TECHNISCHE
UNIVERSITÄT
DARMSTADT

Master Thesis

A Systematic Comparison of HE-based Private Function Evaluation Protocols

Marco Holz

November 11, 2019

ENCRYPTO
CRYPTOGRAPHY AND
PRIVACY ENGINEERING

Engineering Cryptographic Protocols
Department of Computer Science
Technische Universität Darmstadt

Supervisors: M.Sc. Ágnes Kiss
Prof. Dr.-Ing. Thomas Schneider

Erklärung zur Abschlussarbeit gemäß §23 Abs. 7 APB der TU Darmstadt

Hiermit versichere ich, Marco Holz, die vorliegende Master Thesis ohne Hilfe Dritter und nur mit den angegebenen Quellen und Hilfsmitteln angefertigt zu haben. Alle Stellen, die Quellen entnommen wurden, sind als solche kenntlich gemacht worden. Diese Arbeit hat in gleicher oder ähnlicher Form noch keiner Prüfungsbehörde vorgelegen.

Mir ist bekannt, dass im Falle eines Plagiats (§38 Abs.2 APB) ein Täuschungsversuch vorliegt, der dazu führt, dass die Arbeit mit 5,0 bewertet und damit ein Prüfungsversuch verbraucht wird. Abschlussarbeiten dürfen nur einmal wiederholt werden.

Bei der abgegebenen Thesis stimmen die schriftliche und die zur Archivierung eingereichte elektronische Fassung überein.

Thesis Statement pursuant to §23 paragraph 7 of APB TU Darmstadt

I herewith formally declare that I, Marco Holz, have written the submitted Master Thesis independently. I did not use any outside support except for the quoted literature and other sources mentioned in the paper. I clearly marked and separately listed all of the literature and all of the other sources which I employed when producing this academic work, either literally or in content. This thesis has not been handed in or published before in the same or similar form.

I am aware, that in case of an attempt at deception based on plagiarism (§38 Abs. 2 APB), the thesis would be graded with 5,0 and counted as one failed examination attempt. The thesis may only be repeated once.

In the submitted thesis the written copies and the electronic version for archiving are identical in content.

Darmstadt, November 11, 2019

Marco Holz

Abstract

Secure function evaluation (SFE) allows two parties to jointly compute a known function on their private data. Private Function Evaluation (PFE) is a technique to obliviously evaluate a private function on private inputs. In the PFE setting, party P_1 inputs a private function f and party P_2 inputs private data x, and at the end of the protocol, P_2 only learns the function's output $y = f(x)$ while P_1 learns nothing. PFE has several applications such as privacy-preserving credit checking or user-specific insurance tariffs. In the last years, PFE protocols based on Universal Circuits (UCs), that have an inevitable superlinear overhead, have been investigated thoroughly. Specialized public key-based protocols with linear complexity were believed to be less inefficient than these UC-based approaches.

In this thesis, we take another look at the linear-complexity protocol by Katz and Malka (ASIACRYPT'11) and propose three efficient instantiations of the protocol using the Damgaard Jurik Nielsen (DJN) cryptosystem, elliptic curve ElGamal encryption and the Brakerski/Fan-Vercauteren (BFV) homomorphic encryption scheme. We show that HE-based PFE is practical with the latest improvements in the field of ECC and RLWE-based homomorphic encryption. The BFV implementation outperforms recent UC-based PFE schemes starting from circuits of size ca. 100 000 gates on. When instantiated with elliptic curve ElGamal encryption, our implementation outperforms UC-based schemes for all circuit sizes in communication.

Acknowledgments

First, I wish to express my sincere gratitude to my supervisors Thomas Schneider and Ágnes Kiss for their great support during the process of writing this thesis. Thanks for the excellent guidance, for pointing out new directions but also for giving me the flexibility to work on my thesis in a very self-determined way and the freedom to experiment and try out new ideas. Their continuous, valuable feedback and comments turned out to be a very inspiring contribution to this thesis.

I would like to thank Deevashwer Rathee from Indian Institute of Technology (BHU) for his valuable ideas and for sharing his knowledge and experiences about RLWE-based homomorphic encryption. My thanks also goes to the whole team of the Cryptography and Privacy Engineering Group at the Department of Computer Science at Technische Universität Darmstadt for their helpful feedback during my intermediate presentation and the welcoming atmosphere in their group. It has been a pleasure and a great honor to work in such an inspiring and supportive environment at ENCRYPTO.

I also place on record, my sense of gratitude to my friends, including the great people I met during my studies, for the stimulating discussions and the wonderful times we had together. Without this ongoing backing and support, this thesis would not have been possible.

Finally, I would like to express my very profound gratitude to my whole family for the unceasing encouragement, backing and support throughout my years of study and the time I spent writing this thesis.

Contents

1 Introduction

Secure multi-party computation (SMPC) is a major enabling technology in the area of privacy-preserving protocols. While computations on a local machine can be secured against malicious eavesdropping by providing physical protection and securing the system against intrusions from a remote location, computations that are performed collaboratively over a communication network on two or more devices typically rely on the trustworthiness of at least one system. This poses privacy-sensitive data supplied by participants at risk. Privacy-preserving protocols aim to mitigate these risks by protecting the data using cryptography approaches in a way that there is no need for a trusted party any more.

Secure function evaluation (SFE) protocols allow two parties to jointly compute a known function on their private data without learning the other party's input to the function. Private function evaluation (PFE) extends on this setting by also hiding the evaluated function and constitutes a very flexible approach to many challenges in the field of privacy-preserving protocols. PFE thereby provides a generic solution to evaluate any kind of private operations on private data and is a special case of multi-party computation (MPC). One party inputs a function f, often represented by a circuit $\mathcal{C}_f$, and the other party inputs their private data on which the function f is evaluated. As in the standard MPC setting, the input data of all participants remains private. In PFE, the function f is only known to one party and will also be kept private.

Past attempts to create efficient private function evaluation protocols were mostly based on so-called universal circuits (UCs) [Val76] and considerable effort has been undertaken to improve on previous results using this technique. Practical PFE based on universal circuits with a logarithmic increase in the size of the circuit $\mathcal{C}_f$ has been implemented in [KS16; GKS17; AGKS19] providing the most efficient UC implementation so far. While these generic schemes already have touched their theoretical lower bound [ZYZL19], alternative approaches still leave room for further improvements.

In the last decade, other private function evaluation protocols were proposed that do not depend on UCs. These specialized protocols are less generic but might achieve better performance in practice. Katz and Malka proposed a PFE scheme based on Yao's garbled circuit protocol in [KM11]. Their protocol utilizes additively homomorphic encryption and achieves linear complexity in the circuit size.

In the last years, tremendous research effort has been put into improving homomorphic encryption schemes. As a result, extensive improvements were made in this field. In particular, schemes based on elliptic curve cryptography [Elg85; Kob87; Mil86] and schemes based on the *learning with errors* problem over polynomial rings (Ring-LWE, or RLWE) [Reg05; LPR10; Bra12; FV12] are promising candidates for efficient instantiations of the [KM11] protocol.

This thesis takes another look at the linear-complexity PFE protocol by Katz and Malka presented in [KM11] and focuses on investigating the practicality of their construction. By implementing their scheme with state-of-the-art optimizations for the building blocks and efficiently instantiating it with modern homomorphic encryption schemes, this thesis shows that linear-complexity private function evaluation is practical and an eligible alternative to UC-based approaches.

1.1 Applications of Private Function Evaluation

PFE has many diverse applications such as privacy-preserving intrusion detection [NSMS14], blinded policy evaluation protocols with hidden policies and hidden credentials [FAL06; FLA06], user-specific insurance tariff calculation [GKSS19], and privacy-preserving credit checking [FAZ05]. All of these protocols rely on two main challenges: keeping the inputs of the participants private and hiding the operations applied to these inputs. Next, we describe some concrete example use-cases in detail.

One can think of PFE as the evaluation of a function f on private data as if the data was provided in plain, while the data is in fact not revealed to the party that evaluates the function.

An application of PFE is *privacy-preserving intrusion detection* [NSMS14]. In this scenario, an intrusion detection system (IDS) server holds a set of sensitive signatures of zero-day attacks and is able to check whether sensitive data (e.g., log files or sensitive documents) uploaded to the IDS matches those signatures. Using PFE, the IDS server will learn nothing about the sensitive data and the IDS client learns nothing about the signatures.

By making use of a private function evaluation scheme, *attribute-based access control* can be enhanced to protect both sensitive credentials and sensitive policies [FAL06]. For such an access control system, it is desirable to not only protect the requester's credentials (such as age, employment or credit status), but also the policy in order to make it harder for an adversary to undermine the security of the system.

Another application of private function evaluation is *privacy-preserving checking for credit worthiness* [FAZ05]. Traditional credit worthiness checking systems operate on highly sensitive customer data. PFE provides the opportunity to privately check the credit worthiness of a customer without disclosing any of its private financial data to the service operator. The original scheme is able to evaluate simple policies. With the high flexibility of PFE, arbitrary credit checking policies can be applied.

Privacy-preserving car insurance rate calculation can also be implemented using private function evaluation [GKSS19]. Here, the data collected by the various sensors is a massive threat for the privacy of users. Still, the data may be of special interest for car insurance companies. By making use of a PFE protocol, the privacy-critical sensor data, as well as the tariff calculation details can remain private.

1.2 Contributions

This thesis shows that using the latest improvements in the field of elliptic curve cryptography (ECC) and RLWE-based homomorphic encryption and several optimizations, linear-complexity PFE is practical. A practical implementation, an extensive performance evaluation and the introduced optimizations were the main focus of this thesis. The implemented protocol is based on the linear-complexity PFE protocol by Katz and Malka [KM11]. We introduce two further improvements to the protocol, propose two efficient instantiation using elliptic curve ElGamal encryption (*short*: EC ElGamal) and the Brakerski/Fan-Vercauteren (BFV) homomorphic encryption scheme [FV12] and compare these implementations with an instantiation using Paillier-based homomorphic encryption and UC-based PFE protocols. We implement several of our protocols using the ABY framework [DSZ15] and thereby provide this first implementation of a linear-complexity PFE scheme. Our experiments show that HE-based PFE is practical when instantiated with modern homomorphic encryption schemes. Our BFV-based implementation outperforms today's most efficient UC-based PFE [AGKS19] on the same platform starting from circuits of size about $n = 100\,000$ gates on. When instantiated with elliptic curve ElGamal encryption, our implementation outperforms UC-based schemes for all circuit sizes in communication while providing respectable runtime. Our ECC-based instantiation achieves the lowest communication of all PFE protocols known to date.

1.3 Outline

The rest of the thesis is organized as follows: In Chapter 2, we give an introduction to the fields of Secure Multi-Party Computation (SMPC) and Private Function Evaluation (PFE), and the building blocks required to build such schemes. The protocol by Katz and Malka [KM11] is described in Chapter 4, along with our improvements and details on efficient instantiation using EC ElGamal and the BFV scheme. In Chapter 3, we summarize related work. Finally, we present our implementation results in Chapter 6, and discuss directions for future work.

2 Preliminaries

In this chapter we given an introduction into the fields of SMPC in Section 2.1 and PFE in Section 2.2, introduce the building blocks required to construct a PFE scheme in Section 2.3 and describe the two main additional challenges of PFE in contrast to SMPC in Section 2.4.

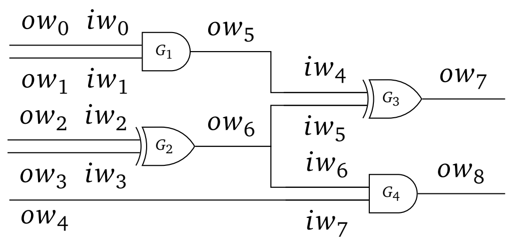
2.1 Secure Multi-Party Computation

Secure multi-party computation (SMPC), also known as *multi-party computation* (MPC), *secure function evaluation* (SFE), or *secure computation*, is a field of research that deals with the evaluation of functions by two or more mutually distrusting parties on their private inputs without revealing anything but the function's output. The function itself is public in the SMPC setting. This reveals the length of the inputs (but not the inputs themselves). More formally, each party P_i $(1 \leq i \leq N)$ knows the function f and has a private input x_i of length l and they jointly compute the output $y = f(x_1, ..., x_N)$ while keeping their own input and all the intermediate results private. We distinguish between two-party protocols with exactly two participants and multi-party protocols which are able to evaluate functions with n private inputs from n different parties. We focus on secure two-party computation (2PC), i.e., MPC with $N = 2$ parties.

A well known example for secure multi-party computation is Yao's millionaires' problem [Yao82] where two millionaires want to determine which of them is richer without revealing their actual wealth. Their goal is to evaluate the function $f(x_1, x_2) = (x_1 \geq x_2)$ where x_1 and x_2 are the private inputs of the two millionaires and they should learn nothing more than the function output.

A SMPC protocol should fulfill two main security properties: *correctness* and *privacy*. Correctness states that the correct value $f(x_1, ..., x_N)$ is computed. The privacy property ensures that each party P_i only learns $f(x_1, ..., x_N)$ and nothing else - in particular, P_i should learn nothing about x_j where $j \neq i$.

This thesis focuses on security against semi-honest (passive) adversaries where all parties are assumed to follow the protocol. This allows highly efficient protocols and is a starting point for constructing protocols with stronger security guarantees. Also, in many application scenarios parties can be assumed to behave semi-honestly, e.g., due to regulations or attestation.



(a)

iw	ow
iw_0	ow_0
iw_1	ow_1
iw_2	ow_2
iw_3	ow_3
iw_4	ow_5
iw_5	ow_6
iw_6	ow_6
iw_7	ow_4

(b)

Figure 2.1: (a) An example Boolean circuit $\mathcal{C}_f$ with $g = 4$ gates $G_1, ..., G_4$, $u = 5$ input wires $ow_0, ..., ow_4$, $o = 2$ output wires ow_7, ow_8, $g + u = 9$ *outgoing wires* $ow_0, ..., ow_8$, and $2g = 8$ *incoming wires* $iw_0, ..., iw_7$. (b) Mapping π representing the wiring of the circuit $\mathcal{C}_f$.

2.1.1 Circuits

A circuit can be defined by the number and type of its gates, the number of input and output wires of the circuit, and the wiring between the gates. We denote the total number of gates in the circuit by g, the number of input wires to the circuit by u and the number of output wires of the circuit by o. The example circuit given in Figure 2.1 consists of $g = 4$ gates, $u = 5$ input wires and $o = 2$ output wires. The input wires of the circuit correspond to the input x, and the output wires to the output of the function $y = f(x)$.

In order to define the wiring of the gates $G_i, i \in \{1, ..., g\}$, previous work has established the terms *incoming wires* and *outgoing wires* [BMR90; KM11; MS13]. In contrast to the terms *input/output wires*, which describe the inputs and outputs of the entire circuit, the terms *incoming/outgoing wires* are used to describe the inputs and outputs of the individual gates. In our example circuit, there are eight incoming wires, two for each gate. Since the fan-in for all gates in our circuit is two, the number of incoming wires is $2g$. In our example circuit, there are eight incoming wires, two for each gate. The outgoing wires include the output wires of the g gates and the u input wires of the circuit. One can imagine an invisible, non-existent dummy *input gate* before each input wire of the circuit. Their u output wires account for the additional u outgoing wires. Since all gates have exactly one output wire and there are u input wires to the circuit, we have a total of $g + u$ outgoing wires.

The wiring of the gates can now be represented by a mapping from the incoming wires to the outgoing wires of the gates. Note that each incoming wire must be connected to exactly one outgoing wire, but an outgoing wire may be connected to one or more incoming wires, which enables arbitrary fan-out of the gates. Also note that it is sufficient to define a mapping from each gate, represented by its outgoing wire, to the incoming wires connected to the two incoming wires of the gate. The mapping that represents the wiring of our example circuit is given in Figure 2.1 (b). We assume that the gates and the wires of our circuit are in topological order so that for two gates G_i and G_j with the outgoing wire of gate G_i connected to some incoming wire of gate G_j we have $i < j$.

2.1.2 Circuit-based Secure Multi-Party Computation

In the past, several MPC protocols have been proposed that rely on a circuit representation of the function f, e.g., Yao's garbled circuit protocol [Yao82; Yao86; LP09] and the GMW protocol by Goldreich, Micali and Wigderson [GMW87].

The first and most widely used approach to MPC is Yao's Garbled Circuit (GC) protocol [Yao86] which uses a Boolean circuit representation of the function f and has a constant number of communication rounds. Here, the two parties P_1 and P_2 act as *circuit garbler* and *circuit evaluator* respectively, where the garbler prepares the circuit for evaluation ("*garbling*"), and the evaluator does the evaluation of the *garbled circuit*. Yao's Garbled Circuit protocol is described in detail in Section 2.3.4.

The GMW protocol requires a round of interaction for evaluating a layer of AND gates and therefore its round complexity depends on the depth of the circuit. While the evaluation of XOR gates does not require any interaction between the parties, interactive oblivious transfers (OTs) are performed during the evaluation of AND gates. The GMW protocol can be adapted to be used in the multi-party setting.

