p_1], \ldots, [p_k]) \leftarrow \mathsf{PreMult}([a_1], \ldots, [a_k])$$

Hence, Mult^* is a special case of PreMult. Both protocols can be implemented with either variable on constant number of rounds.

Operation	Rounds	Invocations	Ref.
Mult^*	$\log(k)$	$k-1$	[6]
	6	$5k$	
after preprocessing	2	k	
PreMult	$\log(k)$	$2k - \log(k)$	[6]
	6	$5k$	
after preprocessing	2	k	

Symmetric boolean functions. Let $[b_1], \ldots, [b_k]$ be an array of binary values $b_i \in \{0, 1\} \subset \mathbb{Z}_q$, and $k < q - 1$. We want to compute a symmetric boolean function of these variables, e.g. AND, OR, XOR. The output of a symmetric boolean function depends only on the number of ones in input and not on their location. The complexity of computing a symmetric boolean function is the same as for PreMult. Similarly to PreMult, we can define functions PreAND and PreOR (in this case $[a_1], \ldots, [a_k]$ are secret bits). The corresponding complexities are shown below:

Operation	Rounds	Invocations	Ref.
Symm. bool. func.	6	$5k$	[21]
after preprocessing	2	k	
PreAND, PreOR	12	$12k + 5\sqrt{k}$	[21, 70]
after preprocessing	8	$4k + \sqrt{k}$	

Bitwise shared integers. In this section we are working with bitwise shared secrets, denoted as $[a]_B = ([a_{k-1}], \ldots, [a_0]), k = \lceil \log q \rceil$.

Operation	Rounds	Invocations	Ref.
$\mathsf{BitLT}([a < b] \leftarrow \mathsf{BitLT}([a]_B, b))$	12	$\approx 13k$	[21]
after preprocessing	8	$\approx 5k$	
$\mathsf{BitLT}([a < b] \leftarrow \mathsf{BitLT}([a]_B, [b]_B))$	14	$15k$	[21]
after preprocessing	10	$7k$	
$\mathsf{RandBitwise}(([r], [r]_B) \leftarrow \mathsf{RandBitwise}(), r \in_R \mathbb{Z}_q)$	13	$15k$	[21]
for statistical privacy	3	$2k$	
after preprocessing	9	$5k$	
$\mathsf{AddBitwise}$ ($[s]_B \leftarrow \mathsf{AddBitwise}([a]_B, [b]_B)$, add in $\mathbb{Z}_q$)	$\log(k) + 1$	$3k - \log(k)$	[1]
	22	$28k \log(k)$	[21, 70]
after preprocessing	18	$28k \log(k)$	
BitDec ($[a]_B \leftarrow \mathsf{BitDec}([a])$)	$\log(k) + 22$	$< 32k$	[21, 70, 57]
after preproc.	$\log(k) + 9$	$7k$	

Integer comparison. In the following, κ is the security parameter.

Operation	Rounds	Invocations	Ref.
CompB ($[a \leq b] \leftarrow \mathsf{CompB}([a], [b])$)	13	$15k + 2\kappa$	[70]
after preprocessing	9	$5k$	
Comp ($[a < b] \leftarrow \mathsf{Comp}([a], [b])$)	25	$84k + 5$	[70, 57]
after preprocessing	12	$12k + 5$	
$\mathsf{EqualOpen}$ ($(a = b) \leftarrow \mathsf{EqualOpen}([a], [b])$)	5	4	[1]
after preprocessing	2	1	
Equal ($[a = b] \leftarrow \mathsf{Equal}([a], [b])$)	16	$20k$	[57]
after preprocessing	3	k	
EqualB ($[a = b] \leftarrow \mathsf{EqualB}([a], [b])$)	7	$7k + 2\kappa$	[1]
after preprocessing	3	k	
EqualP ($[a = b] \leftarrow \mathsf{EqualP}([a], [b]), k = \#tests$)	6	$12k$	[57, 70]
after preprocessing	3	$2k$	

CompB is much more efficient than Comp, but works with bounded k-bit integers ($k \ll \log(q)$) and offers statistical privacy. The latter works with any integers in $\mathbb{Z}_q$ ($k = \lceil \log(q) \rceil$) and provides perfect privacy. Also, Comp can be modified to work with signed integers [1].

Similarly, EqualB works with k-bit integers represented as elements in $\mathbb{Z}_q$ ($k + \kappa < \lceil \log(q) \rceil$) and offers statistical privacy.

The protocol EqualP is probabilistic equality testing with success rate 1 if $a = 0$ and 0.5 if $a \neq 0$. EqualP performs k independent tests in parallel, resulting in the reduced error probability 2^{-k}.

Division. The general division protocol Div has two variants, one that uses exact computation when shifting $[a]$ down m bits; and another that approximates it. Θ-complexities of the respective protocols are equal, but more details can be found in [30]. In the following, n is the number of players, the protocols use k-bit integers, and $\rho \in \mathbb{N}$ is a security parameter.

Operation	Rounds	Communication	Computation	Ref.
Mod ($[amodb] \leftarrow \mathsf{Mod}([a], [b])$)	$\mathcal{O}(1)$	$\mathcal{O}(kn)$	$\mathcal{O}(k^2 n + kn^2 \lg n)$ per player	[2]
Div ($[a/b] \leftarrow \mathsf{Div}([a], [b])$)				
preprocessing	$\Theta(1)$	$\Theta(nk^2 + nk\rho)$	$\mathcal{O}(nk^2)$ mult. in $\mathbb{Z}_p$ per player	[30]
processing	$\Theta(\log(k))$	$\Theta(nk \log(k)$ $+n\rho \log(k))$	$\mathcal{O}(nk^2)$ mult. in $\mathbb{Z}_p$ per player	

Arrays of secrets. Let $[A]$ denote an array of secret shared values:

$$[A] = ([a]_1, \ldots, [a]_k),$$

where k is the number of elements in this array. We denote reading from and writing to the secret array as $[x] \leftarrow [A](i)$ and $[A](i) \leftarrow [x]$ respectively. For reading from and writing to the i-th element of $[A]$, we construct a binary k-element vector $[\mathbf{i}]$. This indexing vector if filled with shares of zeros ($[0]$), except in one position where there is a share of one. This position corresponds to the j-th element that we want to read/write, so $[\mathbf{i}](j) = [1]$.

Protocols MaxL, Max1 and MaxLL [70] find the maximum value in the array of secret shared values $[A]$ and return its secret shared index vector $[\mathbf{i}]$ so that $[A]([\mathbf{i}]) = max(A)$. In these protocols, c_r stands for the round complexity of the comparison protocol used.

Operation	Rounds	Invocations	Ref.
$[x] \leftarrow [A]([\mathbf{i}])$	1	k	[70]
$[A]([\mathbf{i}]) \leftarrow [x]$	1	k	[70]
MaxL ($[\mathbf{i}] \leftarrow \mathsf{MaxL}([A])$)	$\log(k)(c_r + 2)$	$k - 1$ comp. and $2k - 2$ mult.	[70]
Max1 ($[\mathbf{i}] \leftarrow \mathsf{Max1}([A])$)	$< 2c_r$	$k(k-1)/2$ comp.	[70]
MaxLL ($[\mathbf{i}] \leftarrow \mathsf{MaxLL}([A])$)	$\log\log(k)(2c_r + 2)$	$k \log\log(k)$ comp. and $2k \log\log(k)$ mult.	[70]

4.2 Multiparty computation based on additive secret sharing

The Shamir secret sharing scheme described in Section 4.1 works over finite fields. However, when implementing this scheme, we have to use custom data structures and operators to hold and work with the shares. For efficiency we could consider using basic data types (e.g. 32-bit and 64-bit integers) for storing the shares directly, but mathematically, the 2^m-bit integers act like elements in the ring $\mathbb{Z}_{2^m}$ and not like elements of a finite field. Therefore we may consider *additive secret sharing scheme* over ring $\mathbb{Z}_{2^m}$. Given n parties, a sharing of a secret 2^m-bit value s is constructed as follows:

$$\begin{aligned} s_1 &\leftarrow \mathbb{Z}_{2^m} \\ s_2 &\leftarrow \mathbb{Z}_{2^m} \\ &\ldots \\ s_{n-1} &\leftarrow \mathbb{Z}_{2^m} \\ s_n &\leftarrow s - s_1 - \ldots - s_{n-1} \pmod{2^m}, \end{aligned}$$

where $[s]$ denotes the secret shared version of s and $[s]_i$ is the share held by party $\mathcal{P}_i$. Furthermore, it is possible to share individual bits of s and we denote it as $[s_i]$ for the i-th bit. It is easy to see that the additive secret sharing scheme is additively homomorphic, so addition of two secret shared values requires no communication.

Most of the results using this secret sharing scheme are by Bogdanov et al. and the SHAREMIND project [10]. Since SHAREMIND framework is developed with practical efficiency in mind, it uses three parties and passive adversary model with at most one corrupted party.

The most recent set of protocols Sharemind uses can be found in [11]. In this review, we will give an account of the complexity of the protocols in this set. In the following table $\ell = \log_2(m)$[1]. For the division protocol, choosing n' and n'' (denoted as m in the original paper) depend on m, and we refer the reader to the original paper [11] for details.

[1]For most of the SHAREMIND protocols it is required that m is a power of two.

