D13.3
The Full Set of New Protocols

Project number:	609611
Project acronym:	**PRACTICE**
Project title:	Privacy-Preserving Computation in the Cloud
Project Start Date:	1st November, 2013
Duration:	36 months
Programme:	FP7/2007-2013
Deliverable Type:	Report
Reference Number:	ICT-609611 / D13.3 / 1.0
Activity and WP:	Activity 1 / WP13
Due Date:	October 2016 - M36
Actual Submission Date:	3rd November, 2016
Responsible Organisation:	BIU
Editor:	Benny Pinkas
Dissemination Level:	PU
Revision:	1.0
Abstract:	This document describes the full set of new secure computation protocols that were designed by the partners of the PRACTICE project. The document has short descriptions of protocols that were already published in deliverable D13.1, and a detailed description of protocols that were designed in the last year of the project.
Keywords:	Secure multi-party comptutation.

This project has received funding from the European Union's Seventh Framework Programme for research, technological development and demonstration under grant agreement no. 609611.

Editor

Benny Pinkas (BIU)

Contributors (ordered according to beneficiary numbers)

Florian Hahn (SAP)
Florian Kerschbaum (SAP)
Agnes Kiss (TUDA)
Thomas Schneider (TUDA)
Michael Zohner (TUDA)
Pille Pullonen (CYBER)
Claudio Orlandi (AU)

Executive Summary

This report describes the full set of secure multi-party computation protocols that were designed by the members of the PRACTICE project. The main goal of these protocols is to enhance the performance and scalability of the available secure multi-party computation solutions, in order to address the needs of the field, which were described in Deliverable D11.2 of this project. The results that are presented in this report have been published in multiple research papers at top-tier conferences.

The document contains short descriptions of protocols that were already published in deliverable D13.1, which was published after the first two years of the project, and a detailed description of protocols that were designed in the final year of the project.

The first chapter of this report is an introduction. The second chapter describes new methods for generic multi-party computation (generic meaning that these methods can be used for computing arbitrary functions). The third chapter describes improvements to different tools and primitives that are used in secure computation. These improvements affect the performance of each secure computation protocol that will use these tools. The fourth chapter of this report describes new protocols for order preserving encryption, that is a crucial tool for storing encrypted databases and then performing queries on the encrypted data. The fifth and final chapter describes new methods for computing private set intersection, which is a secure protocol which, rather then being generic, solves a specific problem of high interest.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

The main task of WP13 is the specification and design of new protocols, which are intended to improve the state-of-the-art in secure multi-party computation, in directions that are most relevant to secure computation in the cloud. Deliverable D11.2 of WP11 (An evaluation of current secure computation protocols) identified two main issues where current protocols are lacking: (1) the scalability of protocols for generic secure computation, and in particular protocols that are secure against malicious adversaries; (2) there are specific tasks, specifically private set intersection (PSI), where generic protocols are considerably less efficient (perhaps by orders of magnitude) than is required.
The work that is described in this deliverable was carried out by many members of WP13. The work mostly focused on addressing the two issues that were highlighted by deliverable D11.2. Most of the results that were achieved improved the performance and scalability of protocols for generic secure computation. Other results dramatically improved the state-of-the-art protocols for specific tasks, particularly for private set intersection (PSI), and for order preserving encryption, which is an essential tool for secure database queries.
The document contains short descriptions of protocols that were already published in deliverable D13.1, which was published after the first two years of the project, and a detailed description of protocols that were designed in the final year of the project.
The results that are described in this deliverable were published in the most prestigious conferences in security, as is detailed in Section 1.2.

1.1 Contents

Chapter 2 describes new methods for generic multi-party computation (generic meaning that these methods can be used for computing arbitrary functions). These new methods has considerable improved efficiency compared to the former state of the art. Section 2.1 describes a version of Yao's secure computation protocol, which is currently the basis for most secure computation solutions. The new protocol makes use of a new garbling method, which is based on standard assumptions (whereas previous state-of-the-art garbling methods depended on less established cryptographic assumptions). Section 2.2 describes the first constant round secure multi-party computation protocol that is secure against malicious adversaries and has an efficient concrete overhead. Section 2.3 describes a framework for efficient mixed-protocol two-party secure computation: secure computation protocols typically use either arithmetic circuits, where the primitive operations are addition and multiplication, or boolean circuits, where the primitive operations are XOR and AND. Each of these protocol types is better at computing different types of functions. The new framework is the first to enable easy and efficient computation

which combines protocols of both types. Section 2.4 describes new protocols for private function evaluation. Namely, for secure computation which hides the function that is being computed. Section 2.5 describes the TinyTable protocol for secure two-party computation, for maliciously secure two-party computation in the preprocessing model.
Chapter 3 describes improvements to different tools and primitives that are used in secure computation. These improvements affect the performance of each secure computation protocol that will use these tools. Section 3.1 describes a new and extremely simple oblivious transfer protocol. Section 3.2 describes the first efficient oblivious transfer extension protocol that is secure against malicious adversaries. Section 3.3 describes recent improvements of oblivious transfer extension protocols. Section 3.4 describes an efficient implementation of protocols of the GMW family using hardware tokens. Section 3.5 described a new protocol stack for secure unsigned arithmetic computation for two parties, supporting more basic operations than just addition and multiplication, and thus supporting more efficient implementations. Two-Party Unsigned Arithmetic Based on Additive Secret Sharing Section 3.6 describes how to use garbled circuits for running very efficient zero-knowledge proofs of non-arithmetic statements. Section 3.7 describes practically efficient zero-knowledge arguments especially tailored for Boolean circuits.
Chapter 4 describes two new protocols for order preserving encryption, that is a crucial tool for storing encrypted databases and then performing queries on the encrypted data. The first protocol, in Section 4.1 improves the performance of the currently available order preserving encryption protocols. The second protocol, in Section 4.2, achieves a strictly stronger notion of security than any other order-preserving encryption scheme.
Chapter 5 describes improved new methods for private set intersection, that are based on using oblivious transfer extension and advanced hashing schemes. The chapter describes detailed experiments showing a performance improvement of more than an order of magnitude. Section 5.1 describes the most efficient up-to-date PSI protocols, which are based on oblivious transfer extension. Section 5.2 describes new and improved protocols for PSI.

1.2 Publications

The results described in this document were published in the following publications, which were published in the top academic security conferences:

- Ad-Hoc Secure Two-Party Computation on Mobile Devices using Hardware Tokens. Daniel Demmler, Thomas Schneider, and Michael Zohner. Usenix Security 2014.
- Optimal Average-Complexity Ideal-Security Order-Preserving Encryption. Florian Kerschbaum, and Axel Schroepfer. ACM Computer and Communication Security 2014.
- Frequency-Hiding Order-Preserving Encryption. Florian Kerschbaum, ACM Computer and Communication Security 2014.
- ABY – a Framework for Efficient Mixed-Protocol Secure Two-Party Computation. Daniel Demmler, Thomas Schneider and Michael Zohner. NDSS 2015.
- More Efficient Oblivious Transfer Extensions with Security for Malicious Adversaries. Gilad Asharov, Yehuda Lindell, Thomas Schneider, and Michael Zohner. Eurocrypt 2015.
- The Simplest Protocol for Oblivious Transfer. Tung Chou and Claudio Orlandi. Latincrypt 2015.

- Efficient Constant Round Multi-Party Computation Combining BMR and SPDZ. Yehuda Lindell, Benny Pinkas, Nigel P. Smart, and Avishay Yanai. Crypto 2015.
- Fast Garbling of Circuits Under Standard Assumptions. Shay Gueron, Yehuda Lindell, Ariel Nof, and Benny Pinkas. ACM Computer and Communication Security 2015.
- Valiant's Universal Circuit is Practical. Agnes Kiss and Thomas Schneider. Eurocrypt 2016.
- Gate-scrambling Revisited - or: The TinyTable protocol for 2-Party Secure Computation. Ivan Damgaard, Jesper Buus Nielsen, Michael Nielsen and Samuel Ranellucci. IACR Cryptology ePrint Archive 2016/695.
- ZKBoo: Faster Zero-Knowledge for Boolean Circuits. Irene Giacomelli, Jesper Madsen and Claudio Orlandi, USENIX Security Symposium 2016.

Chapter 2

Improved Secure Computation Protocols

2.1 Fast Garbling of Circuits Under Standard Assumptions (short description)

This section is based on [29]. The complete description of this work appears in deliverable D13.1.
A highly important tool in the design of two-party protocols is Yao's garbled circuit construction [68], and multiple optimizations on this primitive have led to performance improvements of orders of magnitude over the last years. However, many of these improvements come at the price of making very strong assumptions on the underlying cryptographic primitives being used. The justification behind making these strong assumptions has been that otherwise it is not possible to achieve fast garbling and thus fast secure computation. This work takes a step back and examines whether it is really the case that such strong assumptions are needed. It provides new methods for garbling that are secure solely under the well established assumption that the primitive used (e.g., AES) is a pseudorandom function. The results show that in many cases, the penalty incurred is not significant, and so a more conservative approach to the assumptions being used can be adopted.

2.1.1 The Results

This work constructed fast garbling methods solely under the assumption that AES behaves like a pseudorandom function. In particular, it does not use fixed-key AES, and uses two AES encryptions per entry in the garbled gates (since using just one encryption requires some sort of related-key security assumption). In addition, it does not use the free-XOR optimization (since this requires a non-standard circularity assumption). In brief, the following improvements are presented:

- **Fast AES-NI without fixing the key:** AES-NI is a hardware instruction supported on modern Intel chips, and implements very efficient AES encryption. AES-NI can greatly benefit from pipelining the blocks that need to be encrypted. In this new work it is shown that, in addition to pipelining encryptions, it is also possible to pipeline the key schedule of AES-NI, in order to achieve very fast garbling times without using a fixed key or any other non-standard AES variant. Namely, the key schedule processing of different keys can be pipelined together, so that the amortized effect of key scheduling on Yao garbling

is greatly reduced. Experiments (described below) show that this and other optimizations of AES operations have become so fast that the benefits of using fixed-key AES are almost insignificant. Thus, in contrast to current popular belief, in most cases fixed-key AES is ***not*** necessary for achieving extremely fast garbling.

- **Low-communication XOR gates:** Over the past years, it has become apparent that in secure protocols, communication is far more problematic than computation. The free-XOR technique is so attractive exactly because it requires no computation but also *no communication* for XOR gates. The paper provides a new garbling method for XOR gates that requires storing only a single ciphertext per XOR gate; the technique is inspired by the work of [42]. The computational cost is 3 AES computations for garbling the gate, and 1-2 AES computations for evaluating it.

- **Fast 4-2 row reduction:** Since the free-XOR technique is no longer used, it is possible to use 4-2 row reduction (GRR) on the non-XOR gates. However, the 4-2 row reduction method of [63] that uses polynomial interpolation is rather complex to implement (requiring finite field operations and precomputation of special constants to make it fast). In addition, even working in $GF(2^n)$ Galois fields and using the PCLMULQDQ Intel instruction, the cost is still approximately half an AES computation. The paper presents a new method for 4-2 row reduction that uses a few XOR operations only, and is trivial to implement.

The paper described implementations of these optimizations and compared them to the well known JustGarble library which is based on non-standard assumptions [8]. There is no doubt that the cost of garbling and evaluation is higher using the new and safer methods, since they have to run AES key schedules, and we pay for computing XOR gates. However, within protocol executions, the difference in performance is insignificant. This is demonstrated by running Yao's protocol for semi-honest adversaries which has nothing but oblivious transfer (for which the fast OT extensions of [2] are used), garbled-circuit evaluation and computation, and communication.

Patent-free garbled circuits. Another considerable advantage of using the new method for computing XOR gates with low communication is that it does not rely on the free XOR technique and thus is not patented. Since patents in cryptography are typically an obstacle to adoption, the search for efficient garbling techniques that are not patented is of great importance.

2.2 Efficient Constant Round Multi-Party Computation Combining the BMR and SPDZ Protocols (short description)

This section is based on [47]. The complete description of this work appears in deliverable D13.1.
There are extremely efficient variants of Yao's protocol for the two party case that are secure against malicious adversaries (e.g., [45,48]). These protocols run in a constant number of rounds and therefore remain fast over slow networks. The BMR protocol [7] is a variant of Yao's protocol that runs in a multi-party setting with more than two parties. This protocol works by the parties jointly constructing a garbled circuit (possibly in an offline phase), and then later

computing it (possibly in an online phase). However, in the case of malicious adversaries this protocol suffers from two main drawbacks: (1) Security is only guaranteed if at most a *minority* of the parties are corrupt; (2) The protocol uses generic protocols secure against malicious adversaries (say, the GMW protocol) that evaluate the pseudorandom generator used in the BMR protocol. This non black-box construction results in an extremely high overhead.
The TinyOT and SPDZ protocols [17,56] follow the GMW paradigm, and have offline and online phases. Both of these protocols overcome the issues of the BMR protocol in that they are secure against any number of corrupt parties, make only black-box usage of cryptographic primitives, and have very fast online phases that require only very simple (information theoretic) operations. In the case of multi-party computation with more than two parties, these protocols are currently the *only practical* approach known. However, since they follow the GMW paradigm, their online phase requires a communication round for every multiplication gate. This results in a large amount of interaction and high latency, especially over slow networks. To sum up, there is no known concretely efficient constant-round protocol for the multi-party case (with the exception of [13] that considers the specific three-party case only). This new work introduces the first protocol with these properties.

Contribution This work provides the first *concretely efficient constant-round* protocol for the general *multi-party* case, with security in the presence of malicious adversaries.
The basic idea behind the construction is to use an efficient non-constant round protocol – with security for malicious adversaries – to compute the gate tables of the BMR garbled circuit (and since the computation of these tables is of constant depth, this step is constant round). A crucial observation, resulting in a great performance improvement, shows that in the offline stage it is *not required to verify the correctness* of the computations of the different tables. Rather, validation of the correctness is an immediate by product of the online computation phase, and therefore does not add any overhead to the computation. Although our basic generic protocol can be instantiated with any non-constant round MPC protocol, we provide an optimized version that utilizes specific features of the SPDZ protocol [17].
In the new general construction, the new constant-round MPC protocol consists of two phases. In the first (offline) phase, the parties securely compute *random shares* of the BMR garbled circuit. If this is done naively, then the result is highly inefficient since part of the computation involves computing a pseudorandom generator or pseudorandom function multiple times for every gate. By modifying the original BMR garbled circuit, it was shown that it is possible to actually compute the circuit very efficiently. Specifically, each party locally computes the pseudorandom function as needed for every gate (the construction uses a pseudorandom function rather than a pseudorandom generator), and uses the results as input to the secure computation. The security proof shows that if a party cheats and inputs incorrect values then no harm is done, since it can only cause the honest parties to abort (which is anyway possible when there is no honest majority). Next, in the online phase, all that the parties need to do is reconstruct the single garbled circuit, exchange garbled values on the input wires and locally compute the garbled circuit. The online phase is therefore very fast.
In a concrete instantiation of the protocol, based on using the SPDZ protocol [17] in the offline phase, there are actually three separate phases, with each being faster than the previous. The first two phases can be run offline, and the last phase is run online after the inputs become known.

- The first (slow) phase depends only on an upper bound on the number of wires and the number of gates in the function to be evaluated. This phase uses Somewhat Homomorphic Encryption (SHE) and is equivalent to the offline phase of the SPDZ protocol.

- The second phase depends on the function to be evaluated but not the function inputs; in our proposed instantiation this mainly involves information theoretic primitives and is equivalent to the online phase of the SPDZ protocol.
- In the third phase the parties provide their input and evaluate the function; this phase just involves exchanging shares of the circuit and garbled values on the input wire and locally computing the BMR garbled circuit.

2.3 ABY: Mixed-Protocol Secure Computation (short description)

This section is based on [19]. The complete description of this work appears in deliverable D13.1.

In generic secure computation, the function to be evaluated is often represented as a circuit. The two most common types of circuits that are used in secure computation are *Arithmetic circuits*, where the primitive operations are addition and multiplication, and *Boolean circuits*, where the primitive operations are XOR and AND. The efficiency of secure computation protocols goes hand-in-hand with an efficient circuit representation of the function. For instance, performing a multiplication between two ℓ-bit values in an Arithmetic circuit is very efficient but requires a circuit of size $O(\ell^2)$ for Boolean circuits.

To overcome the dependence on an efficient function representation, several works proposed to *mix* secure computation protocols based on Arithmetic and Boolean circuits. One of these works is the ABY framework [19] for which an overview is given in Figure 2.1.

The ABY framework enables to convert between different representations of the inputs in a dynamic and efficient manner. The possible representation are Yao sharing, Boolean sharing, and arithmetic sharing. Each input representation is efficient for a different type of computation. (For example, Yao sharing is particularly suited for bit-wise operations.) The ABY framework converts between different representations of the input in order to use the most efficient representation for each phase of the computation.

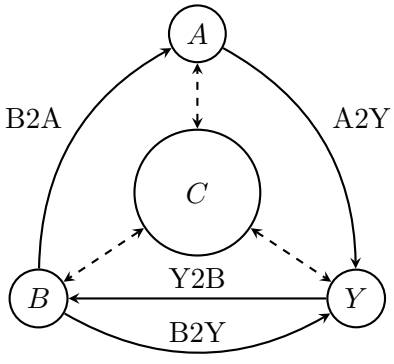


Figure 2.1: Overview of the ABY framework [19] that allows efficient conversions between **C**leartexts and three types of sharings: **A**rithmetic, **B**oolean, and **Y**ao.

2.4 Private Function Evaluation

2.4.1 Private Function Evaluation Using Universal Circuits

Universal circuits (UCs) can be programmed to evaluate any circuit up to a given size k. They provide elegant solutions in various application scenarios, e.g. for private function evaluation (PFE). The optimal size of a universal circuit is proven to be $\Omega(k \log k)$. Valiant proposed a size-optimized UC construction in [67], which has not been put in practice ever since. The only implementation of universal circuits was provided by Kolesnikov and Schneider [44], with size $\mathcal{O}(k \log^2 k)$.
Any computable function $f(x)$ can be represented as a Boolean circuit with input bits $x = (x_1, \ldots, x_u)$. Universal circuits (UCs) are programmable circuits, which means that beyond the u inputs, they receive $p = (p_1, \ldots, p_m)$ program bits as further inputs. Using these program bits, the UC is programmed to evaluate the function, such that $UC(x, p) = f(x)$. The advantage is that one can apply the same UC for computing different functions of the same size.
Efficient constructions considering both the size and the depth of the UC were proposed. The approach that we revisit due to the below detailed application is the optimization of the size by Valiant [67], resulting in a construction with asymptotically optimal size $\mathcal{O}(k \log k)$ and linear depth $\mathcal{O}(k)$, where k denotes the size of the simulated circuits.
The most prominent application of UCs is the evaluation of private functions based on *secure function evaluation* (SFE) or *secure two-party computation*. SFE enables two parties P_1 and P_2 to evaluate a publicly known function $f(x, y)$ on their private inputs x and y, ensuring that none of the participants learns anything about the other participant's input, and that both P_1 and P_2 learn the correct result. Many secure computation protocols use Boolean circuits for representing the desired functionality, such as Yao's garbled circuit protocol [46, 70] and the GMW protocol [28]. In some applications the function itself should be kept secret. This setting is called *private function evaluation* (PFE), where only one of the parties P_1 knows the function $f(x)$, whereas the other party P_2 provides the input to the private function. P_2 learns no information about f besides the size of the circuit defining the function and the number of inputs and outputs.
PFE can be reduced to SFE [44] by securely evaluating a UC that is programmed by P_1 to evaluate the function f on P_2's input x. Thus, P_1 provides the program bits for the UC and P_2 provides his private input x into an SFE protocol that computes a UC. The complexity of PFE in this case is determined mainly by the complexity of the UC construction. The security follows from that of the SFE protocol that is used to evaluate the UC. If the SFE protocol is secure against semi-honest, covert or malicious adversaries, then the PFE protocol is secure in the same adversarial setting. Further applications of UCs and PFE and related work can be found in [39, 40].