Besides the secure evaluation of a given circuit, generating a circuit $\mathcal{C}_f$ for a function f from a specification in a high level programming language is orthogonal and there are several tools for this [MNPS04; SHS$^+$15; DDK$^+$15; HFKV12; BHWK16; BDK$^+$18]. Efficient circuit generation that tries to optimize in the type and the number of gates or the complexity of the wiring of the circuit is outside the focus of this thesis.

2.2 Private Function Evaluation

Private function evaluation (PFE), also called *secure evaluation of private functions* (PF-SFE), is a special form of Multi-Party Computation (MPC) that in addition to fulfilling all the requirements of MPC, also ensures the privacy of the function f (publicly known in the MPC setting) so that it is only known to exactly one party. Private function evaluation thereby introduces a significant additional degree of complexity compared to standard multi-party computation. In the circuit-based PFE setting, party P_1 holds a circuit $\mathcal{C}_f$, representing a function f, and each party P_i $(2 \leq i \leq N)$ holds an input value x_i.[1]

The goal of the protocol is for the parties P_i $(2 \leq i \leq N)$ to learn $f(x_1, ..., x_N)$ without revealing any information about their inputs and for party P_1 to keep the function f private.

[1]This can be extended to the case were party P_1 also holds an input value in addition to the circuit $\mathcal{C}_f$. Our 2-party PFE implementation actually supports input values for P_1 and P_2.

In practice, private function evaluation reveal the number of gates of $\mathcal{C}_f$, the number of input wires for each party and the number of output wires is publicly known to all parties. If needed, the actual number of gates and wires can be hidden by adding dummy gates and dummy input/output wires to the circuit.

One notable characteristic of PFE protocols is that P_1 typically must not be able to learn the output of the function f. The reason why this is the case is that a malicious party P_1 could reveal the inputs of any other party P_i by defining f as the identity function mapping the other party's input as the output of f, i.e., $f(x_1, ..., x_n) = x_i$. Fortunately, many protocols can be easily modified to only reveal the output of f to (a subset of) the parties P_i $(2 \leq i \leq n)$. We describe how this can be achieved in the protocol by Katz and Malka [KM11] in Section 4.1.

In order to hide the function f represented by the circuit $\mathcal{C}_f$, the type of the individual gates and the wiring of the circuit has to be kept secret from all parties except P_1. We keep up with the terminology introduced in [MS13] and refer to these two challenges as Private Gate Evaluation (PGE) and Circuit Topology Hiding (CTH) and go into detail in Sections 2.4.2 and 2.4.1.

The most common approach by prior work is to reduce the problem of PFE to classical secure computation by making use of universal circuits (UC, see Section 2.3.3). The concept of using universal circuits to build a PFE scheme is explained in Section 3.1. An alternative approach based on Yao's garbled circuits and homomorphic encryption has been presented by Katz and Malka in [KM11] (see Section 4.1).

2.3 Building Blocks

In this section, an overview over the building blocks of private function evaluation protocols is given. Homomorphic encryption schemes are described in Section 2.3.1. Section 2.3.2 gives an introduction into oblivious transfers (OTs), a technique to obliviously receive exactly one out of n elements from another participant in the protocol without revealing which element has been requested. Universal circuits, an intensively well-researched primitive that can be used to build efficient PFE schemes, are described in Section 2.3.3. The technique of Yao's garbled circuits, on which the [KM11] protocol bases upon, is explained in Section 2.3.4.

2.3.1 Homomorphic Encryption

Homomorphic encryption (HE) is a class of encryption schemes where computations on encrypted data are possible. The operations performed on the ciphertexts are reflected in the output of decryption, i.e., underlying plaintexts, as if they were applied directly on the plaintexts.

In the last years, some approaches to secure multi-party computation have been proposed that are based on homomorphic encryption. Damgard et al. [DPSZ12] proposed a multiparty computation protocol secure against active adversaries based on somewhat homomorphic encryption (SHE), a class of homomorphic encryption schemes supporting only a limited number of operations on encrypted data. Gentry proposed the very first fully homomorphic encryption (FHE) scheme, which is able to compute arbitrary operations on encrypted data [Gen09]. Private function evaluation based on fully homomorphic encryption (FHE) became possible when Gentry proposed the very first FHE encryption scheme, which is able to compute arbitrary operations on encrypted data [Gen09].

Partially Homomorphic Encryption. Due to the high computation cost of fully homomorphic encryption, Katz and Malka proposed a PFE scheme based on partially homomorphic encryption i.e. a homomorphic encryption scheme that supports only one type of homomorphic operation [KM11]. Their HE scheme is required to provide additively homomorphic properties in order to be used in their PFE scheme. The additive property allows us to add a two values under encryption by applying operations to their ciphertexts only. In particular, a party is able to add the plaintext of two encrypted values without having access to the secret key required for decryption.

Katz and Malka suggest to make use of the following two homomorphic properties of Paillier's probabilistic public-key system [Pai99]:

$$\mathsf{Dec}(\mathsf{Enc}(m_1) \cdot \mathsf{Enc}(m_2) \bmod n^2) = m_1 + m_2 \bmod n \tag{2.1}$$

$$\mathsf{Dec}(\mathsf{Enc}(m_1)^{m_2} \bmod n^2) = m_1 \cdot m_2 \bmod n \tag{2.2}$$

The additive property (Equation (2.1)) allows us to add a two values under encryption by multiplying their ciphertexts. In particular, a party is able to add the plaintext of two encrypted values without having access to the secret key required for decryption. The scalar-multiplicative property (Equation (2.2)) allows a party to multiply two values, whereof one value is encrypted, by raising the encrypted value to the power of the unencrypted value. Note that this property does not allow one to multiply two values under encryption when both values are encrypted (a property provided by some other homomorphic encryption schemes).

Today, more recent HE schemes are available that satisfy the additive homomorphic property, out of which Elliptic Curve Cryptography (ECC)-based and Ring-LWE (RLWE)-based schemes are the most efficient. In our implementation, we use EC ElGamal encryption (see Section 4.4) and the BFV cryptosystem (see Section 4.5) to achieve better performance than by using the Damgaard Jurik Nielsen cryptosystem (DJN) [DJN10], a generalization of [Pai99], and compare the three schemes in Section 6.1.

Homomorphic Encryption based on Elliptic Curve Cryptography. EC ElGamal encryption offers exceptionally small ciphertexts while still offering practicable computation and fulfills the homomorphic requirements of the [KM11] protocol. The use of elliptic curves over finite fields as a basis for a cryptosystem was suggested independently from each other by both

Koblitz [Kob87] and Miller [Mil86]. An elliptic curve $E_p(a, b)$ over the Galois field $GF(p)$ consists of the elements $P = (x, y)$ that satisfy the equation

$$y^2 = x^3 + ax + b \pmod p \tag{2.3}$$

where $x, y, a, b \in GF_p = \{1, ..., p-1\}$ together with the extra point O called the *point at infinity*. An elliptic curve cryptosystem defines the ECC arithmetic over the Galois field. It assures that any operation performed on two points on the elliptic curve results in a third element that is also a point on the curve. The addition of two elements on the elliptic curve is the counterpart to a modular multiplication in the arithmetic of classical public-key cryptosystems and the scalar multiplication of an integer and a point on the elliptic curve corresponds to modular exponentiation. Elliptic curve cryptosystems typically do not introduce novel cryptographic algorithms but define the arithmetic to apply existing algorithms, such as ElGamal [Elg85], on elliptic curves and thereby base their security on a different underlying problem. In elliptic curve arithmetic, inverting a scalar multiplication, i.e. finding the discrete logarithm of a given point on the elliptic curve with respect to a publicly known generator point, is hard. This problem is known as the *elliptic curve discrete logarithm problem* (ECDLP) and corresponds to the problem of inverting an exponentiation in traditional public-key cryptosystem arithmetic, known as the *discrete logarithm problem* (DLP).

Homomorphic Encryption based on the Ring-Learning with Errors problem. Next to ECC-based encryption, we specifically focus on a Ring-LWE-based scheme called the Brakerski/Fan-Vercauteren (BFV) scheme [FV12; Lai17] in our work. We choose BFV simply because it has the most efficient publicly available implementation, the Microsoft SEAL library [Sea19], and note that our discussion also applies to other popular Ring-LWE-based HE schemes such as BGV [BGV12]. Here, we present a high level overview of the BFV scheme restricted to only a part of BFV's functionality which is relevant for our application. For additional details, see [Lai17].

The BFV scheme operates on polynomial rings of the form $R = \mathbb{Z}[x]/(x^n + 1)$, where the *polynomial modulus degree* n is a power of 2. For a *plaintext modulus* t, the plaintext space is defined as $R_t = R/tR = \mathbb{Z}_t[x]/(x^n + 1)$, which consists of polynomials of degree $n-1$ with coefficients in $\mathbb{Z}_t$. Similarly, the ciphertext space is defined as $(R_q)^2$, where q is called the *coefficient modulus* and $R_q = R/qR$. The encryption function Enc is a probabilistic function that takes a public key pk and a message $m \in R_t$ as inputs, and outputs a ciphertext $c \in (R_q)^2$. The ciphertexts output by Enc have a noise component associated with them which is necessary for maintaining security. The decryption function Dec takes the secret key sk and a ciphertext $c \in (R_q)^2$ as inputs, and outputs a message $m \in R_t$. Decryption works, i.e., $m = \mathsf{Dec}(sk, \mathsf{Enc}(pk, m))$ if the ciphertext noise is below a certain threshold defined by the scheme parameters. For ease of exposition, we omit the keys from the invocation of the encryption and decryption function, and assume a single key-pair throughout the paper, which makes the functions compatible.

Enc is a homomorphic map from $(R_t, +)$ to $((R_q)^2, +)$, which provides the scheme with its additive homomorphic properties. This means that, given ciphertexts $c_1 = \mathsf{Enc}(m_1)$ and

$c_2 = \mathsf{Enc}(m_2)$, we have $\mathsf{Dec}(c_1 + c_2) = \mathsf{Dec}(c_1) + \mathsf{Dec}(c_2)$. The noise component grows as we perform homomorphic operations on the ciphertext until it reaches a threshold, beyond which decryption is not possible and the ciphertext is rendered useless. This is not a problem since addition does not grow the noise by much. The scheme described so far only provides IND-CPA security against parties other than the key owner. In order to hide the operations applied to ciphertext from the key owner, which may include some private inputs from other parties, and only reveal the result of decryption, the ciphertext needs to be flooded with extra noise (cf. [Lai17], Section 9.4). This requires taking larger parameters to accommodate the extra noise, and has been taken into account in our parameter selection.

2.3.2 Oblivious Transfer

Oblivious Transfer (OT) is a basic building block for many privacy preserving protocols and allows a party P_r, called receiver, to *obliviously* reveal k elements out of a set of n elements (where $k < n$) held by a party P_s, called sender. The oblivious transfer protocol is designed in a way that P_r does not learn anything about the $n-k$ other elements from P_s. However, the receiver P_r is allowed to freely choose which k elements is wants to reveal. After the protocol run, the sender P_s has learned nothing about the choice of P_r, i.e. it has no knowledge which k out of the n elements were chosen by P_r.

In the special case of a so-called *1 out of two oblivious transfer*, the sender inputs two l-bit strings (m_0, m_1) and the receiver inputs a selection bit $b \in \{0,1\}$ and obliviously learns m_b but learns nothing about m_{1-b} and the sender learns nothing about b (see Figure 2.2).

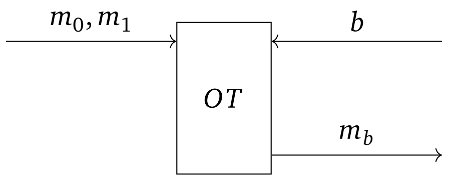


Figure 2.2: A simple 1-out-of-2 oblivious transfer

Oblivious transfer protocols have been well-researched in the last years. While a single 1-out-of-2 oblivious transfer is a relatively expensive operation relying on public-key cryptography, it is possible to perform a larger number of OTs more efficiently using OT extensions [IKNP03; ALSZ13]. This technique allows to extend some *base OTs* to obtain many oblivious transfers based on cheap symmetric cryptographic operations.

In the private function evaluation protocol by Katz and Malka [KM11], oblivious transfer is used to obliviously send the correct value of the input wires to the circuit as in Yao's garbled circuit protocol (see Section 2.3.4).

2.3.3 Universal Circuits

A Universal Circuit (UC) is a special circuit that is able to simulate any Boolean circuit up to size n. The UC can be programmed to evaluate a specific Boolean circuit $\mathcal{C}_f$ by specifying so-called programming bits (or control-bits). These programming bits represent the functionality of $\mathcal{C}_f$. The evaluation of the Boolean circuit described by the programming bits p with input x will then be equivalent to the evaluation of the universal circuit $\mathcal{UC}(p, x)$.

By evaluating a UC in a MPC protocol where party P_1 inputs the programming bits p and P_2 inputs x, the two parties can perform a private function evaluation of the function f represented by p. Since the MPC protocol ensures that the inputs p and x are kept secret, this reduces PFE to standard MPC (see Section 3.1).

In recent years, a lot of research was put into optimizing and implementing UC-based PFE. These efforts have now touched lower bounds, so no more significant performance improvements are expected: Valiant [Val76] proposed two recursive UC constructions with 2 and 4 substructures (so-called 2-way and 4-way UCs), and asymptotic sizes $\sim 5n \log_2 n$ and $\sim 4.75n \log_2 n$ respectively, which are close to the asymptotic lower bound of $\Omega(n \log n)$ on the size of a UC. The most recent work of Zhao et al. [ZYZL19] showed that their 4-way split building block achieves the smallest concrete complexity of $\sim 4.5n \log_2 n$. Also, for Yao's protocol Zahur et al. showed a lower bound of 256 bits of communication per AND gate using their *half gates* optimization [ZRE15]. Lipmaa et al. [LMS16] generalized Valiant's constructions to any k-way UC. A hybrid UC construction combining 2-way and 4-way UCs and optimizations from [KS16; GKS17; ZYZL19] and an implementation providing the most efficient UC implementation with size $\sim 4.5n \log_2 n$ was given in [AGKS19].

Alhassan et al. [AGKS19] suggested various applications for size-optimized universal circuits, such as private function evaluation (see Section 3.1), efficient verifiable computation on encrypted data [FGP14; GGPR13], multi-hop homomorphic encryption [GHV10], Attribute-Based Encryption (ABE) [GGHZ14; Att14] and secure two-party computation in the batch execution setting [LMS16; HKK$^+$14; LR15; MR17].

2.3.4 Yao's Garbled Circuit Protocol

The *garbled circuit* protocol [Yao86] is a circuit-based two-party secure computation protocol that enables two parties P_1 and P_2 to obliviously evaluate a public function f over their private inputs $x \in \{0, 1\}^{u_1}$ and $x' \in \{0, 1\}^{u_2}$. The parties learn the output $y = f(x, x')$ but will learn nothing about the input of the other party. The protocol operates on a Boolean circuit representation $\mathcal{C}_f$ of the function f.

The idea behind the garbled circuit protocol was first introduced by Yao in an oral presentation of his paper "How to generate and exchange secrets" in 1986 [Yao86], as mentioned in [Gol03]. However, Yao did not publish any paper describing his idea. The first written description of the protocol was given by Goldreich, Micali, and Wigderson in [GMW87]. The term "garbled circuit" was introduced by Beaver, Micali, and Rogaway in [BMR90]. A PRF-based instantiation of the garbled circuit protocol is given by Naor, Pinkas, and Sumner [NPS99] as an alternative to the primitive based on symmetric double encryption by Yao. Lindell and Pinkas provide the first proof of Yao's protocol [LP09]. A novel formalization of garbling schemes that recaps previous approaches and respects optimizations like PRF-based garbling [NPS99], *point-and-permute* [BMR90], *garbled row reduction* [NPS99; PSSW09] and *free-XOR* [KS08b] is given by Bellare, Hoang and Rogaway in [BHR12]. More recent optimizations to the well-studied garbled circuit protocol include *fixed-key AES garbling* [BHKR13] and *half gates* [ZRE15].

In the garbled circuit protocol described by Yao, each wire of the circuit is represented by a pair of so-called *wire keys*. These labels represent the potential values 0 and 1 of the corresponding wire. Each wire key is a random bit-string of length σ, where σ is our security parameter. We denote the wire key of wire w_i representing the value 0 by s_i^0 and the wire key representing the value 1 by s_i^1.

As a first step, party P_1 randomly samples and assigns the wire keys to the wires of the circuit. P_1 is now able to create the *garbled gates*, i.e., *garbled* representations of the individual gates, using the wire key pairs for the input wires and the output wire of each gate. We assume, that gate G_i has two input wires, w_j and w_k, and one output wire w_i. We remember from Section 2.1.1 that each input wire of a gate is identical ("connected") to one outgoing wire of the circuit. Our example gate G_i is depicted in Figure 2.3. Creation of a garbled gate is depicted in Figure 2.4.