Protocol		Rounds	Communication
Addition	$[a+b] \leftarrow \mathsf{Add}([a],[b])$	0	0
Multiplication	$[ab] \leftarrow \mathsf{Mult}([a],[b])$	1	$15m$
Share conversion $\mathbb{Z}_2 \to \mathbb{Z}_{2^m}$	$[a] \leftarrow \mathsf{ShareConv}([a]^{\mathbb{Z}_2})$	2	$5m+4$
Equality testing	$[a \stackrel{?}{=} b] \leftarrow \mathsf{Equal}([a],[b])$	$\ell+2$	$22m+6$
Shift right (public shift)	$[a \gg p] \leftarrow \mathsf{ShiftR}([a],p)$	$\ell+3$	$12(\ell+4)m+16$
Bit extraction	$([a_0],\ldots,[a_{m-1}]) \leftarrow \mathsf{BitExtr}([a])$	$\ell+3$	$5m^2+12(\ell+1)m$
Division (public divider)	$[a/b] \leftarrow \mathsf{PubDiv}([a],b)$	$\ell+4$	$(108+30\ell)m+18$
Division (shared divider)	$[a/b] \leftarrow \mathsf{Div}([a],[b])$	$4\ell+9$	$2n''m+6n''\ell+39\ell m+35\ell n'+126m+32n'+24$

Other necessary protocols are implemented by combining the protocols above. For example, the greater-than-or-equal operator GTE on $(m-1)$-bit values can be implemented as

$$\mathsf{GTE}(x,y) = \begin{cases} 1, & \text{if the highest bit of the difference } x-y \text{ is } 0, \\ 0, & \text{otherwise.} \end{cases}$$

that uses ShiftR functionality.

4.3 Multi-party computation based on homomorphic encryption

In this Section we will consider protocols that use properties of homomorphic encryption schemes. We will mostly consider additively homomorphic encryption schemes $[\cdot]$, such that $[m_1][m_2] = [m_1+m_2]$ (e.g. lifted ElGamal, Paillier [60], Damgård-Jurik [22]). In this case addition of encryptions can be done locally by each party. However, multiplication requires a separate interactive protocol. We will also consider Franklin-Haber cryptosystem [28] that is XOR-homomorphic. All of the protocols covered in this Section are also covered in more detail in [1, Section 5].

4.3.1 Circuit evaluation

First, we will consider a secure multi-party circuit evaluation protocol by Franklin and Haber [28] that is secure against passive adversary. Next, we will briefly introduce three protocols that work against active adversary. These are by Cramer et al. [18]; Jakobsson et al. [39]; and Schoenmakers et al. [65] respectively.

Franklin and Haber's protocol. In [28], Franklin and Haber propose a XOR-homomorphic cryptosystem and a protocol that allows n parties to collaboratively evaluate a boolean circuit C. The protocol works in passive adversary model and needs $\mathcal{O}(|C|n)$ encrypted bits of communication.

Cramer et al's. protocol. The protocol securely evaluates a function f using threshold homomorphic encryption in a multi-party setting. This protocol is provably secure against static active adversary that is able to corrupt minority of parties. It has $\mathcal{O}(d(C))$ rounds and $\mathcal{O}(n|C|k)$ communication complexity, where C denotes the circuit computing function f, k is a security parameter, $|C|$ is the number of gates in C and $d(C)$ is the depth of circuit C. Multiplication gates on the same level are evaluated in parallel. The authors of [18] argue that the construction of the protocol requires a sufficiently efficient threshold cryptosystem, e.g. Paillier [60], Damgård-Jurik [22] or a variant of Franklin-Haber, detailed in the same paper. For more details, refer to [18]. The protocol of Cramer et al. is improved upon in several papers. Most relevant of those are the work of Damgård and Nielsen [23], who show that a similar protocol is secure against adaptive active adversary; and the work of Hirt and Nielsen [35], who build a protocol with the same complexities that guarantees that every party only learns the correct result and nothing else.

Mix and match protocol. The mix and match protocol by Jakobsson et al. [39] uses the gate-by-gate approach to evaluate the circuit C that computes function f. The protocol first blinds and mixes all the rows in the logical table of each gate in C and then uses a secure multi-party plaintext equality test to evaluate each gate. All gate input and output values are encrypted, so no intermediate results are revealed. The communication complexity of the proposed protocol depends on the number of rows in the logical tables of the circuit gates. Since the latter grows exponentially in the input size, it is reasonable to restrict the approach to binary inputs. Then the communication complexity of the protocol is $\mathcal{O}(n|C|)$ group elements (e.g. using ElGamal cryptosystem) and the round complexity is $\mathcal{O}(n + d(C))$, where n is the number of parties, $|C|$ is the size of the circuit C and $d(C)$ is its depth.

Conditional gate. Schoenmakers and Tuyls [65] propose another approach for circuit evaluation, by constructing a *conditional gate*, a special kind of multiplication gate that efficiently multiplies two values x and y, where x is restricted to a two-valued domain, e.g. $x \in \{0, 1\}$. The conditional gate works under decisional Diffie-Hellman assumption (e.g. using ElGamal cryptosystem) and the choice of y is not restricted, so one may have $y \in \mathbb{Z}_q$ for large prime q. Using conditional gates to evaluate a circuit C corresponding to the function $f : \mathbb{Z}_q \to \mathbb{Z}_q$ has round complexity $\mathcal{O}(nd(C))$ and communication complexity $\mathcal{O}(nk|C| \log(q))$, where n is the number of parties, k is a security parameter and $d(C)$ and $|C|$ are the depth and size of circuit C respectively.

4.3.2 Rational numbers

Given Paillier cryptosystem with modulus $N = pq$ for $p < q$, we can encode a rational number $t = r/s$ as $t' = rs^{-1} \pmod N$, where $-R < r < R$, $0 < s < S < p$ and $RS < N$. We can uniquely recover (r, s) for any such t' using Gauss Algorithm on lattices [72], provided that $2RS < N$. The computation with this values can be performed as normal, provided that the numerator and denominator do not exceed the given limit. For details refer to [27].

4.3.3 Basic computation primitives

This section will briefly describe protocols for jointly generating random numbers, evaluating inverses and decomposing encrypted values into bits by using threshold homomorphic cryptosystems. Again, this overview is based on the work of [1, Section 5.3]. As stated in [18, 66], it is important to notice that some of the protocols represented here require decryption of random values and, therefore, cannot be implemented using cryptosystems that are able to decrypt only a limited subset of messages (e.g. lifted ElGamal). These protocols are designed for and work with Paillier cryptosystem.

Random element generation (RN-Gate). This protocol generates $[r]$, where r is a random element from the encryption scheme's message space [18].

Inversion and modular division. Given $[x]$ and $[y]$, such that $x, y \in \mathbb{Z}_N$, we want to find $[x/y] = [xy^{-1}]$. Here it follows that finding modular division $[x/y]$ requires first finding the encryption of the inverse of y and one multiplication. The inversion protocol is given in [18], with slight optimization in [1].

Unbounded fan-in multiplication. This allows us to compute $[\prod_{i=1}^{l} x_i]$ given $[x_1], \ldots, [x_l]$. Again, the protocol is introduced in [18], with slight optimization in [1].

Random bit generation (RB-Gate). For generating random $[b]$, where $b \in \{0, 1\}$. The protocol description is given in [18] and its communication complexity is shown for both constant and variable round construction in the table below.

Bitwise addition. This is a special protocol for computing $[x + y]$ given x and y in their binary representation, where each bit is encrypted separately: $[x_0], \ldots, [x_m]$ and $[y_0], \ldots, [y_m]$. The protocol is outlined in [1].

Least significant bit (LSB). Schoenmakers and Tuyls [66] introduced a protocol for extracting the least significant bit of x, where both the input and output are encrypted. This protocol is also extended for extracting l least significant bits $[x_0], \ldots, [x_{l-1}]$. The round and communication complexity depends on which construction one uses for the random bit generation. This least significant bit extraction gate can be used to extract all bits of $[x]$, however, it offers only statistical privacy. For perfect privacy, use the following bit representation gate.

Bit representation. Given $[x]$ it extracts all of its bits in encrypted form: $[x_0], \ldots, [x_m]$, providing perfect privacy [66]. It has the same Big-Oh complexity as extracting least significant bits (where $l = m$), but with higher constants.

The operation costs of the protocols covered in this Section are summarized in the following table (k is a security parameter):

Operation	Rounds	Communication (bits)
Random element generation	2	$\mathcal{O}(nk)$
Inversion and modular division	$\mathcal{O}(1)$	$\mathcal{O}(nk)$
Unbounded fan-in multiplication	$\mathcal{O}(1)$	$\mathcal{O}(lnk)$
Random bit generation	$\mathcal{O}(1)$ $\mathcal{O}(n)$	$\mathcal{O}(n^2k)$ $\mathcal{O}(nk)$
Bitwise addition	$m + 1$ (depth)	$4mnk$
Least significant bits	$\mathcal{O}(l)$ $\mathcal{O}(n + l)$	$\mathcal{O}(ln^2k)$ $\mathcal{O}(lnk)$
Bit representation	$\mathcal{O}(m)$ $\mathcal{O}(n + m)$	$\mathcal{O}(mn^2k)$ $\mathcal{O}(mnk)$

4.3.4 Circuits for primitives

This Section covers the circuit construction for $sgn(x-y)$, $x > y$ and $x = y$ operations for bitwise encrypted m-bit x and y. All the constructions described in here are by Schoenmakers et al. [65]. It is important to notice that the circuit complexities are given in terms of depth and number of multiplication gates (e.g. conditional gates), whereas the circuit evaluation protocols are described in Section 4.3.1.

Operation	Circuit depth	# of multiplication gates
$sgn(x - y)$	m	$2m - 2$
Equality $(x = y)$	m	$2m - 1$
Inequality $(x > y)$	m	$2m - 1$

4.4 Two-party additive sharing

There exists a line of work mostly by Mikhail Atallah and his co-workers defining and using a set of composable operations over data shared over a ring $\mathbb{Z}_N$ by two parties (P_1 and P_2), where $\mathbb{Z}_N$ is a ring of plaintexts for a homomorphic encryption scheme. If Paillier's scheme [60] is used (as we assume in this section), then N is an RSA modulus and the ciphertexts belong to the set $\mathbb{Z}_{N^2}$. Let $k = \lceil \log N \rceil$. For a number $a \in \mathbb{Z}_N$ let $[a] = ([a]_1, [a]_2)$ be its representation. Here $[a]_1, [a]_2 \in \mathbb{Z}_N$ and $[a] \equiv [a]_1 + [a]_2 \pmod N$. Let $E_i(x)$ denote the encryption of the item $x \in \mathbb{Z}_N$ with the public key of party P_i for homomorphic encryption. In stating the computation costs of operations possible with shared numbers, we use the following notation.