2.4.2 Valiant's Universal Circuit Construction

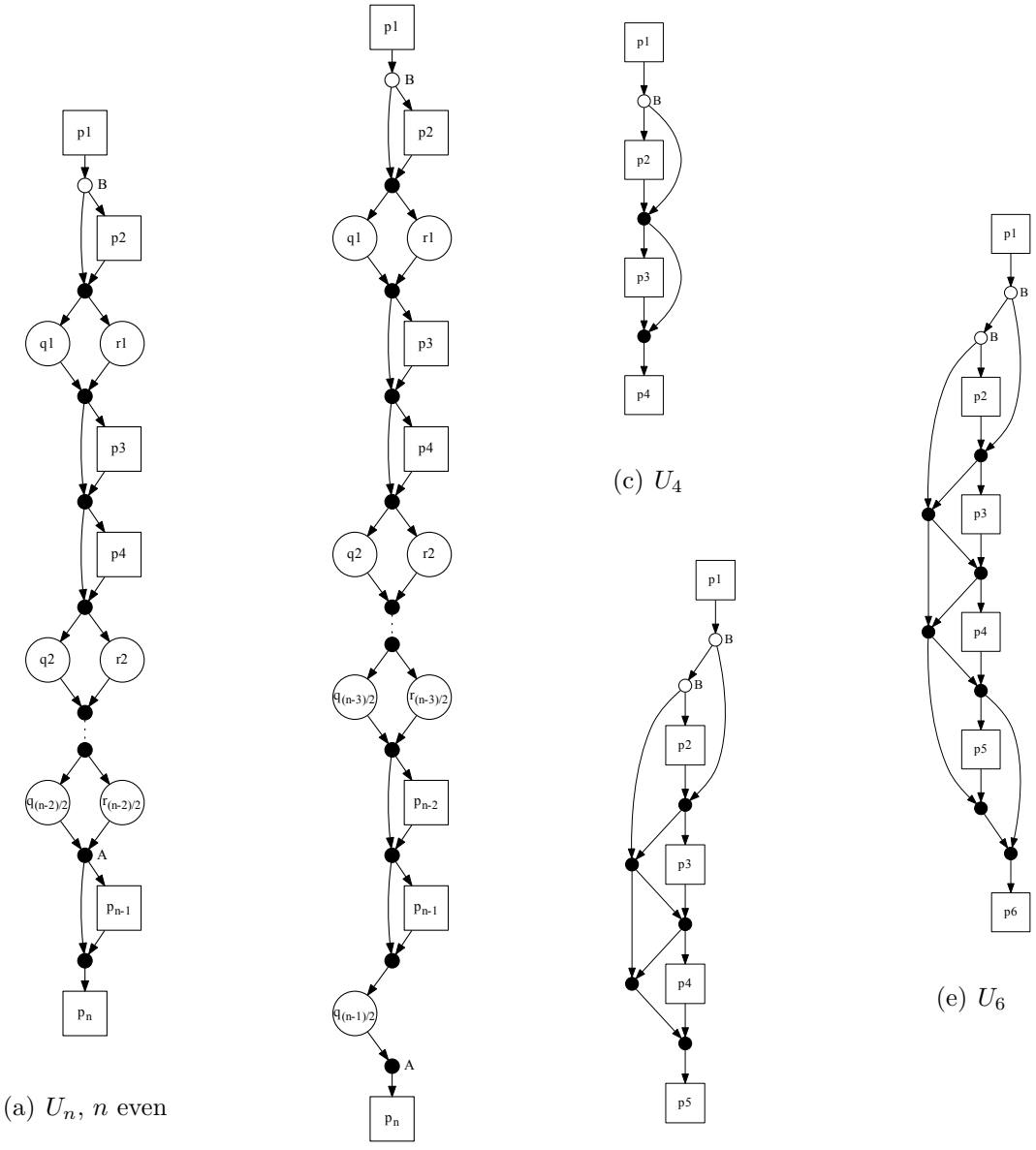
In this section, we describe Valiant's edge-universal graph construction for graphs for which all nodes have at most one incoming and at most one outgoing edge and detail how two such graphs can be used for constructing universal circuits [67].

Edge-Universal Graphs.

$G = (V, E)$ is a directed graph with the set of nodes $V = \{1, \ldots, n\}$ and the set of edges $E \subseteq V \times V$. A directed graph has *fanin* or *fanout* ℓ if each of its nodes has at most ℓ edges

directed into or out of it, respectively. $\Gamma_\ell(n)$ denotes the set of all acyclic directed graphs with n nodes and fanin and fanout ℓ. Further on, we require a labelling of the nodes in a topological order, i.e., $i > j$ implies that there is no directed path from i to j. In a graph in $\Gamma_\ell(n)$, a topological ordering can always be found with computational complexity $\mathcal{O}(n + \ell n)$.
An *edge-embedding* of graph $G = (V, E)$ into $G' = (V', E')$ is a mapping that maps V into V' one-to-one, with possible additional nodes in V', and E into directed paths in E', such that they are pairwise edge-disjoint, i.e., an edge can be used only in one path. A graph G' is *edge-universal* for $\Gamma_\ell(n)$ if it has distinguished poles $\{p_1, \ldots, p_n\} \subseteq V'$ and every graph $G \in \Gamma_\ell(n)$ with node set $V = \{1, \ldots, n\}$ can be edge-embedded into G' by a mapping φ^G such that $\varphi^G : i \mapsto p_i$ and $\varphi^G : (i, j) \mapsto \{\text{path from pole } p_i \text{ to pole } p_j\}$ for each $i, j \in V$.
Here, we recapitulate Valiant's construction for acyclic edge-universal graph for $\Gamma_1(n)$, denoted by U_n, that has fewer than $2.5n \log_2 n$ nodes, fanin and fanout 2 and poles with fanin and fanout 1. Valiant presents another edge-universal graph construction with a lower multiplicative constant $2.375n \log_2 n$. We omit that version of the algorithm for two reasons: firstly, our aim is to show the practicality of Valiant's approach and secondly, including all the optimizations even in the simpler construction is a challenging task in practice. The more efficient algorithm uses four subgraphs instead of two at each recursion and utilizes a skeleton with a more complex structure. For more details on this improved algorithm, the reader is referred to [49, 67]. We leave showing the practicality of the improved method as future work.

Valiant's Edge-Universal Graph Construction of Size $2.5n \log_2 n$ for $\Gamma_1(n)$ Graphs: The edge-universal graph for $\Gamma_1(n)$, denoted by U_n, is constructed with poles $\{p_1, \ldots, p_n\}$ with fanin and fanout 1, which are connected according to the skeleton shown in Figure 2.2. The poles are emphasized as special nodes with squares, and the additional nodes are shown as circles. The recursive construction works as follows: the nodes denoted by $\{q_1, \ldots, q_{\lceil \frac{n-2}{2} \rceil}\}$ and $\{r_1, \ldots, r_{\lfloor \frac{n-2}{2} \rfloor}\}$ are considered as the poles of two smaller edge-universal graphs called subgraphs $Q_{\lceil \frac{n-2}{2} \rceil}$ and $R_{\lfloor \frac{n-2}{2} \rfloor}$, respectively, that are otherwise not shown. Since they are poles of the two subgraphs with such a skeleton but not of U_n, they will have at most the allowed fanin and fanout 2: they inherit one incoming and one outgoing edge from the outer skeleton, and at most one incoming and one outgoing edge from the subgraph. $Q_{\lceil \frac{n-2}{2} \rceil}$ (and $R_{\lfloor \frac{n-2}{2} \rfloor}$) is then constructed similarly: the skeleton is completed and two smaller graphs with sizes $\lceil \frac{\lceil \frac{n-2}{2} \rceil - 2}{2} \rceil$ and $\lfloor \frac{\lceil \frac{n-2}{2} \rceil - 2}{2} \rfloor$ (and sizes $\lceil \frac{\lfloor \frac{n-2}{2} \rfloor - 2}{2} \rceil$ and $\lfloor \frac{\lfloor \frac{n-2}{2} \rfloor - 2}{2} \rfloor$) are constructed. For starting off the recursion, U_1 is a graph with a single pole while U_2 and U_3 are graphs with two and three connected poles, respectively. Valiant gives special constructions for U_4, U_5 and U_6 and shows that it is possible to obtain the respective edge-universal graphs with altogether $3, 7$ and 9 additional nodes, respectively, as shown in Figure 2.2.
We recapitulate the proof from [67] that U_n is edge-universal for $\Gamma_1(n)$, such that any graph with n nodes and fanin and fanout 1 can be edge-embedded into U_n. According to the definition of edge-embedding, it has to be shown that given any $\Gamma_1(n)$ graph G with set of edges E, for any $(i, j) \in E$ and $(k, l) \in E$ we can find pairwise edge-disjoint paths from p_i to p_j and from p_k to p_l in U_n. As before, the labelling of nodes $V = \{1, \ldots, n\}$ in the $\Gamma_1(n)$ graph is according to a topological order of the nodes.
Firstly, each two neighbouring poles of the edge-universal graph, p_{2s} and p_{2s+1} for $s \in \{1, \ldots, \lceil \frac{n}{2} \rceil\}$, are thought of as merged superpoles, with their fanin and fanout becoming 2. In a similar manner, any $G \in \Gamma_1(n)$ graph can be regarded as a $\Gamma_2(\lceil \frac{n}{2} \rceil)$ graph with supernodes, i.e. each pair $(2s, 2s + 1)$ will be merged into one node in a $\Gamma_2(\lceil \frac{n}{2} \rceil)$ graph $G' = (V', E')$. If there are edges



(a) U_n, n even

(b) U_n, n odd

(c) U_4

(d) U_5

(e) U_6

Figure 2.2: Skeleton of Valiant's edge-universal graph and optimized cases.

between the nodes in G, they are simulated with loops.[1] The set of edges of this graph G is partitioned to sets E_1 and E_2, s.t. $G_1 = (V, E_1)$ and $G_2 = (V, E_2)$ are instances of $\Gamma_1(\lceil \frac{n}{2} \rceil)$ and $\Gamma_1(\lfloor \frac{n}{2} \rfloor)$, respectively. This can be done efficiently, as shown later in this section. The edges in E_1 are embedded as directed paths in Q, and the edges in E_2 as directed paths in R. Both E_1 and E_2 have at most one edge directed into and at most one directed out of any supernode and therefore, there is only one edge from E_1 and one from E_2 to be simulated going through any superpole in U_n as well. Thus, the edge coming into a superpole (p_{2s}, p_{2s+1}) in E_1 is embedded as a path through q_{s-1}, while the edge going out of the pole in E_1 is embedded as a path through q_s in the appropriate subgraph. Similarly, the edges in E_2 are simulated as edges through r_{s-1} and r_s. These paths can be chosen disjoint according to the induction hypothesis. Finally, the paths from q_{s-1} and r_{s-1} to superpole (p_{2s-1}, p_{2s}) as well as the paths from (p_{2s-1}, p_{2s}) to q_s and r_s can be chosen edge-disjoint due to the skeleton shown in Figure 2.2. With this, Valiant's graph construction is a valid edge-universal graph construction with asymptotically optimal size $\mathcal{O}(n \log n)$, and depth $\mathcal{O}(n)$ [67].

Valiant's Edge-Universal Graph Construction of Size $5n \log_2 n$ for $\Gamma_2(n)$ Graphs: Given a directed acyclic graph $G \in \Gamma_2(n)$, the set of edges E can be separated into two distinct sets E_1 and E_2, such that graphs $G_1 = (V, E_1)$ and $G_2 = (V, E_2)$ are instances of $\Gamma_1(n)$, having fanin and fanout 1 for each node [67]. Given the set of nodes $V = \{1, \ldots, n\}$, one constructs a bipartite graph $\overline{G} = (\overline{V}, \overline{E})$ with nodes $\overline{V} = \{m_1, \ldots, m_n, m'_1, \ldots, m'_n\}$ and edges $\overline{E}$ such that $(m_i, m'_j) \in \overline{E}$ if and only if $(i, j) \in E$. The edges of $\overline{G}$ and thus the corresponding edges of G can be colored in a way that the result is a valid two-coloring. Having fanin and fanout at most 2, such coloring can be found directly with the following method, used in the proof of Konig-Hall theorem in [50]:

1: **while** There are uncolored edges in $\overline{G}$ **do**
2: Choose an uncolored edge $e = (m_i, m'_j)$ randomly and color the path or cycle that contains it in an alternating manner: the neighbouring edge(s) of an edge of the first color will be colored with the second color and vice versa.
3: **end while**

This coloring can be performed in $\mathcal{O}(n)$ steps and it defines the edges in E_1 and E_2, s.t. E_1 contains the edges colored with color one and E_2 the ones with color two and $G_1 = (V, E_1)$ and $G_2 = (V, E_2)$ (cf. [40]).
With this method, the problem of constructing edge-universal graphs for $\Gamma_2(n)$ can be reduced to the $\Gamma_1(n)$ construction. After constructing two edge-universal graphs for $\Gamma_1(n)$ (i.e. $U_{n,1}$ and $U_{n,2}$), their poles are merged and an edge-universal graph for $\Gamma_2(n)$ is obtained. The merged poles now have fanin and fanout 2, since the poles of $U_{n,1}$ and $U_{n,2}$ previously had fanin and fanout 1. E_1 can then be edge-embedded using the edges of $U_{n,1}$ and E_2 using the edges of $U_{n,2}$.

Universal Circuits.

We now describe how to construct UCs by means of Valiant's edge-universal graph construction for $\Gamma_2(n)$ graphs [67]. Our goal is to obtain an acyclic circuit built from special gates that simulate any acyclic Boolean circuit with u inputs, v outputs and k gates. In the circuit, the inputs of the gates are either connected to an input variable, to the output of another gate or are assigned a fixed constant. Due to the nature of Valiant's edge-universal graph construction, we have two restrictions on the original circuit. Firstly, all the gates must have at most two

[1] We note that these G' graphs are constructed from the original $\Gamma_1(n)$ graph G in order to define the correct embedding. Therefore, they are not required to be acyclic.

inputs and secondly, the fanout of inputs and gates must be at most 2, i.e., each input of the circuit and each output of any gate can only be the input of at most two later gates. This is necessary in order to guarantee that the graph of the original circuit has fanin and fanout 2. We note that the first restriction was present in case of the construction in [44] as well, but the output of any input or any gate could be used multiple times. However, it was proven in [67] that the general case, where the fanout of the circuit can be any integer $m \geq 2$, can be transformed to the special case when $m \leq 2$. The algorithm places a binary tree of identity gates in place of each gate with fanout larger than 2, following Valiant's proposition: *„Any gate with fanout $x+2$ can be replaced by a binary fanout tree with $x+1$ gates"* [67, Corollary 3.1]. These identity gates are so-called *copy gates*, each of them eliminating one from the extra fanout of the original gate. The resulting circuit will have k^* gates with $k \leq k^* \leq 2k+v$, where k denotes the number of gates and v the number of outputs in the circuit.
After this transformation, given a circuit C with u inputs, v outputs and k^* gates with fanin and fanout 2, the graph of C, denoted by G^C consists of a node for each gate, input and output variable and thus is in $\Gamma_2(u+v+k^*)$. The wires of circuit C are represented by edges in G^C. A *topological ordering* of the gates is chosen, which ensures that gate g_i has no inputs that are outputs of a later gate g_j, where $j > i$. The inputs and the outputs can be ordered arbitrarily within themselves as long as the inputs are kept before the topologically ordered gates and the outputs after them. Even though the output nodes cause an overhead in Valiant's UC, they are required to fully hide the topology of the circuit in the corresponding universal circuit. If, in the fanout-2 circuit, one can observe which gates provide the output of the computation, it might reveal information about the structure of the circuit, e.g. how many times is the result of an output gate used after being calculated. We ensure by adding nodes corresponding to the outputs that the last v nodes in U_{u+v+k^*} are the ones providing the outputs. We note that our understanding of universal circuits here slightly differs from Valiant's, since he constructs U_{u+k^*} [67].
Therefore, after obtaining G^C a Γ_2 edge-universal graph U_{u+v+k^*} is constructed, into which G^C is edge-embedded. Valiant shows in [67] how to obtain the universal circuit corresponding to U_{u+v+k^*} and how to program it according to the edge-embedding of G^C. Firstly, the first u poles become inputs, the next k^* poles are so-called universal gates, and the last v poles are outputs in the universal circuit. A *universal gate* denoted by $U(\mathrm{in}_1, \mathrm{in}_2; c_0, c_1, c_2, c_3)$, can compute any function with two inputs in_1 and in_2 and four control bits c_0, c_1, c_2 and c_3 as in Equation 2.1.

$$\mathrm{out}_1 = U(\mathrm{in}_1, \mathrm{in}_2; c_0, c_1, c_2, c_3) = c_0\overline{\mathrm{in}_1}\overline{\mathrm{in}_2} \oplus c_1\overline{\mathrm{in}_1}\mathrm{in}_2 \oplus c_2\mathrm{in}_1\overline{\mathrm{in}_2} \oplus c_3\mathrm{in}_1\mathrm{in}_2. \tag{2.1}$$

The rest of the nodes of the edge-universal graph are translated into *universal switches* or X *gates*, denoted by $(\mathrm{out}_1, \mathrm{out}_2) = X(\mathrm{in}_1, \mathrm{in}_2; c)$ that are defined by one control bit c and return the two input values either in the same or in reversed order as in Equation 2.2.

$$\mathrm{out}_1 = \bar{c}\,\mathrm{in}_1 \oplus c\,\mathrm{in}_2, \qquad \mathrm{out}_2 = c\,\mathrm{in}_1 \oplus \bar{c}\,\mathrm{in}_2. \tag{2.2}$$

Since ABY uses the free-XOR optimization from [43], we construct universal gates and switches with low ANDsize and ANDdepth given in Section 2.4.3. With the cost metric we consider, X and Y gates have the same AND complexity, optimized in [43] and are obtained as

$$\begin{aligned} \mathrm{out}_1 &= Y(\mathrm{in}_1, \mathrm{in}_2; c) = (\mathrm{in}_1 \oplus \mathrm{in}_2)c \oplus \mathrm{in}_1 \\ (\mathrm{out}_1, \mathrm{out}_2) &= X(\mathrm{in}_1, \mathrm{in}_2; c) = (e \oplus \mathrm{in}_1, e \oplus \mathrm{in}_2) \text{ with } e = (\mathrm{in}_1 \oplus \mathrm{in}_2)c \end{aligned} \tag{2.3}$$

with ANDsize and ANDdepth of 1 for both universal switches. X gates are realized with one additional XOR gate compared to Y gates. Our efficient implementation of generic universal

Algorithm 1 SUPERGRAPH(G)

Input: $\Gamma_1(n)$ graph G with set of nodes $V = \{1, \ldots, n\}$
Output: $\Gamma_1(n)$ supergraph

Create a graph H with $\lceil \frac{n}{2} \rceil - 1$ nodes ▷ H Γ_2 graph (with possible loops)
if there exist an edge (i, j) in G **and** $\lceil \frac{j}{2} \rceil - 1 \geq \lceil \frac{i}{2} \rceil$ **then**
    Add edge $\left(\lceil \frac{i}{2} \rceil, \lceil \frac{j}{2} \rceil - 1\right)$ in H ▷ each pair of nodes in G is one node in H
end if
Partition H into two Γ_1 graphs G_1 of size $\lceil \frac{n}{2} \rceil - 1$ and G_2 of size $\lfloor \frac{n}{2} \rfloor - 1$ using Konig's theorem as in Section 2.4.2
    ▷ in odd case, the $(e, \lceil \frac{n}{2} \rceil - 1)$ edge in H for arbitrary e will be added in G_1
if size(G_1) $\neq 0$ **then**
    SUPERGRAPH(G_1)
    Store G_1 as the left subgraph of G
end if
if size(G_2) $\neq 0$ **then**
    SUPERGRAPH(G_2)
    Store G_2 as the right subgraph of G
end if
delete H
return G

gates uses Y gates yielding

$$\text{out}_1 = U(\text{in}_1, \text{in}_2; c_0, c_1, c_2, c_3) = Y[Y(c_0, c_1; \text{in}_2), Y(c_2, c_3; \text{in}_2); \text{in}_1] \tag{2.4}$$

with ANDsize(U) = 3 and ANDdepth(U) = 2. This universal gate implementation is generic and works in all secure computation protocols. However, for Yao's garbled circuits protocol, one can further optimize it to ANDsize(U) = ANDdepth(U) = 1, as in some garbling schemes such as the garbled 3-row-reduction [54] the gate being evaluated remains oblivious to the evaluator. The programming of the universal circuit means specifying the control bit of each universal switch and the four control bits of each universal gate. The universal gates are programmed according to the simulated gates in C and the universal switches according to the paths defined by the edge-embedding of the graph of the circuit G^C in the edge-universal graph U_{u+v+k^*}. Depending on if the path takes the same direction during the embedding (e.g. arrives from the left and continues on the left) or changes its direction at a given node (e.g. arrives from the left and continues on the right), the control bit of the universal switch can be programmed accordingly.

In the following, we detail our concrete method for programming the universal circuit and discuss efficient implementations of universal gates and switches. The Γ_2 graph of the circuit G^C with n nodes is partitioned into two $\Gamma_1(n)$ graphs G_1^C and G_2^C which are embedded into the two edge-universal graphs for $\Gamma_1(n)$ that build up U_n. Valiant proved in [67] that for any topologically ordered $\Gamma_1(n)$ graph, for any $(i, j) \in E$ and $(k, l) \in E$ edges there exist edge-disjoint paths in U_n between the i^{th} and the j^{th} poles and between the k^{th} and the l^{th} poles. We described Valiant's method in Section 2.4.2 and here we show the algorithm that uniquely defines these paths in U_n.

