Linear-Complexity Private Function Evaluation is Practical

Marco Holz[1](✉), Ágnes Kiss[1], Deevashwer Rathee[2], and Thomas Schneider[1]

[1] ENCRYPTO, Technische Universität Darmstadt, Darmstadt, Germany
{holz,kiss,schneider}@encrypto.cs.tu-darmstadt.de
[2] Department of Computer Science, IIT (BHU) Varanasi, Varanasi, India
deevashwer.student.cse15@iitbhu.ac.in

Abstract. Private function evaluation (PFE) allows to obliviously evaluate a private function on private inputs. PFE has several applications such as privacy-preserving credit checking and user-specific insurance tariffs. Recently, 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 efficient than UC-based approaches.

In this paper, we take another look at the linear-complexity PFE protocol by Katz and Malka (ASIACRYPT'11): We propose several optimizations and split the protocol in different phases that depend on the function and inputs respectively. We show that HE-based PFE is practical when instantiated with state-of-the-art ECC and RLWE-based homomorphic encryption. Our most efficient implementation outperforms the most recent UC-based PFE implementation of Alhassan et al. (JoC'20) in communication for all circuit sizes and in computation starting from circuits of a few thousand gates already.

Keywords: Private function evaluation · Homomorphic encryption · Secure computation

1 Introduction

While computations on a local machine can be secured against malicious eavesdropping, computations that are performed collaboratively on two or more devices typically rely on the trustworthiness of remote systems. This poses a risk to the sensitive data supplied by the participants. Privacy-preserving protocols aim to mitigate these risks by protecting the data using cryptographic approaches such that there is no need for a trusted remote party anymore.

Secure two-party computation (STPC) or secure function evaluation (SFE) protocols allow two parties to jointly compute a function on private data without learning the other party's inputs. Private function evaluation (PFE) extends this setting by also hiding the evaluated function from one of the parties: P_1 inputs a private function f, typically represented by a circuit $\mathcal{C}_f$, and P_2 inputs private data x and learns only $f(x)$ but no additional information on f (except its size).

L. Chen et al. (Eds.): ESORICS 2020, LNCS 12309, pp. 401–420, 2020.
https://doi.org/10.1007/978-3-030-59013-0_20

PFE has diverse applications that require to keep the participants' inputs private and hide the operations applied to these inputs from one of the participants. We describe a few example applications. In a *privacy-preserving intrusion detection system (IDS)* [Nik+14], a server holds a set of zero-day signatures (including regular expressions matching the payload) and is able to check whether sensitive data uploaded to the IDS matches those signatures such that the server learns nothing about the data and the client learns nothing about the signatures. Using PFE, *attribute-based access control* can be enhanced to protect both sensitive credentials and sensitive policies [FAL06]. PFE can be used for *privacy-preserving credit worthiness checking* [FAZ05], disclosing neither the customer's private financial data nor the private criteria of the loaner. In *privacy-preserving car insurance rate calculation* [Gün+19] the privacy-critical customer data, as well as the tariff calculation details remain private.

The most common approach for PFE is to reduce it to classical SFE by securely evaluating a public universal circuit (UC) [Val76,KS08a,KS16,LMS16,GKS17, Alh+20,Zha+19,Liu+20]. This series of works on optimizations and implementations of UCs has shown that UC-based PFE can be practical, but UCs introduce an inevitable logarithmic overhead [Val76]. Katz and Malka [KM11] propose a linear-complexity PFE scheme based on homomorphic encryption (HE) and Yao's garbled circuit protocol. They expect their scheme to be "easier to implement and more efficient (for larger circuits) than approaches relying on universal circuits". However, their scheme has not been implemented yet.

Our Contributions. Our paper takes another look at the linear-complexity PFE protocol by Katz and Malka [KM11]. We split the protocol into several phases so that parts of the protocol can be precomputed knowing, e.g., only the size of the private function or the private function itself. For instance, for a privacy-preserving IDS it is reasonable to precompute any function-dependent part so that the online phase where the client provides its input is fast. We optimize, instantiate, and implement their scheme using three state-of-the-art homomorphic encryption (HE) schemes: Elliptic curve (EC) ElGamal [Elg85], the Brakerski/Fan-Vercauteren (BFV) scheme [FV12], and the cryptosystem by Damgård/Jurik/Nielsen (DJN) [DJN10]. We implement our protocols using the ABY framework [DSZ15] and thereby provide the first implementation of a linear-complexity PFE scheme. Our experiments show that HE-based PFE outperforms today's most efficient UC-based PFE implementation [Alh+20] on the same platform already starting from circuits with only a few thousand gates.