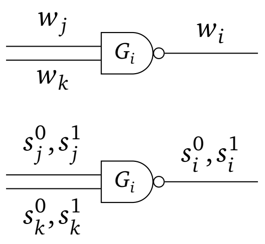


Figure 2.3: Example XOR gate G_i connected to the input wires w_j and w_k and its assigned wire keys

Each garbled gate GG_i is represented by a *garbled computation table (GCT)*, also called *garbled table (GT)*, that is constructed from the gate's truth table as follows: First, the values in the truth table of gate G_i are replaced by the corresponding wire keys representing the potential values of the wires. Next, the output values of the truth table are encrypted using the wire keys of the input wires for any combination of the input wire keys. Encryption is done using a symmetrical encryption scheme and the wire keys are used as encryption keys. This hides the output wire keys of gate G_i. Party P_2 will later only be able to decrypt one of the output wire keys, either s_i^0 *or* s_i^1. Which wire key will be retrieved depends on the input wire keys known to party P_2. If, for example, P_2 knows the wire keys s_j^1 and s_k^0, it can use the ciphertext

$\mathsf{Enc}_{s_j^1}(\mathsf{Enc}_{s_k^0}(s_i^1))$ to learn s_i^1. However, P_2 is not able to learn s_i^0 from the four ciphertexts of the encrypted truth table and the wire keys s_j^1 and s_k^0.

w_j	w_k	$w_j \oplus w_k$		s_j	s_k	s_i		enc. truth table		GARBLED TABLE
0	0	0		s_j^0	s_k^0	s_i^0		$\mathsf{Enc}_{s_j^0}(\mathsf{Enc}_{s_k^0}(s_i^0))$		$\mathsf{Enc}_{s_j^0}(\mathsf{Enc}_{s_k^1}(s_i^1))$
0	1	1	$\rightarrow$	s_j^0	s_k^1	s_i^1	$\rightarrow$	$\mathsf{Enc}_{s_j^0}(\mathsf{Enc}_{s_k^1}(s_i^1))$	$\rightarrow$	$\mathsf{Enc}_{s_j^1}(\mathsf{Enc}_{s_k^1}(s_i^0))$
1	0	1		s_j^1	s_k^0	s_i^1		$\mathsf{Enc}_{s_j^1}(\mathsf{Enc}_{s_k^0}(s_i^1))$		$\mathsf{Enc}_{s_j^0}(\mathsf{Enc}_{s_k^0}(s_i^0))$
1	1	0		s_j^1	s_k^1	s_i^0		$\mathsf{Enc}_{s_j^1}(\mathsf{Enc}_{s_k^1}(s_i^0))$		$\mathsf{Enc}_{s_j^1}(\mathsf{Enc}_{s_k^0}(s_i^1))$

Figure 2.4: Creation of a garbled XOR gate

The entries of the encrypted truth table are permuted in random order to hide which of its entries corresponds to which entry of the unencrypted table. The permuted truth table is called *garbled table* and encodes the *garbled gate*. This procedure is therefore also called *garbling* of a gate. Since P_1 performs this operation for all gates of the circuit, it is also referenced as the *circuit garbler*.

Note that even if creation of a garbled gate is not limited to any specific gate logic it is conventional to use only XOR and AND gates in Yao's garbled circuit protocol. This is due to the *free-XOR* optimization of [KS08b], which allows XOR gates to be evaluated locally, i.e., essentially "for free".

The garbled gates of the circuit together form the *garbled circuit (GC)* representing the function f. The GC is transferred to party P_2, who acts as the *circuit evaluator*. Besides the GC, P_2 needs to learn its share of the input values x and x'. For each bit of the input values, there is one input wire of the circuit $\mathcal{C}_f$ that represents its bit value. These input wires have been used as encryption keys during GC generation. For P_1's input value $x = x_1||\ldots||x_{u_1}$ of length u_1, P_1 simply sends the wire keys $s_1^{x_1}, \ldots, s_{u_1}^{x_{u_1}}$. Note, that since the wire keys representing the values 0 and 1 are uniform and random, P_2 does not learn anything about the value of x.

P_1 also generated the wire key pairs corresponding to P_2's input x'. In order to retrieve the wire keys representing x', P_1 and P_2 evaluate a series of 1-out-of-2 oblivious transfers, one for each bit of x'. During each oblivious transfer, P_1 acts as the *sender*, holding the two wire keys $s_{u_1+i}^0$ and $s_{u_1+i}^1$ where $i \in [1, u_2]$. These wire keys represent the potential bit values for P_2's input value x_i'. P_2 acts as the *receiver*, holding the actual bit value of its input x_i'. After the oblivious transfers, P_2 holds either $s_{u_1+i}^0$ or $s_{u_1+i}^1$ and P_1 learns nothing about P_2's input value x_i'.

P_2 now knows the wire keys representing the input values of P_1 and P_2 and is able to evaluate the garbled circuit. As described in Section 2.1.1, the gates are ordered in topological order, i.e. for the gates G_j and G_k connected to the inputs of gate G_i it holds that $j < i$ and $k < i$.

Therefore, the evaluation of the circuit can be done gate by gate. For each garbled gate, P_2 knows two wire keys s_j and s_k, one for each of its incoming wires, and tries to decrypt the four entries in the garbled table. For exactly one of the entries, decryption succeeds and P_2 learns the outgoing wire of the gate. We denote this as $s_i = \mathsf{decYao}(s_j, s_k, GG_i)$. P_2 is now able to use this outgoing wire in the evaluation of one (or more) of the following gates. When P_2 completely evaluated the garbled circuit, it holds the wire keys of the output wires of the circuit that represent the output of the function f.

In the first step of the protocol, party P_1 created two wire keys for each output wire of the circuit, one of them representing the value 0 and one of them representing the value 1. P_1 can now send the wire key pairs corresponding to the output wires of the circuit to P_2 and P_2 is then able to compare these wire keys to those retrieved during circuit evaluation. Thereby, P_2 learns the output $y = f(x, x')$. Alternatively, P_2 could send its retrieved output wire keys to P_1 and P_1 could do the comparison. This way, P_1 learns the output of the function f.

2.4 Challenges of Private Function Evaluation

Private function evaluation introduces a significant additional degree of complexity compared to standard multi-party computation. In addition to hiding the inputs to a circuit $\mathcal{C}_f$, the circuit itself is only known to one of the parties in the PFE setting. In order to hide the function f represented by the circuit $\mathcal{C}_f$, the type of the individual gates and the wiring of the circuit has to be kept secret from all parties except P_1.

Mohassel and Sadeghian [MS13] introduce a framework for designing circuit-based PFE protocols. They identify two main challenges facing private function evaluation. First, the topology of the circuit has to be hidden from all parties except party P_1 holding the circuit as its input to the protocol. Since the topology of the circuit can be represented as a mapping $\pi_{\mathcal{C}}$ from the incoming wires to the outgoing wires of the gates (see Section 2.1.1), this challenge can be reduced to the problem of hiding $\pi_{\mathcal{C}}$ from $P_{i,1<i\leq n}$ while still enabling the evaluation of the circuit. We adopt to the term introduced in [MS13] and refer to this task as Circuit Topology Hiding (CTH).

Second, hiding the functionality of the individual gates is another challenge of PFE not present in the classical MPC setting. In order to reveal no information about the circuit $\mathcal{C}_f$, it is of vital importance that no gate is distinguishable from any other gate in the circuit during evaluation. Each gate has to be evaluated *blindly* without leaking any information about its internal functionality to a party $P_{i,1<i\leq n}$. Again, we maintain the terminology introduced in [MS13] and refer to this task as Private Gate Evaluation (PGE).

2.4.1 Circuit Topology Hiding (CTH)

Hiding the topology of a circuit $\mathcal{C}_f$ is of vital importance in order to hide the function f. In the PFE setting, it is usually accepted that the size of the circuit, the number of input bits for each party and the number of output bits is commonly known by all parties. However, the mapping π_C that encodes the wiring of the gates leaks some information about the functionality of the whole circuit and must therefore be hidden from all parties except P_1.

There are multiple approaches to hide the wiring of the circuit, represented by the mapping π_C. Katz and Malka propose a hiding technique based on homomorphic encryption [KM11]. Intuitively, they assign labels to each wire of the circuit (wire keys) and let party P_1 blind the codomain of the mapping π_C under homomorphic encryption so that P_1 does not learn the wire keys and all other parties do not learn the mapping π_C. A full description of their protocol is given in Section 4.1.

Mohassel and Sadeghian propose another approach to CTH based on switching networks [MS13]. In their approach, a switching network that corresponds to an extended permutation is evaluated obliviously using their oblivious switching network (OSN) construction. The oblivious witching network is then used to obliviously evaluate the mapping π_C.

2.4.2 Private Gate Evaluation (PGE)

Besides hiding the wiring of the gates, it is important to hide the functionality of the individual gates of the circuit in order to prevent any leakage functionality of the circuit C_f. A simple but effective solution is to only use one type of gates in the circuit, e.g. only NAND gates. Thereby, hiding the functionality of the gates can be reduced to hiding the output of a singe gate by applying Yao's garbled table technique solely to NAND gates. As opposed to the original protocol by Yao, the circuit garbler does not need to differentiate between the types of the gates when creating the garbled tables for NAND gates only. Since there is no need to have any knowledge about the functionality of the gate when ungarbling them in the garbled circuit protocol, this makes the gates indistinguishable for the circuit evaluator in terms of their functionality.

3 Related Work

Previous work on private function evaluation can be categorized in two main approaches: 1) Evaluating a universal circuit (UC) with a Secure Function Evaluation (SFE) protocol like Yao's garbled circuit protocol [Yao82; Yao86; LP09] or the so-called GMW protocol [GMW87] reduces the task of oblivious evaluation of a private function to standard secure function evaluation SFE where the function is the publicly known UC. These protocols rely mostly on symmetric cryptography but UCs have superlinear complexity in the size of the private function. 2) The second category of PFE protocols are linear-complexity protocols based on public-key cryptography, such as [KM11] and [MS13].

The protocols in this thesis provide security against semi-honest adversaries. There are also first approaches to PFE focusing on malicious adversaries. Mohassel et al. extend the framework from [MS13] to security against malicious adversaries with linear complexity of $O(g)$ based on linear zero-knowledge proofs [MSS14]. They provide the first actively secure PFE protocol besides the generic approach of evaluating universal circuits using an actively secure MPC protocol.

3.1 Private Function Evaluation based on Universal Circuits

Private evaluation of a function f can be reduced to the less complex problem of Secure Function Evaluation by using universal circuits to evaluate the function f. In many SFE protocols, the function f is represented as a Boolean circuit $\mathcal{C}_f$. Universal circuits can be programmed to evaluate such a Boolean circuit by defining a representation p of the circuit $\mathcal{C}_f$ as the UC's input (see Section 2.3.3). This representation p, also called *programming bits of the UC*, can be used next to the input x of the function f to evaluate $\mathcal{UC}(p, x)$.

When using a secure computation protocol to evaluate $\mathcal{UC}(p, x)$, where the universal circuit is the function to be evaluated, p is the input of party P_1 and x is the input of party P_2, the input of the other party mutually remains oblivious and either one or both parties receive the output $y = \mathcal{UC}(p, x)$ of the universal circuit. The construction of the UC and the correctness assumption of the MPC protocol guarantee that the output of $\mathcal{UC}(p, x)$ is identical to the output of $f(x)$. Thereby, we have constructed a PFE protocol that relies on universal circuits and a SFE protocol to privately evaluate the function f, specified by party P_1, on the private input x of party P_2.

In the PFE protocol based on a universal circuit, the UC is publicly known to both parties. All functions f that can be represented by the UC can be evaluated in the PFE protocol. We remember from Section 2.3.3 that a given UC is able to evaluate Boolean circuits up to size n. Constructing size-optimized universal circuits that are able to evaluate large Boolean circuits is an active field of research and strongly influences the performance of the PFE protocol.

UC-based PFE can be instantiated with any circuit-based MPC protocol and also benefits from recent improvements to those protocols. A construction of PFE using universal circuits was first proposed by Abadi and Feigenbaum in [AF90]. In the last years, considerable effort has been undertaken to improve on previous results [SYY99; Pin02; AGKS19]. The main challenge in improving on the performance of UC-based PFE protocols today relies in optimizing the size of a universal circuit capable of evaluating a Boolean circuit of size n. The first implementation of PFE based on universal circuits was provided in [KS08a; Sch08]. Past efforts on constructing size-optimized universal circuits have been summarized, extended and combined to lead to the most efficient UC-based PFE scheme known so far in [AGKS19].

3.2 Private Function Evaluation with Linear Complexity

Private function evaluation schemes based on universal circuits have inevitable logarithmic overhead and already have touched their theoretical lower bound [ZYZL19]. In the last years, other approaches have been introduced that do not rely on UCs. We specially focus on showing the practicality of the scheme by Katz and Malka [KM11]. Their scheme bases on partially homomorphic encryption and has linear complexity in the size of the circuit $\mathcal{C}_f$ representing the function f. Katz and Malka extend Yao's garbled circuit protocol by introducing so-called *encrypted garbled gates* (encGG) in order to hide the topology of the circuit $\mathcal{C}_f$. By applying a homomorphic blinding technique to the wire keys encrypted in these encrypted garbled gates, the topology of the circuit becomes oblivious to the circuit garbler. We give a full description of the scheme in Section 4.

Biçer at al. introduce the first reusable private function evaluation scheme with linear complexity [BBKL18a]. Their scheme achieves significant cost reduction when the same private function f is evaluated multiple times between the same two parties or between the party holding the function f and another varying party. The reusability feature is achieved by deriving the wire keys from a set of generators in a cyclic group $\mathcal{G}$ of large prime order instead of choosing random values for the wire keys.

Mohassel and Sadeghian introduce another approach based on the oblivious evaluation of a switching network of size $\Theta(n \log n)$ [MS13]. In their scheme, the topology of the circuit, described by the mapping $\pi_{\mathcal{C}}$, is represented by a switching network of an extended permutation (see Figure 3.1). Their novel technique to obliviously evaluate such a switching network uses oblivious transfer to evaluate the individual switches in the network. As shown in [AGKS19], the communication of both [MS13] and [BBKL18a] is worse than their UC-based private function evaluation protocol.

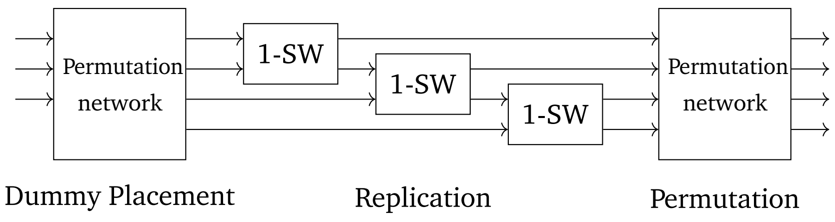


Figure 3.1: A switching network implementing an extended permutation

Bingöl at al. adapt the half gates optimization by Zahur et al. [ZRE15] to the PFE setting [BBKL18b]. Their work bases on the OT-based approach by Mohassel and Sadeghian and significantly reduces the number of oblivious transfers during the evaluation of the switching network. Similar to the optimized variant of the [KM11] protocol, Bingöl at al. introduce a global random shift r, randomly choose only one wire key s_0 and set the other wire key as $s_1 = s_0 \oplus r$. This significantly reduces the size of the switching network and leads to a considerable decrease in communication compared to [MS13]. Their scheme also reduces the size of the garbled tables for output gates by half and for non-output gates from four to only three ciphertexts.

4 Homomorphic Encryption-based Private Function Evaluation

In this section, we recapitulate the protocol of Katz and Malka [KM11] and its improved version in Sections 4.1 and 4.2, introduce two further improvements to the protocol in Sections 4.3.1 and 4.5.3 and propose an efficient instantiation of the protocol using the BFV homomorphic encryption scheme in Section 4.5.

4.1 The [KM11] Protocol

Katz and Malka [KM11] proposed a Private Function Evaluation protocol which combines homomorphic encryption with Yao's Garbled Circuit protocol to hide the topology of the circuit in addition to the parties' inputs.

The protocol, summarized in Figure 4.2, can be divided into three parts: (i) hiding the topology of the circuit by homomorphically blinding the wire keys used to create the garbled tables from the original Yao's Garbled Circuit protocol, (ii) creation and evaluation of the garbled tables, and (iii) output decryption. The main contribution of [KM11] lies in the first phase where the encrypted wire keys are blinded using homomorphic encryption and combined into so-called *encrypted garbled gates* in order to hide the topology of the circuit. These encrypted garbled gates contain the wire keys to create the garbled tables as in the original protocol by Yao. Determining the output of the circuit is done very similar to Yao's original garbled circuit protocol but respects the blinding of the wire keys when deriving the keys to decrypt the garbled tables.