For the description of our algorithm, we first define a $\Gamma_1(n)$ supergraph, which is a $\Gamma_1(n)$ graph with additionally a binary tree of Γ_1 graphs of decreasing size. These Γ_1 graphs uniquely define the embedding of the edges into U_n. When embedding an edge (i, j) of the topologically ordered

graph G into the edge-universal graph, one needs to construct the supergraph of G as described in Algorithm 1 and then look at the binary tree in the supergraph. The path of the edge (i, j) defines the edge-embedding uniquely. This means that if edge $(\lceil \frac{i}{2} \rceil, \lceil \frac{j}{2} \rceil - 1)$ is in the left subgraph of G, then it can be embedded through subgraph Q in U_n, otherwise it is in the right subgraph of G and can be embedded through subgraph R in U_n. The unique embedding happens through $\{r_{\lceil \frac{i}{2} \rceil}, r_{\lceil \frac{j}{2} \rceil - 1}\}$ or through $\{q_{\lceil \frac{i}{2} \rceil}, q_{\lceil \frac{j}{2} \rceil - 1}\}$, utilizing the unique shortest path between them, through subpoles further identified by smaller subgraphs of G.
The embedding of (i, j) is ready in one of the following three scenarios:

1. **Leaf:** there are no subgraphs in G anymore,

2. **Superpole:** $\lceil \frac{j}{2} \rceil - 1 < \lceil \frac{i}{2} \rceil$, and therefore $(\lceil \frac{i}{2} \rceil, \lceil \frac{j}{2} \rceil - 1)$ cannot be found in any of the supergraphs anymore, in which case i is odd and $j = i + 1$, and the path between p_i and p_{i+1} in the skeleton as in Figures 2.2 goes directly through one switching node without entering a subgraph, or

3. **Subpole:** $\lceil \frac{j}{2} \rceil - 1 = \lceil \frac{i}{2} \rceil$ and therefore is represented by a loop in a subgraph, in which case i is even and $j = i + 1$, and the path between p_i and p_{i+1} as in Figures 2.2 goes directly through one subpole and two to four switching nodes. In this case, which subpole is used is defined by the supergraph G.

When the embedding is done, for defining the control bits, each node x has at most two nodes that have ingoing edges to x, one is represented as the left parent and one as the right parent of x in the edge-universal graph. The two consecutive nodes are also saved as left and right children of x. Now, when x is a switching node and we take edges (v, x) and (x, w) in the path, we save for x if parent v and child w are on the same or on the opposite side in the edge-universal graph. This defines the control bit of each universal switch in the translated universal circuit, where left and right parent and child translate to first and second input and output, respectively.

2.4.3 The Size and the Depth of Valiant's UC

In this section, we obtain new formulae for the size and the depth of Valiant's construction: for the Γ_1 edge-universal graph construction and for the universal circuit construction. The *size of the edge-universal graph* is the number of nodes, counting all the poles and nodes created while using Valiant's construction. The *depth of the edge-universal graph* is the number of nodes on the longest path between any two nodes. When considering UCs and the PFE application, since XOR gates can be evaluated for free in secure computation [43], the *ANDsize of the universal circuit* is the number of AND gates that are needed to realize the UC in total. The *ANDdepth of the universal circuit* in this scenario is the maximum number of AND gates between any input and output. For the sake of generality, we give the *total size* and *depth* of Valiant's UC construction with respect to both the AND and XOR gates that are used. Our implementation of universal gates and switches is optimized for PFE (cf. [39]) and therefore uses the fewest AND gates possible. However, the total size and depth can be relevant when optimizing for other applications, in which case our implementation gives an upper bound that can be improved. For instance, when XOR and AND gates have the same costs, one needs to minimize the total number of gates instead of the number of AND gates.

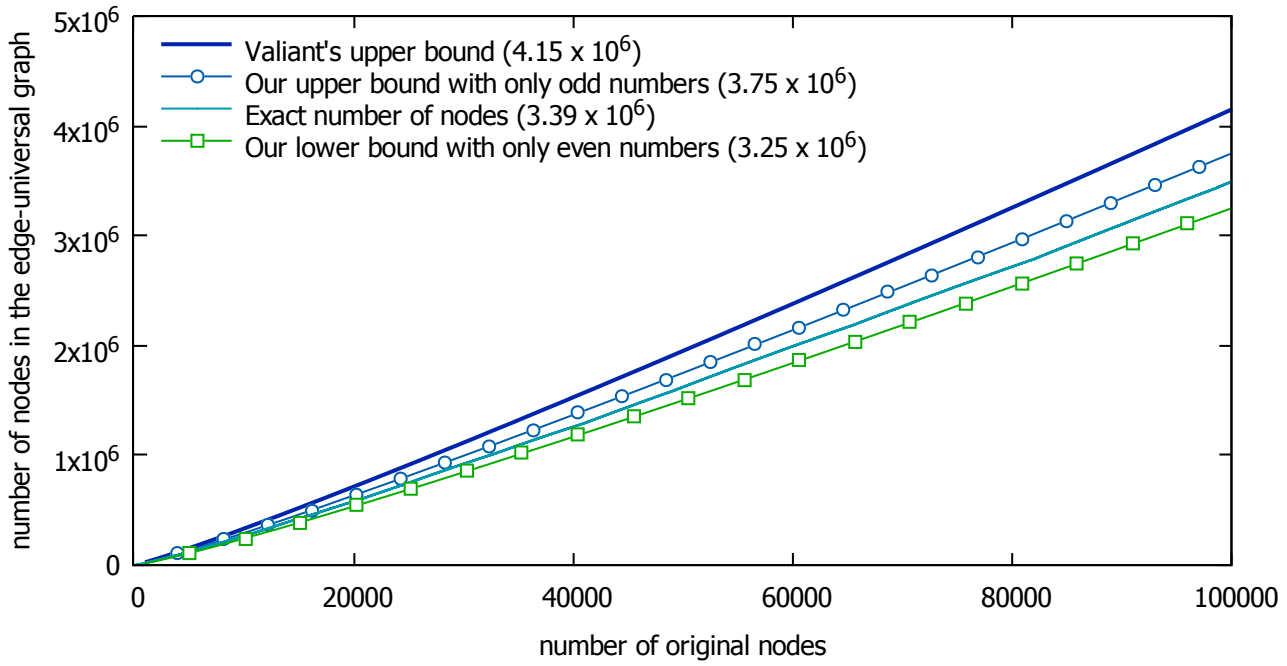


Figure 2.3: Our upper and lower bounds for the size of Valiant's edge-universal graph construction for $\Gamma_1(n)$ graphs, along with Valiant's upper bound on the same construction and the exact size $\textsc{Exact}(n)$, considering the size of the embedded graph $n \in \{1, \ldots, 100{,}000\}$.

The Size and the Depth of the Γ_1 Edge-Universal Graph

In the skeleton for even n, node A in Figure 2.2 is redundant, since one can choose to embed the edge $(y, n-1)$ as (p_y, p_{n-1}) through Q, and (z, n) as (p_z, p_n) through R for any y and z nodes [67]. Thus, the number of nodes other than poles $\textsc{Exact}(n)$, for even n becomes

$$\textsc{Exact}(n) = 2 \cdot \textsc{Exact}\left(\frac{n-2}{2}\right) + 5 \cdot \frac{n-2}{2}. \tag{2.5}$$

For odd n, the construction makes use of $\frac{n-1}{2}$ poles in Q and $\frac{n-3}{2}$ poles in R. Then, edge (y, n) is embedded as (p_y, p_n) through Q for any y node, and node A is again redundant. Thus,

$$\textsc{Exact}(n) = \textsc{Exact}\left(\frac{n-1}{2}\right) + \textsc{Exact}\left(\frac{n-3}{2}\right) + 5 \cdot \frac{n-3}{2} + 3. \tag{2.6}$$

Using these recursive formulae, given the value n, it is possible to obtain the exact number of nodes other than poles in U_n. Valiant includes optimizations for starting off the recursion: for $1, 2, 3, 4, 5$ and 6 nodes; the respective number of additional nodes are $0, 0, 0, 3, 7$ and 9 (cf. Figure 2.2). Thus, a simple algorithm using dynamic programming based on the recursion relations of Equations 2.5-2.6 yields the exact number of nodes other than the original n poles that are created during the edge-universal graph construction. It depends on the parity of the input n at each iteration and unfortunately does not yield a closed formula for the size of Valiant's edge-universal graph construction, which is $n + \textsc{Exact}(n)$.
Valiant states that using his method, an edge-universal graph for $\Gamma_1(n)$ can be found *„with fewer than* $\frac{19}{8} n \log_2 n$ *nodes, and fanin and fanout 2"* [67]. As mentioned in Section 2.4.2, we consider the more detailed algorithm that yields the result with a slightly larger prefactor of $2.5n \log_2 n$

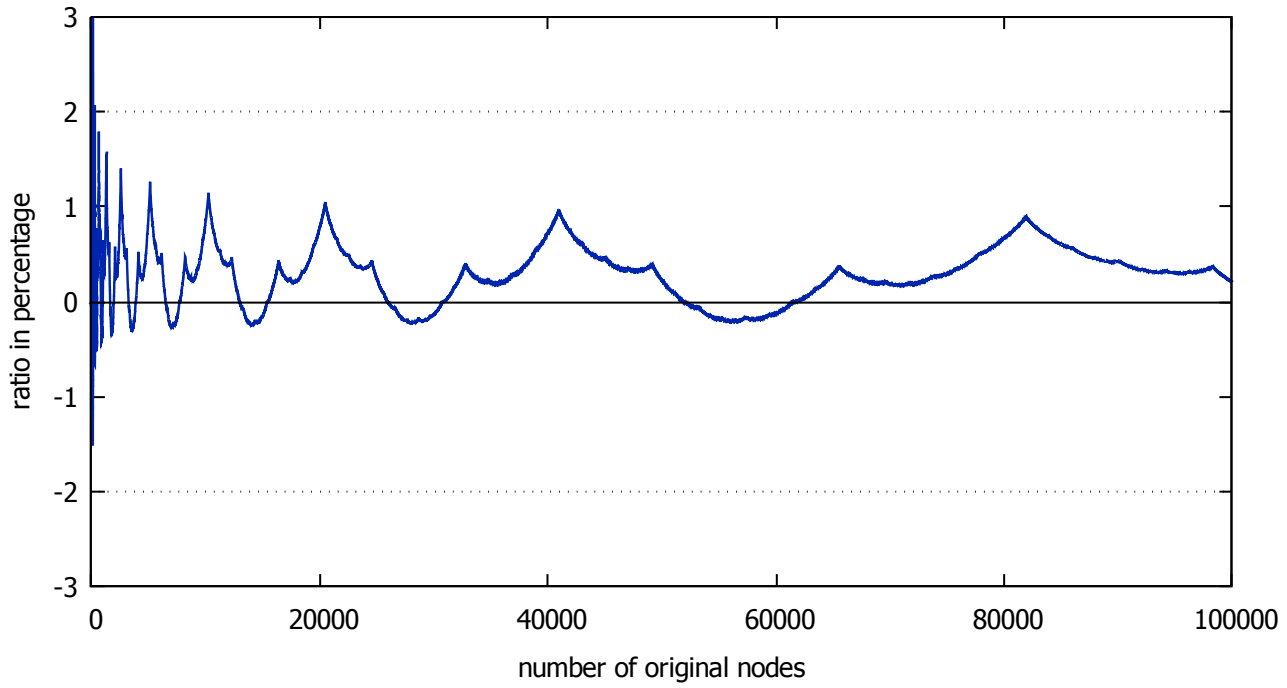


Figure 2.4: The deviation of the mean of our upper and lower bounds (Equation 2.7 and Equation 2.8) from the exact size of the edge-universal graph $\textsc{Exact}(n) + n$, considering the size of the embedded graph $n \in \{1, \ldots, 100{,}000\}$.

instead of $2.375n \log_2 n$. In this section, we sharpen this bound and give an approximate closed formula for the size of the construction. We first give upper and lower bounds, and then derive an approximation for a closed formula. For our lower bound, we consider the case when only the formula for even numbers, i.e., Equation 2.5, is considered. This yields our lower bound of

$$n + 5\left(\sum_{i=0}^{\log_2 n-1} 2^i \left(\frac{n}{2^{i+1}} - \frac{2(2^{i+1}-1)}{2^{i+1}}\right)\right) = 2.5n\log_2 n - 9n + 5\log_2 n + 10. \tag{2.7}$$

The upper bound can be obtained similarly, considering the case when only the formula for odd numbers with $5 \cdot \left(\frac{n-1}{2}\right)$ is considered

$$n + 5\left(\sum_{i=0}^{\log_2 n-1} 2^i \left(\frac{n}{2^{i+1}} - \frac{2^{i+1}-1}{2^{i+1}}\right)\right) = 2.5n\log_2 n - 4n + 2.5\log_2 n + 5. \tag{2.8}$$

Figure 2.3 depicts our upper and lower bounds along with Valiant's upper bound on the same construction for up to 100,000 nodes. We observe that the mean of our bounds is very close to the exact number of nodes. Figure 2.4 shows that already after a couple of hundreds of poles, it only slightly deviates from the exact number of nodes $\textsc{Exact}(n)$. Thus, we accept

$$\text{size}(U_n) \approx 2.5n\log_2 n - 6.5n + 3.75\log_2 n + 7.5 \tag{2.9}$$

as a good approximation of the closed formula for the size of the construction, noting that an estimated deviation of at most 2% compared to the exact number of nodes, i.e., $\varepsilon \leq 0.02 \cdot \text{size}(U_n)$ may occur.

The depth of the edge-universal graph, i.e., the maximum number of nodes between any two nodes is defined by the number of nodes between p_1 and p_n in the skeleton (cf. Figure 2.2). Thus, $\text{depth}(U_n) = 3n - 3$ for even n and $\text{depth}(U_n) = 3n - 2$ for odd n.

The Size and the Depth of Valiant's UC

As described in Section 2.4.2, a universal circuit is constructed by means of an edge-universal graph for graphs with fanin and fanout 2, which is in turn constructed from two Γ_1 edge-universal graphs with poles merged together and thus taken only once into consideration. When constructing a UC, the number of inputs u, the number of outputs v and the number of gates k is public. We set k^* as the number of gates in the equivalent fanout-2 circuit, where $k \leq k^* \leq 2k + v$, in order to be able to later fairly compare with the UC construction of [44]. We consider k^* as the public parameter instead of k, since without the knowledge of the original number of simulated gates, it does not reveal information about the simulated circuit. If the original k is public, one can hide k^* by setting it to its maximal value $2k + v$. Thus, using Valiant's UC construction, a Γ_2 edge-universal graph with $u + v + k^*$ poles is constructed and thus, our approximative formula for the size of the Γ_2 edge-universal graph corresponding to the graph of the circuit would become $2 \cdot \text{size}(U_{u+v+k^*}) - (u + v + k^*)$ and the exact number would be $u + v + k^* + 2 \cdot \textsc{Exact}(u + v + k^*)$, i.e., the $u + v + k^*$ merged poles of the two edge-universal graphs plus the exact number of nodes other than poles. Therefore, the size of Valiant's UC is

$$\begin{aligned}\text{size}(UC^{\text{Valiant}}_{u,v,k^*}) \approx [5(u+v+k^*)\log_2(u+v+k^*) - 15(u+v+k^*) \\ + 7.5\log_2(u+v+k^*) + 15] \cdot \text{size}(X) + k^* \cdot \text{size}(U) \quad (2.10)\end{aligned}$$

and the depth stays

$$\text{depth}(UC^{\text{Valiant}}_{u,v,k^*}) \approx [2(u+v+k^*) - 2] \cdot \text{depth}(X) + k^* \cdot \text{depth}(U). \quad (2.11)$$

When transforming the Γ_2 edge-universal graph into a UC, the first u poles are associated with inputs, the last v poles with outputs, and the k^* poles between are realized with universal gates (cf. Equation 2.1) and their programming is defined by the corresponding gates in the simulated circuit. The rest of the nodes of the edge-universal graph are translated into universal switches (cf. Equation 2.2), whose programming is defined by the edge-embedding of the graph of the circuit into the Γ_2 edge-universal graph. Thus, the size and depth of Valiant's UC can be directly derived from the size of the Γ_2 edge-universal graph. However, we include two optimizations to obtain a smaller size of the UC. The first optimization improves already the size of the edge-universal graph and the second optimization is applied when translating the edge-universal graph into a UC description (cf. [39]).

1. **Optimization for Input and Output Nodes:** We observe that obviously circuit inputs need no ingoing edges and circuit outputs need no outgoing edges. Therefore, since u, v and k^* are publicly known, we optimize by deleting nodes that become redundant while canceling the edges going to the first u (input) and coming from the last v (output) nodes. Depending on the parity of u, v and $u + v + k^*$, the number of redundant switching nodes is $u + v - 3 \pm 1$ in both Γ_1 edge-universal graphs that build up the graph of the UC. Therefore, we have, on average, $2(u + v - 3)$ redundant nodes, which number we use in our calculations further on. This optimization also affects the depth by, on average, $u + v - 3$.

2. **Optimization for Fanin-1 Nodes:** We observe that in the skeleton of the Γ_1 edge-universal graph construction there is a fanin-1 node (denoted with B in Figures 2.2). Such fanin-1 nodes exist in the base-cases for a small number of poles as well (cf. Figure 2.2). These nodes are important to achieve fanin and fanout 2 of each nodes in the graph, but can be ignored and replaced with wires when translated into a circuit description, essentially resulting in the same UC. According to Valiant's construction, these gates would translate into universal switches with one real input (and an other arbitrary one). Instead, we translate each of them into two wires and therefore set the second input to the same as the first one. Since at least one such node can be ignored in each subgraph when nodes are translated into gates, this results in altogether around

$$2 \cdot \left(\sum_{i=0}^{\log_2(u+v+k^*)-1} 2^i \right) - 1 = 2(u+v+k^*) - 3 \tag{2.12}$$

less gates for the two Γ_1 edge-universal graphs. This improvement has no effect on the depth of the construction.

Since both the size and the depth are dependent on the underlying representation of the circuit building blocks (of the universal gate U and of the universal switch or X gate), and the secure computation protocol, we express the size of the universal circuit with the size and depth of U and of X as parameters. Including the above optimizations of altogether $4(u+v)+2k^*-9$, the approximate formula for the size of Valiant's optimized UC construction becomes

$$\begin{aligned}\text{size}(UC^{\text{opt}}_{u,v,k^*}) \approx [5(u+v+k^*)\log_2(u+v+k^*) - 17k^* - 19(u+v) \\ + 7.5\log_2(u+v+k^*) + 24] \cdot \text{size}(X) + k^* \cdot \text{size}(U).\end{aligned} \tag{2.13}$$

To obtain the exact size of the UC, we use the recursive relations depicted in Equations 2.5-2.6 and include our optimizations. Thus, the depth of the UC becomes

$$\text{depth}(UC^{\text{opt}}_{u,v,k^*}) \approx [u+v+2k^*+3] \cdot \text{depth}(X) + k^* \cdot \text{depth}(U). \tag{2.14}$$

Depending on the application, size(X) and size(U) as well as depth(X) and depth(U) can be optimized. Due to the PFE application, where XOR gates can be evaluated for free, we assess the ANDsize and ANDdepth of our AND-optimized implementations of universal gates and switches (cf. [39]). In general, a universal gate can be realized with 3 AND gates (and 6 XOR gates), and ANDdepth of 2 (total depth of 6). Universal switches can be realized with only one AND gate (and 3 XOR gates), and ANDdepth of 1 (total depth of 3) [43].
For private function evaluation, the size and the depth of U can be further optimized depending on the underlying secure computation protocol. In case the SFE implementation uses Yao's garbled circuit protocol [70], both ANDsize(U) and ANDdepth(U) can be minimized to 1, due to the fact that in some garbling schemes the evaluator does not learn the type of the evaluated gate such as in case of garbled 3-row-reduction [54]. Therefore, a universal gate can be implemented with one 2-input non-XOR gate [57].