- M_p — the cost of multiplying two plaintexts (two elements of $\mathbb{Z}_N$).

- M_c — the cost of multiplying two ciphertexts. As ciphertexts have twice the length of plaintexts, we have $M_c \approx 4M_p$.
- E_c — the cost of exponentiating a ciphertext with an element of $\mathbb{Z}_N$. We have $E_c \approx (3k/2) \cdot M_c$.
- E — the cost of an encryption. For Paillier cryptosystem, $\mathsf{E} = 2E_c + M_c$.
- D — the cost of a decryption. For Paillier cryptosystem, $\mathsf{D} = E_c + 2M_p$.
- OT — the cost of an oblivious transfer. For Lipmaa's homomorphic encryption based OT scheme [47], $\mathsf{OT} = 2\mathsf{E} + \mathsf{D} + 2E_c + M_c$.

Addition. To add two shared numbers $[a]$ and $[b]$, both parties add their shares. Cost — one addition; no communication is necessary.

Multiplying two private numbers. Let a be known to P_1 and b to P_2. The goal is to obtain $[c]$, where $c = ab$.

Party P_1 sends $E_1(a)$ to P_2. Party P_2 randomly generates $r \in \mathbb{Z}_N$ and sends back $E_1(a)^b \cdot E_1(r) = E_1(ab + r)$. Party P_1 decrypts and obtains $[c]_1 = ab + r$. Party P_2 sets $[c]_2 = -r$.

Multiplying two shared numbers. If $a = [a]_1 + [a]_2$ and $b = [b]_1 + [b]_2$ then $ab = [a]_1[b]_1 + [a]_2[b]_1 + [a]_1[b]_2 + [a]_2[b]_2$. The first and last of those can be computed by P_1 and P_2, respectively. For the middle two multiplications, the previous protocol (in parallel) has to be used.

Asymmetric blind-and-permute. Given a vector of shared numbers $([a_1], \ldots, [a_n])$, the goal of this protocol is to obtain a new vector $([a'_1], \ldots, [a'_n])$, such that $(a'_1, \ldots, a'_n)$ is a random permutation of $(a_1, \ldots, a_n)$, and only P_2 knows the permutation.

Party P_1 sends $E_1([a_1]_1), \ldots, E_1([a_n]_1)$ to P_2. Party P_2 picks a random permutation π of n elements, and random numbers $r_1, \ldots, r_n \in \mathbb{Z}_N$. Party P_2 computes $e_i = E_1([a_{\pi(i)}]_1) \cdot E_1(r_i) = E_1([a_{\pi(i)}]_1 + r_i)$ and sends $(e_1, \ldots, e_n)$ back to P_1. Party P_1 decrypts e_i and obtains $[a'_i]_1 = [a_{\pi(i)}]_1 + r_i$. Party P_2 sets $[a'_i]_2 = [a_{\pi(i)}]_2 - r_i$.

Blind-and-permute. Given a vector of shared numbers $([a_1], \ldots, [a_n])$, the goal of this protocol is to obtain a new vector $([a'_1], \ldots, [a'_n])$, such that $(a'_1, \ldots, a'_n)$ is a random permutation of $(a_1, \ldots, a_n)$ and neither party knows the permutation. To achieve this, asymmetric blind-and-permute is run once; and then for a second time with the roles of the parties reversed [45]. Damgård-Jurik encryption scheme [22] must be used, such that all parties can use the same modulus N.

Divide by public. Let $[a]$ be a shared number, let C be a public constant. The goal of the protocol is to obtain $[b]$, such that $b = \lceil a/C \rceil$ [3].

We assume that $a < 2^\ell$ for some $\ell < \log N$. Both P_1 and P_2 divide their share by C and rerandomize (they agree on a random number r that P_1 adds to, and P_2 subtracts from its share). If both $[a]_1$ and $[a]_2$ are less than 2^ℓ then this would complete the computations. If either $[a]_1 \geq 2^\ell$ or $[a]_2 \geq 2^\ell$ (or both), then $[a]_1 + [a]_2 = a + N$. When dividing by C, we get an error term N/C which must be subtracted from $[b]_1$ or $[b]_2$. This error term is subtracted by P_2 who will obtain it from P_1 by means of oblivious transfer — P_1 prepares the pair $(0, N/C)$ (if $[a]_1 < 2^\ell$) or $(N/C, N/C)$ (if $[a]_1 \geq 2^\ell$); P_2 will obtain the first (if $[a]_2 < 2^\ell$) or the second (if $[a]_2 \geq 2^\ell$) component of it.

The costs of operations are summarized in the following table.

Operation	Computation	Communication	Rounds
Add shared numbers	*negl.*	0	0
Multiply private numbers	$2\mathsf{E} + \mathsf{D} + M_{\mathrm{c}} + E_{\mathrm{c}}$	$4k$	2
Multiply shared numbers	$4\mathsf{E} + 2\mathsf{D} + 2M_{\mathrm{c}} + 2E_{\mathrm{c}} + 2M_{\mathrm{p}}$	$8k$	2
Blind-and-permute	$4n\mathsf{E} + 2nM_{\mathrm{c}} + 2n\mathsf{D}$	$4nk$	2
Divide by public	$2M_{\mathrm{p}} + \mathsf{OT}$	$6k$	2

This set of operations is extended with comparison (based on garbled circuits) and scaled reciprocal ($x \mapsto \lceil 2^{\ell}/x \rceil$; based on the Newton-Rhapson method for finding roots of functions). A number of applications have been built atop of this set of operations, including linear programming, collaborative planning, forecasting and replenishment, capacity allocation, etc.

Chapter 5

Server-assisted secure computation

In this chapter we review techniques for solving various computational problems, where the set-up and the techniques have the following form.

- A low-power entity (the *client*) has a computational problem C.
- The client has access to a different entity (the *server*) with much higher computational power. The client does not trust the server.
- The client generates some randomness R and constructs a different computational problem $C' = f(C, R)$ by applying the transformation f to C.
- The client sends C' to the server. The server solves C' and sends the solution s' back to the client.
- The client applies a transformation g to s', C and R and obtains $s = g(s', C, R)$ which is a solution to C.

The client does not trust the server, hence C' should not leak information about C. If the server is malicious then the client may also want to be able to verify that s' is a correct solution to C'.

We are interested in these techniques because they are applicable to SMC scenarios, particularly if a virtual machine based technology is used. In this case, the client's computations correspond to computations performed with private data, and server's computations to computations performed with public data. A possible issue is the different performance profile of the virtual machine, compared to the client platforms considered in the literature.

All protocols described here aim to securely implement the ideal functionalities $\mathcal{F}_f$ for various f-s with two inputs and two outputs, where the second input (from the server) is not used and the second output is always empty. They can be integrated into virtual machine based solutions as described above.

5.1 Linear algebra

5.1.1 Matrix multiplication

In [4], Atallah and Frikken propose a method for outsourcing the operation of multiplying two $n \times n$ matrices M_1 and M_2 (with entries from a field over which Shamir's secret sharing schemes [68] exist), such that the client only performs $O(n^2)$ operations with the entries of the matrices. The client shares each entry of both matrices using a polynomial of degree t (as we see below, here $t \in \mathbb{N}$ serves as a sort of *security parameter*). For matrix M_i, let $P_i(j)$ denote the matrix of the shares of all entries of M_i at the point j. Someone that knows $t+1$ matrices $P_i(j_1), \ldots, P_i(j_{t+1})$ and the points $j_1, \ldots, j_{t+1}$ is able to recover M_i.

The product of matrices $P_1(j) \cdot P_2(j)$ contains shares for the entries of $M_1 \cdot M_2$, where each share has been constructed using some polynomial of degree $2t$. The knowledge of $2t+1$ matrices $P_1(j_1) \cdot$

$P_2(j_1), \ldots, P_1(j_{2t+1}) \cdot P_2(j_{2t+1})$, as well as the points $j_1, \ldots, j_{2t+1}$ allows one to recover $M_1 \cdot M_2$ by linear interpolation.

The idea of Atallah and Frikken is to construct $2t+1$ shares of M_1 and M_2 using polynomials of degree t, and send all shares *to a single server*. By itself, this construction is obviously insecure. Thus Atallah and Frikken introduce a computational problem that postulates, that if enough chaff is added to these shares (linear in the amount of real data), then the server cannot figure out which pieces of data it has received are actual shares of M_1 and M_2, and which are random numbers.

Formally, the *Strong Secret Hiding Assumption* is the following. Let $U(t, e, m, p)$ denote the uniform distribution over matrices of size $(2t + 2e + 2) \times m$ with entries from $\mathbb{Z}_p$. Let $R(t, e, m, p)$ be a distribution over the same set of matrices, defined as follows:

- uniformly generate $t + e + 1$ different points $j_1, \ldots, j_{t+e+1}$ from $\mathbb{Z}_p^*$.
- uniformly generate m polynomials $f_1, \ldots, f_m$ of degree at most t over $\mathbb{Z}_p$, such that $f_i(0) = 0$ for all i.
- For $k \in \{1, \ldots, t + e + 1\}$, let R_k be the (row) vector $(f_1(j_k), \ldots, f_m(j_k))$.
- For $k \in \{t + e + 2, \ldots, 2t + 2e + 2\}$, let R_k be a row vector of size m with rows uniformly chosen from $\mathbb{Z}_p$.
- Output a matrix consisting of rows $R_1, \ldots, R_{2t+2e+2}$, taken in random order.