In the protocol by Katz and Malka, party P_2 holds the input value $x = x_1||\dots||x_{u_2}$ of length u_2 and acts as the *circuit garbler*. P_1 holds the circuit $\mathcal{C}_f$ of g gates and (optionally) another input value $x' = x'_1||\dots||x'_{u_1}$ of length u_1 and acts as the *circuit evaluator*[1]. Since P_1 holds the circuit $\mathcal{C}_f$ and P_2 is unaware of the circuit wiring, P_2 cannot directly garble the gates. Instead, P_1 creates an encrypted garbled gate for each gate in the circuit and P_2 decrypts theses encrypted garbled gates to learn the keys required to create the garbled tables. By creating the encrypted garbled gates, P_1 obliviously connects two *outgoing wires* to each gate of the circuit (the wire keys for the outgoing wires are provided by P_2 beforehand). We denote the encrypted garbled gate corresponding to gate i as encGG_i. Creation of the

[1]Note, that the roles of P_1 and P_2 are defined in the reversed way in [KM11].

encrypted garbled gates is done under homomorphic encryption in order to hide this mapping from P_2.

We remember the two main challenges of PFE from Section 2.4: Circuit Topology Hiding (CTH) and Private Gate Evaluation (PGE). In order to hide the functionality of the individual gates, only NAND gates are used in the Boolean circuit. This reduces the PGE problem to creating garbled tables for NAND gates only. This is not a restriction because arbitrary gates can be replaced by a few NAND gates and there are even established hardware synthesis tools that optimize for a small number of NAND gates. Since all gates have the same logical functionality, they are indistinguishable from each other and no further effort is needed to accomplish the challenge of PFE. CTH is realized using homomorphic encryption by creating and evaluating the *encrypted garbled gates* in phase (i) of the protocol.

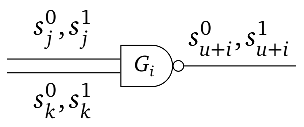


Figure 4.1: A single gate with two incoming wires and one outgoing wire

In phase (i) of the protocol, P_2 creates two wire keys for each of the $N = g + u = g + (u_1 + u_2)$ outgoing wires - one of them representing the bit value 0 and one representing the bit value 1. The wire keys of all $g + u$ outgoing wires except the o output wires of the circuit are essential to define the mapping representing the topology of the circuit. We denote the wire key corresponding to the bit value $b \in \{0, 1\}$ on outgoing wire $i \in \{1, \ldots, N\}$ by s_i^b. The security of the protocol depends on the indistinguishability of the two keys. Therefore, the wire keys are chosen at random. P_2 initiates the protocol by sending those wire keys under encryption using a homomorphic encryption scheme to P_1:

$$[\mathsf{Enc}(s_1^0), \mathsf{Enc}(s_1^1)], \ \ldots, \ [\mathsf{Enc}(s_{g+u-o}^0), \mathsf{Enc}(s_{g+u-o}^1)]. \tag{4.1}$$

Using the homomorphic property of the encryption scheme, party P_1 is now able to create the encrypted garbled gates. In order to hide the wiring of the circuit from P_2, each wire key has to be blinded. Creation of the encrypted garbled gates is then done as follows: P_1 randomly samples the blinding values a_i, b_i, a_i', b_i' for each gate G_i. Say the outgoing wires j and k are connected to the incoming wires of gate G_i (as depicted in Figure 4.1), then P_1 constructs the encrypted garbled gate encGG_i as

$$\mathsf{encGG}_i = \begin{pmatrix} [\mathsf{Enc}(a_i * s_j^0 + b_i), \ \mathsf{Enc}(a_i * s_j^1 + b_i)] \\ [\mathsf{Enc}(a_i' * s_k^0 + b_i'), \ \mathsf{Enc}(a_i' * s_k^1 + b_i')] \end{pmatrix} \tag{4.2}$$

by making use of the homomorphic properties of the homomorphic encryption scheme. After the creation of the encrypted garbled gates, P_1 sends $\mathsf{encGG}_1, \ldots, \mathsf{encGG}_{g+u}$ to P_2.

P_1 (holds $\mathcal{C}_f$)		P_2 (holds input $x \in \{0,1\}^l$)
$a_i, b_i, a'_i, b'_i \leftarrow_\$ \{0,1\}^\sigma$		$(\mathsf{pk}, \mathsf{sk}) \leftarrow_\$ \mathsf{KGen}(1^n)$ $(s_i^0, s_i^1) \leftarrow_\$ \{0,1\}^\sigma \ \forall i \in \{1, ..., N\}$ where $\sigma = 128$ is our symmetric security parameter
	$\xleftarrow{\mathsf{pk}, \forall i : \{\mathsf{Enc}(s_i^0), \mathsf{Enc}(s_i^1)\}}$	
$\forall i \in \{1, ..., g\}$: *create encrypted garbled gate* $encGG_i$		
	$\mathsf{Enc}(a_i * s_j^0 + b_i),\ \mathsf{Enc}(a_i * s_j^1 + b_i)$ $\xrightarrow{\mathsf{Enc}(a'_i * s_k^0 + b'_i),\ \mathsf{Enc}(a'_i * s_k^1 + b'_i)}$	
		compute garbled table GG_i $L_i^0 = \mathsf{Dec}(\mathsf{Enc}(a_i * s_j^0 + b_i)$ $L_i^1 = \mathsf{Dec}(\mathsf{Enc}(a_i * s_j^1 + b_i)$ $R_i^0 = \mathsf{Dec}(\mathsf{Enc}(a'_i * s_k^0 + b'_i)$ $R_i^1 = \mathsf{Dec}(\mathsf{Enc}(a'_i * s_k^1 + b'_i)$ GARBLED TABLE GG_i (entries in random order) $\mathsf{sEnc}_{L_i^0}(\mathsf{sEnc}_{R_i^0}(s_{u+i}^1))$ $\mathsf{sEnc}_{L_i^0}(\mathsf{sEnc}_{R_i^1}(s_{u+i}^1))$ $\mathsf{sEnc}_{L_i^1}(\mathsf{sEnc}_{R_i^0}(s_{u+i}^1))$ $\mathsf{sEnc}_{L_i^1}(\mathsf{sEnc}_{R_i^1}(s_{u+i}^0))$
	$\xleftarrow{GG_i,\ s_1^{x_1}, \ldots, s_l^{x_1}}$	
$L_i = a_i * s_j + b_i$ $R_i = a'_i * s_k + b'_i$ $s_{u+i} = \mathsf{decYao}(L_i, R_i, GG_i)$ *once all output wires have been determined:*		
	$\xrightarrow{s_{N-o+1}, \ldots, s_N}$	
		compare $s_{N-o+1}, \ldots, s_N$ *to* $[s_{N-o+1}^0, s_{N-o+1}^1], \ldots, [s_N^0, s_N^1]$ to receive the output bits

Figure 4.2: The [KM11] protocol (original, non-improved version)

Note that the values of the keys s^0_{u+i} and s^1_{u+i} assigned to the outgoing wire of gate G_i do not need to be included in encGG_i since P_2 already knows the plaintext values of these keys.

P_2 is now able to create the garbled tables in phase (ii) of the protocol and thereby acts as the *circuit garbler*. For each gate G_i, P_2 decrypts the corresponding encrypted garbled gate and retrieves one blinded key for the left and one for the right incoming wire of the gate without learning the blinding values:

$$[L^0_i, L^1_i] = [\mathsf{Dec}(\mathsf{Enc}(a_i * s^0_j + b_i)),\ \mathsf{Dec}(\mathsf{Enc}(a_i * s^1_j + b_i))] \tag{4.3}$$

for the left incoming wire and

$$[R^0_i, R^1_i] = [\mathsf{Dec}(\mathsf{Enc}(a'_i * s^0_k + b'_i)),\ \mathsf{Dec}(\mathsf{Enc}(a'_i * s^1_k + b'_i))] \tag{4.4}$$

for the right incoming wire. These keys are used to encrypt the garbled table GG_i for gate G_i. Note that the blinded wire keys L^0_i, L^1_i and R^0_i, R^1_i are random and independent of the keys assigned to the outgoing wires of the gates G_j and G_k. This hides the topology of the circuit to P_2 while still enabling P_2 to create the garbled tables using these blinded wire keys and the outgoing wires s^0_{u+i} and s^1_{u+i} of gate G_i. As in the original Yao's garbled circuit protocol, the values of the gate's truth table are first replaced by the corresponding wire keys. Next, the keys of the outgoing wire of gate G_i are encrypted according to the truth table using the symmetric encryption sEnc and the keys L^0_i, L^1_i, R^0_i and R^1_i as the symmetric encryption keys. We use AES 128-bit symmetric encryption in our implementation. The four resulting ciphertexts are permuted in a random order to obtain the garbled table for gate G_i, also known as the *garbled gate GG_i* (see Figure 4.3). Note that P_2 does not learn anything about the functionality of the individual gates, since the circuit consists solely of NAND gates.

j	k	$\overline{j \wedge k}$
0	0	1
0	1	1
1	0	1
1	1	0

$\rightarrow$

L^j_i	R^k_i	$s^{\overline{j\wedge k}}_{u+i}$
L^0_i	R^0_i	s^1_{u+i}
L^0_i	R^1_i	s^1_{u+i}
L^1_i	R^0_i	s^1_{u+i}
L^1_i	R^1_i	s^0_{u+i}

$\rightarrow$

encrypted truth table
$\mathsf{sEnc}_{L^0_i}(\mathsf{sEnc}_{R^0_i}(s^1_{u+i}))$
$\mathsf{sEnc}_{L^0_i}(\mathsf{sEnc}_{R^1_i}(s^1_{u+i}))$
$\mathsf{sEnc}_{L^1_i}(\mathsf{sEnc}_{R^0_i}(s^1_{u+i}))$
$\mathsf{sEnc}_{L^1_i}(\mathsf{sEnc}_{R^1_i}(s^0_{u+i}))$

$\rightarrow$

GARBLED TABLE GG_i
$\mathsf{sEnc}_{L^0_i}(\mathsf{sEnc}_{R^1_i}(s^1_{u+i}))$
$\mathsf{sEnc}_{L^1_i}(\mathsf{sEnc}_{R^1_i}(s^0_{u+i}))$
$\mathsf{sEnc}_{L^0_i}(\mathsf{sEnc}_{R^0_i}(s^1_{u+i}))$
$\mathsf{sEnc}_{L^1_i}(\mathsf{sEnc}_{R^0_i}(s^1_{u+i}))$

Figure 4.3: Creation of a garbled table in [KM11]

After creating the garbled gates $\mathsf{GG}_1, \ldots, \mathsf{GG}_g$, P_2 sends them to P_1 alongside the wire keys $s_1^{x_{u_1+1}}, \ldots, s_l^{x_{u_1+u_2}}$ of the circuit input wires corresponding to P_2's input bits $x_1, \ldots, x_{u_2}$. Note, that since the wire keys representing the values 0 and 1 are uniform and random, P_1 does not learn anything about the value of x. If party P_1 also holds an input value x', oblivious transfer is used to retrieve only the wire keys $s_1^{x'_1}, \ldots, s_l^{x'_{u_1}}$ corresponding to the input bits of x'. This way P_1 only learns the wire keys necessary to evaluate the circuit with one input value x' and P_2 learns nothing about x'.

P_1 is now able to evaluate the garbled tables and determine the wire keys of the output wires of the circuit representing the output bits of the function f encoded by the circuit $\mathcal{C}_f$. As noted in Section 2.1.2, we assume the gates of the circuit to be in topological order. This guarantees that the outgoing wire keys of two gates G_j and G_k connected to the two incoming wires of a gate G_i will be evaluated before the evaluation of gate G_i. Therefore, we can assume their availability during the evaluation of gate G_i. For the first gates which input wires are not connected to another gate but an input wire of the circuit, these wire keys are also available since they have been send in the last step by party P_2.

In order to evaluate gate G_i, P_1 has to reconstruct the keys used to encrypt one entry of the garbled table. Starting with the first gate in topological order, P_1 uses the keys $s_j \in \{s_j^0, s_j^1\}$ and $s_k \in \{s_k^0, s_k^1\}$ and the blinding values a_i, b_i, a_i', b_i' for gate G_i to calculate $L_i = a_i * s_j + b_i$ and $R_i = a_i' * s_k + b_i'$. P_1 is now able to decrypt the garbled gate GG_i to learn $s_{u+i} = \mathsf{decYao}(L_i, R_i, \mathsf{GG}_i)$ (the wire key of the outgoing wire of gate G_i) and continue with the next gate in topological order. Once all gates have been evaluated, P_1 has obtained the wire keys $s_{N-o+1}, \ldots, s_N$ of the output wires.

Phase (iii) of the protocol, output determination, can now be done in the same way as in Yao's garbled circuit protocol. Since P_1 holds the private function that is being evaluated, P_1 could however just define f as $f(x, x') = x$ and potentially learns P_2's input value x which violates the security assumptions of the PFE protocol. Therefore, the output of the circuit should be determined by party P_2 which has no influence on the choice of f. Hence, P_1 sends the wire keys of the output wires to P_2. When comparing the individual output wire keys to the wire key pairs for these wires generated in the first step of the protocol, P_2 learns the output bits of the circuit $\mathcal{C}_f$ and thereby the output of the function f.

4.2 Improved Version of the KM11 Protocol

Katz and Malka describe a more efficient variant of their protocol in their paper that is inspired by the *Free XOR* technique of Kolesnikov and Schneider [KS08b]. We summarize the more efficient variant in Figure 4.4.

Instead of the $g + u$ wire key pairs, P_2 only chooses $g + u$ single wire keys $s_1^0, ..., s_{g+u}^0$ and, similar to the *free XOR* technique [KS08b], defines a global random shift r of the same size as the wire keys. P_2 then sets $s_i^1 = s_i^0 + r$ for $i \in \{1, \ldots, g + u\}$ and sends those wire keys under encryption using a homomorphic encryption scheme to P_1:

$$\mathsf{Enc}(s_1^0),\ \ldots,\ \mathsf{Enc}(s_{g+u-o}^0). \tag{4.5}$$

Creation of the encrypted garbled gates is then done as follows: P_1 creates two random blinding values, b_i and b_i', for each gate G_i. Say the outgoing wires j and k are connected to

P_1 (holds $\mathcal{C}_f$)		P_2 (holds input $x \in \{0,1\}^l$)
$b_i, b'_i \leftarrow_\$ \{0,1\}^\sigma$		$(\mathsf{pk}, \mathsf{sk}) \leftarrow_\$ \mathsf{KGen}(1^n)$ $(s_i^0) \leftarrow_\$ \{0,1\}^\sigma \ \forall i \in \{1,...,N\}$ where $\sigma = 128$ is our symmetric security parameter $r \leftarrow_\$ \{0,1\}^\sigma$
	$\xleftarrow{\mathsf{pk}, \forall i : \mathsf{Enc}(s_i^0)}$	
$\forall i \in \{1,...,g\}$: *create encrypted garbled gate* $encGG_i$		
	$\xrightarrow{\mathsf{Enc}(s_j^0 + b_i),\ \mathsf{Enc}(s_k^0 + b'_i)}$	
		compute garbled table GG_i $L_i^0 = \mathsf{Dec}(\mathsf{Enc}(s_j^0 + b_i)$ $L_i^1 = L_i^0 + r$ $R_i^0 = \mathsf{Dec}(\mathsf{Enc}(s_k^0 + b'_i)$ $R_i^1 = R_i^0 + r$
		GARBLED TABLE GG_i (entries in random order) $\mathsf{sEnc}_{L_i^0}(\mathsf{sEnc}_{R_i^0}(s_{u+i}^1))$ $\mathsf{sEnc}_{L_i^0}(\mathsf{sEnc}_{R_i^1}(s_{u+i}^1))$ $\mathsf{sEnc}_{L_i^1}(\mathsf{sEnc}_{R_i^0}(s_{u+i}^1))$ $\mathsf{sEnc}_{L_i^1}(\mathsf{sEnc}_{R_i^1}(s_{u+i}^0))$
	$\xleftarrow{GG_i,\ s_1^{x_1}, \ldots, s_l^{x_1}}$	
$L_i = s_j + b_i$ $R_i = s_k + b'_i$ $s_{u+i} = \mathsf{decYao}(L_i, R_i, GG_i)$ *once all output wires have been determined:*		
	$\xrightarrow{s_{N-o+1}, \ldots, s_N}$	
		compare $s_{N-o+1}, \ldots, s_N$ *to* $[s_{N-o+1}^0, s_{N-o+1}^1], \ldots, [s_N^0, s_N^1]$ to receive the output bits

Figure 4.4: The [KM11] protocol (improved version)

the incoming wires of gate G_i (as depicted in Figure 4.1), then P_1 constructs the encrypted garbled gate encGG_i as

$$\mathsf{encGG}_i = \begin{pmatrix} \mathsf{Enc}(s_j^0 + b_i) \\ \mathsf{Enc}(s_k^0 + b_i') \end{pmatrix} \tag{4.6}$$

by making use of the additively homomorphic property of the homomorphic encryption scheme. After the creation of the encrypted garbled gates, P_1 sends $\mathsf{encGG}_1, \dots, \mathsf{encGG}_{g+u}$ to P_2.