2.4.4 Comparison of Valiant's UC for PFE with Other PFE Protocols

Mohassel et al. in [52] design a generic framework for PFE and apply it to three different scenarios: to the m-party GMW protocol [28], to Yao's garbled circuits [70] and to arithmetic

Circuit	u	k	v	$k^* - k$ ($\frac{k^*}{k}$)	Valiant	[44]	OT-based [52]
AES-non-exp	256	31,924	128	14,539 (1.46)	$1.150 \cdot 10^7$	$2.797 \cdot 10^7$	$6.243 \cdot 10^6$
AES-expanded	1,536	25,765	128	11,089 (1.43)	$9.211 \cdot 10^6$	$2.206 \cdot 10^7$	$4.943 \cdot 10^6$
DES-non-exp	128	19,464	64	12,290 (1.63)	$7.502 \cdot 10^6$	$1.560 \cdot 10^7$	$3.639 \cdot 10^6$
md5	512	43,234	128	22,623 (1.52)	$1.700 \cdot 10^7$	$3.995 \cdot 10^7$	$8.681 \cdot 10^6$
add_32	64	187	33	58 (1.31)	35,512	55,341	19,939
comp_32	64	150	1	1 (1.01)	19,384	40,222	15,424
mult_32x32	64	6,995	64	5,079 (1.73)	$2.522 \cdot 10^6$	$4.647 \cdot 10^6$	$1.184 \cdot 10^6$
Branching_18	72	121	4	3 (1.02)	17,312	30,994	11,994
CreditCheck	25	50	1	6 (1.12)	5,056	9,348	4,198
MobileCode	80	64	16	0 (1.00)	12,528	13,727	5,644

Table 2.1: The number of symmetric-key operations using different PFE protocols: Valiant's UC with SFE, the universal circuit construction from [44] or Mohassel et al.'s OT-based method from [52]. u, v and k denote the number of inputs, outputs and gates in the simulated circuit, and k^* denotes the number of gates in the equivalent fanout-2 circuit.

circuits using homomorphic encryption [15]. Both the two-party version of their framework with the GMW protocol and the solution with Yao's garbled circuit protocol has two alternatives: using homomorphic encryption they achieve linear complexity $\mathcal{O}(k)$ in the circuit size k and when using a solution solely based on oblivious transfers (OTs), they obtain a construction with $\mathcal{O}(k \log k)$ symmetric-key operations. The OT-based construction in both cases is more desirable in practice, since using OT extension the number of public-key operations can be reduced significantly [2, 32].

Since the asymptotical complexity of this construction and using Valiant's UC for PFE is the same, we compare these methods for PFE. We revisit the formulas provided in [52] for the PFE protocol based on Yao's garbled circuits and elaborate on the number of symmetric-key operations when the different PFE protocols are used. Mohassel et al. show that the total number of switches in their framework is $4k \log_2(2k) + 1$ that are evaluated using OT extension, for which they calculate $8k \log_2(2k) + 8$ symmetric-key operations together with $5k$ operations for evaluating the universal gates with Yao's protocol. We count only the work of the party that performs most of the work, i.e., $4k$ symmetric-key operations for creating a garbled circuit with k gates and 3 symmetric-key operations (two calls to a hash function and one call to a pseudorandom function (PRF)) for each OT using today's most efficient OT extension of [2]. Hence, according to our estimations, the protocol of [52] requires $12 \log_2(2k) + 4k + 12$ symmetric-key operations.

In the same way, we assume that in our case, for evaluating both the universal gates and switches, the garbler needs $4k$ symmetric-key operations. Thus, for a fair comparison, we essentially update Table 4 from the full version of [52, Appendix J.1], where Valiant's UC size was calculated with assumed $k^* = 2k + v$, without calculating 4 operations for the garbling.

We took our example circuit files of varying size in Table 2.1 from two different sources and elaborate on the resulting number of symmetric-key operations using the different constructions. The first 7 circuits we obtained from the function set of [66] and the last three from the FairplayPF extension of the Fairplay compiler [44, 51]. The example circuits that we took from [66] had to be converted to our desired SHDL format, which was a necessary step in order to be able to elaborate on the performance of these more complicated circuits as well. We included the INV gates in the function table of the consecutive gate and therefore, resulted in smaller gate numbers k for the equivalent SHDL circuits with arbitrary fanout. Then, these SHDL circuits were considered as input circuits for our tool.

Circuit	UC Compile Time (ms)	UC I/O Time (ms)	GMW Time (ms)	GMW Communic. (KB)	Yao Time (ms)	Yao Communic. (KB)
AES-non-exp	2,909.2	6,331.2	5,522.08	137,561.13	2,349.35	88,417.61
AES-expanded	2,103.7	5,063.6	4,136.72	109,033.79	1,878.75	70,097.48
DES-non-exp	1,596.2	4,173.5	2,695.51	76,644.38	1,310.52	48,180.69
md5	4,043.5	8,785.4	7,041.12	169,558.83	3,547.68	110,043.59
add_32	11.4	63.8	31.97	457.77	26.49	224.77
comp_32	5.8	34.1	29.94	340.23	8.90	159.73
mult_32x32	328.9	1,443.2	1,092.46	31,053.53	539.98	18,741.85
Branching_18	4.8	31.4	26.23	307.77	17.34	145.87
CreditCheck	1.2	11.4	26.25	113.35	5.67	45.15
MobileCode	3.2	26.3	25.71	202.50	28.16	103.45

Table 2.2: Running time and communication for our UC-based PFE implementation with ABY. We include the compile time, the I/O time of the UC compiler, and the evaluation time (in milliseconds) and the total communication (in Kilobytes) between the parties in GMW as well as in Yao sharing.

We now compare the size of the three two-party PFE protocols: the two UC-based PFE with secure computation (Valiant's UC [39] and [44]) and the OT-based method of [52]. We assess our findings in Table 2.1. We note that our numbers are estimations, i.e., we do not consider that [52] works with circuits made up solely of NAND gates. Since Valiant's UC construction depends also on the number of gates with fanout more than 2 in the original circuit, we include the number of copy gates, $(k^* - k)$ in the table. We emphasize the ratio between the new number of gates k^* and the original number of gates k and conclude that in general circuits, it is well below the maximal $\frac{k^*}{k} \sim 2$. The size of the UC construction from [44] obviously makes their method less efficient, in our examples using more than twice as many symmetric-key operations as the method with Valiant's UC and four times as many as Mohassel et al.'s efficient OT-based method [52]. We conclude that universal circuits are not the most efficient solution to perform PFE, however, we show the feasibility of generating and evaluating UCs simulating large circuits. We emphasize that even though the PFE-specific protocol from [52] achieves better results for PFE, universal circuits are generic and can be applied for various other scenarios (cf. [39]), and the most efficient UC construction is Valiant's construction.

Our Experimental Results.

We validated the practicality of Valiant's universal circuit construction with an efficient implementation. We ran our experiments on two Desktop PCs, each equipped with an Intel Haswell i7-4770K CPU with 3.5 GHz and 16 GB RAM, that are connected via Gigabit-LAN and give our benchmarks in Table 2.2. We are able to generate UCs up to around 300,000 gates of the simulated circuit, i.e., which results in billions of gates in the UC. Until now, the only implementation of universal circuits was given in [44], which is outperformed by Valiant's construction already for a couple of hundred gates due to its asymptotically larger complexity. We show the real practicality of UCs through experimental results proving the efficiency of our implementation of PFE with the ABY framework [19]. Furthermore, due to its asymptotically smaller depth, we are also able to evaluate our generated UCs with the GMW protocol [28], whereas the construction from [44] was only evaluated with Yao's garbled circuit protocol. We do not directly compare our runtimes with the method of [52], since to the best of our knowledge, their framework has not yet been implemented.

Converting from circuit descriptions and writing into and reading out from files slows down the program significantly, but it still achieves good performance for practical circuits such as AES and DES. Our implementation in ABY can evaluate most of the circuits in both the GMW and Yao's protocols, but for some examples it runs out of memory (e.g. SHA-256). However, improvements on SFE protocols imply improvements on UC-based PFE frameworks as well. As can be seen in Table 2.2, the evaluation time and the communication in case of Yao's garbled cirucit protocol is about a factor of two smaller than that of the GMW protocol. This difference is due to the more efficient universal gate construction with only one gate for the case of Yao's protocol in contrast to the universal gates used in the GMW protocol with ANDsize = 3 and ANDdepth = 2.

2.5 TinyTable Secure Two-Party Computation

We proposed a new protocol, nicknamed TinyTable [16], for maliciously secure 2-party computation in the preprocessing model. One version of the protocol is useful in practice and allows, for instance, secure AES encryption with latency about 1ms and amortized time about 0.5 μs per AES block on a fast cloud set-up. Another version is interesting from a theoretical point of view: we achieve a maliciously and unconditionally secure 2-party protocol in the preprocessing model for computing a Boolean circuit, where both the communication complexity and preprocessed data size needed is $O(s)$ where s is the circuit size, while the computational complexity is $O(k^\epsilon s)$ where k is the statistical security parameter and $\epsilon < 1$ is a constant. For general circuits with no assumption on their structure, this is the best asymptotic performance achieved so far in this model.
The idea of the protocol is to implement each (non-linear) gate by a scrambled version of its truthtable. Players will do look-ups in the tables using bits that are masked by uniformly random bits that are chosen in the preprocessing phase together with the tables. This first version of our protocol has data and communication complexity $O(s)$ and computational complexity $O(ks)$. Although this is asymptotically inferior to previous protocols when counting elementary bit operations, it works much better in practice: XOR and NOT gates require no communication, and for each non-linear gate, each player sends and receives 1 bit and XORs 1 word from memory into a register. This means that TinyTable saves a factor of at least 2 in communication in comparison to standard *passively* secure protocols in the preprocessing model, such as GMW with precomputed OT's or the protocol using circuit randomization via Beaver-triples.
We implemented a version of this protocol that was optimised for AES computation, by using tables for each S-box. This is more costly in preprocessing but, compared to a Boolean circuit implementation, it reduces the round complexity of the on-line phase dramatically (to about 10).

Construction for passive security

Let C be a Boolean circuit with fan-in 2 gates denoted $G_1, \ldots, G_N$ and let $w_1, \ldots, w_W$ be the wires. The gates are fixed in an order that allows the circuit to be evaluated gate by gate, such that the output gates come last, and such that when we are about to evaluate gate i, its inputs have already been computed. We call the wires coming out of the output gates *output wires* and assume for simplicity that both parties are to learn the output.
We first specify a functionality for preparing preprocessing material that will allow computation of the circuit with semi-honest security, see Fig. 2.5.

Preprocessing Functionality F_{sem}^{pre}.

1. On input C from both players, do the following: For each wire w_u, choose a random masking bit r_u. This bit will be used to mask the bit b_u that will actually be on w_u when we do the computation, i.e., $e_u = b_u \oplus r_u$ will become known to the players. If w_u is an input wire, give r_u to the player who owns w_u.
2. For each gate G_i, with input wires w_u, w_v and output wire w_o, we will construct two tables A_i, B_i each with 4 entries, indexed by bits (c,d). This is done as follows: for each of the 4 possible values of bits (c, d), do:
   (a) If both player are honest, choose a random bit $s_{c,d}$. Otherwise take $s_{c,d}$ as input from the adversary. Let $G_i(\cdot,\cdot)$ denotes the function computed by gate G_i.
   (b) If both parties are honest, or if A is corrupt, set $A_i[c,d] = s_{c,d}$ and $B_i[c,d] = s_{c,d} \oplus (r_o \oplus G_i(c \oplus r_u, d \oplus r_v))$.
   (c) If B is corrupt, set $B_i[c,d] = s_{c,d}$ and $A_i[c,d] = s_{c,d} \oplus (r_o \oplus G_i(c \oplus r_u, d \oplus r_v))$.
3. For each gate G_i, hand A_i to player A and B_i to player B. For each output wire w_u, send r_u to both players.

Figure 2.5: Functionality for preprocessing, semi-honest security.

The idea behind the construction of the tables is that when the time comes to compute gate G_i, both players will know "encrypted bits" $e_u = b_u \oplus r_u$ and $e_v = b_v \oplus r_v$, for the input wires, where b_u,b_v are the actual "cleartext" bits going into G_i, and r_u, r_v are random masking bits that are chosen in the preprocessing. In addition, the preprocessing sets up two tables A_i, B_i for each gate, one held by party A and one held by party B. These tables are used to get hold of a similar encrypted bit for the output wire: $e_o = b_o \oplus r_o$, where $b_o = G_i(b_u, b_v)$. This works because the tables are set up such that $A_i[e_u,e_v], B_i[e_u,e_v]$ is an additive sharing of $b_o \oplus r_o$, i.e.,

$$A_i[b_u \oplus r_u,b_v \oplus r_v] \oplus B_i[b_u \oplus r_u,b_v \oplus r_v] = b_o \oplus r_o \ .$$

These considerations lead naturally to the protocol for computing C securely shown in Fig 2.6.

Protocol π_{sem}.

1. A and B send C as input to F and get a set of tables $\{A_i, B_i |\ i = 1 \ldots N\}$, as well as a bit b_u for each input wire w_u.
2. For each input wire w_u, if A holds input x_u for this wire, send $e_u = x_u \oplus b_u$ to B. If B holds input x_u, send $e_u = x_u \oplus b_u$ to A.
3. For $i = 1$ to N, do: Let G_i have input wires w_u, w_v and output wire w_o (so that e_u, e_v have been computed). A sends $A_i[e_u,e_v]$ to B, and B sends $B_i[e_u,e_v]$ to A. Set $e_o = A_i[e_u,e_v] \oplus B_i[e_u,e_v]$.
4. The parties output the bits $\{e_o \oplus r_o |\ w_o$ is an output wire$\}$.

Figure 2.6: Protocol for semi-honest security.

Chapter 3

Tools with Improved Efficiency

3.1 Simple and Efficient Oblivious Transfer (short description)

This section is based on [14]. The complete description of this work appears in deliverable D13.1.

Oblivious Transfer (OT) is a cryptographic primitive defined as follows: in its simplest flavour, 1-out-of-2 OT, a sender has two input messages M_0 and M_1 and a receiver has a choice bit c. At the end of the protocol the receiver is supposed to learn the message M_c and nothing else, while the sender is supposed to learn nothing. Perhaps surprisingly, this extremely simple primitive is sufficient to implement any cryptographic task [38]. OT is also one of the main building blocks in most secure-two party computation protocol such as Yao's garbled circuits, the GMW protocol etc.

This work designed a novel, extremely *simple*, *efficient* and *secure* OT protocol, which is a simple tweak of the celebrated Diffie-Hellman (DH) key exchange protocol. Given a group $\mathbb{G}$

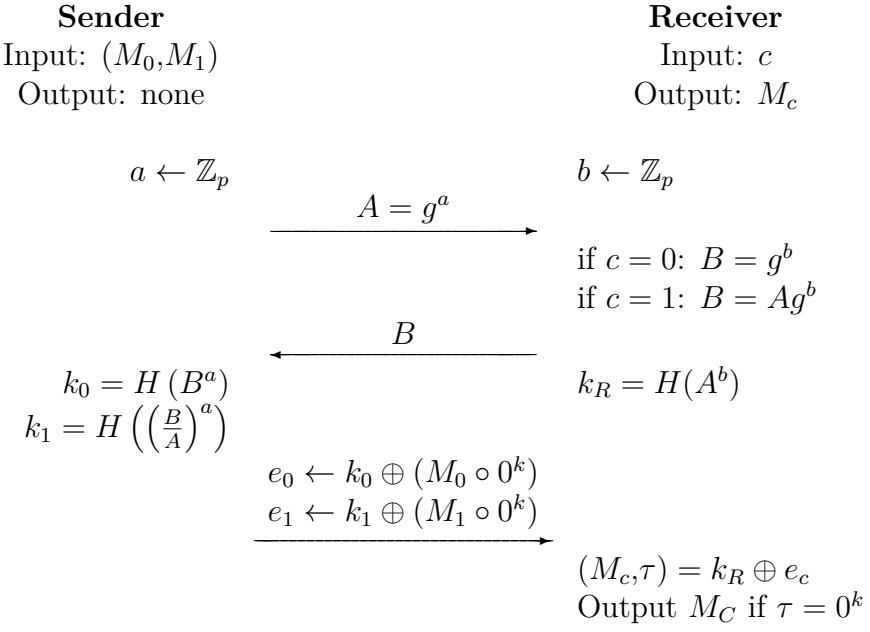


Figure 3.1: Our protocol in a nutshell

and a generator g, the DH protocol allows two players Alice and Bob to agree on a key as follows: Alice samples a random a, computes $A = g^a$ and sends A to Bob. Symmetrically Bob samples a random b, computes $B = g^b$ and sends B to Alice. Now both parties can compute $g^{ab} = A^b = B^a$ from which they can derive a key k. The key observation is now that Alice can also derive a different key from the value $(B/A)^a = g^{ab-a^2}$, and that Bob cannot compute this group element (assuming that the computational DH problem is hard).
We can now turn this into an OT protocol by letting Alice play the role of the sender and Bob the role of the receiver (with choice bit c) as shown in Figure 3.1. The first message (from Alice to Bob) is left unchanged (and can be reused over multiple instances of the protocol) but now Bob computes B as a function of his choice bit c: if $c = 0$ Bob computes $B = g^b$ and if $c = 1$ Bob computes $B = Ag^b$. At this point Alice derives two keys k_0,k_1 from $(B)^a$ and $(B/A)^a$ respectively. It is easy to check that Bob can derive the key k_c corresponding to his choice bit from A^b, but cannot compute the other one. This can be seen as a *random OT* i.e., an OT where the sender has no input but instead receives two random messages from the protocol, which can be used later to encrypt his inputs.
Combining the above random OT protocol with the right symmetric encryption scheme (e.g., a *robust encryption scheme*) achieves security in a strong, simulation based sense and in particular the protocol can be proven UC-secure against active and adaptive corruptions in the random oracle model.

3.2 Actively Secure Oblivious Transfer Extension (short description)

This section is based on [4]. The complete description of this work appears in deliverable D13.1.
Oblivious Transfer (OT) is a cryptographic primitive that is fundamental for secure computation. In an 1-out-of-2 OT, a sender P_S holds a pair of n-bit strings (x^0, x^1) of which a receiver P_R with choice bit r wants to obtain x^r such that P_S does not learn r and P_R gains no information about x^{1-r}.
It has been proven that OT can not be based on one-way functions alone [30] and hence computing an OT requires public-key cryptography. However, public-key cryptography is very costly and many applications in secure computation typically require millions up to billions of OTs, so getting efficient OT is of high importance. In [6] it was shown that a small number of real *base-OTs*, that were computed using OT protocols based on expensive public-key cryptography, can be extended to an arbitrarily large number of OTs using efficient symmetric cryptography only. Due to their nature, these protocols are called *OT extension* protocols.
While the feasibility result of [6] was still costly in concrete terms, the work of [31] showed how to extend OTs at a relatively low cost. The main protocol that was introduced was secure against passive (or semi-honest) adversaries, which try to learn as much information as possible but honestly follow the protocol description. An extension for security against active (or malicious) adversaries, which can arbitrarily deviate from the protocol description, was also given but incurred a huge cost overhead (around factor 40 compared to the passively secure variant).
Several follow-up works reduced the cost for both the passively- and actively secure variants of the protocol. One of these works is [4], which reduced the cost overhead of the active secure protocol over the passive secure protocol to factor 1.4.

3.3 Improved Actively Secure Oblivious Transfer Extension

This section is a continuation of Section 3.2, which was also part of deliverable D13.1 [36], and where the active secure OT extension protocol of [4] was described. In this section we detail recent improvements of oblivious transfer (OT) extension protocols.
In regular OT protocols, such as Protocol 1 in §3.2 in D13.1), the sender P_S holds two messages of which the receiver P_R obliviously obtains one. (For completeness, we describe this protocol in Fig 1.)
In the following section, which was published in [3], we present further optimizations that are specifically tailored to the use of OT extensions in secure computation protocols summarized in Table 3.1: Correlated OT (Section 3.3.1), Sender Random OT (Section 3.3.2), Receiver Random OT (Section 3.3.3), and Random OT (Section 3.3.4). We first give the intuition and overview of the functionalities and then present a formal definitions and proofs of security.

Protocol	**Applicability**	$P_R \to P_S$	$P_S \to P_R$	H
Original [21]+[31]	All applications	$m\ell$	$2mn$	CR
C-OT Section 3.3.1	x_j^0 random; x_j^1 correlated with Δ_j	$m\ell$	mn	RO
SR-OT Section 3.3.2	x_j^0, x_j^1 random, r_j chosen	$m\ell$	0	RO
RR-OT Section 3.3.3	x_j^0, x_j^1 chosen, r_j random	$m(\ell-1)$	$2mn$	RO
R-OT Section 3.3.4	x_j^0, x_j^1, r_j random	$m(\ell-1)$	0	RO

Table 3.1: Bits sent for sender P_S and receiver P_R for m 1-out-of-2 OT extensions of n-bit strings and security parameter κ for the semi-honest OT extension protocol of [31] with our optimizations.