The Strong Secret Hiding Assumption states that for all m and e, if p scales reasonably with t, the families of probability distributions $\mathcal{U}_{m,e} = \{U(t, e, m, p)\}_{t\in\mathbb{N}}$ and $\mathcal{R}_{m,e} = \{R(t, e, m, p)\}_{t\in\mathbb{N}}$ are computationally indistinguishable. If this assumption holds then the matrix multiplication operation described below is privacy-preserving.

Verification To verify that server has performed the computations correctly, Atallah and Frikken suggest that the client submits not a single task of multiplying $n \times n$-matrices M_1 and M_2 to it, but several tasks simultaneously, with shares of different tasks in random order, such that the client already knows the solutions to all but one multiplication tasks.

Another, generic way to compare two $n \times n$ matrices A and B with $O(n^2)$ scalar operations is to generate a random vector v of length n and compare the vectors Av and Bv. If the elements of A, B and v belong to a field of size p, then the probability of $A \neq B$ and $Av = Bv$ simultaneously holding is at most $1/p$. Given three $n \times n$ matrices A, B and C, the verification of $AB = C$ can be performed by generating a random n-element vector v and checking that $A(Bv) = Cv$. Obviously, the check can be repeated to lower the error probability.

5.1.2 Iterative solution of linear equation systems

A cloud-assisted iterative solution of linear equation systems $Ax = b$ has been proposed by Wang et al. [73]. Here A is a non-singular $n \times n$ matrix, b is a (column) vector of length n and x is the vector of unknowns. The system can be represented as $(D + R)x = b$, where D is a diagonal matrix and $R = A - D$. This is equivalent to $x = D^{-1}b - D^{-1}Rx$; this system may be solved iteratively by defining $x^{(k+1)} = c + Tx^{(k)}$, where $c = D^{-1}b$ and $T = -D^{-1}R$.

Wang et al. first mask the vector of unknowns: the first step is to generate a secret random vector r, set $y = x + r$ and define a new system $A(y - r) = b$, which is equivalent to $Ay = b'$, where $b' = b + Ar$. This system will then be solved iteratively as $y^{(k+1)} = c' + Ty^{(k)}$, where $c' = D^{-1}b'$. The entries of the matrix T are encrypted using homomorphic encryption and sent to server. The client picks an initial estimate $y^{(0)}$.

At k-th iteration:

- The client sends $y^{(k)}$ to the server.
- The server uses homomorphic properties of the encryption to compute the encrypted entries of the vector $Ty^{(k)}$. These are sent back to the client.

- The client decrypts and adds the vector c', thus obtaining $y^{(k+1)}$.

The security analysis in [73] is informal. The approach is claimed to protect the actual solution x, because during the iteration, an independent value y is targeted. As the client sees the iterates $y^{(k)}$, it can directly observe whether they are approaching the zero vector.

5.1.3 Matrix inversion

To invert a $n \times n$ matrix M, the client generates a random invertible $n \times n$ matrix R and sends $M \cdot R$ to the server. The server replies with $(M \cdot R)^{-1} = R^{-1} \cdot M^{-1}$; the correctness of this computation can be verified by multiplying the result with MR and checking that it gives the unit matrix. The client computes $R \cdot (R^{-1} \cdot M^{-1}) = M^{-1}$. The client thus has to perform two $n \times n$ matrix multiplications. In the second multiplication, one of the arguments is public. This method was already published in [6]. A random invertible matrix can be generated as described in [63], at the cost that is dominated by one $n \times n$ matrix multiplication.

5.1.4 Linear programming

The task of linear programming is to minimize $c^T x$, subject to conditions $Ax \leq b$ and $x \geq 0$. Here x is an n-element vector of unknowns taking values from the set of real numbers. A is a $m \times n$ matrix, b is a m-element and c and n-element vector, all with real entries. The inequalities between vectors and matrices are defined pointwise.

In a more general setting, the conditions on x may be both linear inequalities and equalities. Also, typical solution algorithms transform inequalities to equalities by introducing extra *slack variables*. Hence the conditions on x may be given as $A_1 x = b_1$, $A_2 x \leq b_2$, $x \geq 0$, where A_i is a $m_i \times n$ matrix and b_i is a m_i-element vector.

Dreier and Kerschbaum [24] propose the following method for the client to hide A_1, A_2, b_1, b_2 and c from the server as follows.

- Let Q be a $n \times n$ *positive monomial* matrix. A square matrix is *monomial* if it contains exactly one non-zero entry in each row and in each column. It is *positive* if all those non-zero entries are positive.
- Let r be a positive random n-element vector.
- Let S be a strictly positive $n \times n$ diagonal matrix. I.e. it has positive entries on the main diagonal and zeroes elsewhere.
- Let P be a $(m_1 + m_2 + n) \times (m_1 + m_2 + n)$ nonsingular matrix.
- Define A', b' and c' as

$$A' = P \cdot \begin{pmatrix} M_1 Q & 0 \\ M_2 Q & I_{m_2+n} \\ -S & \end{pmatrix} \quad b' = P \cdot \begin{pmatrix} b_1 + M_1 Q r \\ b_2 + M_2 Q r \\ -Sr \end{pmatrix} \quad c' = \begin{pmatrix} Q^T c \\ 0 \end{pmatrix}$$

The server then has to solve the problem of minimizing $c'^T z$, subject to the constraints $A'z = b'$ and $z' \geq 0$, where z is a vector of unknowns of size $2n + m_2$.

Dreier and Kerschbaum perform a security analysis of the transformation, based on the notion of channel capacity [69]. Each operation in constructing A', b' and c' can be seen as a noisy channel, For certain definitions of noise, the bounds on the capacity of the composite channel can be found from the bounds on the elementary channels. More analysis is probably necessary before this solution can be applied in practice.

5.2 Set intersection

There exists a very simple protocol for server-aided two- or multiparty set intersection computation, if all parties are semi-honest. Let party P_i have the set $S_i = \{s_{i,1}, \ldots, s_{i,\ell_i}\}$. Let K be a key for a pseudorandom function F; K is shared by all parties P_i, but unknown to the server. Party P_i sends $F_K(s_{i,1}), \ldots, F_K(s_{i,\ell_i})$ to the server (randomly permuted). The server finds the intersection of all values received by different parties and communicates back the values $F_K(x)$ in the intersection. The only thing server learns are the cardinalities of the sets $S_{i_1} \cap \cdots \cap S_{i_k}$ for all combinations $(i_1, \ldots, i_k)$.

To verify that the server performs correctly, the parties replace each element x in their sets with η copies $x|1, \ldots, x|\eta$ of it, where η is the security parameter. Now the elements of the set intersection received from the server must contain all η copies of any element in the intersection. This verification fails only if the server claims that the intersection is empty, or that it contains all elements it received. To prevent this, the parties add dummy elements to their sets to rule out those two cases.

Kamara et al. [41] show how to make this construction secure also against malicious parties. The parties also have a second key K' shared between each other, they run the protocol described in the previous paragraph on the sets $F_{K'}(S_i) = \{F_{K'}(s_{i,1}), \ldots, F_{K'}(s_{i,\ell_i})\}$. But before the server makes the intersection result available to parties, it checks that all parties have followed protocol: the server first commits to the result and then all parties send K and the sets $F_{K'}(S_i)$ to the server. Only if these match the messages the server has received so far, will it open the committed result.

5.3 General server-aided computation from garbled circuits

In a two-party secure computation protocol based on garbled circuits, party P_1 generates the keys corresponding to both parties' inputs (as well as for all intermediate results), garbles the circuit, and sends the circuit and the keys for inputs to party P_2 (using oblivious transfer for P_2's inputs). Party P_2 evaluates the garbled circuit, learns the result of the computation and sends it back to P_1. It is possible to offload some of these computations to a third party (the server S).

Feige et al. [25] propose a setting where P_1 and P_2 share a common source of randomness (e.g. a random seed that can be fed to a pseudorandom generator) that is unknown to S. Party P_1 will use this source of randomness to generate the keys for inputs and intermediate results, and to garble the circuit. P_1 will then send the garbled circuit and keys corresponding to its own inputs to S. Party P_2 will also use that source of randomness to generate only the keys that correspond to its own inputs. Thanks to using the common source of randomness, P_1 and P_2 will generate the same keys for the same inputs. P_2 sends the keys it generated to S. The server S evaluates the garbled circuit, learns the output and communicates it back to P_1 and P_2. The protocol is secure against semi-honest adversaries.

Kamara et al. [41] improve the previous protocol in several aspects. First, they note that if the translation table for the outputs of the garbled circuit is not sent to S then S will learn the result of the computation only in encrypted fashion. The encryption can be sent back to P_1 and P_2 who learn the result. Second, cut-and-choose techniques can be used to gain security against deviations from the protocol. Also, the protocol can be extended to more than two parties if they all share the source of randomness.

5.4 General server-aided computation from FHE

Fully homomorphic encryption [31] is intended to be used in a manner where one party performs the computations, in this sense it is also a server-aided secure computation technique. It is intended to be used in the two-party setting where the client C encrypts its inputs with its own public key K_C, and sends the encrypted inputs a the server S. The server executes a circuit, possibly together with its own inputs, producing encrypted outputs. These are sent back to the client who can decrypt those.

López-Alt et al. [50] have shown how to extend this technique to the multi-party setting where several clients send their encrypted inputs to a server. Normally, there would-be the issue of different clients having

different public keys. To cope with this, Gentry [31] proposed the clients to execute separate SMC protocols to jointly generate the key, and to jointly decrypt the results. These protocols are made straightforward by the use of *key-homomorphic* FHE proposed in [50] as a combination of ideas from earlier papers. The proposed protocols have four rounds in the semi-honest and five rounds in the malicious setting. Because of the use of FHE, the communication and computation requirements of these protocols are currently not practical.