P_2 is now able to decrypt the encrypted garbled gates and receives $L_i^0 = s_j^0 + b_i$ and $R_i^0 = s_k^0 + b_i'$ without learning b_i and b_i'. Particularly, P_2 cannot distinguish the L_i^0 and R_i^0 from a random bit string and therefore does not learn anything about the wiring of the circuit. P_2 obtains the blinded versions of the wire keys s_j^1 and s_k^1 by defining $L_i^1 = L_i^0 + r$ and $R_i^1 = R_i^0 + r$. It can now create the garbled gate GG_i using the blinded wire keys $L_i^0, L_i^1, R_i^0, R_i^1$ and the outgoing wires s_{u+i}^0 and s_{u+i}^1 of gate G_i.

The protocol continues as in the standard version. After receiving the garbled gates, P_1 uses the wire keys $s_j \in \{s_j^0, s_j^1\}$ and $s_k \in \{s_k^0, s_k^1\}$ and the blinding values b_i, b_i' for gate G_i to calculate $L_i = s_j + b_i$ and $R_i = s_k + b_i'$. P_1 can now decrypt the garbled gate GG_i to learn $s_{u+i} = \mathsf{decYao}(L_i, R_i, \mathsf{GG}_i)$.

The authors of [KM11] theoretically analyze that this improved version of the protocol is roughly twice as efficient as the original protocol. Our practical performance evaluation in Section 6.1 substantiates this claim.

4.3 Further Optimizations of [KM11]

In this section, we introduce two further optimizations to the [KM11] protocol, one that shifts computational effort from the online phase to an offline phase (Section 4.3.1) and one that describes the application of pipelining techniques to the [KM11] protocol. Together with the BFV-specific improvements introduced in section 4.5, we thereby achieve the best practical computational performance for a PFE protocol starting from ca. 100 000 gates.

4.3.1 Precomputation of the Encrypted Wire Keys

We observe that the creation of the *encrypted wire keys* $\mathsf{Enc}(s_i^0)$, where $1 \leq i \leq N$, by party P_2 can be done in an offline phase before the start of the protocol. The protocol has no special requirement for the construction of the wire keys except that they should be sampled randomly. In particular, no wire key depends on the inputs x to the PFE protocol nor on the circuit $\mathcal{C}_f$ that is being evaluated. Since encryption is a relatively expensive operation, this drastically reduces the protocol runtime (see Section 6.1). Also, the encryption of the wire keys is done only by the party generating the public/private key pair and therefore no communication needs to take place before precomputation of the encrypted wire keys.

Also, the blinding values b_i, b'_i, where $1 \le i \le g$, generated by party P_1 can be prepared before the actual protocol run begins. Since the blinding values for the encrypted garbled gates need to be encrypted before they can be applied to the wire keys, these encrypted values can also be precomputed beforehand. Here, it is necessary to exchange the public key of the underlying homomorphic encryption scheme first. As with the wire keys, the blinding values do not depend on the inputs x nor the circuit $\mathcal{C}_f$ used in the PFE protocol. We argue that the exchange of a public key prior to the protocol run is a feasible practice and divide the protocol in an offline phase where the wire keys and the blinding values are encrypted and an online phase where the encrypted wire keys are sent to P_1 and the other steps of the protocol are executed.

4.3.2 GC Pipelining

While in UC-based PFE the universal circuit is known to both parties before the start of the protocol and can therefore be garbled in an offline phase, in [KM11] the garbling depends on the private circuit $\mathcal{C}_f$ only known by P_1. Thus, pipelining in the online phase gives benefits here: Decryption of the *encrypted garbled gates* is the most expensive operation in the online phase of the protocol. Since they are created and evaluated in the same (topological) order, P_1 can send each *encrypted garbled gate* directly when it has been created and P_2 can start decryption for this gate without delay. After receiving the encrypted garbled gate for gate i, P_2 has all the information it needs to create the garbled table for gate i. In our experiments, we saw that pipelining reduces the runtime in the online phase by about 25%.

4.4 Instantiating [KM11] with EC ElGamal

Katz and Malka suggest to use standard ElGamal encryption to instantiate their protocol. We implemented the protocol using elliptic curve ElGamal instead for performance reasons. Since the only requirement for the choice of the wire keys in the [KM11] protocol is indistinguishability, we can choose random points equally distributed on the elliptic curve as plaintext values and do not need to define a mapping from arbitrary-formed values (i.e arbitrary bit strings) to elements on the elliptic curve and vice versa. In our implementation, this is done by multiplying a randomly chosen scalar value k with the base point P on the elliptic curve. Since P_1 needs to apply the blinding value to the plaintext wire keys when determining the symmetric encryption keys of the garbled tables, we also choose the blinding value as a point on the elliptic curve and perform the addition of a wire key and the blinding value using the ECC arithmetic over the Galois field.

In the following, we denote integers by lowercase letters and points on the elliptic curve by capital letters. The equivalent of choosing a random element of the residue field as the private key in standard ElGamal encryption is choosing a random integer a from the Galois field $GF(p)$ as the private key in the elliptic curve version. The public key A is then computed as $A = a * P$ where P is the base point of the elliptic curve.

Analogous to standard ElGamal, we define encryption of a message M, where M is a point on the elliptic curve, as follows:

$$\mathsf{Enc}(M) = (K, C) = (k * P, k * A + M). \tag{4.7}$$

Decryption of the ciphertext (C_0, C_1) can now be done as follows:

$$\begin{aligned} \mathsf{Dec}(K, C) &= C - a * K \\ &= k * A + M - a * k * P \\ &= k * a * P + M - a * k * P \\ &= M. \end{aligned} \tag{4.8}$$

Our elliptic curve variant of the ElGamal algorithm needs to provide the additively homomorphic property. Therefore we need to define a way to homomorphically add two ciphertexts and verify that the decryption of the resulting ciphertext indeed represents the addition of the underlying plaintexts. More formally, the cryptosystem is homomorphic if the encryption and decryption functions and a binary operation $\oplus$ satisfy the equation:

$$\mathsf{Dec}(\mathsf{Enc}(M_1) \oplus \mathsf{Enc}(M_2)) = M_1 + M_2. \tag{4.9}$$

Since we only use points on the elliptic curve as our plaintext, addition of the two plaintexts M_1 and M_2 can simply be performed in the ECC arithmetic. We define the homomorphic addition of two ciphertexts as

$$\mathsf{Enc}(M_1) \oplus \mathsf{Enc}(M_2) = (K_1, C_1) \oplus (K_2, C_2) = (K_1 + K_2, C_1 + C_2) \tag{4.10}$$

and show that this satisfies Equation (4.9):

$$\begin{aligned} \mathsf{Dec}(\mathsf{Enc}(M_1) \oplus \mathsf{Enc}(M_2)) &= \mathsf{Dec}((k_1 * P, k_1 * A + M_1) \oplus (k_2 * P, k_2 * A + M_2)) \\ &= \mathsf{Dec}((k_1 * P + k_2 * P, k_1 * A + M_1 + k_2 * A + M_2)) \\ &= \mathsf{Dec}((k_1 + k_2) * P, (k_1 + k_2) * A + M_1 + M_2)) \\ &= (k_1 + k_2) * A + M_1 + M_2 - a * (k_1 + k_2) * P \\ &= M_1 + M_2. \end{aligned} \tag{4.11}$$

4.5 Instantiating [KM11] with BFV

In 2011, Katz and Malka [KM11] proposed a PFE scheme based on additively homomorphic encryption, and suggested an instantiation based on the ElGamal cryptosystem [Elg85] or the Paillier cryptosystem [Pai99], which were the most efficient HE schemes available then. Since then, significant improvements have been made in the area of Ring-LWE (RLWE) based homomorphic encryption, which substantially improved over Paillier and ElGamal for many applications. Thus, we revise the protocol from [KM11] with an HE instantiation based on these efficient Ring-LWE HE schemes. We specifically use the Brakerski/Fan-Vercauteren scheme (cf. Section 2.3.1) as implemented in Microsoft's SEAL library [Sea19] to instantiate the [KM11] protocol.

In 2012, Fan and Vercauteren introduced this homomorphic encryption scheme [FV12], a variant of Brakerski's scale-invariant scheme [Bra12]. We will refer to the scheme as the *BFV scheme*. While Brakerski's scheme is based on the Learning With Errors (LWE) problem [Reg05], Fan and Vercauteren ported the approach to the Ring-Learning With Errors (RLWE) setting [LPR10].

In the BFV encryption scheme, the plaintext space is the polynomial quotient ring $R_t = \mathbb{Z}_t[x]/(x^n + 1)$ of polynomials of degree less than n with coefficients reduced modulo t (see Section 2.3.1). Plaintexts are encoded as polynomials of that ring. Ciphertexts are vectors of at least two polynomials in R_q. Following the terminology of [Sea19], we refer to n as the *polynomial modulus degree*, to t as the *plaintext modulus* and to q as the *coefficient modulus*.

We take the plaintext modulus as $t = 2$, which results in the smallest possible polynomial modulus degree, and consequently ciphertext size in our scenario. The coefficient modulus q is chosen as a product of primes $q_1 = 12\,289$ and $q_2 = 1\,099\,510\,054\,913$. q_1 is the smallest prime that is large enough to allow homomorphic blinding of the key values and satisfies $q_1 \equiv 1 \bmod 2n$. For function privacy, which is necessary to prevent P_2 from learning the permutation of the keys employed by P_1, we flood the ciphertext with noise (cf. [Lai17], Section 9.4) that is 40-bits larger than the noise of the output ciphertext, ensuring a statistical security of 40-bits against P_2. Thus, we require an additional 40-bits (in the form of q_2) in the coefficient modulus to contain the extra noise. Consequently, we choose $n = 2048$ as the polynomial modulus degree, which is the smallest n that maintains a security of 128-bits for a q of 54-bits (cf. [Lai17], Table 3).

4.5.1 Encoding of the Wire Keys

In our instantiation of the BFV scheme, each bit of the plaintext value is then encoded as one coefficient of the polynomial. Our encoding is as follows: Say, we have a wire key v with a binary representation of $v = v_{127}||v_{126}||\dots||v_0$. We then define our plaintext polynomial as $v_{127}x^{127} + \dots + v_1x + v_0$.

Our encoding is similar to the integer encoder provided by SEAL with integer base $B = 2$. However, our implementation directly encodes plaintexts from a buffer instead of using SEAL's `BigUInt` class for performance reasons. Since we use a plaintext modulus of $t = 2$ and homomorphic addition is done coefficient-wise in the BFV scheme, addition becomes equivalent to a *homomorphic XOR operation*.

Due to the special requirement that each wire key has to be utilized separately when creating the encrypted garbled gates, CRT batching, as provided by SEAL, becomes inefficient for our use case. Using batching, one can pack n integers modulo t into one plaintext polynomial and apply SIMD (Single Instruction, Multiple Data) operations on those values. However this would require us to select a much larger value for t. A multiplication operation (by a one-hot encoded vector), that is needed to extract one wire key from the ciphertext containing n wire keys, is less efficient than encrypting and decrypting a smaller ciphertext on its own. We therefore decided against CRT batching.

4.5.2 Efficient Packing of the Ciphertexts

The encoding of the wire keys described in Section 4.5.1 uses exactly 128 coefficients of the BFV ciphertext. Since the degree of the polynomial modulus (`poly_modulus_degree`) is set to 2048, we only use $\frac{1}{16}$ of the coefficients of each ciphertext. Even though we decided not to use CRT batching, utilising the unused coefficients for packing additional 15 wire keys in a ciphertext seems desirable in order to reduce the communication of the protocol by a factor of 16.

There are two types of BFV ciphertexts transferred during one protocol run: First, the encrypted wire keys are sent from P_2 to P_1, and second, the encrypted garbled gates are sent back from P_1 to P_2 after blinding of the wire keys (see Figure 4.2). Both types have different requirements to enable packing multiple wire keys into one ciphertext and in fact, packing is only possible for the second type of ciphertexts. The two main challenges for packing are (i) the combination of multiple wire keys into one ciphertext and (ii) the extraction of a single wire key from the packed ciphertext containing multiple wire keys.

In the first case, where the wire keys are encrypted by P_2 and sent to P_1, party P_1 need to utilize the wire keys independently from another when creating the encrypted garbled gates. Therefore, P_1 needs to be able to split the wire keys, which are packed into one ciphertext, upon receiving them. However, since P_1 has no access to the secret key of the encryption scheme, it cannot simply decrypt the ciphertexts to tell the wire keys apart. Unfortunately, it is not possible for P_1 to homomorphically (without access to the secret key) extract a subset of coefficients of the underlying plaintext, and thus a wire key. To the best of our knowledge, it is not possible to apply any operation other than decryption to a ciphertext in order to extract only one of the wire keys from the ciphertext. Therefore, multiple wire keys can only be packed in a response to P_2 holding the secret key.

In the second case, we need to combine multiple encrypted wire keys into one ciphertext and party P_2, that has access to the secret key, needs to tell the wire keys apart. Traditionally, each of the encrypted garbled gates consists of two ciphertexts, holding the blinded wire keys for the two incoming wires of that gate. First, we describe a way to combine the encrypted wire keys, $\mathsf{Enc}(s_j)$ and $\mathsf{Enc}(s_k)$, into one ciphertext $\mathsf{Enc}(s_j||s_k)$. All of the required operations to do so have to be performed under homomorphic encryption. Since in the plaintexts, the wire keys of length 128 bits are followed by 15×128 coefficients set to zero, we can use these coefficients to encode further wire keys. This can be achieved by applying a "homomorphic (right) bit shift" of 128 bits (respectively coefficients) to one of the wire keys (by multiplying a ciphertext by 2^{128}) and adding both wire keys afterwards. Addition of two ciphertexts is natively provided by the BFV scheme. The homomorphic bit shift can be achieved by homomorphically multiplying a ciphertext by 2^{128}.

In Figure 4.5, we illustrate packing with two wire keys of length 4 bits (for simplicity). All operations are applied to plaintext values here. By homomorphically applying the same operations to their equivalent ciphertexts, two wire keys are packed into a single ciphertext.

$$\begin{aligned} s_j \text{ plain} &= \texttt{1011 0000 0000 0000} \ldots \\ s_k \text{ plain} \times 2^4 &= \texttt{0000 0111 0000 0000} \ldots \\ \hline s_j \text{ plain} + s_k \text{ plain} \times 2^4 &= \texttt{1011 0111 0000 0000} \ldots \end{aligned}$$

Figure 4.5: Packing of two wire keys into one ciphertext

We now have combined two wire keys for incoming wires s_j, s_k of gate G_i into one ciphertext. These wire keys still have to be blinded to form the encrypted garbled gate encGG_i. Since both wire keys reside in one ciphertext, this can now be achieved by only one homomorphic addition. Therefore, we concatenate the blinding values b_j and b_k and homomorphically add them to $\mathsf{Enc}(s_j||s_k)$ to receive the encrypted garbled gate $\mathsf{encGG}_i = \mathsf{Enc}(s_j||s_k) + (b_j||b_k)$.

Since P_2 is in charge of telling the wire keys apart, extraction of a single wire key can be solved simply by decrypting the ciphertext and assign the first 128 bits to the first and the next 128 bits to the second wire key.

Analogously, we can pack additional encrypted garbled gates into the same ciphertext and thereby use all 2048 coefficients to pack 8 encrypted garbled gates. This can be done efficiently using Horner's method as described in [KSS13]. When packing the ith of the 8 garbled gates into the ciphertext, we multiply (the ciphertexts of) the left and right incoming wire key by $2^{(2\times i)\times 128}$ and $2^{(2\times i+1)\times 128}$ and add them to the same ciphertext. Blinding of the wire keys can now be applied for 16 wire keys in a single step by concatenating the 16 blinding values and add them to the ciphertext in a single homomorphic addition.

Compared to not using this packing technique, we require the same number of homomorphic additions (15 additions to pack the 16 wire keys + 1 addition for the combined blinding value instead of one addition of a blinding value per wire key) and 15 multiplications by 2^{256}, but we also eliminated 15 decryptions since P_2 only receives one ciphertext instead of 16. Since for our instantiation of the BFV protocol, decryption is more expensive than homomorphic scalar multiplication, this also improves computation.

4.5.3 Wire Key Generation using Seed Expansion

Since the wire keys are encrypted by the key owner P_2, they could be symmetrically encrypted using the secret key to have smaller noise from encryption. Additionally, symmetric encryption also allows some savings in ciphertext size. In symmetric encryption, the second element of the ciphertext is a uniformly random element from R_q. Using a pseudo-random function, one could sample this element by expanding seed, which is sent to P_1 in place of the second element. This optimization halves the ciphertext size of the encrypted wire keys, which is very significant given that communication in this step of the protocol is the major bottleneck of the scheme.