3.3.1 Correlated OT (C-OT)

When performing OT extension, often the sender does not need to transfer two independent n-bit strings (x_j^0, x_j^1). In some protocols, x_j^0 and x_j^1 only need to be correlated by a value Δ_j and a correlation function f_{Δ_j}, while one of the two strings can be constant and publicly known or random. For instance, the Private Set-Intersection protocol of [20] fixes $x_j^0 = 0$ and transfers only x_j^1 (hence, we can set $\Delta_j = x_j^1$ and $f_{\Delta_j}(x_j^0) = \Delta_j$) and the Hamming Distance Protocol of [12] requires a random x_j^0 and a correlated $x_j^1 = f_{\Delta_j}(x_j^0) = x_j^0 + \Delta_j$. We can alter the functionality of our OT extension protocols to compute correlated OT as follows. Since x_j^0 is just a random value, P_S can set $x_j^0 = H(j,\mathbf{q}_j)$ and $x_j^1 = f_{\Delta_j}(x_j^0)$ and can send the *single* value $y_j = x_j^1 \oplus H(j,\mathbf{q}_j \oplus \mathbf{s})$. P_R defines its output as $H(j,\mathbf{t}_j)$ if $r_j = 0$ or as $y_j \oplus H(j,\mathbf{t}_j)$ if $r_j = 1$. For OT on n-bit strings, we thereby reduce the communication from P_S to P_R from $2n + \ell$ to $n + \ell$ per OT.

Defining the functionality. The input x_j^0 of the sender is implicitly defined by the protocol. Nevertheless, the sender may choose x_j^1 in any arbitrarily way, including as an arbitrary function of x_j^0. That is, in the protocol the sender has the freedom to choose x_j^1 as a function of x_j^0. When defining the corresponding functionality, we need to model this fact. As a result, the functionality C-OT is defined as a *reactive* functionality, where the functionality chooses x_j^0 at random, gives it to the sender, and then the sender replies with its choice for x_j^1. We proceed with a formal description of the functionality (Functionality 1), the protocol (Protocol 2) and its proof of security (Theorem 3.3.1).

PROTOCOL 1 (Active secure OT extension protocol of [4].)

- **Input of P_S:** m pairs (x_j^0, x_j^1) of n-bit strings, $1 \le j \le m$.
- **Input of P_R:** m selection bits $\mathbf{r} = (r_1, \ldots, r_m)$.
- **Common Input:** Symmetric security parameter κ and statistical security parameter ρ. It is assumed that the number of base-OTs is $\ell = \kappa + \rho$.
- **Oracles and cryptographic primitives: The parties use an ideal $\ell \times OT_\kappa$ functionality, which computes ℓ OTs on κ-bit input values, pseudorandom generator $G : \{0,1\}^\kappa \to \{0,1\}^m$, and random-oracle H.**

1. *Initial OT Phase:*
   (a) P_S initializes a random vector $\mathbf{s} = (s_1, \ldots, s_\ell) \in \{0,1\}^\ell$ and P_R chooses ℓ pairs of seeds $\mathbf{k}_i^0, \mathbf{k}_i^1$ each of size κ.
   (b) The parties invoke the $\ell \times OT_\kappa$-functionality, where P_S acts as the *receiver* with input $\mathbf{s}$ and P_R acts as the *sender* with inputs $(\mathbf{k}_i^0, \mathbf{k}_i^1)$ for every $1 \le i \le \ell$.

   For every $1 \le i \le \ell$, let $\mathbf{t}^i = G(\mathbf{k}_i^0)$. Let $T = [\mathbf{t}^1 | \ldots | \mathbf{t}^\ell]$ denote the $m \times \ell$ bit matrix where its ith column is $\mathbf{t}^i$ for $1 \le i \le \ell$. Let $\mathbf{t}_j$ denote the jth row of T for $1 \le j \le m$.
2. *OT Extension Phase (Part I):*
   (a) P_R computes $\mathbf{t}^i = G(\mathbf{k}_i^0)$ and $\mathbf{u}^i = \mathbf{t}^i \oplus G(\mathbf{k}_i^1) \oplus \mathbf{r}$, and sends $\mathbf{u}^i$ to P_S for every $1 \le i \le \ell$.
3. *Consistency Check of $\mathbf{r}$:*
   (a) For every pair $\alpha, \beta \subseteq [\ell]^2$, P_R defines the four values:
   $$h_{\alpha,\beta}^{0,0} = H(G(\mathbf{k}_\alpha^0) \oplus G(\mathbf{k}_\beta^0)) \qquad h_{\alpha,\beta}^{0,1} = H(G(\mathbf{k}_\alpha^0) \oplus G(\mathbf{k}_\beta^1)) ,$$
   $$h_{\alpha,\beta}^{1,0} = H(G(\mathbf{k}_\alpha^1) \oplus G(\mathbf{k}_\beta^0)) \qquad h_{\alpha,\beta}^{1,1} = H(G(\mathbf{k}_\alpha^1) \oplus G(\mathbf{k}_\beta^1)) .$$
   It then sends $\mathcal{H}_{\alpha,\beta} = (h_{\alpha,\beta}^{0,0}, h_{\alpha,\beta}^{0,1}, h_{\alpha,\beta}^{1,0}, h_{\alpha,\beta}^{1,1})$ to P_S.
   (b) For every pair $\alpha, \beta \subseteq [\ell]^2$, P_S knows $s_\alpha, s_\beta, \mathbf{k}_\alpha^{s_\alpha}, \mathbf{k}_\beta^{s_\beta}, \mathbf{u}^\alpha, \mathbf{u}^\beta$ and checks that:
      i. $h_{\alpha,\beta}^{s_\alpha, s_\beta} = H(G(\mathbf{k}_\alpha^{s_\alpha}) \oplus G(\mathbf{k}_\beta^{s_\beta}))$.
      ii. $h_{\alpha,\beta}^{\overline{s_\alpha}, \overline{s_\beta}} = H(G(\mathbf{k}_\alpha^{s_\alpha}) \oplus G(\mathbf{k}_\beta^{s_\beta}) \oplus \mathbf{u}^\alpha \oplus \mathbf{u}^\beta)$ $\quad (= H(G(\mathbf{k}_\alpha^{\overline{s_\alpha}}) \oplus G(\mathbf{k}_\beta^{\overline{s_\beta}}) \oplus \mathbf{r}^\alpha \oplus \mathbf{r}^\beta))$.
      iii. $\mathbf{u}^\alpha \ne \mathbf{u}^\beta$.

   In case one of these checks fails, P_S aborts and outputs $\perp$.
4. *OT Extension Phase (Part II):*
   (a) For every $1 \le i \le \ell$, P_S defines $\mathbf{q}^i = (s_i \cdot \mathbf{u}^i) \oplus G(\mathbf{k}_i^{s_i})$. (Note that $\mathbf{q}^i = (s_i \cdot \mathbf{r}) \oplus \mathbf{t}^i$.)
   (b) Let $Q = [\mathbf{q}^1 | \ldots | \mathbf{q}^\ell]$ denote the $m \times \ell$ bit matrix where its ith column is $\mathbf{q}^i$. Let $\mathbf{q}_j$ denote the jth row of the matrix Q. (Note that $\mathbf{q}^i = (s_i \cdot \mathbf{r}) \oplus \mathbf{t}^i$ and $\mathbf{q}_j = (r_j \cdot \mathbf{s}) \oplus \mathbf{t}_j$.)
   (c) P_S sends (y_j^0, y_j^1) for every $1 \le j \le m$, where:
   $$y_j^0 = x_j^0 \oplus H(j, \mathbf{q}_j) \quad \text{and} \quad y_j^1 = x_j^1 \oplus H(j, \mathbf{q}_j \oplus \mathbf{s})$$
   (d) For $1 \le j \le m$, P_R computes $x_j = y_j^{r_j} \oplus H(j, \mathbf{t}_j)$.
5. **Output:** P_R outputs $(x_1^{r_1}, \ldots, x_m^{r_m})$; P_S has no output.

FUNCTIONALITY 1 (The Correlated OT Functionality C-OT)

1. P_R sends its input $\mathbf{r} = (r_1, \ldots, r_m)$.
2. The functionality chooses m random n-bit strings $x_1^0, \ldots, x_m^0$ and sends them to P_S.
3. P_S sends $x_1^1, \ldots, x_m^1$ to the functionality.
4. P_R gets as output $x_1^{r_1}, \ldots, x_m^{r_m}$.

PROTOCOL 2 (Implementing Correlated OT (C-OT))
We follow Protocol 1 in §3.2 in D13.1, where the sender has input $f_{\Delta_1}, \ldots, f_{\Delta_m}$ instead of $(x_1^0, x_1^1), \ldots, (x_m^0, x_m^1)$. In Step 4 (c), we have the following modification:

1. P_S defines $x_j^0 = H(j, \mathbf{q}_j)$ and $x_j^1 = f_{\Delta_j}(x_j^0)$.
2. P_S sends y_j for every $1 \leq j \leq m$, where: $y_j = x_j^1 \oplus H(j, \mathbf{q}_j \oplus \mathbf{s})$.
3. For $1 \leq j \leq m$, P_R computes $x_j = H(j, \mathbf{t}_j)$ if $r_j = 0$, and $x_j = y_j \oplus H(j, \mathbf{t}_j)$ otherwise.

Output: P_S outputs $(x_1^0, x_1^1), \ldots, (x_m^0, x_m^1)$, P_R outputs $\mathbf{r}$.

Theorem 3.3.1 *Assuming that H is a programmable random oracle and G is a pseudorandom generator, then Protocol 2 securely computes the C-OT functionality (Functionality 1) in the $\ell \times OT_\kappa$-hybrid model in the presence of a static malicious adversary.*

Proof Sketch: We sketch the simulator and the proof, and relate to the full proof of the protocol [3].

The case of corrupted Sender. The case of corrupted sender here is more subtle than the proof of the general protocol, and the functionality is now a reactive one. Moreover, we prove security in the *programmable* random oracle model.
The simulator chooses a random input $\mathbf{r}$ and follows the execution of the protocol with the corrupted sender and with an honest receiver with input $\mathbf{r}$. Specifically, the adversary first outputs a vector $\mathbf{s}$ of size ℓ. The simulator chooses random $\{\mathbf{k}_i^0, \mathbf{k}_i^1\}_{i=1}^{\ell}$ and sends them back to the adversary, together with the $\mathbf{u}^i$ messages and the necessary checks $\mathcal{H}_{\alpha,\beta}$, all set according to the protocol specifications. Note that this determines the matrices T and Q.
The simulator then receives the inputs $x_1^0, \ldots, x_m^0$ from the trusted party, and it programs the random oracle H such that for every $1 \leq j \leq m$, $H(j, \mathbf{q}_j) = x_j^0$ and chooses random output for $H(j, \mathbf{q}_j \oplus \mathbf{s})$. In case the adversary has already queried H for one of these values before the simulator programs it, the simulator fails. The simulator receives from the adversary the messages $y_1, \ldots, y_m$, defines for every $1 \leq j \leq m$ the input $x_j^1 = y_j \oplus H(j, \mathbf{q}_j \oplus \mathbf{s})$, and sends the inputs $x_1^1, \ldots, x_m^1$ to the trusted party.
Clearly, the probability that the adversary that makes at most q queries to $H(j, \mathbf{q}_j)$ or $H(j, \mathbf{q}_j \oplus \mathbf{s})$ before it receives the messages $\mathbf{u}^1, \ldots, \mathbf{u}^\ell$ is bounded by $q \cdot 2^{-\ell}$, and therefore the probability that the simulation fails is bounded by this amount.

The case of corrupted Receiver. The simulator is the same as in Theorem 5.2 in [3], where in the last step, instead of sending to the adversary the two messages y_j^0, y_j^1, it sends only y_j^1. Note that if no critical query then the input $x_j^{1-r_j}$ is hidden from the adversary or

the distinguisher. Specifically, in case $r_j = 0$, the value $\mathbf{t}_j = \mathbf{q}_j$ and therefore $H(j,\mathbf{q}_j \oplus \mathbf{s})$ is distributed uniformly, and the value $y_j = x_j^1 \oplus H(j,\mathbf{q}_j \oplus \mathbf{s})$ is distributed uniformly as well. In case $r_j = 1$ it holds that $\mathbf{t}_j = \mathbf{q}_j \oplus \mathbf{s}$, which implies that $x_j^0 = H(j,\mathbf{q}_j) = H(j,\mathbf{t}_j \oplus \mathbf{s})$ is distributed uniformly and hidden from the adversary. ■

3.3.2 Sender Random OT (SR-OT)

When using OT extensions for implementing the OT-based Private Set Intersection (PSI) protocol of [58, 60], the efficiency can be improved even further. In this case, the inputs for P_S in every OT are *independent random* strings m^0 and m^1. Thus, the sender can allow the OT extension protocol (functionality) Sender Random OT (SR-OT) to determine *both* of its inputs randomly. This is achieved in the OT extension protocol by having P_S define $m^0 = H(j,\mathbf{q}_j)$ and $m^1 = H(j,\mathbf{q}_j \oplus \mathbf{s})$. Then, P_R computes m^{r_j} just as $H(j,\mathbf{t}_j)$. With this optimization, we obtain that the entire communication in the OT extension protocol consists only of the initial base-OTs, together with the messages $\mathbf{u}^1,\ldots,\mathbf{u}^\kappa$, and there are *no* y_j messages. This is a dramatic improvement of bandwidth. In particular, for the OT-PSI protocol of [58, 60], which performs $O(n\sigma)$ OTs on $\rho + 2\log_2(n)$ bit-strings, where n is the number of elements in both parties' sets, σ is the bit-length of each element, and ρ is the statistical security parameter, the communication from P_S to P_R is reduced from $O(n\sigma)$ to $O(n)$.

Formal description of the functionality. We proceed with a formal description of the functionality (Functionality 2), the protocol (Protocol 3) and its proof of security (Theorem 3.3.2).

FUNCTIONALITY 2 (Sender Random OT)

- **Input:** P_S holds no input, P_R holds $\mathbf{r} = (r_1,\ldots,r_m)$.
- **The functionality:** The functionality chooses m pairs of random strings of size n each, $(x_1^0,x_1^1),\ldots,(x_m^0,x_m^1)$.
- **Output:** P_S outputs $(x_1^0,x_1^1),\ldots,(x_m^0,x_m^1)$. P_R outputs $(x_1^{r_1},\ldots,x_m^{r_m})$.

PROTOCOL 3 (Implementing Sender Random OT (SR-OT))
We follow Protocol 1 in §3.2 in D13.1, where the sender does not have any input. In Steps 4 (c) and 4 (d), we have the following modification:

1. P_S defines $x_j^0 = H(j,\mathbf{q}_j)$ and $x_j^1 = H(j,\mathbf{q}_j \oplus \mathbf{s})$ for every $1 \le j \le m$.
2. P_R defines $x_j^{r_j} = H(j,\mathbf{t}_j)$ for every $1 \le j \le m$. Note that there is no interaction between the parties in this step.

Output: The sender outputs (x_j^0,x_j^1), the receiver outputs $x_j^{r_j}$.

Theorem 3.3.2 *Assuming that H is a programmable random oracle, G is a pseudorandom generator, Protocol 3 securely computes the SR-OT functionality (Functionality 2) in the $\ell \times OT_\kappa$-hybrid model in the presence of a static malicious adversary.*

Proof Sketch:

The case of corrupted Sender. We prove security in the programmable random oracle model.
The simulator chooses a random input $\mathbf{r}$ and follows the execution of the protocol with the corrupted sender and with an honest receiver with input $\mathbf{r}$. Specifically, the adversary first outputs a vector $\mathbf{s}$ of size ℓ. The simulator chooses random $\{\mathbf{k}_i^0,\mathbf{k}_i^1\}_{i=1}^{\ell}$ and sends them back to the adversary, together with the $\mathbf{u}^i$ messages and the necessary checks $\mathcal{H}_{\alpha,\beta}$, all set according to the protocol specifications. Note that this determines the matrices T and Q.
The simulator then receives the inputs $(x_1^0,x_1^1),\ldots,(x_m^0,x_m^1)$ from the trusted party, and it *programs* the random oracle H such that for every $1 \leq j \leq m$, $H(j,\mathbf{q}_j) = x_j^0$ and $H(j,\mathbf{q}_j \oplus \mathbf{s}) = x_j^1$.

The case of corrupted Receiver. The simulator is the same as in Theorem 5.2 in [3], where the only modification is that the simulator does not send the receiver any message in the transfer phase. Assuming that the receiver or the distinguisher do not make any critical query, the value $H(j,\mathbf{t}_j \oplus \mathbf{s})$ is hidden and distributed uniformly. In case where $r_j = 0$, this value is x_j^1 and in case where $r_j = 1$, it is x_j^0. The theorem follows. ■

3.3.3 Receiver Random OT (RR-OT)

Analogously to the Sender Random OT, in the Receiver Random OT (RR-OT), P_R obtains his input choice bits $\mathbf{r}$ as random output of the protocol execution. Our instantiation of RR-OT in OT extension allows P_R to save one bit of communication per OT. Recall that in Step 2(a) in Protocol 1 in §3.2 in D13.1, P_R sends $\mathbf{u}^i = G(\mathbf{k}_i^0) \oplus G(\mathbf{k}_i^1) \oplus \mathbf{r}$ for $1 \leq i \leq \ell$. However, if we allow $\mathbf{r}$ to be randomly chosen, we can set $\mathbf{r} = G(\mathbf{k}_1^0) \oplus G(\mathbf{k}_1^1)$ and $\mathbf{t}^1 = G(\mathbf{k}_1^0)$ and only need to transfer $\mathbf{u}^{i'} = G(\mathbf{k}_{i'}^0) \oplus G(\mathbf{k}_{i'}^1) \oplus \mathbf{r}$ for $2 \leq i' \leq \ell$. P_S can then compute $\mathbf{q}^1 = G(\mathbf{k}_1^{s_1})$ and, as before, $\mathbf{q}^{i'} = (s_{i'} \cdot \mathbf{u}^{i'}) \oplus G(\mathbf{k}_{i'}^{s_{i'}})$. Thereby, the communication from P_R to P_S is reduced by one bit per OT.
We proceed with a formal description of the functionality (Functionality 3), protocol (Protocol 4) and its proof of security (Theorem 3.3.3).

FUNCTIONALITY 3 (The Receiver Random OT Functionality (RR-OT))

- **Input:** P_S holds m pairs $(x_1^0,x_1^1),\ldots,(x_m^0,x_m^1)$ of n-bit strings.
- **In case of corrupted receiver:** P_R sends m-bits $\mathbf{r} = (r_1,\ldots,r_m)$ to the functionality.
- **In case of honest receiver:** P_R has not input. The functionality chooses m random bits $\mathbf{r} = (r_1,\ldots,r_m)$.
- **Output:** P_S has no output; P_R outputs $(x_1^{r_1},\ldots,x_m^{r_m})$ and $\mathbf{r}$.

Theorem 3.3.3 *Assuming that H is a random oracle, G is a pseudorandom generator, Protocol 4 securely computes the RR-OT functionality (Functionality 3) in the $\ell \times OT_\kappa$-hybrid model in the presence of a static malicious adversary.*

Proof Sketch: Note that the random oracle does not have to be programmable. Regarding correctness, for every $2 \leq i \leq \ell$ it holds that $\mathbf{q}^i = \mathbf{t}^i \oplus (s_i \cdot \mathbf{r})$. For $i = 1$, if $s_1 = 0$ then $\mathbf{q}^1 = G(\mathbf{k}_1^0) = \mathbf{t}^i$; in case $s_1 = 1$ then $\mathbf{q}^1 = G(\mathbf{k}_1^1) = G(\mathbf{k}_1^0) \oplus \mathbf{r} = \mathbf{t}^i \oplus \mathbf{r}$, and therefore $\mathbf{q}^1 = \mathbf{t}^1 \oplus (s_1 \cdot \mathbf{r})$ as well.

PROTOCOL 4 (Implementing Receiver Random OT (RR-OT))
We follow Protocol 1 in §3.2 in D13.1 with the following modifications:

1. P_R has no input.
2. Given the chosen keys $\{\mathbf{k}_i^0,\mathbf{k}_i^1\}_{i=1}^{\ell}$, P_R sets $\mathbf{r} = G(\mathbf{k}_1^0) \oplus G(\mathbf{k}_1^1)$.
3. For every $2 \leq i \leq \ell$, P_R sets $\mathbf{u}^i = G(\mathbf{k}_i^0) \oplus G(\mathbf{k}_i^1) \oplus \mathbf{r}$, and sends $\mathbf{u}^2,\ldots,\mathbf{u}^\ell$ to P_S. Note that $\mathbf{u}^1$ is not sent.
4. In case of our actively secure OT extension protocol, the parties check consistency as previously.
5. P_R defines $T = [\mathbf{t}^1 \mid \ldots \mid \mathbf{t}^\ell]$ where $\mathbf{t}^i = G(\mathbf{k}_i^0)$ for every $1 \leq i \leq \ell$ as in Protocol 1 in §3.2 in D13.1.
6. P_S defines $Q = [\mathbf{q}^1 \mid \ldots \mid \mathbf{q}^\ell]$ where $\mathbf{q}^1 = G(\mathbf{k}_1^{s_1})$, and for every $2 \leq i \leq \ell$, $\mathbf{q}^i$ is defined as in Protocol 1 in §3.2 in D13.1, i.e., $\mathbf{q}^i = G(\mathbf{k}_i^0)$ if $s_i = 0$; otherwise, set $\mathbf{q}^i = \mathbf{u}^i \oplus G(\mathbf{k}_i^1)$.
7. The parties proceed with the execution as in Protocol 1 in §3.2 in D13.1.