Chapter 6

Protocols for specific tasks

This chapter gives an overview of techniques for specific secure computation problems that have more efficient protocols than what would be obtainable from the generic constructions. The primary aim of this chapter is to cover different techniques, not different problems. Hence we do not specifically consider problems that can be solved by a clever combination of the techniques described in previous chapters.

As a rule, the techniques in this chapter provide secure implementations for ideal functionalities $\mathcal{F}_f$, where f is the function that we want to compute. An exception are the secure operations on multisets by Kissner and Song [42] described in Sec. 6.3. These operations work on secure encodings of multisets and deliver their results in the same format, allowing to securely realize a functionality similar to $\mathcal{F}_{\mathsf{ABB}}$.

6.1 Scalar product

Let party A have a vector v of length d with elements from a suitable ring R. Let party B have a vector w of the same length over the same ring. A private scalar product protocol allows A to learn only $v \cdot w$ with B learning nothing.

Scalar products can be easily evaluated using the cryptographic methods of previous chapters. Additionally, non-cryptographic methods have been proposed [71, 36], where the data is scrambled and later de-scrambled using operations of linear algebra. These methods can have much lower computational costs than methods based on e.g. homomorphic encryption, but their communication costs may be higher and they typically do not fully hide the vectors and the relationships between their elements.

Scalar product is often used as a sub-protocol in larger computations, e.g. if a vector $\{A_i\}_{i=1}^n$ has been additively shared among several parties, for example $A_i = A_{i,1} + A_{i,2}$; the sharing is between two parties, then scalar product helps to compute the sum $\sum_{i=1}^n A_i^2$. Indeed,

$$\sum_{i=1}^{n} A_i^2 = \sum_{i=1}^{n} (A_{i,1} + A_{i,2})^2 = \underbrace{\sum_{i=1}^{n} A_{i,1}^2}_{P_1 \text{ computes}} + \underbrace{\sum_{i=1}^{n} A_{i,2}^2}_{P_2 \text{ computes}} + 2\underbrace{\sum_{i=1}^{n} A_{i,1} A_{i,2}}_{\text{scalar prod.}}$$

The sum of squares is often required in e.g. geometric applications, where (the squares of) Euclidean distances between private points have to be computed.

6.2 Applying homomorphic properties of the Goldwasser-Micali encryption

For an RSA modulus $N = pq$, let $\mathcal{J}_N \subseteq \mathbb{Z}_N$ be the set of all $x \in \mathbb{Z}_N$, such that the Jacobi symbol $\left(\frac{x}{N}\right)$ equals 1. Let $\mathcal{QR}_N$ be the set of quadratic residues *modulo* N and $\mathcal{QNR}_N = \mathcal{J}_N \backslash \mathcal{QR}_N$. The *quadratic residuosity assumption* states that without the knowledge of p and q, the uniform distribution over $\mathcal{QR}_N$ is indistinguishable from the uniform distribution over $\mathcal{QNR}_N$.

The public-key encryption scheme introduced in the original paper by Goldwasser and Micali [33], the public key is (N, z), where N is an RSA modulus and $z \in \mathcal{QNR}_N$. The secret key are the factors p and q of N. Possible plaintexts are the bits 0 and 1, the ciphertexts belong to $\mathbb{Z}_N$. To encrypt a bit b, select a random $r \in \mathbb{Z}_N^*$ and output $z^b r^2 \in \mathbb{Z}_N$. In other words, the encryptions of 0 are random elements of $\mathcal{QR}_N$, while the encryptions of 1 are random elements of $\mathcal{QNR}_N$.

The following operations can be performed with the ciphertexts.

Rerandomize If $y \in \mathbb{Z}_N$ is a ciphertext then $y \cdot r^2$ is also a ciphertext encrypting the same bit, but otherwise uniformly distributed (if r is chosen uniformly from $\mathbb{Z}_N^*$).

XOR By the properties of the Jacobi symbol, if y_1 encrypts b_1 and y_2 encrypts b_2 then $y_1 \cdot y_2 \in \mathbb{Z}_N$ encrypts $b_1 \oplus b_2$.

Negation Similarly, if y encrypts b then $y \cdot z$ encrypts $\neg b$.

We call this version of the GM-encryption the *XOR-homomorphic encryption.*

We can also define a different version of GM-encryption which we call the *AND-homomorphic encryption.* Let κ be an additional security parameter. The encryption of bit 1 is a vector of length κ of uniformly randomly chosen elements of $\mathcal{QR}_N$ (computed by randomly selecting $r \in \mathbb{Z}_N$ and outputting r^2). The encryption of bit 0 is a vector of length κ of uniformly randomly chosen elements of $\mathcal{J}_N$ (computed by randomly selecting $r \in \mathbb{Z}_N$ until $\left(\frac{r}{N}\right) = 1$). With this scheme, decryption may fail, but only with probability $2^{-\kappa}$.

The following operations can be preformed with the ciphertexts.

Rerandomize XOR-rerandomize each component of the ciphertext. Then permute the elements of the vector.

AND Given two ciphertexts of the AND-homomorphic encryption, rerandomize them both (actually, it is sufficient to rerandomize only one of them), giving $Y_1 = (y_{1,1}, \ldots, y_{1,\kappa})$ and $Y_2 = (y_{2,1}, \ldots, y_{2,\kappa})$. Then output $(y_{1,1} \cdot y_{2,1}, \ldots, y_{1,\kappa} \cdot y_{2,\kappa})$. If at least one of the inputs represents the bit 1, then the operation works flawlessly. If both inputs are 0, then with probability $2^{-\kappa}$, the result may represent 1.

Finally, a XOR-ciphertext can be transformed into an AND-ciphertext. Given a XOR-ciphertext y, let each of the components $y_1, \ldots, y_\kappa$ of the AND-ciphertext be

- either a XOR-rerandomization of y_1;
- or a random element of $\mathcal{QR}_N$

with equal probability. In this way, if y XOR-encrypts the bit 1, then yz is an element of $\mathcal{QR}_N$. its rerandomizations are also elements of $\mathcal{QR}_N$. If y XOR-encrypts the bit 0 then yz is an element of $\mathcal{QNR}_N$. The components of the AND-ciphertext are thus randomly chosen from $\mathcal{J}_N$.

Let $F(x_1, \ldots, x_n)$ be a boolean formula of the form

$$F(x_1, \ldots, x_n) = \bigwedge_{i=1}^{m} (c_{i0} \oplus c_{i1} \wedge x_1 \oplus \cdots \oplus c_{in} \wedge x_n)$$

for some (public) $m, n \in \mathbb{N}$ and boolean $c_{10}, \ldots, c_{mn}$. Given XOR-encrypted bits $[b_1]_{\mathrm{XOR}}, \ldots, [b_n]_{\mathrm{XOR}}$, it is possible to compute the AND-encryption of $F(b_1, \ldots, b_n)$ without interacting with the owner of the secret key. One would first compute the XOR-encryptions of all exclusive OR-s in the formula, transform those to AND-encryptions, and compute the conjunction.

An important example of a formula F in almost such a form is the *less-than*-functionality. Let $a_1, \ldots, a_n$ and $b_1, \ldots, b_n$ be bit-wise representations of two natural numbers, with a_1, b_1 being the most significant bits. In this case

$$a > b \equiv \bigvee_{i=1}^{n} \left(a_i \wedge \neg b_i \wedge \bigwedge_{j=1}^{i-1} (a_j = b_j) \right) .$$

Here $(a_j = b_j)$ is equivalent to $\neg(a_j \oplus b_j)$. Note that if the disjunction is true then exactly one component $a_i \wedge \neg b_i \wedge \bigwedge_{j=1}^{i-1}(a_j = b_j)$ of the disjunction is true. We can thus compute all n components as AND-encryptions and then permute them (again, as AND-encryptions). If $a > b$ then the list of AND-encryptions contains exactly one AND-encryption of the bit 1. If $a \leq b$ then only 0-s are AND-encrypted.

The computation method described here is due to Fischlin [26].

6.3 Privacy-preserving set-theoretic operations

Kissner and Song [42] have proposed a set of representations and algorithms for manipulating multisets. The protection mechanism in their approach is based on threshold homomorphic encryption. The protocols are secure against semi-honest adversaries, but zero-knowledge proofs may be added to protect against malicious ones.

Let N be the modulus for addition, supported by the homomorphic encryption. We consider the multisets Υ of elements from a publicly known set $S \subseteq \mathbb{Z}_N$, such that $|S|/N$ is negligible.

Multisets are represented by polynomials $f \in \mathbb{Z}_N[x]$. A polynomial f represents a set Υ, such that $x \in S$ is an element of Υ iff $f(x) = 0$. Moreover, the multiplicity of $x \in \Upsilon$ is equal to the multiplicity of the root x of f. The polynomial f may have other roots outside the set S. These roots have no effect on the represented set Υ.

The following operations on multisets can be performed on their polynomial representations.

Union If f_i represents the multiset Υ_i then $\biguplus_i \Upsilon_i$ is represented by $\prod_i f_i$.

Intersection If f_i represents the multiset Υ_i ($i \in \{1, \ldots, n\}$) then $\bigcirc\!\!\!\!\!+_i \Upsilon_i$ (here $\oplus$ denotes the multiset intersection) is represented by $\gcd(f_1, \ldots, f_n)$, as well as by any multiple of it. In [42], random polynomials $g_1, \ldots, g_n$ are generated, where $\deg g_i = 2(\max_j \deg f_j) - \deg f_i$, and $\bigcap_i \Upsilon_i$ is represented by $\sum_i f_i g_i$.