While the BFV scheme supports encryption using the secret key, this is not implemented in SEAL. We therefore give theoretical numbers of this optimization in Figure 6.1 and defer the implementation to future work.

5 Implementation

We have implemented the optimized [KM11] protocol described in Section 4.1, instantiated with different homomorphic encryption schemes using the ABY secure computation framework [DSZ15]. We thereby not only extend the framework to the PFE setting but also provide, to the best of our knowledge, the first implementation of a linear-complexity PFE scheme. Using the ABY framework gives us direct comparability with the results from the efficient UC-based PFE implementation from [AGKS19] that uses ABY to evaluate today's most efficient universal circuits for PFE with complexity $\Theta(n \log n)$.

We give an introduction into the ABY framework in Section 5.1, explain insides about our implementation in Section 5.2 and go into details of the implemented cryptosystems in Sections 5.2.1 (DJN), Section 5.2.2 (EC ElGamal) and Section 5.2.3 (BFV).

5.1 The ABY Framework

To implement the PFE scheme described in Section 4 and conduct our performance measurements, we use the *ABY* framework [DSZ15]. ABY combines secure computation schemes based on Arithmetic sharing, Boolean sharing, and Yao sharing. The framework abstracts from the underlying secure computation protocol and allows to efficiently convert between the three types of sharings. ABY provides many important primitives, such as Yao's garbled circuit technique and oblivious transfer, required to implement the protocol by Katz and Malka [KM11].

ABY is also able to evaluate universal circuits which gives us great comparability of the performance measurements from Section 6 to highly optimized UC-based PFE approaches.

5.2 Implementation of KM11-PFE

Our implementation of the [KM11] protocol bases on the Yao sharing implementation in ABY [DSZ15]. The framework defines two classes that implement Yao's garbled circuit protocol for the two participating parties, a *server* and a *client*. In ABY, the *server* acts as the *circuit garbler* and the client acts as the *circuit evaluator*. In the [KM11] protocol, party P_1 evaluates the garbled tables and party P_2 garbles the gates (see Figure 4.4). When adding the [KM11]-specific hiding of the circuit topology before the creation and evaluation of the garbled tables as in Yao's original protocol, it follows that party P_1 holding the circuit $\mathcal{C}_f$ becomes the *client* that is able to evaluate the garbled tables by making use of the information about the circuit's topology it holds. Party P_2 becomes the *server* that holds the private input x, creates the random wire keys and garbles the circuit.

Throughout the implementation, we instantiate all primitives with 128 bit security, i.e. all wire keys (and the blinding values) are of length $\sigma = 128$ bit and we use `AES-128` in Cipher Block Chaining (CBC) mode and choose the instantiation of the homomorphic encryption algorithms accordingly (see below).

In order to meet the requirements for the additional overhead of the [KM11] protocol compared to standard secure computation, several new methods have been introduced in the implementation of the Yao sharing. The server and client part of the protocol reside in the `YaoServerSharing` and `YaoClientSharing` classes located in `src/abycore/sharing/`.

In Listing 5.1, the added methods to the `YaoServerSharing` class are given. The creation and encryption of the wire keys by P_2 is done in the `CreateEncryptedWireKeys` method. This can be done during an offline phase prior to the protocol run. A helper function `AddGlobalRandomShift` is used by the DJN instantiation to add the global random shift r to a wire key (e.g., required to calculate $L_i^1 = L_i^0 + r$) where the wire key is given as a pointer to a block of memory. This method is not used by the BFV and ElGamal instantiation since for BFV, the addition is equivalent to a simple `XOR` operation and for ElGamal encryption, the operation is performed on point objects in the semantics of the elliptic curve. The last method, added to the `YaoServerSharing` class is the `EvaluateKM11Gate` method which decrypts the encrypted garbled gates and creates the garbled table for each gate (formalized by the `encYao` method in Section 4).

The methods added to the `YaoClientSharing` class are given in Listing 5.2 and implement the functionality of party P_1. First, the blinding values are created using the `CreateBlinding-Values` method. Precomputation (encryption) of the blinding values is done in the `Pre-computeBlindingValues` method. In contrast to the `YaoServerSharing` class, were the wire keys are created and encrypted in one method, there are two seperate methods in the `YaoClientSharing` class since precomputation of the blinding values depends on the availability of the public key of the chosen homomorphic cryptosystem. The creation of the encrypted garbled gates based on the encrypted wire keys is done in method `CreateEnc-GarbledGates`. The decryption of the garbled tables (formalized by the `decYao` method in Section 4) using the blinded versions of the received respectively already decrypted wire keys is handled by the `EvaluateKM11Gate` method.

Next to the methods added to the `YaoClientSharing` and `YaoServerSharing` implementation, the individual steps that form the whole protocol are put together in the `Prepare-SetupPhase` and `PerformSetupPhase` methods of the client and server implementation. The decryption of the garbled gates in the BFV instantiation, where one ciphertext contains several encrypted garbled gates at once, is coordinated in the `CreateAndSendGarbledCircuit` method and the transmission of the input wire keys of the client using oblivious transfer (OT) is done in the `FinishCircuitLayer` (server) and `AssignClientInputKeys` (client) methods.

Listing 5.1: New methods in the `YaoServerSharing` class

```
/**
 Create the encrypted wire keys for each wire that will be
 sent to the client (KM11)
 **/
void CreateEncryptedWireKeys();
/**
 Add global random shift r to a wire key buffer (KM11, only
 used for DJN homomorphic encryption)
 \param keyout  buffer to which the shifted wire key will
                be written
 \param keyin   buffer holding the wire key to be shifted
 **/
void AddGlobalRandomShift(BYTE* keyout, BYTE* keyin);
/**
 Method for evaluating a KM11 gate (all gate types) for the
 given gateid. Decrypts the received encrypted garbled gate
 and creates the garbled table for the gate to be sent to
 the client.
 \param gateid  Gate Identifier
 \param setup   ABYSetup Object
 **/
void EvaluateKM11Gate(uint32_t gateid, ABYSetup* setup);
```

Listing 5.2: New methods in the `YaoClientSharing` class

```
/**
 Method for creating the blinding values used to blind the
 encrypted wire keys when forming the encrypted garbled
 gates (KM11)
 **/
void CreateBlindingValues();
/**
 Method for encrypting the blinding values to speedup the
 creation of the encrypted garbled gates in the online
 phase (KM11)
 **/
void PrecomputeBlindingValues();
/**
 Method for evaluating a KM11 gate for the given gateid.
 This method decrypts the garbled table for the given gate
 and assigns the resulting wirekey to the gates outkey
 ("decYao" from KM11 paper).
 \param gateid  Gate Identifier
 **/
void EvaluateKM11Gate(uint32_t gateid);
```

```
/**
 Method for creating all encrypted garbled gates that will
 be sent to the server using homomorphic encryption (KM11)
 \param setup   ABYSetup Object
 **/
void CreateEncGarbledGates(ABYSetup* setup);
```

Private function evaluation can be enabled and configured by uncommenting the according C preprocessor macros in `src/abycore/sharing/yaosharing.h`, as shown in Listing 5.3. When the `KM11_GARBLING` macro is defined, the [KM11] protocol is used to evaluate the circuit defined in the usual way. One can select the instantiation of the homomorphic cryptosystem used to encrypt the wire keys and build the encrypted garbled gates by setting one of the three available values for the `KM11_CRYPTOSYSTEM` macro. For DJN (Paillier) encryption, it is possible to choose between the original, non-improved variant of the [KM11] protocol or the more efficient variant that creates only one wire key per outgoing wire and derives the second wire key using the global random shift r. The latter is used when the `KM11_IMPROVED` macro is enabled. This macro only has an effect when DJN encryption is used. For BFV and EC ElGamal, only the more efficient variant is implemented. The same holds true for the `KM11_PRECOMPUTEB` option. To improve the performance in the online phase, the encrypted versions of the blinding values b and b' are precomputed in the offline phase by party P_1. This can be disabled by uncommenting the definition of the `KM11_PRECOMPUTEB` macro for DJN encryption only. Pipelining the sending of the encrypted garbled gates and the creation of the garbled tables, as described in Section 4.3.2, can be enabled using the `KM11_PIPELINING` macro.

Listing 5.3: `YaoClientSharing` class

```
// enable KM11 garbling (private function evaluation. Please
// note: In ABY, the client holds the circuit and the server
// acts as the garbler)
#define KM11_GARBLING

// choose between DJN/Paillier, RLWE-based (BFV) or ECC-based
// (EC ElGamal) encryption
#define KM11_CRYPTOSYSTEM_DJN 1
#define KM11_CRYPTOSYSTEM_BFV 2
#define KM11_CRYPTOSYSTEM_ECC 3
#define KM11_CRYPTOSYSTEM KM11_CRYPTOSYSTEM_DJN

// enable improved variant of KM11
// (see section 3.2 "A More Efficient Variant" in KM11 paper)
// With this optimization turned on, the wire key representing
// the value '1' is derived from the wire key representing '0'
// using the global random shift r.
// BFV and ECC cryptosystems only implement the improved variant
// of KM11.
#define KM11_IMPROVED
```

```

// enable precomputation (encryption) of blinding value b (this
// cannot be disabled for BFV and ECC cryptosystems)
#define KM11_PRECOMPUTEB

// enable pipelined sending of the encrypted garbled gates
#define KM11_PIPELINING
```

Debug information, such as intermediate values, can be shown by enabling the `DEBUGYAOCLIENT` and `DEBUGYAOSERVER` macros in `yaoclientsharing.h` and `yaoserversharing.h` respectively.

5.2.1 Damgaard Jurik Nielsen Cryptosystem

In their paper Katz and Malka [KM11] suggest to use Paillier encryption [Pai99] to instantiate their protocol. The first instantiation of the protocol therefore bases on a generalization of the Paillier cryptosystem, the Damgaard Jurik Nielsen cryptosystem [DJN10]. The DJN cryptosystem offers a fixed base encryption that improves performance over Paillier's scheme by a factor of 4.

5.2.2 Elliptic Curve ElGamal

For the Elliptic Curve-based instantiation of the [KM11] protocol, we make use of the Koblitz-283 curve from the FIPS 186-2 standard [NIST00] already implemented in ABY. As we will show later in Section 6.1, the elliptic curve ElGamal instantiation offers the lowest communication and improves in communication over the most efficient UC-based PFE implementation [AGKS19] by a factor of ~ 8. The elliptic curve arithmetic is implemented using the MIRACL cryptographic library for elliptic curve cryptography. [1] The instantiation of the curve is shown in Listing 5.4.

Listing 5.4: Initialization of the Koblitz-283 curve

```
// initialize MIRACL with Koblitz curve from FIPS 186-2 standard
fparams->m = 283;
fparams->a = 12;
fparams->b = 7;
fparams->c = 5;
*fparams->BA = 0;
*fparams->BB = 1;
ecurve2_init(fparams->m, fparams->a, fparams->b, fparams->c,
             fparams->BA->getbig(), fparams->BB->getbig(), false, MR_BEST);
```

[1] https://github.com/miracl/MIRACL/

5.2.3 Brakerski/Fan-Vercauteren Cryptosystem

We instantiate the [KM11] protocol using the SEAL library [Sea19] with the parameters given in Section 4.5, as shown in Listing 5.5.

Listing 5.5: Initialization of the context and key generation for the BFV scheme

```
const size_t m_nBFVpolyModulusDegree = 2048;
seal::SmallModulus m_nBFVplainModulus = seal::SmallModulus(2);
std::vector<seal::SmallModulus> m_nBFVCoeffModulus = {seal::SmallModulus(12289),
    seal::SmallModulus(1099511590913)};

seal::EncryptionParameters parms(seal::scheme_type::BFV);
parms.set_poly_modulus_degree(m_nBFVpolyModulusDegree);
parms.set_plain_modulus(m_nBFVplainModulus);
parms.set_coeff_modulus(m_nBFVCoeffModulus);
m_nWirekeySEALcontext = seal::SEALContext::Create(parms);

seal::KeyGenerator keygen(m_nWirekeySEALcontext);
m_nWirekeySEALpublicKey = keygen.public_key();
m_nWirekeySEALsecretKey = keygen.secret_key();
m_nSEALdecryptor = new seal::Decryptor(m_nWirekeySEALcontext,
    m_nWirekeySEALsecretKey);
```

One special feature of the BFV encryption scheme is that the public key is not needed for the homomorphic addition of a cleartext value to an encrypted value. We therefore do not transfer it in the first step of the protocol.

Since SEAL comes with a non-optimized binary encoding of the ciphertexts. We implemented our own encoding and thereby reduce the size of the BFV ciphertexts by 75%. When looking at the hex dump of a ciphertext produced by the SEAL library in Listing 5.6, one can see a quite inefficient encoding of the elements from the ciphertext space for the specific BFV parameters we chose.

Listing 5.6: Hex dump of a BFV ciphertext as produced by SEAL

```
35 05 00 00 00 00 00 00 dc 1e 00 00 00 00 00 00
f8 26 00 00 00 00 00 00 e6 1d 00 00 00 00 00 00
fd 0e 00 00 00 00 00 00 12 1c 00 00 00 00 00 00
67 24 00 00 00 00 00 00 7b 2a 00 00 00 00 00 00
c2 11 00 00 00 00 00 00 43 09 00 00 00 00 00 00
40 01 00 00 00 00 00 00 d3 1b 00 00 00 00 00 00
ff 2c 00 00 00 00 00 00 bb 1a 00 00 00 00 00 00
bd 1f 00 00 00 00 00 00 6b 13 00 00 00 00 00 00
...
```

SEAL encodes each coefficient of the ciphertext polynomials as a 64bit value regardless of the chosen coefficient modulus q. Since the coefficients are always values modulo q, a large portion of the allocated 64bit remain unused. We therefore implemented a more efficient encoding that used only two byte per coefficient and thereby saves a lot of communication overhead, as shown in Listing 5.7.

Listing 5.7: Hex dump of a BFV ciphertext as produced by our more efficient encoder

```
35 05 dc 1e f8 26 e6 1d fd 0e 12 1c 67 24 7b 2a
c2 11 43 09 40 01 d3 1b ff 2c bb 1a bd 1f 6b 13
...
```

6 Performance Analysis

In this section, we compare our measurements for the different instantiations of the [KM11] protocol and point out bottlenecks and advantages of the different instantiations. We also compare the best previous UC-based PFE implementation of [AGKS19]. The results of our performance tests show that linear-complexity PFE based on homomorphic encryption is on a par with UC-based approaches, is able to outperform those protocols in either computation or communication with acceptable impairment of performance in the other. Because UC-based PFE approaches have touched lower bounds, linear-complexity PFE is a viable alternative that has the potential to entirely supersede these schemes in the future.

We used two identical machines with a physical connection of 10 Gbit/s bandwidth and a round-trip time of 1ms. We refer to this setting as the LAN setting and also did performance tests in a simulated WAN setting with 100 Mbit/s bandwidth and a round-trip time of 100ms. Each machine is equipped with an Intel Core i9-7960X CPU with 2.8 GHz and 128 GB RAM. All measurements are averaged over 10 executions.

In the next section, we compare the homomorphic encryption-based instantiations to recent UC-based PFE protocols. In Section 6.2, we give the total runtime and communication split up by the different phases of the protocol. In Section 6.3, we give concrete numbers for our measurements.

6.1 Comparison of PFE Implementations

In Figure 6.1, we depict the communication of the protocols measured during our practical performance evaluation. In Figure 6.2 and Figure 6.3, we depict the runtime of our implementation compared to the most recent PFE implementation based on UCs from [AGKS19]. Alhassan et al. [AGKS19] evaluated their universal circuit constructions using the same implementation of Yao's garbled circuit protocol as we do. We denote the total runtime by "total" and our total runtime excluding the generation of the wire keys (which can be outsourced to a local offline phase which requires no interaction, see Section 4.3.1) by "online". For completeness, we give concrete numbers for our measurements in Section 6.3. If not stated otherwise, we give relative numbers comparing the different settings for $n = 1\,000\,000$ gates.

Communication. Our measurement results for communication of the different PFE protocols are depicted in Figure 6.1. The elliptic curve ElGamal instantiation clearly outperformes all other implementations, including the well-researched UC-based approaches and thereby offers the best PFE scheme in terms of communication complexity known so far. Its communicational overhead is lower than UC-based PFE by a factor of $\sim 8.8\times$. EC ElGamal-based PFE becomes a valid choice when communicational efficiency is of greater importance than runtime (which is behind UC-based PFE by a factor of $\sim 2.3\times$ in the WAN setting).