The case of corrupted sender. The simulator is exactly the same as in Theorem 5.2 in [3], i.e., the simulator chooses a random $\mathbf{r}'$ and plays the role of an honest receiver with input $\mathbf{r}'$. There is no contradiction between the simulated execution (where the input of the receiver is $\mathbf{r}'$) and the actual value $\mathbf{r}$ chosen by the trusted party, for the same reasons that the simulator in Theorem 5.2 in [3] succeeds with the simulation for some random input $\mathbf{r}'$ whereas the receiver uses its true input $\mathbf{r}$ to the trusted party.

The case of corrupted receiver. The only difference is that the simulator sends the messages $\mathbf{u}^2,\ldots,\mathbf{u}^\ell$ (excluding $\mathbf{u}^1$). In particular, the input $\mathbf{r}$ that the simulator extracts is the most repeated $\mathbf{r}^i$ value according to the messages $\mathbf{u}^2,\ldots,\mathbf{u}^\ell$, and define $\mathbf{r}^1$ as $G(\mathbf{k}_i^0) \oplus G(\mathbf{k}_i^1)$. The theorem follows from the correctness argument as above, and Theorem 5.2 in [3]. ■

3.3.4 Random OT (R-OT)

In a random OT, both P_S and P_R obtain their input as random output of the protocol. The random OT functionality can be obtained by combining the SR-OT protocol with the RR-OT protocol. Random OT can be used in the GMW protocol when pre-computing random multiplication triples [2]. We proceed with a formal description of the functionality (Functionality 4), the protocol (Protocol 5) and its proof of security (Theorem 3.3.4).

FUNCTIONALITY 4 (Functionality Random OT R-OT)

- **Inputs:** P_S has no input, and the functionality chooses m pairs of n-bit strings $(x_1^0,x_1^1),\ldots,(x_m^0,x_m^1)$.
- **In case of corrupted receiver:** P_R sends m-bits $\mathbf{r} = (r_1,\ldots,r_m)$ to the functionality.
- **In case of honest receiver:** P_R has no input. The functionality chooses m random bits $\mathbf{r} = (r_1,\ldots,r_m)$.
- **Output:** P_S outputs $(x_1^0,x_1^1),\ldots,(x_m^0,x_m^1)$. P_R outputs $(x_1^{r_1},\ldots,x_m^{r_m})$ and $\mathbf{r}$.

PROTOCOL 5 (Implementing Random-OT R-OT)
This is a simple combination of Protocols 3 and 4. Specifically, P_R defines its input as $G(\mathbf{k}_1^0) \oplus G(\mathbf{k}_1^1)$, and P_S defines its inputs x_j^0, x_j^1 according to $H(j, \mathbf{q}_j), H(j, \mathbf{q}_j \oplus \mathbf{s})$, respectively, for every $1 \leq j \leq m$. There is no transmission of $\mathbf{u}^1$ from P_R to P_S, and there is no transmission of y_j^0, y_j^1 from P_S to P_R for every $1 \leq j \leq m$.

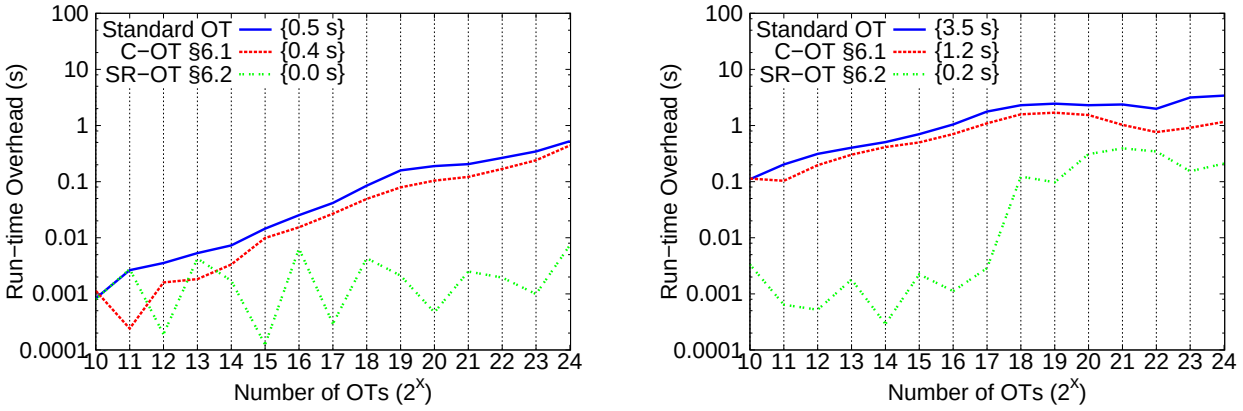
Theorem 3.3.4 *Assuming that H is a programmable random oracle, G is a pseudorandom generator, Protocol 5 securely computes the R-OT functionality (Functionality 4) in the $\ell \times OT_\kappa$-hybrid model in the presence of a static malicious adversary.*

Proof Sketch: The proof follows from Theorems 3.3.3 and 3.3.2. In particular, in case of corrupted sender the simulator receives the inputs $(x_1^0, x_1^1), \ldots, (x_m^0, x_m^1)$ from the trusted party, and it *programs* the random oracle H such that for every $1 \leq j \leq m$, $H(j, \mathbf{q}_j) = x_j^0$ and $H(j, \mathbf{q}_j \oplus \mathbf{s}) = x_j^1$. In case of a corrupted receiver, the input $\mathbf{r}$ that the simulator extracts and sends to the trusted party is the most repeated $\mathbf{r}^i$ value according to the messages $\mathbf{u}^2, \ldots, \mathbf{u}^\ell$ (where $\mathbf{r}^1$ is defined as $G(\mathbf{k}_1^0) \oplus G(\mathbf{k}_1^1)$). ■

Summary The original OT extension protocol of [31] and our proposed improvements for m OTs on n-bit strings are summarized in Tab. 3.1. We compare the communication complexity of P_R and P_S for m parallel 1-out-of-2 OT extensions of n-bit strings, with security parameter κ and ℓ base-OTs (we omit the cost of the initial κ base-OTs on κ-bit strings). We also compare the assumption on the function H needed in each protocol, where CR denotes Correlation Robustness and RO denotes Random Oracle.

3.3.5 Evaluation of Special Purpose OT Functionalities

We evaluate the performance of the special purpose OT functionalities, outlined in the previous section. We use the performance of the Random OT (R-OT) (cf. Section 3.3.4) as base-line and evaluate the overhead that is added when using the the original OT, Correlated OT (C-OT) (cf. Section 3.3.1), and Sender Random OT (SR-OT) (cf. Section 3.3.2) functionalities. We vary the number of OTs from 2^{10} (=1,024) to 2^{24} (=16,777,216) and fix the bit-length of the transferred strings to 128. The results for a LAN (two Desktop PCs with an Intel Haswell i7-4770K CPU, connected via Gigabit LAN) and a WAN (two Amazon EC2 m3.xlarge instances, connected via a 120MBit/s bandwidth and 100 ms ping latency link) scenario are given in Figure 3.2.
From the results we can observe that the standard OT functionality and the C-OT functionality are both slower than the R-OT functionality. The SR-OT, on the other hand, has a similar performance as the R-OT since R-OT only reduces the communication by a single bit per OT. In the LAN setting, the performance difference is nearly negligible (2^{24} R-OTs need 13.1 s while the same number of OTs require 13.6 s), since the improvements from R-OT mainly affect the communication complexity which is not the bottleneck in the LAN setting. In the WAN setting, however, the performance improvements of (S)R-OT are higher, since the communication is the bottleneck and the C-OT and standard OT functionality have to send messages from the sender to the receiver. Evaluating 2^{24} OTs in the WAN setting requires 23.0 s for the standard OT functionality, 20.7 s for the C-OT functionality, 19.7 s for SR-OT, and 19.5 s for R-OT.



(a) LAN Setting

(b) WAN Setting

Figure 3.2: Run-time overhead over R-OT for different OT flavors using the semi-honest OT extension on 128-bit strings in the LAN (a)- and WAN (b) setting.

3.4 Token-Aided Mobile GMW (short description)

This section is based on [18]. The complete description of this work appears in deliverable D13.1.

In the two-party case, the Goldreich-Micali-Wigderson (GMW) protocol [27] enables two mutually-distrusting parties $\mathcal{A}$ and $\mathcal{B}$ to securely evaluate a function which is expressed as a Boolean circuit. The GMW protocol relies heavily on OT and requires two OTs to compute a *multiplication triple* [5] that is used to evaluate an AND gate. Using OT pre-computation [5], the computation and communication intensive operations can be pre-computed in an interactive *setup phase* that is independent of the function. We describe a hardware token-aided GMW-based protocol for mobile phones [18]. Our goal is to minimize the *ad-hoc time* of the protocol, i.e., the time from establishing the communication channel between party $\mathcal{A}$ and party $\mathcal{B}$ until receiving the results of the secure computation.

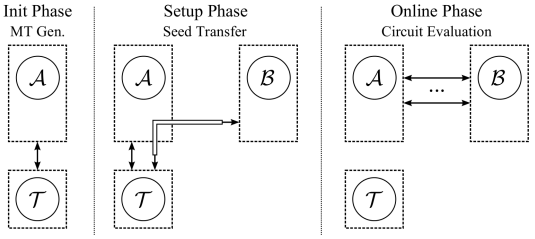


Figure 3.3: The three phases, workload distribution, and communication in our token-aided scheme.

An overview of our protocol is given in Figure 3.3. The general idea is to let the hardware token $\mathcal{T}$, held by $\mathcal{A}$, generate multiplication triples from two (or more) seeds in the init phase that are independent of the later communication partner. In the setup phase, $\mathcal{T}$ then sends one seed to $\mathcal{A}$ and the other seed over an encrypted channel to $\mathcal{B}$. The token thereby replaces the OT protocol in the setup phase and allows pre-computing the multiplication triples independently of the communication partner. The online phase of the GMW protocol remains unchanged.

3.5 Two-Party Unsigned Arithmetic Based on Additive Secret Sharing (short description)

The complete description of this work appears in deliverable D13.1.
This work proposes a protocol stack for secure unsigned arithmetic computation for two parties. Although in theory addition and multiplication suffice for all computation, it is often more efficient to use specialized protocols also for other operations. We describe common arithmetic protocols like addition, multiplication, integer division, exponentiation and comparisons. In addition, we include bit level operations like shifts and rotations. Some operations have protocols for different flavours where some inputs can be public instead of private to get more efficiency. For example, division has two versions with either public (PubDiv) or private divisor (PrivDiv). We focus on computations in rings Z_{2^k} for some integer $k > 0$ which correspond to the k-bit unsigned integer data types. An additive secret sharing of element x is denoted as $[\![x]\!]$. Sharing $[\![x]\!]$ is made up of two shares $[\![x]\!]_1$ and $[\![x]\!]_2$ where $x = [\![x]\!]_1 + [\![x]\!]_2$. These shares are distributed so that party $\mathcal{CP}_i$ for $i \in \{1,2\}$ has share $[\![x]\!]_i$ and no knowledge about the other share. Usually, the ring where the sharing is computed is understandable from the context, but it is also possible to denote it as $[\![x]\!]_{\bmod 2^k}$ for $x \in \mathbb{Z}_{2^k}$ where $[\![x]\!]_1, [\![x]\!]_2 \in \mathbb{Z}_{2^k}$. Furthermore, it is possible to access single bits of individual shares or extract them from the shared elements. For that, $x[k]$ denotes the k'th bit of the value x, where $k = 1$ means the least significant bit. In case of the share of the first party, the k'th bit would be denoted as $[\![x]\!]_1[k]$. However, in case the shared bit is extracted from the shared value the result is denoted as $[\![x[k]]\!]_{\bmod 2}$.
We assume that at most one of the parties is passively corrupted and ensure the security of the computations in this case. Current protocols rely on the security proof framework of passively secure protocols from [10]. A significant step when using this framework is to determine secure protocols that are a suitable finishing step of the composition. Two straightforward candidates for this role are resharing and declassify. Declassify is more meaningful in the two-party setting as it is a natural finishing step of any protocol. Clearly, two-party declassify is a secure protocol as knowing $[\![x]\!]_1$ and x the second share is uniquely fixed as $x - [\![x]\!]_1$ and can be perfectly simulated. The rest of the protocols in the computation are allowed to be input private.

3.6 Zero-Knowledge from Garbled Circuits – and GC for ZK (short description)

This section is based on [22]. The complete description of this work appears in deliverable D13.1.
Zero-knowledge protocols are one of the fundamental concepts in modern cryptography and have countless applications. However, after more than 30 years from their introduction, there are only very few languages (essentially those with a group structure) for which we can construct zero-knowledge protocols that are efficient enough to be used in practice. This is problematic, since zero-knowledge protocols (ZK) are one of the main building blocks in constructing MPC protocols which are secure against malicious corruptions.
The work described here was done in AU and proposes a solution to this problem. It is based on a slightly earlier work of AU and SAP [34] that presented a protocol based on Yao's garbled circuit technique that supports efficient zero-knowledge proofs for generic languages (e.g., to prove statements of the form "I know x s.t. $y = \text{SHA-256}(x)$" for a common input y). The new work in [22] shows that garbled circuits can be optimized for this specific application.
In the last few years garbled circuits have been elevated from being merely a component in

Yao's protocol for secure two-party computation, to a cryptographic primitive in its own right, following the growing number of applications that use GCs, including the zero-knowledge example described above. Our current work shows that due to the property of this particular application (i.e., one of the parties knows all the secret input bits, and therefore all intermediate values in the computation), we can construct more efficient garbling schemes specifically tailored to this goal.
As a highlight of the results in [22], in one of the constructions only *one ciphertext* per gate needs to be communicated and XOR gates never require any cryptographic operations. In addition to making a step forward towards more practical ZK, we believe that this contribution is also interesting from a conceptual point of view: in the terminology of Bellare *et al.* [9] these garbling schemes achieve authenticity, but no privacy nor obliviousness, therefore representing the first *natural* separation between those notions.

3.7 ZKBoo – Practically Efficient Zero-Knowledge Arguments

We describe ZKBoo [25], a proposal for practically efficient zero-knowledge arguments especially tailored for Boolean circuits. As an highlight, with ZKBoo it is possible to generate (resp. verify) a non-interactive proof for the SHA-1 circuit in approximately 13ms (resp. 5ms), with a proof size of 444KB.
ZKBoo is based on the "MPC-in-the-head" approach to zero-knowledge of Ishai et al. [33], which has been successfully used to achieve significant asymptotic improvements.
In IKOS, a prover simulates an MPC protocol between a number of "virtual" servers (at least 3) and then commits to the views and internal state of the individual servers. Now the verifier challenges the prover by asking to open a subset of these commitments. The privacy guarantee of the underlying MPC protocol guarantees that observing the state of a (sufficiently small) subset of servers does not reveal any information. At the same time, the correctness of the MPC protocol guarantees that if the prover tries to prove a false statement, then the joint views of some of the server must necessarily be inconsistent, and the verifier can efficiently check that. By plugging different MPC protocols into this approach, [33] shows how to construct ZK protocols with good asymptotic properties. However, prior to our work, no one had investigated whether the IKOS approach can be used to construct practically efficient ZK protocols.

3.7.1 (2,3)-Function Decomposition

Given an arbitrary function $\phi : X \to Y$ and an input value $\mathbf{x} \in X$ we want to compute the value $\phi(\mathbf{x})$ splitting the computation in 3 branches such that the values computed in 2 branches reveals no information about the input $\mathbf{x}$. In order to achieve this, we start by splitting the value $\mathbf{x}$ in three values $\mathbf{x}_1$,$\mathbf{x}_2$,$\mathbf{x}_3$ (called *input shares*) using a surjective function that we indicate with Share. These input shares as well as all the intermediate values are stored in 3 string $\mathbf{w}_1$,$\mathbf{w}_2$,$\mathbf{w}_3$ called the *views*. More precisely, $\mathbf{w}_i$ contains the values computed in the computation branch i. In order to achieve the goal and compute the value $\mathbf{y} = \phi(\mathbf{x})$, we use a finite family of efficiently computable functions that we indicate with $\mathcal{F} = \bigcup_{j=1}^{N} \{\phi_1^{(j)},\phi_2^{(j)},\phi_3^{(j)}\}$. The function $\phi_m^{(j)}$ takes as inputs specific values from the views $\mathbf{w}_m$,$\mathbf{w}_{m+1}$ with $m = \{1,2,3\}$ and where $3+1=1$. The functions are used in the following way: we use functions $\phi_1^{(j)},\phi_2^{(j)},\phi_3^{(j)}$ to compute the next value to be stored in each view $\mathbf{w}_m$: The function $\phi_m^{(1)}$ takes as input $\mathbf{w}_m$,$\mathbf{w}_{m+1}$ (which at this point contain only the shares $\mathbf{x}_m$,$\mathbf{x}_{m+1}$) and outputs one value which is saved in position 1 of

the views $\mathbf{w}_m$. We continue like this for all N functions, with the difference that in step $j > 1$, the function $\phi_m^{(j)}$ can receive as input (any subset of) the current views $\mathbf{w}_m$,$\mathbf{w}_{m+1}$. The initial function Share and all subfunctions $\phi_m^{(j)}$ are allowed to be randomized, and they get their coins from $\mathbf{k}_1$,$\mathbf{k}_2$,$\mathbf{k}_3$, three random tapes which correspond to the three branches. Finally, after the N steps described, the 3 functions $\mathsf{Output}_1, \mathsf{Output}_2, \mathsf{Output}_3$ are used to compute the values $\mathbf{y}_i = \mathsf{Output}_i(\mathbf{w}_i)$ that we call *output shares*. From these three values we compute the final output $\mathbf{y} = \phi(\mathbf{x})$ using the function Rec.

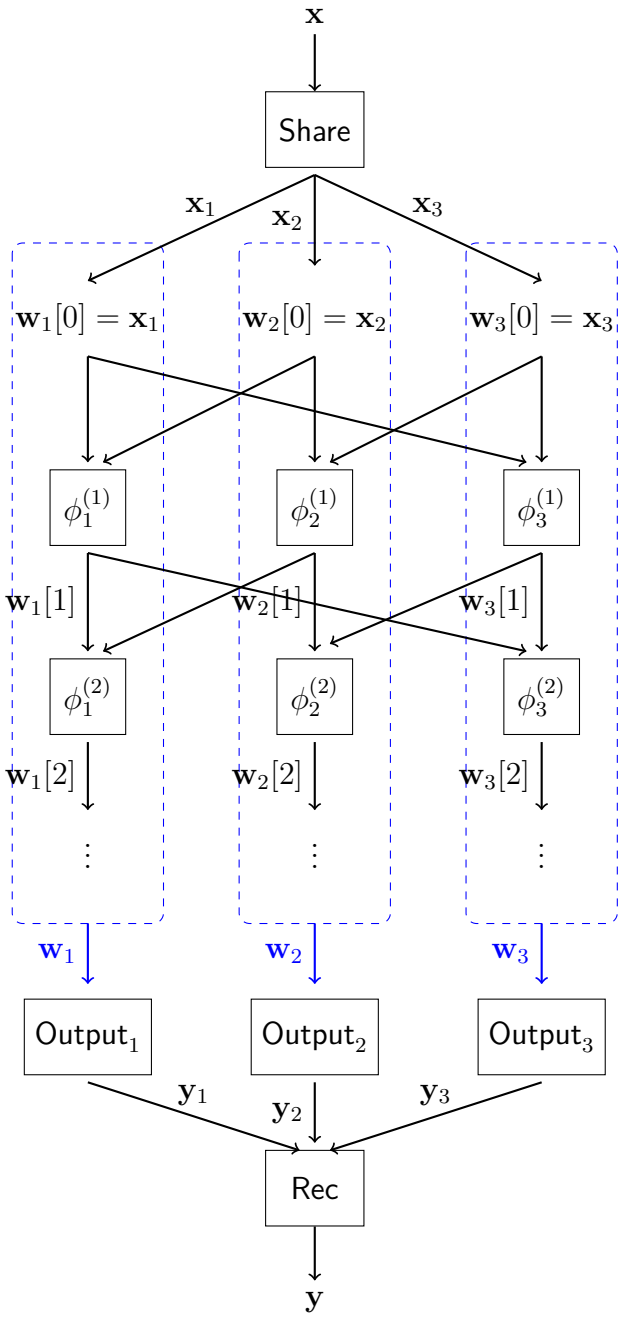


Figure 3.4: Pictorial representation of a (2,3)-decomposition of the computation $\mathbf{y} = \phi(\mathbf{x})$ showing the three branches.