Element reduction by d If Υ is a multiset then $\mathrm{Rd}_d(\Upsilon)$ denotes a multiset, such that each element $a \in S$ that occurs b times in Υ, occurs $\max\{b-d, 0\}$ times in $\mathrm{Rd}_d(\Upsilon)$. If Υ is represented by f then $\mathrm{Rd}_d(\Upsilon)$ can be represented by $\gcd(f, f', f'', \ldots, f^{(d)})$, where $f^{(i)}$ denotes the i-th derivative of f. A random multiple of this greatest common divisor is computed in the same way as computing the intersection.

In a privacy-preserving representation, each polynomial is represented as the sequence of encryptions of its coefficients (i.e. the degree of the polynomial, or an upper bound of it is public). Standard HE protocols are used to manipulate the polynomials. The representation also allows to determine whether some $a \in S$ belongs to some Υ represented by $[f]$. The value $[r \cdot f(a)]$ can be computed, where $r \in \mathbb{Z}_N$ is a collaboratively generated random number (i.e. no allowed coalition of parties knows its value). The decryption $r \cdot f(a)$ is 0 if $a \in \Upsilon$, and a random number otherwise.

If we have just two parties P_1 and P_2 holding sets S_1 and S_2, and the goal of the protocol is for P_1 to learn $S_1 \cap S_2$ (and P_2 to learn nothing at all) then protocols based on *oblivious pseudorandom functions (OPRF)* [53] can be used. An OPRF consists of a pseudorandom function family F and a protocol between two parties, the first of them holding the key k and the second holding the input x. After running the protocol, the second party learns $F_k(x)$, while the first party learns nothing. This can be turned into a set intersection protocol by letting P_2 select a key k, send $\{F_k(x) \,|\, x \in S_2\}$ to P_1, and then obliviously evaluate $F_k(y)$ for all $y \in S_1$ with P_1 learning the results. De Cristofaro et al. [20, 19] have presented several instantiations of that idea, where the total computational load is a couple of modular exponentiations per element of S_1 or S_2.

Brickell and Shmatikov [14] have proposed a protocol (even two of them) for privacy-preserving set union in a similar setting: parties P_1 and P_2 have the sets S_1 and S_2; both will learn the set $S_1 \cup S_2$ and nothing more. The protocol computes the set $S_1 \cup S_2$ element-by-element, starting from the smallest. Each round requires the computation of the minimum of two elements, one held by P_1 and another by P_2; the result

of the comparison is public. If garbled circuits are used for the computation, the whole protocol requires $O(|S_1 \cup S_2|)$ rounds and $O(\ell \cdot |S_1 \cup S_2|)$ computation, where ℓ is the bit-length of the elements of S_1 and S_2.

Neugebauer et al. [54] use the privacy-preserving multiset operations to implement the selection of a common choice from the sets of choices of a number of parties, such that the selected choice maximizes an utility function that is defined from the preference orders of each party on its set of choices. If parties $P_1, \ldots, P_n$ have sets $S_1, \ldots, S_n$ with preference relations $\leq_i$ (where the smallest is the most preferred), then let $r_i(a)$ be the rank of the element a for the party P_i, defined as follows. If $a \notin S_i$ then $r_i(a) = 0$. Otherwise, if a is the k-th *largest* elements of S_i then $r_i(a) = k$.

To find an element $c \in \bigcap_i S_i$ that maximizes the number $\min_i r_i(c)$, each party P_i will define the multiset Υ_i that contains each element $a \in S_i$ for $r_i(a)$ times. The parties will then collaboratively find such c as some element in $\mathrm{Rd}_t(\biguplus_i \Upsilon_i)$, where t is as large as possible. The protocol proceeds in rounds; in each round t is reduced by 1. In each round, a polynomial f representing $\mathrm{Rd}_t(\biguplus_i \Upsilon_i)$ is computed and made public. Each party P_i then checks whether some element of S_i is a root of f. If it is, the element c has been found.

To find an element $c \in \bigcap_i S_i$ that maximizes the number $\sum_i r_i(c)$, each party P_i will define the multiset Υ_i as above, as well as the multiset Υ'_i that contains nk copies of each $a \in S_i$, where k is some upper bound on the cardinalities of $S_1, \ldots, S_n$. The element c is the found as some element of the multiset $\mathrm{Rd}_t((\biguplus_i \Upsilon_i) \uplus \biguplus_i \Upsilon'_i)$, where t is as large as possible.

6.4 Parsing regular languages

Several authors have studied the problem of parsing, where both the input string as well as the language description is private. We give an overview of [9], where both the string and the finite automaton are additively shared between two parties P_1 and P_2. As the main operation in performing a step of a finite automaton is the selection of the next state based on the previous state and the next letter of the input string, oblivious transfer is the main component of the protocol.

Let the secret input string be $w = w_1 w_2 \cdots w_n$, where $x_i \in \Sigma$ for a public n and Σ. Let $m = |\Sigma|$. Let $(Q, \Sigma, \Delta, q_0, F)$ be a deterministic finite automaton, where the set of states Q, as well as the initial state q_0 is public, but the transition function $\Delta : Q \times \Sigma \to Q$ and the set of final states F are private. W.l.o.g we identify Σ with $\mathbb{Z}_{|\Sigma|}$, Q with $\mathbb{Z}_{|Q|}$ and set $q_0 = 0$.

The elements of Δ are shared additively *modulo* $|Q|$ between P_1 and P_2. The letters of the input string w are shared additively *modulo* $|\Sigma|$ between P_1 and P_2. Let q (secret) be the state of the DFA before parsing the k-th letter $w_k = x$ (secret) of the input string. Let $x = x_1 + x_2$, $q = q_1 + q_2$ and $\Delta = \Delta_1 + \Delta_2$ be the sharings of the current letter, state and the transition function between parties. The goal of the parties is to determine random q'_1 and q'_2, such that $q'_1 + q'_2 = \Delta(q, x) \bmod |Q|$.

- Party P_i generates a random number $r_i \in \mathbb{Z}_{|Q|}$ and defines Δ' by $\Delta'_i(a, b) = \Delta(a + q_i, b + x_i) - r_i$.
- P_1 and P_2 execute a 1-out-of-$|Q| \cdot |\Sigma|$ OT protocol, such that P_1 can learn $s_1 = \Delta'_2(q_1, x_1)$. A second execution of the OT protocol with reversed roles allows P_2 to learn $s_2 = \Delta'_1(q_2, x_2)$.
- P_i defines $q'_i = r_i + s_i$.

Hence the cost of parsing one letter is two parallel executions of 1-out-of-$|Q| \cdot |\Sigma|$ OT protocols. There are other protocols for performing the first transition (where the starting state q_0 is public) and for checking whether the final state belongs to F. These protocols are no more complex than the general transition protocol shown above.

Mohassel et. al [51] propose a method for obliviously evaluating a DFA if party P_1 has the input string w (with public length n) and party P_2 the finite automaton $(Q, \Sigma, \Delta, q_0, F)$. The method, similar to Yao's garbled circuits, allows party P_1 to find out whether P_2's automaton accepts the string w. Consider the "Δ-gate": it takes two inputs: $q \in Q$ and $x \in \Sigma$, and returns $\Delta(q, x) \in Q$. Similarly, we can consider the F-gate that takes $q \in Q$ as input and returns the bit $q \stackrel{?}{\in} F$. Party P_2 prepares a circuit with n Δ-gates and one F-gate. The first Δ-gate has the inputs q_0 (a constant) and w_1. The i-th Δ-gate has the output

of the $(i-1)$-st Δ-gate and w_i as its inputs. The F-gate takes the output of the last Δ-gate as its input. Party P_2 garbles this circuit (note that for each wire carrying n possible values, party P_2 needs to generate n keys). It sends the garbled circuit to P_1, as well as the keys corresponding to the inputs $w_1, \ldots, w_n$, using n instances of 1-out-of-$|\Sigma|$ oblivious transfer. Party P_1 evaluates the garbled circuit.

This protocol requires only 2 rounds of communication (equal to the round complexity of oblivious transfer). For each Δ-gate, party P_2 has to perform $2 \cdot |Q| \cdot |\Sigma|$ encryptions (or evaluations of the random oracle). Party P_1 has to perform a similar number of decryptions. The F-gate can be combined with the last Δ-gate.

6.5 Private database search

There exist specific protocols for the *Private Information Retrieval* (PIR) task where one party (the client) wants to access an item in the database held by the other party (the server), but the server may not know which item the client wants to have. If the client also may not learn anything about the other items in the database then we have an instance of the *Symmetric PIR* (SPIR) problem, which coincides with the oblivious transfer problem.

In [59], Ostrovsky and Skeith survey the PIR and SPIR protocols available at that time (2007). There exist PIR protocols where the communication complexity is just $O(\log^2 n)$, where n is the number of items in the database [47, 32]. If a total of m queries against the same database are provided, the communication complexity can be lowered to $O(m \log n)$ [34].

The computational complexity of a PIR protocol must be at least $O(n)$, as the server must "touch" all bits of the database. In the protocols referenced so far, this actually means that $O(n)$ public-key operations must be executed. If there are large chunks of constant data, then the number of public-key operations may actually be smaller, as shown by Lipmaa [49]. In his solution, the mapping of the index to the data item is seen as a function from natural numbers to bit-strings, a branching program is constructed to evaluate this function, and the evaluation is made secure according to the techniques described in Sec. 3.2. Frikken and Li [29] instead use a Boolean circuit (in the form of a search tree) to evaluate this function, and garble this circuit to obtain a secure evaluation method. The garbling can be done off-line.

In [5], Atallah and Li consider the following problem. There is a public $\sigma \times \sigma$ table S that contains values from the ring $\mathbb{Z}_N$ for some RSA-modulus N. There is a value $x \in \mathbb{Z}_\sigma$ and a vector μ with elements from $\mathbb{Z}_\sigma$. The value x and the elements of μ are additively shared between parties P_1 and P_2. The parties want to obtain a sharing of a vector T of length $|\mu|$, such that $T[i] = S[x, \mu[i]]$. We denote the shares of the i-th party by x_i, $\mu_i[j]$, $T_i[j]$.