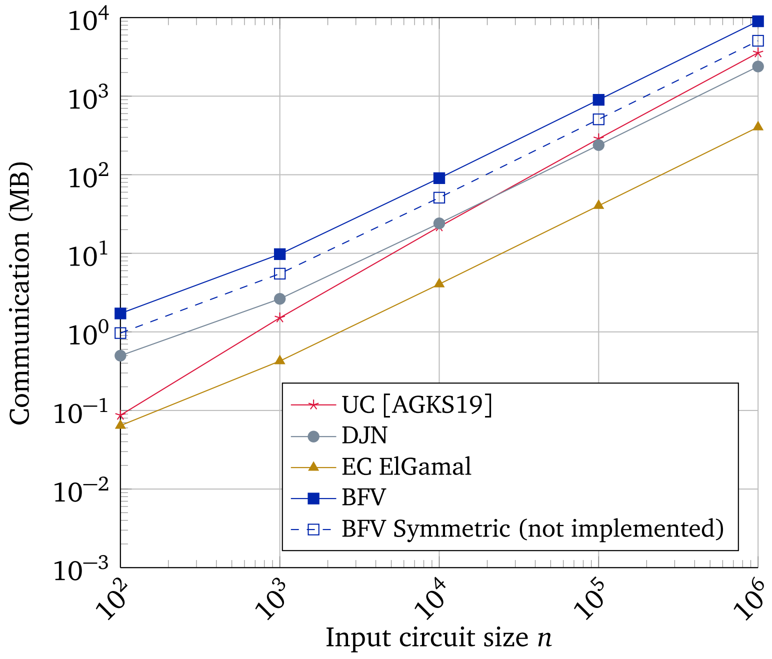


Figure 6.1: Total communication of private function evaluation protocols in megabytes.

We observe that DJN-based PFE communication complexity is on a par with UC-based approaches. Due to its large ciphertext size, BFV-based encryption has worst communication of our instantiations but its communication still is only higher than PFE with universal circuits by a factor of about 2.5×. Its communication could be improved to rank behind UC-based PFE by only a factor of about 1.4× when making use of symmetric encryption, as described in Section 4.5.3. We estimate the expected communication when using symmetric encryption in combination with our PRF optimization. These numbers are estimated based on our practical measurements and are depicted by a dashed line.

LAN runtimes. The practical performance analysis in the LAN setting in Figure 6.2 shows BFV-based private function evaluation leads to outstanding runtime measurements and offers better practical performance than state-of-the-art UC-based PFE starting from $n = 100\,000$ gates. Elliptic curve ElGamal encryption also offers promising runtime even if through is less efficient than BFV-instantiated PFE by a factor of $\sim 12.8\times$ (online runtime) respectively

$\sim 3.9\times$ (total runtime) for $n = 1\,000\,000$ gates. With about 70 minutes of runtime for $n = 100\,000$ gates, DJN-instantiated PFE has impracticable computational overhead.

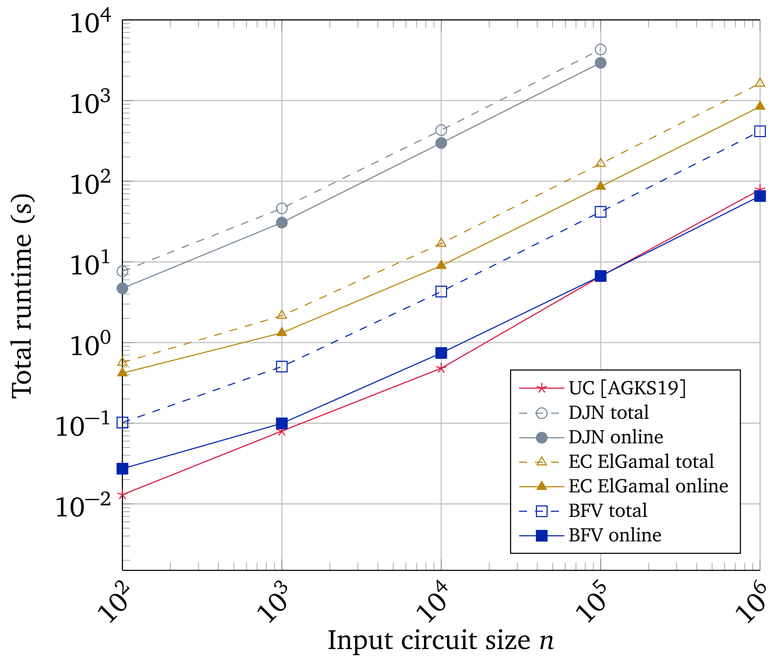


Figure 6.2: Total runtime in seconds of PFE protocols in the LAN setting.

WAN runtimes. In the WAN setting, depicted in Figure 6.3, DJN-based private function evaluation has a very similar runtime and therefore is not practical either (the runtime of DJN-based PFE differs by less than 1% between the LAN and WAN setting). The fact that runtimes are on a very similar level as in the LAN setting shows that the performance of the DJN instantiation is primarily dominated by computation rather than communication. This is also the case for the ECC-based implementation where runtime only differs by about 1% compared to the LAN setting, too. However, elliptic curve ElGamal encryption still offers feasible computational performance and thereby nearly outperforms BFV-based PFE. Its runtime is longer than that of UC-based approaches only by a factor of $\sim 2.3\times$, which is noticeable due to its lowest communication.

The BFV-based instantiation is faster than the ECC approach by about 6% only in the WAN setting. This is remarkable considering its faster runtime by a factor of $\sim 12.8\times$ in the LAN setting compared to the EC ElGamal instantiation. These findings show that communication complexity becomes an important factor in this setting. Here, the computational advances of BFV cannot compensate its larger ciphertext sizes any more.

Even in this more realistic setting for real-world applications, both the BFV-based instantiation and the one using elliptic curve ElGamal, nearly achieve the performance of UC-based PFE

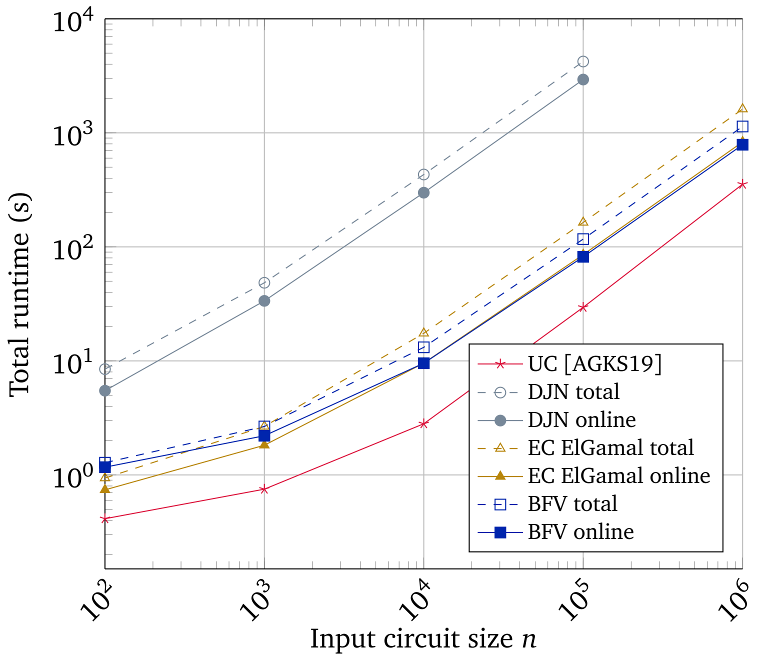


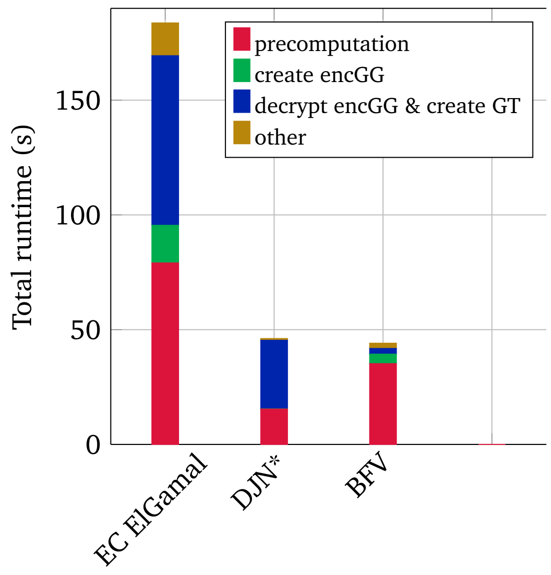
Figure 6.3: Total runtime in seconds of PFE protocols in the WAN setting.

(performance of HE-based PFE is worse than recent UC-based approaches by about a factor of 2 only). While PFE based on homomorphic encryption was believed to be way less efficient than UC-based approaches, we show that this is no longer the case.

Summary. To summarize, we notice that while providing better communication than recent UC-based approaches, DJN-based homomorphic encryption has very high runtime due to its high computation overhead. When precomputation of the encrypted wire keys is shifted to an offline phase, a BFV-based instantiation of the [KM11] protocol outperforms UC-based approaches for circuits of size about $n \geq 100\,000$ gates and has acceptable communication. EC ElGamal-based PFE has extremely small communication overhead and a decent computation overhead. The results of our evaluation also support the claim that homomorphic encryption-based PFE leads to linear-complexity in terms of computation and communication in practice.

6.2 Performance Split by the Different Protocol Phases

In Figure 6.4, we depict the total runtime in the LAN setting split up by the different phases of the protocol and the total communication split up by the different types of data transferred during one protocol run for the EC ElGamal, DJN, and BFV scheme. Since DJN-based PFE has



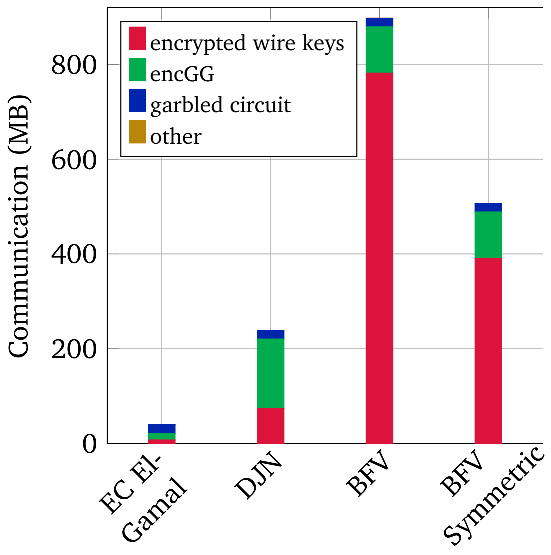


Figure 6.4: Runtime in the LAN setting split up by the different phases of the protocol and total communication split up by the different types of data transferred during one protocol run for the EC ElGamal, DJN, and BFV scheme for $n = 100\,000$ gates each - with the exception that runtime for the DJN-based instantiation is given for $n = 1\,000$ gates only due to its infeasible runtime

extremely high runtime, we depict it for $n = 1\,000$ gates only. Runtimes for the DJN-based instantiation therefore cannot be compared to other schemes in this graph. Here, the protocol is executed without pipelining to tell the computation of the creation of the encrypted garbled gates by party P_1 and their decryption by party P_2 apart.

One can see that both for computation and communication, the first step of encrypting the encrypted garbled gates, that can be shifted to an offline phase, makes up the most costly part of the protocol, except for the EC ElGamal-based instantiation where the decryption of the encrypted garbled gates (decryption operation of the cryptosystem) plays a more important role. The inefficiency of the decryption operation in the EC ElGamal cryptosystem is the main drawback of this scheme and is the main reason for its unfavorable runtime. In contrast to other cryptosystems, the size of the encrypted wire keys is less than the size of the encrypted garbled gates and the garbled tables for EC ElGamal encryption.

In the BFV setting, one could improve the most expensive operation of creating and sending the encrypted wire keys by using the secret key encryption described in Section 4.5.3.

6.3 Concrete Performance Measures of our Implementation

In Table 6.1 we provide the concrete performance measures (averaged over 10 executions) used for depicting the runtime and communication in Figure 6.2, Figure 6.3 and Figure 6.1.

Input circuit size n	100	1 000	10 000	100 000	1 000 000
UC LAN [AGKS19] (s)	0.01	0.08	0.48	6.63	78.73
UC WAN [AGKS19] (s)	0.32	0.75	2.81	29.49	354.32
KM11-DJN [KM11] LAN total (s)	7.65	46.11	430.18	4305.31	-
KM11-DJN [KM11] LAN online (s)	4.70	30.73	297.90	2939.66	-
KM11-DJN [KM11] WAN total (s)	8.47	48.42	431.14	4222.15	-
KM11-DJN [KM11] WAN online (s)	5.48	33.59	298.66	2933.36	-
KM11-ECC (this work) LAN total (s)	0.56	2.16	16.90	186.84	1630.44
KM11-ECC (this work) LAN online (s)	0.42	1.32	8.96	85.80	787.45
KM11-ECC (this work) WAN total (s)	0.94	2.66	17.46	163.67	1614.78
KM11-ECC (this work) WAN online (s)	0.74	1.82	9.56	85.45	832.13
KM11-BFV (this work) LAN total (s)	0.10	0.50	4.29	41.77	416.23
KM11-BFV (this work) LAN online (s)	0.03	0.10	0.74	6.69	65.57
KM11-BFV (this work) WAN total (s)	1.28	2.66	13.16	116.98	1136.99
KM11-BFV (this work) WAN online (s)	1.17	2.21	9.54	81.80	786.41
UC comm. [AGKS19] (MB)	0.09	1.51	21.79	287.51	3562.21
KM11-DJN [KM11] comm. (MB)	0.50	2.63	24.06	238.52	-
KM11-ECC (this work) comm. (MB)	0.06	0.43	4.05	40.35	403.42
KM11-BFV (this work) comm. (MB)	1.72	9.76	90.44	897.89	8973.81
KM11-BFV Sym. (this work) comm. (MB)	0.97	5.51	51.07	507.02	5067.32

Table 6.1: Runtime and communication of PFE with UC and HE

7 Conclusion and Future Work

7.1 Conclusion

By introducing several improvements both on the protocol level and the concrete instantiation of the [KM11] protocol, we achieve highly efficient linear-complexity private function evaluation that can compete with the most efficient PFE protocols which are based on universal circuits. In the last years, a lot of effort has been put into researching RLWE-based encryption schemes and enormous progress has been made in this field. Our practical performance measurements show that HE-based PFE is practical with the latest improvements in the field of ECC and RLWE-based homomorphic encryption. Our BFV instantiation outperforms recent UC-based PFE schemes in runtime. When instantiated with elliptic curve ElGamal encryption, our implementation outperforms UC-based schemes in communication.

Linear-complexity private function evaluation based on homomorphic encryption is heavily dependent on the performance of the chosen cryptosystem and its implementation. The performance of the decryption and homomorphic addition operation is of key importance for the online runtime of the protocol since encryption can be shifted to an offline phase and therefore is not as much performance-critical as the other operations are. Also, the size of the ciphertexts of the homomorphic encryption scheme is a key factor when it comes to practical applicability of the protocol instantiation, especially in environments where the network bandwidth is limited.

When precomputation and pipelining techniques are applied, the BFV instantiation outperforms recent UC-based PFE schemes starting from circuits of size ca. $n = 100\,000$ gates on.

In contrast to BFV encryption, elliptic curve ElGamal encryption offers extremely small ciphertext sizes. When instantiated with elliptic curve ElGamal encryption, our implementation outperforms UC-based schemes for all circuit sizes in communication and thereby achieves the lowest communication of all PFE protocols known to date. At the same time, computational complexity stays on an acceptable level and competes with latest UC-based approaches.

The results of our evaluation also support the claim that homomorphic encryption-based PFE leads to linear-complexity in terms of computation and communication not only in theory, but also in practice. The assumption that PFE based on homomorphic encryption is way less efficient than UC-based approaches clearly does not hold true any more.

7.2 Future Work

This thesis has shown that HE-based PFE is practical when instantiated with modern homomorphic encryption schemes. The findings of this thesis motivate for further research in the field of HE-based private function evaluation.

7.2.1 Implementation of Variants of the Protocol

Due to time constraints, we left the implementation of further variants of the protocol for future work. Implementing the symmetric encryption of the BFV scheme (see Section 4.5.3) to reduce the communication in the BFV instantiation of the protocol and instantiating the PFE protocol with the homomorphic DGK encryption scheme [DGK07; DGK08] and other ECC-based schemes could outperform UC-based PFE not only in either runtime or communication but in both.

7.2.2 Parallelization

The [KM11] protocol is very suitable for parallelization techniques, e.g., the creation of the encrypted wire keys (by P_2) in the first step of the protocol could be perfectly optimized by massive parallelization. Also, the creation of the encrypted garbled gates (by P_1) is perfectly parallelizable since there are no dependencies between the encrypted garbled gates. Additionally, the decryption of the encrypted garbled gates and the creation of the garbled tables (by P_2) could also be parallelized. Only the evaluation of the garbled tables depends on the wire keys retrieved from previous garbled tables and therefore is not fully suitable for parallelization. Implementing these parallelization techniques to speed up the implementation of the protocol was out of scope of this work and would be an interesting direction for further research. Parallelization could lead to lower runtime than UC-based PFE for BFV-based and EC ElGamal-based instantiations of our protocol even in the WAN setting. Especially for the EC ElGamal-based instantiation, parallelization could lead to a huge performance increase. We remember from Section 6.1 that for EC ElGamal-based PFE computation is much more performance-critical than its (small) communication overhead.