A (2,3)-*decomposition* for the function ϕ must satisfy *correctness* (informally, the output final output is $\phi(x)$) and *privacy* (informally, any 2 threads in the decomposition can be simulated

without knowing the input).

The Linear Decomposition

We present here an explicit example of a convenient (2,3)-decomposition. Let $\mathbb{Z}$ be an arbitrary finite ring such that $\phi : \mathbb{Z}^k \to \mathbb{Z}^\ell$ can be expressed by an arithmetic circuit over the ring using addition by constant, multiplication by constant, binary addition and binary multiplication gates[1]. The total number of gates in the circuit is N, the gates are labelled with indices in $[N]$. The *linear (2,3)-decomposition* of ϕ is defined as follows:

- $\mathsf{Share}^{\mathbb{Z}}(\mathbf{x}; \mathbf{k}_1,\mathbf{k}_2,\mathbf{k}_3)$ samples random $\mathbf{x}_1,\mathbf{x}_2,\mathbf{x}_3$ such that $\mathbf{x} = \mathbf{x}_1 + \mathbf{x}_2 + \mathbf{x}_3$;
- The family $\mathcal{F}^{\mathbb{Z}} = \bigcup_{c=1}^{N} \{\phi_1^{(c)},\phi_2^{(c)},\phi_3^{(c)}\}$ is defined in the following way. Assume that the c-th gate has input wires coming from the gate number a and the gate number b (or only gate number a in the case of a unary gate), then the function $\phi_i^{(c)}$ is defined as follows: If the c-th gate is a $(\forall \alpha \in \mathbb{Z})$
  - unary "add α" gate, then $\forall\, i \in [3]$:

$$\mathbf{w}_i[c] = \phi_i^{(c)}(\mathbf{w}_i[a]) = \begin{cases} \mathbf{w}_i[a] + \alpha \text{ if } i = 1 \\ \mathbf{w}_i[a] \text{ else} \end{cases}$$

  - unary "mult. α" gate, then $\forall\, i \in [3]$:

$$\mathbf{w}_i[c] = \phi_i^{(c)}(\mathbf{w}_i[a]) = \alpha \cdot \mathbf{w}_i[a]$$

  - binary addition gate, then $\forall\, i \in [3]$:

$$\mathbf{w}_i[c] = \phi_i^{(c)}(\mathbf{w}_i[a], \mathbf{w}_i[b]) = (\mathbf{w}_i[a] + \mathbf{w}_i[b])$$

  - binary multiplication gate, then $\forall\, i \in [3]$:

$$\begin{aligned} \mathbf{w}_i[c] &= \phi_i^{(c)}\Big(\mathbf{w}_i[a,b], \mathbf{w}_{i+1}[a,b]\Big)\Big) \\ &= \mathbf{w}_i[a] \cdot \mathbf{w}_i[b] + \mathbf{w}_{i+1}[a] \cdot \mathbf{w}_i[b] \\ &+ \mathbf{w}_i[a] \cdot \mathbf{w}_{i+1}[b] + R_i(c) - R_{i+1}(c) \end{aligned}$$

    where $R_i(c)$ is a uniformly random function sampled using $\mathbf{k}_i$.
- For all $i \in [3]$, $\mathsf{Output}_i^{\mathbb{Z}}(\mathbf{w}_i,\mathbf{k}_i)$ simply selects all the shares of the output wires of the circuit;
- Finally, $\mathsf{Rec}^{\mathbb{Z}}(\mathbf{y}_1,\mathbf{y}_2,\mathbf{y}_3)$ outputs $\mathbf{y} = \mathbf{y}_1 + \mathbf{y}_2 + \mathbf{y}_3$

Given such a decomposition, ZKBoo can be instantiated as in Figure 3.5: If $\mathbf{y} \in L_\phi$ is the public input of the proof, then the prover P uses his private input $\mathbf{x}$ (with $\phi(\mathbf{x}) = \mathbf{y}$) to run "in his head" the protocol Π_ϕ^*. After the emulation of the protocol, P commits to each of the 3 produced views $\mathbf{w}_1,\mathbf{w}_2,\mathbf{w}_3$. Now the verifier challenges the prover to open 2 of the commitments. Finally, the verifier accepts if the opened views are consistent with the decomposition used and with output $\mathbf{y}$.

[1] Note that Boolean circuits are a special case of this, with the XOR, AND and NOT gate.

ZKBoo Protocol

The verifier and the prover have input $\mathbf{y} \in L_\phi$. The prover knows $\mathbf{x}$ such that $\mathbf{y} = \phi(\mathbf{x})$. A (2,3)-decomposition of ϕ is given. Let Π^*_ϕ be the protocol related to this decomposition.

Commit: The prover does the following:

1. Sample random tapes $\mathbf{k}_1$,$\mathbf{k}_2$,$\mathbf{k}_3$;
2. Run $\Pi^*_\phi(\mathbf{x})$ and obtain the views $\mathbf{w}_1$,$\mathbf{w}_2$,$\mathbf{w}_3$ and the output shares $\mathbf{y}_1$,$\mathbf{y}_2$,$\mathbf{y}_3$;
3. Commit to $\mathbf{c}_i = \mathsf{Com}(\mathbf{k}_i,\mathbf{w}_i)$ for all $i \in [3]$;
4. Send $\mathbf{a} = (\mathbf{y}_1,\mathbf{y}_2,\mathbf{y}_3,\mathbf{c}_1,\mathbf{c}_2,\mathbf{c}_3)$.

Prove: The verifier choose an index $\mathbf{e} \in [3]$ and sends it to the prover. The prover answers to the verifier's challenge sending opening $\mathbf{c}_e$,$\mathbf{c}_{e+1}$ thus revealing $\mathbf{z} = (\mathbf{k}_e,\mathbf{w}_e,\mathbf{k}_{e+1},\mathbf{w}_{e+1})$.

Verify: The verifier runs the following checks:

1. If $\mathsf{Rec}(\mathbf{y}_1,\mathbf{y}_2,\mathbf{y}_3) \neq \mathbf{y}$, output reject;
2. If $\exists,i \in \{e,e+1\}$ s.t. $\mathbf{y}_i \neq \mathsf{Output}_i(\mathbf{w}_i)$, output reject;
3. If $\exists j$ such that
$$\mathbf{w}_e[j] \neq \phi_e^{(j)}\left(\mathbf{w}_e,\mathbf{w}_{e+1},\mathbf{k}_e,\mathbf{k}_{e+1}\right)$$
output reject;
4. Output accept;

Figure 3.5: ZKBoo protocol for the language L in the commitment-hybrid model.

Chapter 4

Order-Preserving Encryption for Secure Database Qeuries

4.1 Optimal Average-Complexity Ideal-Security Order-Preserving Encryption (short description)

This section is based on [37]. The complete description of this work appears in deliverable D13.1.
Order-preserving encryption enables performing many classes of queries – including range queries – on encrypted databases. Popa et al. recently presented an ideal-secure order-preserving encryption (or encoding) scheme [64], but their cost of insertions (encryption) is very high. Kerschbaum et al. presented an also ideal-secure, but significantly more efficient order-preserving encryption scheme [37]. This scheme is inspired by Reed's referenced work on the average height of random binary search trees. The scheme improves the average communication complexity from $O(n \log n)$ to $O(n)$ under uniform distribution. It also integrates efficiently with adjustable encryption as used in CryptDB. In their experiments for database inserts Kerschbaum et al. achieve a performance increase of up to 81% in LANs and 95% in WANs.
Kerschbaum et al.'s order-preserving encryption algorithm builds a binary search tree as does Popa et al.'s. The tree is however not necessarily balanced and relies on the uniformity assumption about the input distribution. They only balance the tree when necessary, i.e., then an update operation is performed. This enables them to maintain the dictionary on the client and therefore achieve a significant performance gain and compatibility with adjustable onion encryption.

4.2 Frequency-Hiding Order-Preserving Encryption (short description)

This section is based on [35]. The complete description of this work appears in deliverable D13.1.
This work presents a scheme that achieves a strictly stronger notion of security than any other order-preserving encryption scheme so far. The basic idea is to randomize the ciphertexts to hide the frequency of plaintexts. Still, the client storage size remains small, up to 1/15 of the plaintext size. As a result, one can more securely outsource large data sets, since the new scheme also show that their security increases with larger data sets. They clearly increase security while preserving the functionality for most queries relying on the ordering information.

However, they also increase client storage size and introduce a small error in some queries. The new work presents a definition of a new, stronger security notion for order-preserving encryption than indistinguishability under chosen plaintext attack which they call *indistinguishability under frequency-analyzing ordered chosen plaintext attack*. It also presents a scheme implementing this notion including compression mechanisms.

Chapter 5

Protocols for Private Set Intersection

Private set intersection (PSI) allows two parties $\mathcal{A}$ and $\mathcal{B}$ with respective input sets X and Y to compute the intersection $X \cap Y$ of their sets without revealing any information but the intersection itself. Although PSI has been widely studied in the literature, many real-world applications today use an insecure hash-based protocol instead of a secure PSI protocol, mainly because of the insufficient efficiency of current PSI protocols.
In a sequence of works we presented improved PSI protocols that were more efficient by the state of the art by at least an order of magnitude. The most advanced family of protocols that we presented, denoted *Phasing* for Permutation-based Hashing Set Intersection, is a new approach for constructing PSI protocols based on a hashing technique that ensures that hashed elements can be represented by short strings without any collisions. The overhead of recent PSI protocols depends on the length of these representations, and this new structure of construction, together with other improvements, results in very efficient performance that is only moderately larger than that of the *insecure* protocol that is in current real-world usage.

5.1 PSI Protocols based on Oblivious Transfer (short description)

This section is based on [58]. The complete description of this work appears in deliverable D13.1.
The goal of this work was to enable PSI computations for large scale sets that were previously beyond the capabilities of state-of-the-art protocols. The constructions that were designed improve performance by more than an order of magnitude. These improvements were obtained by generalizing the hashing approach of [61] and applying it to generic secure computation-based PSI protocols. The hash function in [61] were replaced by a permutation which enables to reduce the bit-length of internal representations. Moreover, several improvements to the OT-based PSI protocol of [61] were presented. The contributions are next explained in more detail:

5.2 A New Set of Protocols for PSI

Private set intersection (PSI) allows two parties $\mathcal{A}$ and $\mathcal{B}$ holding sets X and Y, respectively, of σ-bit elements to identify the intersection $X \cap Y$ without revealing any information about elements that are not in the intersection. In this section, we outline protocols that use an *oblivious*

pseudo-random function (OPRF) to perform PSI and that extend previous PSI protocols, outlined in Section 7 in D11.1 and Section 5 in D13.1. An OPRF [23] $F : \{0,1\}^\sigma \times \{0,1\}^\kappa \mapsto \{0,1\}^\ell$ is a function which, given a key k from $\mathcal{A}$ and an input element e from $\mathcal{B}$, computes and outputs $F_k(e)$ to $\mathcal{B}$. $\mathcal{A}$ obtains no output and learns no information about e while $\mathcal{B}$ learns no information about k. OPRFs can be used for PSI by first evaluating the OPRF protocol on the set of $\mathcal{B}$ and then having $\mathcal{A}$, who knows the secret key k, evaluate the OPRF locally on its own set, and send the OPRF output to $\mathcal{B}$, who computes a plaintext intersection.
We outline two protocols for OPRF-based PSI: the first protocol uses generic secure computation techniques (§5.2.1) and the second protocol uses oblivious transfer (OT, §5.2.2). Both protocols appear in [62].

5.2.1 Secure Computation-based OPRF Evaluation

A generic secure computation-based protocol performing PSI was outlined in [23, 59] and uses an OPRF. In this protocol, the parties use secure computation techniques such as Yao's garbled circuits [69] or the GMW protocol [27] to evaluate a pseudo-random function $F_k(y) = z$. The use of secure computation guarantees the obliviousness, i.e., that $\mathcal{A}$ learns no information about y or z while $\mathcal{B}$ learns no information about k. The PSI functionality can then be achieved by evaluating the OPRF on each element in the set of $\mathcal{B}$ and having $\mathcal{A}$ locally evaluate and send $F_k(x_i)$ for all elements $x_i \in X$. $\mathcal{B}$ can then identify the intersection by computing the plaintext intersection between his output of the OPRF with the output sent by $\mathcal{A}$.

Efficiency The efficiency of the circuit-based OPRF construction depends mainly on the instantiation of the pseudo-random function F. While it is possible to instantiate F with a cipher that is optimized for use in secure computation such as [1], we consider an AES-based instantiation in our efficiency analysis, since the security of AES is well researched. The number of AND gates in the AES circuit is 5,440 and its multiplicative depth is 40 [11]. In total, we have to perform n_2 parallel oblivious AES evaluations (wher n_2 is the number of items in the set Y), resulting in $5{,}440n_2$ total AND gates and a depth of 40. $\mathcal{A}$, on the other hand, can perform a plaintext AES evaluation on his elements and only needs to send n_1 collision-resistant strings of $\ell = \sigma + \log(n_1) + \log(n_2)$ bit length. Hence, due to the large constants, the OPRF-based approach is less efficient in concrete terms than existing PSI circuits that were outlined in Section 7.3 in D11.1 and Section 5.1 in D13.1, even though it scales with $O(n)$ while both other circuits scale with $O(n \log n)$. However, we show in [62] that if the set sizes of the parties greatly differ, i.e., when $n_2 \ll n_1$, the OPRF-based approach can be more efficient than other circuit constructions and in fact more efficient than even all other PSI protocols, since the elements in the much larger set of $\mathcal{A}$ can be processed at very low cost.

5.2.2 OT-based OPRF Evaluation

In this section, we describe our new OT-based PSI protocol, of which an earlier version appeared in Section 7.4.3 in D11.1 and Section 5.1 in D13.1. In contrast to the earlier versions, we improve our protocol such that its complexity is now independent of the bit length σ for realistic set sizes. The core of our OT-based PSI protocol is an efficient OPRF instantiation using recent OT extension techniques, in particular the random OT functionality [2, 55] and the 1-out-of-N OT ($\binom{N}{1}$-OT) of [41]. Our protocol operates in three steps: the parties *hash* their elements into hash tables, mask their elements using the *OPRF,* and compute the *plaintext intersection* of these masks to identify the intersecting elements. In the hashing step we use the methods

outlined in [62] and the plaintext intersection is straight-forward and does not affect security. Hence, we only describe the OPRF construction in more detail in the following.

PROTOCOL 6 (Our OT-based PSI Protocol)

- **Input of $\mathcal{A}$:** $X = \{x_1,...,x_{n_1}\}$.
- **Input of $\mathcal{B}$:** $Y = \{y_1,...,y_{n_2}\}$.
- **Common Input:** Bit-length of elements σ; number of bins $b = \epsilon n_2$ (cf. [62] for details on these parameters); k random hash functions $\{h_1,...,h_k\} : \{0,1\}^\sigma \mapsto [1...b]$; reduced bit-length of items in the hash table $\mu = \sigma - \log_2 b + \log_2 k$; symmetric security parameter κ; statistical security parameter σ; mask-length $\ell = \sigma + \log_2(kn_1) + \log_2(n_2)$; $N = 2^\mu$; dummy element d_2; stash size s.
- **Oracles and cryptographic primitives:** Both parties have access to a functionality that computes one 1-out-of-N random OT on ℓ-bit strings ($\binom{N}{1}$-ROT$^1_\ell$).

1. **Hashing:**
   (a) $\mathcal{A}$ maps the elements in its set X into a two-dimensional hash table $T_1[\,][\,]$ using simple hashing and k hash functions $\{h_1, ..., h_k\}$. The first dimension has size b and addresses the bin in the table while the second dimension addresses the elements in the bins.
   (b) $\mathcal{B}$ maps the elements in its set Y into a one-dimensional hash table $T_2[\,]$ and stash $S[\,]$ using Cuckoo hashing and k hash functions $\{h_1, ..., h_k\}$. The hash table has size b and the stash has size s. $\mathcal{B}$ then fills all empty entries in T_2 and S with d_2.

   Let $|T_1[i]|$ be the number of elements that are stored in the i-th bin of the hash table T_1 and μ be the bit-length of these elements for $1 \leq i \leq b$.

2. **OPRF evaluation (via OT):**
   For each bin $1 \leq i \leq b$, the parties perform the following steps:
   (a) Let $v_j = T_1[i][j]$ and $w = T_2[i]$ for $1 \leq j \leq |T_1[i]|$.
   (b) The parties evaluate an OPRF using the $\binom{N}{1}$-ROT$^1_\ell$ functionality, where $\mathcal{A}$ has no inputs and obtains a random N-entry look-up table L and $\mathcal{B}$ inputs w as choice bits and obtains a random mask $L[w]$.
   (c) $\mathcal{A}$ computes $M_1[i][j] = L[v_j]$ and $\mathcal{B}$ computes $M_2[i] = L[w]$.

   Stash: For each element in the stash S, the parties repeat the same steps where, for the i-th stash position, $\mathcal{A}$ evaluates the OPRF on his whole input set X and obtains n_1 masks $M_{S_1}[i]$ while $\mathcal{B}$ evaluates the OPRF on $S[i]$ and obtains one masks $M_{S_2}[i]$.

3. **Plaintext Intersection**
   (a) Let $V = \bigcup_{1\leq i\leq b, 1\leq j\leq |T_1[i]|} M_1[i][j]$. $\mathcal{A}$ randomly permutes V and sends it to $\mathcal{B}$.
   (b) $\mathcal{B}$ computes the intersection $Z = \{T_2[i] | M_2[i] \in V\}$.

   Stash: The parties perform the same operation to identify whether an element on the stash is in the intersection: $\mathcal{A}$ permutes and sends $M_{S_1}[i]$ to $\mathcal{B}$, who adds $S[i]$ to the intersection Z if $M_{S_2}[i] \in M_{S_1}[i]$.

- **Output:** $\mathcal{A}$ has no output; $\mathcal{B}$ outputs $Z = X \cap Y$.

In the first step of our OT-based PSI protocol, the parties have mapped their elements into hash tables T_1 and T_2 where the elements in the tables have bit-length $\mu = \sigma - \log_2 b + \log_2 k$ due to permutation-based mapping (cf. D13.1 Section 5.1 and [62] for more information on permutation-based hashing). $\mathcal{A}$ has used simple hashing and hence its hash table T_1 has two

dimensions, where the first dimension addresses the bins and the second dimension addresses the elements in the bins. $\mathcal{B}$ has used Cuckoo hashing and hence its hash table T_2 has only one dimension, which addresses the bins. Our OT-based PSI protocol then evaluates an OPRF F where, for each bin, $\mathcal{A}$ samples a random key and $\mathcal{B}$ inputs the μ-bit element in bin $T_2[i]$ and obtains the resulting mask $M_2[i] = F_{k_i}(T_2[i])$, for $1 \leq i \leq b$. The OPRF must ensure that $\mathcal{A}$ learns no information on the input of $\mathcal{B}$ and that $\mathcal{B}$ learns no information except the outputs that correspond to its elements.
The main observation is that we can instantiate an OPRF for μ-bit inputs using one random 1-out-of-2^μ random OT on ℓ-bit strings ($\binom{2^\mu}{1}$-ROT$^1_\ell$), where $\mathcal{A}$ plays the sender and obtains a 2^μ-dimensional lookup-table $L : \{0,1\}^\mu \mapsto \{0,1\}^\ell$ while $\mathcal{B}$ plays the receiver who inputs $T_2[i]$ and obtains $L[T_2[i]]$. $\mathcal{A}$ can then evaluate the OPRF on the elements in its bin $T_1[i]$ locally by computing $M_1[i][j] = L[T[i][j]]$, for $1 \leq i \leq b$ and $1 \leq j \leq |T_1[i]|$. After $\mathcal{A}$ has evaluated the OPRF for all bins i, it collects the OPRF outputs $M_1[i]$ for all $|T_1[i]|$ elements in a bin to a set V and permutes and sends V. $\mathcal{B}$ identifies whether $T_2[i]$ is in the intersection by checking whether $M_2[i]$ matches any element in V. If the element $T_2[i]$ matches any element in $T_1[i]$, their OPRF outputs will be equal. If $T_2[i]$ matches no element in $T_1[i]$, their OPRF outputs will differ except with probability $|T_1[i]| \cdot 2^{-\ell}$. The elements in the stash of $\mathcal{B}$ are processed independently in a similar fashion: both parties evaluate the OPRF, $\mathcal{B}$ obtains the output for the elements in its stash, and $\mathcal{A}$ evaluates the OPRF locally on each element of its set and sends the permuted outputs to $\mathcal{B}$, who identifies the intersection.