Let E denote encryption with P_1's public key, using an encryption scheme that is additively homomorphic over $\mathbb{Z}_N$. Party P_1 prepares a $\sigma \times \sigma$ table $\hat{S}$, where $\hat{S}[i,j] = E(S[i + x_1, j])$; this table is send to P_2. Party P_2 discards all but the x_2-th row $\mathbf{v} = \hat{S}[x_2, *]$ of this table. For each $j \in \{1, \ldots, |\mu|\}$, party P_2 generates a random $T_2[j] \in \mathbb{Z}_N$, subtracts (under encryption) it from all components of $\mathbf{v}$ and circularly shifts $\mathbf{v}$ left by $\mu_2[j]$ places, resulting in the vector $\mathbf{v}_j$, where $\mathbf{v}_j[i] = E(S[x_1 + x_2, i + \mu_2[j]] - T_2[j])$. Parties P_1 and P_2 use 1-out-of-σ oblivious transfer, such that P_1 learns the $\mu_1[j]$-th element of $\mathbf{v}_j$. The decryption of this element will be $T_1[j]$.

The total communication complexity of this protocol is $O(\sigma^2 + \sigma \cdot |\mu|)$. The computational complexity may be reduced by the reductions between various OT protocols described below.

6.5.1 Extending oblivious transfers

The protocols described above may involve many oblivious transfers which may be computationally expensive. Ishai et al. [37] have shown how to use a few 1-out-of-2 OTs to obtain many instances of 1-out-of-2 OT.

Let sender S have m pairs of bit-strings $(x_{i,0}, x_{i,1})$, where $i \in \{1, \ldots, m\}$. Let the receiver R have m bits $b_1, \ldots, b_m$. Let b be the m-bit string $b_1 \cdots b_m$. The goal of R is to learn x_{i,b_i} for all i, without S learning

anything and R learning nothing about $x_{i,1-b_i}$. Let H be a random function, modeled as a random oracle. The protocol works as follows.

1. R randomly generates η m-bit strings $t_1, \ldots, t_\eta$, where η is a *security parameter*. Let $t_{i,j}$ be the j-th bit of the string t_i. S randomly generates η bits $s_1, \ldots, s_\eta$.

2. Using η instances of 1-out-of-2 OT, with R as the sender and S as the receiver, S obtains the bit-strings $w_i = t_i \oplus b^{s_i}$, where b^{s_i} is b, if $s_i = 1$, and the bitstring 0^m, if $s_i = 0$. Let $w_{i,j}$ be the j-th bit of the string w_i. By the definitions, $w_{i,j} = t_{i,j} \oplus (b_j \wedge s_i)$.

3. For all $j \in \{1, \ldots, m\}$, S computes the masks $M_{j,0} = H(j, w_{1,j}, \ldots, w_{\eta,j})$ and $M_{j,1} = H(j, w_{1,j} \oplus s_1, \ldots, w_{\eta,j} \oplus s_\eta)$. It sends to R the values $x_{j,0} \oplus M_{j,0}$ and $x_{j,1} \oplus M_{j,1}$.

We have

$$M_{j,b_j} = w_{i,j} \oplus (s_i \wedge b_j) = t_{i,j} \oplus (b_j \wedge s_i) \oplus (s_i \wedge b_j) = t_{i,j} \ .$$

The bits $t_{i,j}$ are known to R, hence it is able to compute M_{j,b_j} and recover x_{j,b_j}. On the other hand,

$$M_{j,\neg b_j} = t_{i,j} \oplus (b_j \wedge s_i) \oplus (s_i \wedge \neg b_j) = \begin{cases} t_{i,j}, & \text{if } s_i = 0 \\ \neg t_{i,j}, & \text{if } s_i = 1 \ . \end{cases}$$

To compute $M_{j,\neg b_j}$, R has to try out all possible values for the bits $(s_1, \ldots, s_\eta)$.

Ishai et al. [37] show that this protocol is secure against malicious S and semi-honest R. Security against malicious R can be obtained by using the cut-and-choose technique after the second step of the previous protocol, which exponentially magnifies the probability of detecting R's misbehaviour.

6.5.2 Efficient 1-out-of-n oblivious transfer

Naor and Pinkas [52] have proposed an 1-out-of-n oblivious transfer protocol, where the cost per protocol session is just two modular exponentiations for the receiver R and one modular exponentiation for the sender S (plus some less expensive operations). Additionally, there is a set-up phase, the cost of which can be amortized over many 1-out-of-n OT sessions.

Let $\mathbb{G}$ be a group with hard Diffie-Hellman problem. Let g be its generator. In the set-up phase, S chooses $n-1$ values $C_1, \ldots, C_{n-1} \in \mathbb{G}$ and sends them to R. It also chooses a random number α and sends $h = g^\alpha$ to R. The sender precomputes $C_1^\alpha, \ldots, C_{n-1}^\alpha$.

In the on-line phase, the sender has n messages $M_0, \ldots, M_{n-1}$ and the receiver has a number $i \in \{0, \ldots, n-1\}$. The receiver selects a random number r and sets $pk_i = g^r$. If $i \neq 0$, then it also computes $pk_0 = C_i / pk_i$. The receiver sends pk_0 to the sender. The sender computes $PK_0 = pk_0^\alpha$ and $PK_i = C_i^\alpha / PK_0$ for all $i \in \{1, \ldots, n-1\}$. The sender chooses a random string R and sends it and $H(PK_i, R, i) \oplus M_i$ back to the receiver (for all $i \in \{0, \ldots, n-1\}$). Here H is a hash function, modeled as a random oracle. The receiver can recompute PK_i as h^r.

6.6 Privacy-preserving Graph Algorithms

In [14], Brickell and Shmatikov consider privacy-preserving graph algorithms in the following setting. There are two parties P_1 and P_2, and a public graph $G = (V, E)$. Each P_i has a private weight function $w_i : E \to \mathbb{N}$. Consider the weight function w, where $w(e) = \min\{w_1(e), w_2(e)\}$, and the weighted graph (G, w). Solve a graph-theoretic problem on (G, w), such that the solution becomes known to both P_1 and P_2, but nothing more leaks about w_1, w_2.

The considered problems are all-pairs shortest distance (APSD), single-source shortest distance (SSSD) and minimum spanning tree. For the last two problems, existing algorithms (Dijkstra; Kruskal or Prim) can be implemented in privacy-preserving manner. For APSD, a new algorithm is proposed, fixing the distances between pairs of vertices one-by-one (actually, all pairs with the same distance are fixed together). If k is the

number of different distances in the solution of the APSD problem and ℓ is the bit-length of weights, then the protocol has $O(k)$ rounds and $O(k(\ell + \log k))$ communication and computation complexity. Brickell and Shmatikov [14] note that in this security model, where the result of the APSD computation is made public, that result contains significant amount of information that helps in the construction of the simulator.

Bibliography

[1] SecureSCM. Technical report D9.1: Secure Computation Models and Frameworks. `http://www.securescm.org`, July 2008.

[2] 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.

[3] Mikhail J. Atallah, Marina Bykova, Jiangtao Li, Keith B. Frikken, and Mercan Topkara. Private collaborative forecasting and benchmarking. In Vijay Atluri, Paul F. Syverson, and Sabrina De Capitani di Vimercati, editors, *WPES*, pages 103–114. ACM, 2004.

[4] 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.

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

[6] Judit Bar-Ilan and Donald Beaver. Non-cryptographic fault-tolerant computing in constant number of rounds of interaction. In *PODC*, pages 201–209, 1989.

[7] Mauro Barni, Pierluigi Failla, Vladimir Kolesnikov, Riccardo Lazzeretti, Ahmad-Reza Sadeghi, and Thomas Schneider. Secure evaluation of private linear branching programs with medical applications. In Michael Backes and Peng Ning, editors, *ESORICS*, volume 5789 of *Lecture Notes in Computer Science*, pages 424–439. Springer, 2009.

[8] Donald Beaver, Silvio Micali, and Phillip Rogaway. The round complexity of secure protocols (extended abstract). In Harriet Ortiz, editor, *STOC*, pages 503–513. ACM, 1990.

[9] Marina Blanton and Mehrdad Aliasgari. Secure outsourcing of dna searching via finite automata. In Sara Foresti and Sushil Jajodia, editors, *DBSec*, volume 6166 of *Lecture Notes in Computer Science*, pages 49–64. Springer, 2010.

[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 multiparty computation for data mining applications. *International Journal of Information Security*, 2011. Submitted.

[12] 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.

[13] Justin Brickell, Donald E. Porter, Vitaly Shmatikov, and Emmett Witchel. Privacy-preserving remote diagnostics. In Peng Ning, Sabrina De Capitani di Vimercati, and Paul F. Syverson, editors, *ACM Conference on Computer and Communications Security*, pages 498–507. ACM, 2007.

[14] 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.

[15] Ran Canetti. Security and composition of multiparty cryptographic protocols. *J. Cryptology*, 13(1):143–202, 2000.

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

[17] Ronald Cramer, Ivan Damgård, and Ueli M. Maurer. General secure multi-party computation from any linear secret-sharing scheme. In Bart Preneel, editor, *EUROCRYPT*, volume 1807 of *Lecture Notes in Computer Science*, pages 316–334. Springer, 2000.

[18] 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.

[19] Emiliano De Cristofaro, Jihye Kim, and Gene Tsudik. Linear-complexity private set intersection protocols secure in malicious model. In Masayuki Abe, editor, *ASIACRYPT*, volume 6477 of *Lecture Notes in Computer Science*, pages 213–231. Springer, 2010.

[20] Emiliano De Cristofaro and Gene Tsudik. Practical private set intersection protocols with linear complexity. In Radu Sion, editor, *Financial Cryptography*, volume 6052 of *Lecture Notes in Computer Science*, pages 143–159. Springer, 2010.