List of Figures

List of Tables

List of Abbreviations

SMPC	Secure Multi-Party Computation
MPC	Multi-Party Computation
SFE	Secure Function Evaluation
PFE	Private Function Evaluation
CTH	Circuit Topology Hiding
PGE	Private Gate Evaluation
PF-SFE	Private Function Secure Function Evaluation
UC	Universal Circuit
LWE	Learning With Errors
RLWE	Ring-Learning With Errors
HE	Homomorphic Encryption
BFV	Brakerski/Fan-Vercauteren Homomorphic Encryption
ECC	Elliptic Curve Cryptography
ECDLP	Elliptic Curve Discrete Logarithm Problem
DLP	Discrete Logarithm Problem
ABY	Arithmetic, Boolean, and Yao sharing framework
DJN	Damgaard Jurik Nielsen cryptosystem
OT	Oblivious Transfer

Bibliography

[AF90] M. Abadi, J. Feigenbaum. **“Secure Circuit Evaluation”**. In: *J. Cryptology* 2.1 (1990), pp. 1–12.

[AGKS19] M. Y. Alhassan, D. Günther, Á. Kiss, T. Schneider. **“Efficient and Scalable Universal Circuits.”** In: *IACR Cryptology ePrint Archive* 2019/348 (2019).

[ALSZ13] G. Asharov, Y. Lindell, T. Schneider, M. Zohner. **“More Efficient Oblivious Transfer and Extensions for Faster Secure Computation”**. In: *20. ACM Conference on Computer and Communications Security (CCS’13)*. Full version: `https://ia.cr/2013/552`. Code: `https://encrypto.de/code/OTExtension`. ACM, 2013, pp. 535–548.

[Att14] N. Attrapadung. **“Fully Secure and Succinct Attribute Based Encryption for Circuits from Multi-linear Maps”**. Cryptology ePrint Archive, Report 2014/772. 2014.

[BBKL18a] O. Biçer, M. A. Bingöl, M. S. Kiraz, A. Levi. **“Highly Efficient and Reusable Private Function Evaluation with Linear Complexity”**. Cryptology ePrint Archive, Report 2018/515. `https://eprint.iacr.org/2018/515`. 2018.

[BBKL18b] M. A. Bingöl, O. Biçer, M. S. Kiraz, A. Levi. **“An Efficient 2-Party Private Function Evaluation Protocol Based on Half Gates”**. In: *The Computer Journal* 62.4 (2018), pp. 598–613.

[BDK$^+$18] N. Büscher, D. Demmler, S. Katzenbeisser, D. Kretzmer, T. Schneider. **“HyCC: Compilation of Hybrid Protocols for Practical Secure Computation”**. In: *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*. CCS’18. ACM, 2018, pp. 847–861.

[BGV12] Z. Brakerski, C. Gentry, V. Vaikuntanathan. **“(Leveled) fully homomorphic encryption without bootstrapping”**. In: *Proceedings of the 3rd Innovations in Theoretical Computer ScienceConference*. 2012.

[BHKR13] M. Bellare, V. T. Hoang, S. Keelveedhi, P. Rogaway. **“Efficient Garbling from a Fixed-Key Blockcipher”**. In: *2013 IEEE Symposium on Security and Privacy*. 2013, pp. 478–492.

[BHR12] M. Bellare, V. T. Hoang, P. Rogaway. **“Foundations of Garbled Circuits”**. In: *Proceedings of the 2012 ACM Conference on Computer and Communications Security*. CCS’12. ACM, 2012, pp. 784–796.

[BHWK16] N. Büscher, A. Holzer, A. Weber, S. Katzenbeisser. **“Compiling Low Depth Circuits for Practical Secure Computation”**. In: *Computer Security – ESORICS 2016*. 2016, pp. 80–98.

[BMR90] D. Beaver, S. Micali, P. Rogaway. **“The Round Complexity of Secure Protocols”**. In: *Proceedings of the Twenty-second Annual ACM Symposium on Theory of Computing*. STOC’90. ACM, 1990, pp. 503–513.

[Bra12] Z. Brakerski. **“Fully Homomorphic Encryption without Modulus Switching from Classical GapSVP”**. In: *Advances in Cryptology – CRYPTO 2012*. 2012, pp. 868–886.

[DDK$^+$15] D. Demmler, G. Dessouky, F. Koushanfar, A.-R. Sadeghi, T. Schneider, S. Zeitouni. **“Automated Synthesis of Optimized Circuits for Secure Computation”**. In: *Proceedings of the 22Nd ACM SIGSAC Conference on Computer and Communications Security*. CCS’15. ACM, 2015, pp. 1504–1517.

[DGK07] I. Damgård, M. Geisler, M. Krøigaard. **“Efficient and Secure Comparison for On-Line Auctions”**. In: *Information Security and Privacy*. 2007, pp. 416–430.

[DGK08] I. Damgård, M. Geisler, M. Krøigaard. **“A correction to “Efficient and Secure Comparison for On-Line Auctions””**. Cryptology ePrint Archive, Report 2008/321. 2008.

[DJN10] I. Damgård, M. Jurik, J. B. Nielsen. **“A generalization of Paillier’s public-key system with applications to electronic voting”**. In: *International Journal of Information Security* 9.6 (2010), pp. 371–385.

[DPSZ12] I. Damgård, V. Pastro, N. Smart, S. Zakarias. **“Multiparty Computation from Somewhat Homomorphic Encryption”**. In: *Advances in Cryptology – CRYPTO 2012*. 2012, pp. 643–662.

[DSZ15] D. Demmler, T. Schneider, M. Zohner. **“ABY – A Framework for Efficient Mixed-Protocol Secure Two-Party Computation”**. In: *NDSS’15*. Code: `https://encrypto.de/code/ABY`. The Internet Society, 2015.

[Elg85] T. Elgamal. **“A public key cryptosystem and a signature scheme based on discrete logarithms”**. In: *IEEE Transactions on Information Theory* 31.4 (1985), pp. 469–472.

[FAL06] K. B. Frikken, M. J. Atallah, J. Li. **“Attribute-Based Access Control with Hidden Policies and Hidden Credentials”**. In: *IEEE Transactions on Computers* 55.10 (2006), pp. 1259–1270.

[FAZ05] K. B. Frikken, M. J. Atallah, C. Zhang. **“Privacy-preserving credit checking”**. In: *Electronic Commerce (EC’05)*. 2005, pp. 147–154.

[FGP14] D. Fiore, R. Gennaro, V. Pastro. **“Efficiently Verifiable Computation on Encrypted Data”**. In: *CCS’15*. 2014, pp. 844–855.

[FLA06] K. B. FRIKKEN, J. LI, M. J. ATALLAH. **“Trust Negotiation with Hidden Credentials, Hidden Policies, and Policy Cycles”**. In: *NDSS’06*. The Internet Society, 2006, pp. 157–172.

[FV12] J. FAN, F. VERCAUTEREN. **“Somewhat Practical Fully Homomorphic Encryption.”** In: *IACR Cryptology ePrint Archive* 2012/144 (2012).

[Gen09] C. GENTRY. **“A fully homomorphic encryption scheme”**. PhD thesis. Stanford University, 2009.

[GGHZ14] S. GARG, C. GENTRY, S. HALEVI, M. ZHANDRY. **“Fully Secure Attribute Based Encryption from Multilinear Maps”**. Cryptology ePrint Archive, Report 2014/622. 2014.

[GGPR13] R. GENNARO, C. GENTRY, B. PARNO, M. RAYKOVA. **“Quadratic Span Programs and Succinct NIZKs without PCPs”**. In: *EUROCRYPT’13*. Vol. 7881. 2013, pp. 626–645.

[GHV10] C. GENTRY, S. HALEVI, V. VAIKUNTANATHAN. **“i-Hop Homomorphic Encryption and Rerandomizable Yao Circuits”**. In: *CRYPTO’10*. Vol. 6223. 2010, pp. 155–172.

[GKS17] D. GÜNTHER, Á. KISS, T. SCHNEIDER. **“More Efficient Universal Circuit Constructions”**. In: *23. Advances in Cryptology – ASIACRYPT 2017*. Vol. 10625. LNCS. 2017, pp. 443–470.

[GKSS19] D. GÜNTHER, Á. KISS, L. SCHEIDEL, T. SCHNEIDER. **“Framework for Semi-Private Function Evaluation with Application to Secure Insurance Rate Calculation”**. 26. ACM Conference on Computer and Communications Security (CCS’19) Posters/Demos. 2019.

[GMW87] O. GOLDREICH, S. MICALI, A. WIGDERSON. **“How to Play ANY Mental Game”**. In: *Proceedings of the Nineteenth Annual ACM Symposium on Theory of Computing*. STOC’87. ACM, 1987, pp. 218–229.

[Gol03] O. GOLDREICH. **“Cryptography and Cryptographic Protocols”**. In: *Distrib. Comput.* 16.2-3 (2003), pp. 177–199.

[HFKV12] A. HOLZER, M. FRANZ, S. KATZENBEISSER, H. VEITH. **“Secure Two-party Computations in ANSI C”**. In: *Proceedings of the 2012 ACM Conference on Computer and Communications Security*. CCS’12. ACM, 2012, pp. 772–783.

[HKK$^+$14] Y. HUANG, J. KATZ, V. KOLESNIKOV, R. KUMARESAN, A. J. MALOZEMOFF. **“Amortizing Garbled Circuits”**. In: *CRYPTO’14*. Vol. 8617. LNCS. 2014, pp. 458–475.

[IKNP03] Y. ISHAI, J. KILIAN, K. NISSIM, E. PETRANK. **“Extending Oblivious Transfers Efficiently”**. In: *Advances in Cryptology (CRYPTO’03)*. Vol. 2729. LNCS. 2003, pp. 145–161.

[KM11] J. KATZ, L. MALKA. “Constant-Round Private Function Evaluation with Linear Complexity”. In: *Advances in Cryptology–ASIACRYPT 2011*. 2011.

[Kob87] N. KOBLITZ. **"Elliptic curve cryptosystems"**. In: *Mathematics of computation* 48.177 (1987), pp. 203–209.

[KS08a] V. KOLESNIKOV, T. SCHNEIDER. **"A Practical Universal Circuit Construction and Secure Evaluation of Private Functions"**. In: *12. International Conference on Financial Cryptography and Data Security (FC'08)*. Vol. 5143. LNCS. Code: `https://encrypto.de/code/FairplayPF`. 2008, pp. 83–97.

[KS08b] V. KOLESNIKOV, T. SCHNEIDER. **"Improved garbled circuit: Free XOR gates and applications"**. In: *International Colloquium on Automata, Languages, and Programming*. 2008, pp. 486–498.

[KS16] Á. KISS, T. SCHNEIDER. **"Valiant's Universal Circuit is Practical"**. In: *35. Advances in Cryptology – EUROCRYPT 2016*. Vol. 9665. LNCS. Full version: `https://ia.cr/2016/093`. Code: `https://encrypto.de/code/UC`. 2016, pp. 699–728.

[KSS13] V. KOLESNIKOV, A.-R. SADEGHI, T. SCHNEIDER. **"A Systematic Approach to Practically Efficient General Two-Party Secure Function Evaluation Protocols and their Modular Design"**. In: *Journal of Computer Security (JCS)* 21.2 (2013), pp. 283–315.

[Lai17] K. LAINE. **"Simple Encrypted Arithmetic Library 2.3.1"**. In: *Microsoft Research* (2017).

[LMS16] H. LIPMAA, P. MOHASSEL, S. S. SADEGHIAN. **"Valiant's Universal Circuit: Improvements, Implementation, and Applications."** In: *IACR Cryptology ePrint Archive* 2016/17 (2016).

[LP09] Y. LINDELL, B. PINKAS. **"A Proof of Security of Yao's Protocol for Two-Party Computation"**. In: *Journal of Cryptology* 22 (2009), pp. 161–188.

[LPR10] V. LYUBASHEVSKY, C. PEIKERT, O. REGEV. **"On Ideal Lattices and Learning with Errors over Rings"**. In: *Advances in Cryptology – EUROCRYPT 2010*. 2010, pp. 1–23.

[LR15] Y. LINDELL, B. RIVA. **"Blazing Fast 2PC in the Offline/Online Setting with Security for Malicious Adversaries"**. In: *CCS'15*. ACM, 2015, pp. 579–590.

[Mil86] V. S. MILLER. **"Use of Elliptic Curves in Cryptography"**. In: *Advances in Cryptology — CRYPTO '85 Proceedings*. 1986, pp. 417–426.

[MNPS04] D. MALKHI, N. NISAN, B. PINKAS, Y. SELLA. **"Fairplay – A secure Two-Party Computation System"**. In: *USENIX Security'04*. USENIX, 2004, pp. 287–302.

[MR17] P. MOHASSEL, M. ROSULEK. **"Non-interactive Secure 2PC in the Offline/Online and Batch Settings"**. In: *EUROCRYPT'17*. Vol. 10212. LNCS. 2017, pp. 425–455.

[MS13] P. MOHASSEL, S. SADEGHIAN. **"How to hide circuits in MPC an efficient framework for private function evaluation"**. In: *Annual International Conference on the Theory and Applications of Cryptographic Techniques*. 2013, pp. 557–574.

[MSS14] P. Mohassel, S. Sadeghian, N. P. Smart. **"Actively Secure Private Function Evaluation"**. In: *Advances in Cryptology – ASIACRYPT 2014*. 2014, pp. 486–505.

[NIST00] National Institute of Standards and Technology (NIST). **"Recommended Elliptic Curves for Federal Government Use"**. In: *Digital Signature Standard (DSS)* (2000).

[NPS99] M. Naor, B. Pinkas, R. Sumner. **"Privacy Preserving Auctions and Mechanism Design"**. In: *proceedings of the 1st ACM Conference on Electronic Commerce*. ACM. 1999, pp. 129–139.

[NSMS14] S. Niksefat, B. Sadeghiyan, P. Mohassel, S. Sadeghian. **"ZIDS: A Privacy-Preserving Intrusion Detection System Using Secure Two-Party Computation Protocols"**. In: *The Computer Journal* 57.4 (2014), pp. 494–509.

[Pai99] P. Paillier. **"Public-Key Cryptosystems Based on Composite Degree Residuosity Classes"**. In: *Advances in Cryptology — EUROCRYPT'99*. 1999, pp. 223–238.

[Pin02] B. Pinkas. **"Cryptographic Techniques for Privacy-Preserving Data Mining"**. In: *SIGKDD Explorations* 4.2 (2002), pp. 12–19.

[PSSW09] B. Pinkas, T. Schneider, N. P. Smart, S. C. Williams. **"Secure two-party computation is practical"**. In: *International Conference on the Theory and Application of Cryptology and Information Security*. 2009, pp. 250–267.

[Reg05] O. Regev. **"On Lattices, Learning with Errors, Random Linear Codes, and Cryptography"**. In: *Proceedings of the Thirty-seventh Annual ACM Symposium on Theory of Computing*. STOC'05. ACM, 2005, pp. 84–93.

[Sch08] T. Schneider. **"Practical Secure Function Evaluation"**. MA thesis. Friedrich-Alexander University Erlangen-Nürnberg, Germany, 2008.

[Sea19] **"Microsoft SEAL (release 3.3)"**. `https://github.com/Microsoft/SEAL`. 2019.

[SHS+15] E. M. Songhori, S. U. Hussain, A. Sadeghi, T. Schneider, F. Koushanfar. **"TinyGarble: Highly Compressed and Scalable Sequential Garbled Circuits"**. In: *2015 IEEE Symposium on Security and Privacy*. 2015, pp. 411–428.

[SYY99] T. Sander, A. L. Young, M. Yung. **"Non-Interactive CryptoComputing For NC^1"**. In: *FOCS'99*. 1999, pp. 554–567.

[Val76] L. G. Valiant. **"Universal Circuits (Preliminary Report)"**. In: STOC'76. ACM, 1976, pp. 196–203.

[Yao82] A. C. Yao. **"Protocols for secure computations (extended abstract)"**. In: *FOCS'82*. 1982, pp. 160–164.

[Yao86] A. C.-C. Yao. **"How to Generate and Exchange Secrets"**. In: *Foundations of Computer Science (FOCS'86)*. IEEE, 1986, pp. 162–167.

[ZRE15] S. Zahur, M. Rosulek, D. Evans. **"Two Halves Make a Whole - Reducing Data Transfer in Garbled Circuits Using Half Gates"**. In: *EUROCRYPT'15*. Vol. 9057. LNCS. 2015, pp. 220–250.

[ZYZL19] S. Zhao, Y. Yu, J. Zhang, H. Liu. **"Valiant's Universal Circuits Revisited: an Overall Improvement and a Lower Bound"**. ASIACRYPT'19. To appear. 2019.