Efficiency The main computation and communication overhead comes from the OPRF evaluation. The efficiency of the OPRF depends greatly on the underlying instantiation. We instantiate the OPRF that maps μ-bit inputs to ℓ-bit outputs using the $\binom{2^\mu}{1}$-ROT$^1_\ell$ protocol of [41] with the linear BCH code [2^{77}, 512, 129], generated by [53], which encodes up to 2^{77} words to codewords of length 512 with relative Hamming distance κ, which is denoted as a [2^{77}, 512, 129] code.
Overall, the parties evaluate the OPRF $s + b$ times, corresponding to $\binom{2^\mu}{1}$-ROT$^{s+b}_\ell$, where the stash size s and the number of bins $b = \epsilon n_2$ are chosen to achieve negligible Cuckoo hashing error probability (cf. [62] for possible choices of these parameters). Regarding the communication, $\mathcal{B}$ sends $512(s+b)$ bits for the $\binom{2^\mu}{1}$-ROT, while $\mathcal{A}$ sends $k\ell n_1$ bits for the permuted OPRF output, where k is the number of hash functions used for Cuckoo hashing and $\ell = \log_2(kn_1)+\log_2(n_2)+\sigma$. Regarding the computation, note that in a naive $\binom{2^\mu}{1}$-OT evaluation the sender, $\mathcal{A}$, would need to perform 2^μ correlation-robust function evaluations, one for each message. However, since $\mathcal{A}$ only needs to obtain the output for actual elements in its bins, it only needs to perform $(k + s)n_1$ correlation-robust function evaluations, which is independent of μ.

Correctness In the following, we analyze the correctness of the scheme. We assume that in Step 1 in Protocol 6, $\mathcal{A}$ has used simple hashing to map each element k times into the hash table T_1 while $\mathcal{B}$ has used Cuckoo hashing to map each element once into the hash table T_2.
If $x = y$ then $\mathcal{A}$ and $\mathcal{B}$ will have the same item in a bin in their hash tables ($\mathcal{B}$ has mapped the item to one of k bins while $\mathcal{A}$ has mapped the item to all k bins). For this bin, $\mathcal{B}$ obtains $M_x = L[x]$ as output of the OPRF and $\mathcal{A}$ can locally compute $M_y = L[y]$ with $M_x = M_y$, and $\mathcal{B}$ successfully identifies equality.
If $x \neq y$ then the probability that $M_x = M_y$ is $2^{-\ell}$. However, we require that *all* OPRF outputs M_2 for elements in the hash table T_2 of $\mathcal{B}$ are distinct from *all* outputs M_1 for elements in the hash table T_1 of $\mathcal{A}$, which happens with probability $kn_1n_2 2^{-\ell}$. Thus, to achieve correctness with

probability 1-$2^{-\sigma}$, we must increase the bit-length of the OTs to $\ell = \sigma + \log_2(kn_1) + \log_2(n_2)$.

Security $\mathcal{B}$'s security is obvious, since the only information that $\mathcal{A}$ learns are the random values chosen in the random OT, which are independent of $\mathcal{B}$'s input.
As for $\mathcal{A}$'s security, note that $\mathcal{B}$'s view in the protocol consists of its outputs M_2 of the $\binom{N}{1}$-ROT protocols, and of the values M_1 sent by $\mathcal{A}$. If there are two elements $x \in X$ and $y \in Y$ with $x = y$, then there are outputs $M_x = M_y$. Otherwise, for $x \neq y$, these values are uniformly distributed and $\mathcal{B}$ can gain no information about M_x, which is guaranteed by the properties of the $\binom{N}{1}$-ROT protocol. In both cases, the view of $\mathcal{B}$ can be easily simulated given the output of the protocol (i.e., knowledge whether $x = y$). The protocol is therefore secure according to the common security definitions of secure computation [26].

Chapter 6

Conclusion

The report described new protocols for secure multi-party computation. The main goal of the work that was performed was to address the needs of applications, by improving the performance of protocols which fit the requirements of the application scenarios that were described in Deliverable D11.2 of this project. The protocols that were presented in this report have been published in multiple research papers at top-tier conferences.
The results that was presented in this deliverable can be categorized into several categories:

- Chapter 2 described improved protocols for generic secure multi-party computation, namely protocols that can be used for securely computing any function.
- Chapter 3 described improved building blocks for constructing secure protocols. Namely better constructions of oblivious transfer, token-aided computation, secure arithmetic operations, and zero-knowledge proofs, which are tools that are used by secure computation protocols.
- Chapter 4 described new protocols for the specific problem of search on encrypted data. In that setting the input (an encrypted database) is of a huge size, and therefore generic protocols are not sufficiently efficient.
- Chapter 5 described improved protocols for the specific problem of computing the intersection of two private sets. This is a problem of high interest and therefore we designed protocols that are tailored for solving this problem and are significantly more efficient that applying generic protocols to this problems.

Chapter 7

List of Abbreviations

2PC	Two party computation
ABY	Arithmetic-Boolean-Yao
AES	Advanced encryption standard
BMR	the Beaver-Micali-Rogaway protocol
DH	Diffie-Hellman
EC	European Commission
ECC	Elliptic curve cryptography
FHE	Fully homomorphic encryption
GC	Garbled circuit
GMW	the Goldreich-Micali-Wigderson protocol
GRR	Garbled row reduction
IR	Ireland
MPC	Multi-party computation
OT	Oblivious transfer
SCS	Sort-compare-shuffle
SPDZ	the Damgard-Pastro-Smart-Zakarias protocol
PRF	Pseudo random function
PSI	Private set intersection
VA	Virginia
ZK	Zero knowledge

Bibliography

[1] M. R. Albrecht, C. Rechberger, T. Schneider, T. Tiessen, and M. Zohner. Ciphers for MPC and FHE. In *Advances in Cryptology – EUROCRYPT'15*, volume 9056 of *LNCS*, pages 430–454. Springer, 2015.

[2] G. Asharov, Y. Lindell, T. Schneider, and M. Zohner. More efficient oblivious transfer and extensions for faster secure computation. In *ACM Computer and Communications Security (CCS'13)*, pages 535–548. ACM, 2013. Code: http://encrypto.de/code/OTExtension.

[3] G. Asharov, Y. Lindell, T. Schneider, and M. Zohner. More efficient oblivious transfer extensions. *Journal of Cryptology*, 2016. Accepted for publication.

[4] Gilad Asharov, Yehuda Lindell, Thomas Schneider, and Michael Zohner. More efficient oblivious transfer extensions with security for malicious adversaries. In *EUROCRYPT (1)*, volume 9056 of *Lecture Notes in Computer Science*, pages 673–701. Springer, 2015.

[5] D. Beaver. Efficient multiparty protocols using circuit randomization. In *Advances in Cryptology – CRYPTO'91*, volume 576 of *LNCS*, pages 420–432. Springer, 1991.

[6] D. Beaver. Correlated pseudorandomness and the complexity of private computations. In *Symposium on Theory of Computing (STOC'96)*, pages 479–488. ACM, 1996.

[7] Donald Beaver, Silvio Micali, and Phillip Rogaway. The round complexity of secure protocols. In Harriet Ortiz, editor, *22nd STOC*, pages 503–513. ACM, 1990.

[8] M. Bellare, V. Hoang, S. Keelveedhi, and P. Rogaway. Efficient garbling from a fixed-key blockcipher. In *Symposium on Security and Privacy (S&P'13)*, pages 478–492. IEEE, 2013.

[9] Mihir Bellare, Viet Tung Hoang, and Phillip Rogaway. Foundations of garbled circuits. In *ACM Conference on Computer and Communications Security*, pages 784–796, 2012. Full version at http://eprint.iacr.org/2012/265.

[10] Dan Bogdanov, Peeter Laud, Sven Laur, and Pille Pullonen. From input private to universally composable secure multi-party computation primitives. In *IEEE 27th Computer Security Foundations Symposium, CSF 2014, Vienna, Austria, 19-22 July, 2014*, pages 184–198, 2014.

[11] J. Boyar and R. Peralta. A small depth-16 circuit for the AES S-Box. In *Information Security and Privacy Conference (SEC'12)*, volume 376 of *IFIP Advances in Information and Communication Technology*, pages 287–298. Springer, 2012.

[12] J. Bringer, H. Chabanne, and A. Patey. SHADE: secure hamming distance computation from oblivious transfer. In *Financial Cryptography and Data Security (FC'13)*, volume 7862 of *LNCS*, pages 164–176. Springer, 2013.

[13] Seung Geol Choi, Jonathan Katz, Alex J. Malozemoff, and Vassilis Zikas. Efficient three-party computation from cut-and-choose. In Garay and Gennaro [24], pages 513–530.

[14] Tung Chou and Claudio Orlandi. The simplest protocol for oblivious transfer. In *Progress in Cryptology - LATINCRYPT 2015 - 4th International Conference on Cryptology and Information Security in Latin America, Guadalajara, Mexico, August 23-26, 2015, Proceedings*, pages 40–58, 2015.

[15] Ronald Cramer, Ivan Damgård, and Jesper Buus Nielsen. Multiparty computation from threshold homomorphic encryption. In *Advances in Cryptology – EUROCRYPT'01*, volume 2045 of *LNCS*, pages 280–299. Springer, 2001.

[16] Ivan Damgård, Jesper Buus Nielsen, Michael Nielsen, and Samuel Ranellucci. Gate-scrambling revisited - or: The tinytable protocol for 2-party secure computation. *IACR Cryptology ePrint Archive*, 2016:695, 2016.

[17] Ivan Damgård, Valerio Pastro, Nigel P. Smart, and Sarah Zakarias. Multiparty computation from somewhat homomorphic encryption. In Safavi-Naini and Canetti [65], pages 643–662.

[18] D. Demmler, T. Schneider, and M. Zohner. Ad-hoc secure two-party computation on mobile devices using hardware tokens. In *USENIX Security Symposium (USENIX Security'14)*, pages 893–908. USENIX, 2014.

[19] Daniel Demmler, Thomas Schneider, and Michael Zohner. ABY – a framework for efficient mixed-protocol secure two-party computation. In *Network and Distributed System Security (NDSS'15)*. The Internet Society, 2015. Code: `http://encrypto.de/code/ABY`.

[20] C. Dong, L. Chen, and Z. Wen. When private set intersection meets big data: An efficient and scalable protocol. In *ACM Computer and Communications Security (CCS'13)*, pages 789–800. ACM, 2013.

[21] S. Even, O. Goldreich, and A. Lempel. A randomized protocol for signing contracts. In *Communications of the ACM*, volume 28(6), pages 637–647. ACM, 1985.

[22] Tore Kasper Frederiksen, Jesper Buus Nielsen, and Claudio Orlandi. Privacy-free garbled circuits with applications to efficient zero-knowledge. *EUROCRYPT*, 2015:598, 2015.

[23] M. J. Freedman, Y. Ishai, B. Pinkas, and O. Reingold. Keyword search and oblivious pseudorandom functions. In *Theory of Cryptography Conference (TCC'05)*, volume 3378 of *LNCS*, pages 303–324. Springer, 2005.

[24] Juan A. Garay and Rosario Gennaro, editors. *Advances in Cryptology - CRYPTO 2014 - 34th Annual Cryptology Conference, Santa Barbara, CA, USA, August 17-21, 2014, Proceedings, Part II*, volume 8617 of *Lecture Notes in Computer Science*. Springer, 2014.

[25] Irene Giacomelli, Jesper Madsen, and Claudio Orlandi. Zkboo: Faster zero-knowledge for boolean circuits. In *25th USENIX Security Symposium, USENIX Security 16, Austin, TX, USA, August 10-12, 2016.*, pages 1069–1083, 2016.

[26] O. Goldreich. *Foundations of Cryptography*, volume 2: Basic Applications. Cambridge University Press, 2004.

[27] O. Goldreich, S. Micali, and A. Wigderson. How to play any mental game or a completeness theorem for protocols with honest majority. In *Symposium on Theory of Computing (STOC'87)*, pages 218–229. ACM, 1987.

[28] Oded Goldreich, Silvio Micali, and Avi Wigderson. How to play any mental game or a completeness theorem for protocols with honest majority. In *ACM Symposium on Theory of Computing (STOC'87)*, pages 218–229. ACM, 1987.

[29] Shay Gueron, Yehuda Lindell, Ariel Nof, and Benny Pinkas. Fast garbling of circuits under standard assumptions. In Indrajit Ray, Ninghui Li, and Christopher Kruegel, editors, *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security, Denver, CO, USA, October 12-6, 2015*, pages 567–578. ACM, 2015.

[30] R. Impagliazzo and S. Rudich. Limits on the provable consequences of one-way permutations. In *ACM Symposium on Theory of Computing (STOC'89)*, pages 44–61. ACM, 1989.

[31] Y. Ishai, J. Kilian, K. Nissim, and E. Petrank. Extending oblivious transfers efficiently. In *Advances in Cryptology – CRYPTO'03*, volume 2729 of *LNCS*, pages 145–161. Springer, 2003.

[32] Yuval Ishai, Joe Kilian, Kobbi Nissim, and Erez Petrank. Extending oblivious transfers efficiently. In *Advances in Cryptology – CRYPTO'03*, volume 2729 of *LNCS*, pages 145–161. Springer, 2003.

[33] Yuval Ishai, Eyal Kushilevitz, Rafail Ostrovsky, and Amit Sahai. Zero-knowledge from secure multiparty computation. In *Proceedings of the Thirty-ninth Annual ACM Symposium on Theory of Computing*, STOC '07, pages 21–30. ACM, 2007.

[34] Marek Jawurek, Florian Kerschbaum, and Claudio Orlandi. Zero-knowledge using garbled circuits: how to prove non-algebraic statements efficiently. In *ACM Conference on Computer and Communications Security*, pages 955–966, 2013.

[35] Florian Kerschbaum. Frequency-hiding order-preserving encryption. In *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security*. ACM, 2015.

[36] Florian Kerschbaum, Florian Hahn, Thomas Schneider, Michael Zohner, Pille Pullonen, and Claudio Orlandi. PRACTICE Deliverable D13.1: a set of new protocols, 2015. Available from http://www.practice-project.eu.

[37] Florian Kerschbaum and Axel Schröpfer. Optimal average-complexity ideal-security order-preserving encryption. In *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security*, pages 275–286. ACM, 2014.

[38] Joe Kilian. Founding cryptography on oblivious transfer. In Janos Simon, editor, *Proceedings of the 20th Annual ACM Symposium on Theory of Computing, May 2-4, 1988, Chicago, Illinois, USA*, pages 20–31. ACM, 1988.

[39] Ágnes Kiss and Thomas Schneider. Valiant's universal circuit is practical. In *Advances in Cryptology – EUROCRYPT'16*, LNCS. Springer, 2016.

[40] Ágnes Kiss and Thomas Schneider. Valiant's universal circuit is practical. Cryptology ePrint Archive, Report 2016/093, 2016. http://eprint.iacr.org/2016/093.

[41] V. Kolesnikov and R. Kumaresan. Improved OT extension for transferring short secrets. In *Advances in Cryptology – CRYPTO'13 (2)*, volume 8043 of *LNCS*, pages 54–70. Springer, 2013.

[42] Vladimir Kolesnikov, Payman Mohassel, and Mike Rosulek. Flexor: Flexible garbling for XOR gates that beats free-xor. In Garay and Gennaro [24], pages 440–457.

[43] Vladimir Kolesnikov and Thomas Schneider. Improved garbled circuit: Free XOR gates and applications. In *International Colloquium on Automata, Languages and Programming (ICALP'08)*, volume 5126 of *LNCS*, pages 486–498. Springer, 2008.

[44] Vladimir Kolesnikov and Thomas Schneider. A practical universal circuit construction and secure evaluation of private functions. In *Financial Cryptography and Data Security (FC'08)*, volume 5143 of *LNCS*, pages 83–97. Springer, 2008. Code: http://encrypto.de/code/FairplayPF.

[45] Yehuda Lindell. Fast cut-and-choose based protocols for malicious and covert adversaries. In Ran Canetti and Juan A. Garay, editors, *Advances in Cryptology - CRYPTO 2013*, volume 8043 of *LNCS*, pages 1–17. Springer, 2013.

[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] Yehuda Lindell, Benny Pinkas, Nigel P. Smart, and Avishay Yanai. Efficient constant round multi-party computation combining BMR and SPDZ. In Rosario Gennaro and Matthew Robshaw, editors, *Advances in Cryptology - CRYPTO 2015 - 35th Annual Cryptology Conference, Santa Barbara, CA, USA, August 16-20, 2015, Proceedings, Part II*, volume 9216 of *Lecture Notes in Computer Science*, pages 319–338. Springer, 2015.

[48] Yehuda Lindell and Ben Riva. Cut-and-choose yao-based secure computation in the online/offline and batch settings. In Garay and Gennaro [24], pages 476–494.

[49] Helger Lipmaa, Payman Mohassel, and Saeed Sadeghian. Valiant's universal circuit: Improvements, implementation, and applications. Cryptology ePrint Archive, Report 2016/017, 2016. http://ia.cr/2016/017.

[50] L. Lovász and M.D. Plummer. *Matching Theory*. AMS Chelsea Publishing Series. American Mathematical Soc., 2009.

[51] Dahlia Malkhi, Noam Nisan, Benny Pinkas, and Yaron Sella. Fairplay - secure two-party computation system. In *USENIX Security'04*, pages 287–302. USENIX, 2004.

[52] Payman Mohassel and Seyed Saeed Sadeghian. How to hide circuits in MPC an efficient framework for private function evaluation. In *Advances in Cryptology – EUROCRYPT'13*, volume 7881 of *LNCS*, pages 557–574. Springer, 2013.

[53] Robert H Morelos-Zaragoza. *The art of error correcting coding*. John Wiley & Sons, 2006. Code generation tools online at http://eccpage.com.

[54] Moni Naor, Benny Pinkas, and Reuban Sumner. Privacy preserving auctions and mechanism design. In *ACM Electronic Commerce (EC'99*, pages 129–139, 1999.

[55] J. B. Nielsen, P. S. Nordholt, C. Orlandi, and S. S. Burra. A new approach to practical active-secure two-party computation. In *Advances in Cryptology – CRYPTO'12*, volume 7417 of *LNCS*, pages 681–700. Springer, 2012.

[56] Jesper Buus Nielsen, Peter Sebastian Nordholt, Claudio Orlandi, and Sai Sheshank Burra. A new approach to practical active-secure two-party computation. In Safavi-Naini and Canetti [65], pages 681–700.

[57] Annika Paus, Ahmad-Reza Sadeghi, and Thomas Schneider. Practical secure evaluation of semi-private functions. In *Applied Cryptography and Network Security (ACNS'09)*, volume 5536 of *LNCS*, pages 89–106. Springer, 2009.

[58] B. Pinkas, T. Schneider, G. Segev, and M. Zohner. Phasing: Private set intersection using permutation-based hashing. In *USENIX Security'15*, pages 515–530. USENIX, 2015.

[59] B. Pinkas, T. Schneider, N. P. Smart, and S. C. Williams. Secure two-party computation is practical. In *Advances in Cryptology – ASIACRYPT'09*, volume 5912 of *LNCS*, pages 250–267. Springer, 2009.

[60] B. Pinkas, T. Schneider, and M. Zohner. Faster private set intersection based on ot extension. In *USENIX Security'14*, pages 797–812. USENIX, 2014.

[61] B. Pinkas, T. Schneider, and M. Zohner. Faster private set intersection based on OT extension. In *USENIX Security Symposium*, pages 797–812. USENIX, 2014.

[62] B. Pinkas, T. Schneider, and M. Zohner. Scalable private set intersection based on ot extension. *IACR Cryptology ePrint Archive*, 2016:930, 2016.

[63] Benny Pinkas, Thomas Schneider, Nigel P. Smart, and Stephen C. Williams. Secure two-party computation is practical. In *Proceedings of the 15th International Conference on the Theory and Application of Cryptology and Information Security: Advances in Cryptology*, ASIACRYPT '09, pages 250–267, Berlin, Heidelberg, 2009. Springer-Verlag.

[64] Raluca Ada Popa, Frank H. Li, and Nickolai Zeldovich. An ideal-security protocol for order-preserving encoding. In *34th IEEE Symposium on Security and Privacy*, S&P, 2013.

[65] Reihaneh Safavi-Naini and Ran Canetti, editors. *Advances in Cryptology - CRYPTO 2012*, volume 7417 of *LNCS*. Springer, 2012.

[66] Stefan Tillich and Nigel Smart. Circuits of basic functions suitable for MPC and FHE, 2015.
http://www.cs.bris.ac.uk/Research/CryptographySecurity/MPC/.

[67] Leslie G. Valiant. Universal circuits (preliminary report). In *ACM Symposium on Theory of Computing (STOC'76)*, pages 196–203. ACM, 1976.

[68] A. C. Yao. Protocols for secure computations. In *FOCS'82*, pages 160–164. IEEE, 1982.

[69] A. C. Yao. How to generate and exchange secrets. In *Foundations of Computer Science (FOCS'86)*, pages 162–167. IEEE, 1986.

[70] Andrew Chi-Chih Yao. How to generate and exchange secrets (extended abstract). In *Foundations of Computer Science (FOCS'86)*, pages 162–167. IEEE, 1986.