[21] 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.

[22] Ivan Damgård and Mads Jurik. A length-flexible threshold cryptosystem with applications. In Reihaneh Safavi-Naini and Jennifer Seberry, editors, *ACISP*, volume 2727 of *Lecture Notes in Computer Science*, pages 350–364. Springer, 2003.

[23] Ivan Damgård and Jesper Buus Nielsen. Universally composable efficient multiparty computation from threshold homomorphic encryption. In Boneh [12], pages 247–264.

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

[25] Uriel Feige and Joe Kilian. Two prover protocols: low error at affordable rates. In Frank Thomson Leighton and Michael T. Goodrich, editors, *STOC*, pages 172–183. ACM, 1994.

[26] Marc Fischlin. A cost-effective pay-per-multiplication comparison method for millionaires. In David Naccache, editor, *CT-RSA*, volume 2020 of *Lecture Notes in Computer Science*, pages 457–472. Springer, 2001.

[27] Pierre-Alain Fouque, Jacques Stern, and Jan-Geert Wackers. Cryptocomputing with rationals. In Matt Blaze, editor, *Financial Cryptography*, volume 2357 of *Lecture Notes in Computer Science*, pages 136–146. Springer, 2002.

[28] Matthew K. Franklin and Stuart Haber. Joint encryption and message-efficient secure computation. *J. Cryptology*, 9(4):217–232, 1996.

[29] Keith B. Frikken and Boyang Li. Private database search with sublinear query time. In Yingjiu Li, editor, *DBSec*, volume 6818 of *Lecture Notes in Computer Science*, pages 154–169. Springer, 2011.

[30] Strange L. From and Thomas Jakobsen. Secure multi-party computation on integers. *Master's thesis, University of Aarhus, Denmark, BRICS, Department of Computer Science*, 2006.

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

[32] Craig Gentry and Zulfikar Ramzan. Single-database private information retrieval with constant communication rate. In Luís Caires, Giuseppe F. Italiano, Luís Monteiro, Catuscia Palamidessi, and Moti Yung, editors, *ICALP*, volume 3580 of *Lecture Notes in Computer Science*, pages 803–815. Springer, 2005.

[33] Shafi Goldwasser and Silvio Micali. Probabilistic encryption. *J. Comput. Syst. Sci.*, 28(2):270–299, 1984.

[34] Jens Groth, Aggelos Kiayias, and Helger Lipmaa. Multi-query computationally-private information retrieval with constant communication rate. In Nguyen and Pointcheval [55], pages 107–123.

[35] Martin Hirt and Jesper Buus Nielsen. Robust multiparty computation with linear communication complexity. In Cynthia Dwork, editor, *CRYPTO*, volume 4117 of *Lecture Notes in Computer Science*, pages 463–482. Springer, 2006.

[36] Ioannis Ioannidis, Ananth Grama, and Mikhail J. Atallah. A secure protocol for computing dot-products in clustered and distributed environments. In *ICPP*, pages 379–384. IEEE Computer Society, 2002.

[37] Yuval Ishai, Joe Kilian, Kobbi Nissim, and Erez Petrank. Extending oblivious transfers efficiently. In Boneh [12], pages 145–161.

[38] Yuval Ishai and Anat Paskin. Evaluating branching programs on encrypted data. In Salil P. Vadhan, editor, *TCC*, volume 4392 of *Lecture Notes in Computer Science*, pages 575–594. Springer, 2007.

[39] Markus Jakobsson and Ari Juels. Mix and match: Secure function evaluation via ciphertexts. In Tatsuaki Okamoto, editor, *ASIACRYPT*, volume 1976 of *Lecture Notes in Computer Science*, pages 162–177. Springer, 2000.

[40] Stanislaw Jarecki and Vitaly Shmatikov. Efficient two-party secure computation on committed inputs. In Moni Naor, editor, *EUROCRYPT*, volume 4515 of *Lecture Notes in Computer Science*, pages 97–114. Springer, 2007.

[41] Seny Kamara, Payman Mohassel, and Mariana Raykova. Outsourcing multi-party computation. Cryptology ePrint Archive, Report 2011/272, 2011. `http://eprint.iacr.org/`.

[42] Lea Kissner and Dawn Xiaodong Song. Privacy-preserving set operations. In Victor Shoup, editor, *CRYPTO*, volume 3621 of *LNCS*, pages 241–257. Springer, 2005.

[43] Louis Kruger, Somesh Jha, Eu-Jin Goh, and Dan Boneh. Secure function evaluation with ordered binary decision diagrams. In Ari Juels, Rebecca N. Wright, and Sabrina De Capitani di Vimercati, editors, *ACM Conference on Computer and Communications Security*, pages 410–420. ACM, 2006.

[44] Sven Laur and Helger Lipmaa. On the feasibility of consistent computations. In Nguyen and Pointcheval [55], pages 88–106.

[45] Jiangtao Li and Mikhail Atallah. Secure and Private Collaborative Linear Programming. In Ling Liu, Carla Simone, Enrico Blanzieri, and Tao Zhang, editors, *Proc. 2nd International Conference on Collaborative Computing: Networking, Applications and Worksharing (CollaborateCom 2006)*, pages 19–26, 2006.

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

[47] Helger Lipmaa. An oblivious transfer protocol with log-squared communication. In Jianying Zhou, Javier Lopez, Robert H. Deng, and Feng Bao, editors, *ISC*, volume 3650 of *Lecture Notes in Computer Science*, pages 314–328. Springer, 2005.

[48] Helger Lipmaa. Private branching programs: On communication-efficient cryptocomputing. *IACR Cryptology ePrint Archive*, 2008:107, 2008.

[49] 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.

[50] Adriana Lopez-Alt, Eran Tromer, and Vinod Vaikuntanathan. Cloud-assisted multiparty computation from fully homomorphic encryption. Cryptology ePrint Archive, Report 2011/663, 2011. `http://eprint.iacr.org/`.

[51] Payman Mohassel, Salman Niksefat, Seyed Saeed Sadeghian, and Babak Sadeghiyan. An efficient protocol for oblivious dfa evaluation and applications. In Orr Dunkelman, editor, *CT-RSA*, volume 7178 of *Lecture Notes in Computer Science*, pages 398–415. Springer, 2012.

[52] Moni Naor and Benny Pinkas. Efficient oblivious transfer protocols. In S. Rao Kosaraju, editor, *SODA*, pages 448–457. ACM/SIAM, 2001.

[53] Moni Naor and Omer Reingold. Number-theoretic constructions of efficient pseudo-random functions. In *FOCS*, pages 458–467. IEEE Computer Society, 1997.

[54] Georg Neugebauer, Ulrike Meyer, and Susanne Wetzel. Fair and privacy-preserving multi-party protocols for reconciling ordered input sets. In Mike Burmester, Gene Tsudik, Spyros S. Magliveras, and Ivana Ilic, editors, *ISC*, volume 6531 of *Lecture Notes in Computer Science*, pages 136–151. Springer, 2010.

[55] Phong Q. Nguyen and David Pointcheval, editors. *Public Key Cryptography - PKC 2010, 13th International Conference on Practice and Theory in Public Key Cryptography, Paris, France, May 26-28, 2010. Proceedings*, volume 6056 of *Lecture Notes in Computer Science*. Springer, 2010.

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

[57] Takashi Nishide and Kazuo Ohta. Multiparty computation for interval, equality, and comparison without bit-decomposition protocol. In Okamoto and Wang [58], pages 343–360.

[58] Tatsuaki Okamoto and Xiaoyun Wang, editors. *Public Key Cryptography - PKC 2007, 10th International Conference on Practice and Theory in Public-Key Cryptography, Beijing, China, April 16-20, 2007, Proceedings*, volume 4450 of *Lecture Notes in Computer Science*. Springer, 2007.

[59] Rafail Ostrovsky and William E. Skeith III. A survey of single-database private information retrieval: Techniques and applications. In Okamoto and Wang [58], pages 393–411.

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

[61] 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.

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

[63] Dana Randall. Efficient generation of random nonsingular matrices. *Random Struct. Algorithms*, 4(1):111–118, 1993.

[64] T. Schneider. Practical secure function evaluation. *Master's thesis, University of Erlangen-Nuremberg*, 2008.

[65] Berry Schoenmakers and Pim Tuyls. Practical two-party computation based on the conditional gate. In Pil Joong Lee, editor, *ASIACRYPT*, volume 3329 of *Lecture Notes in Computer Science*, pages 119–136. Springer, 2004.

[66] 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.

[67] Axel Schröpfer and Florian Kerschbaum. Forecasting run-times of secure two-party computation. In *QEST*, pages 181–190. IEEE Computer Society, 2011.

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

[69] Geoffrey Smith. On the foundations of quantitative information flow. In Luca de Alfaro, editor, *FOSSACS*, volume 5504 of *Lecture Notes in Computer Science*, pages 288–302. Springer, 2009.

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

[71] Jaideep Vaidya and Chris Clifton. Privacy preserving association rule mining in vertically partitioned data. In *KDD*, pages 639–644. ACM, 2002.

[72] Brigitte Vallée. Gauss' algorithm revisited. *Journal of Algorithms*, 12(4):556–572, 1991.

[73] Cong Wang, Kui Ren, Jia Wang, and Karthik Mahendra Raje Urs. Harnessing the cloud for securely solving large-scale systems of linear equations. In *ICDCS*, pages 549–558. IEEE Computer Society, 2011.

[74] I. Wegener. *Branching programs and binary decision diagrams: theory and applications.* Number 4 in SIAM Monographs on Discrete Mathematics and Applications. Society for Industrial Mathematics, 2000.

[75] Andrew C. Yao. How to generate and exchange secrets (extended abstract). In *27th Annual Symposium on Foundations of Computer Science*, pages 162–167, Toronto, Ontario, Canada, October 27–29, 1986. IEEE Computer Society Press.