2 Related Work

In this paper, we focus on PFE protocols that provide security against semi-honest adversaries. These can be categorized as follows:

UC-Based PFE. A universal circuit (UC) is a circuit that can be programmed to evaluate any Boolean circuit up to size n by specifying a set of program bits as its input. In recent years, a lot of research was put into optimizing and implementing UC-based PFE, which reduces the task of PFE to standard SFE that relies mostly on symmetric cryptography where the function is the publicly known UC. Valiant [Val76] proposed two recursive UCs with sizes $\sim 5n \log_2 n$ and $\sim 4.75n \log_2 n$ in the size of the simulated circuit n, which are optimal up to a constant factor because any UC must have size at least $\Omega(n \log n)$. Zhao et al. [Zha+19] present a UC with size $\sim 4.5n \log_2 n$. A hybrid UC with size $\sim 4.5n \log_2 n$, combining optimizations from [KS16, GKS17, Zha+19] was implemented in [Alh+20]. The most recent UC from [Liu+20] has size $\sim 3n \log_2 n$. These constructions have reached lower bounds for the most common ways UCs are constructed [Zha+19, Liu+20], so no significant improvements are expected.

OT-Based PFE. Mohassel and Sadeghian introduce an oblivious transfer (OT)-based approach based on the oblivious evaluation of a switching network of size $\Theta(n \log n)$ that hides the topology of the Boolean circuit [MS13]. Bingöl at al. [Bin+18] adapt the half gates optimization [ZRE15] to the OT-based approach of [MS13] and reduce the number of OTs by half. As shown in [Alh+20], the communication of both [MS13] and [Bin+18] is worse than that of UC-based PFE. PFE schemes based on both UCs and switching networks have an inevitable logarithmic overhead.

TEE-Based PFE. Felsen et al. [Fel+19] propose private function evaluation with a different trust assumption and implement PFE using Intel SGX as trusted execution environment (TEE), by evaluating a UC within the SGX enclave.

HE-Based PFE. The protocol by Katz and Malka [KM11] has linear complexity $\mathcal{O}(n)$, but its concrete practicality has not yet been explored. The authors use homomorphic encryption to hide the topology of the circuit $\mathcal{C}_f$ from the party that obliviously garbles the circuit (cf. Sect. 4.1). Mohassel and Sadeghian [MS13] include a linear-complexity protocol in their generic framework for PFE. They optimize the baseline protocol of [KM11], but their protocol is not more efficient than the improved protocol of [KM11] which we use. Mohassel et al. [MSS14] extend the protocol from [KM11, MS13] to security against malicious adversaries using zero-knowledge proofs while maintaining linear complexity. Biçer et al. present a reusable linear-complexity PFE scheme [Biç+18] based on the protocol of [KM11] which is efficient if the same private function f is evaluated multiple times. Their protocol in the first execution has slightly lower total communication, but around a factor four higher online computation than [KM11] (cf. [Biç+18, Table 1]). Later runs of the protocol with the same function are more efficient both in communication and computation than [KM11]. We leave investigating the concrete efficiency of the protocol of [Biç+18] for applications where the same function can be reused as future work.

In our paper, we resurrect the neglected line of research on linear-complexity HE-based PFE protocols and show that the protocol of [KM11] is practical.

3 Preliminaries

In this section, we describe preliminaries to our work from the fields of secure function evaluation (SFE) in Sect. 3.1 and private function evaluation (PFE) in Sect. 3.2, and recapitulate the homomorphic encryption (HE) schemes we use in Sect. 3.3.

3.1 Circuit-Based Secure Function Evaluation

We focus on security against semi-honest (passive) adversaries where all parties are assumed to follow the protocol. This allows for highly efficient protocols and is a starting point for constructing protocols with stronger security guarantees.

In the past, several SFE protocols have been proposed that rely on a circuit representation of the function f which is known to both parties, e.g., Yao's garbled circuit (GC) protocol [Yao82, Yao86, LP09] and the GMW protocol [GMW87]. In Yao's protocol, party P_1, *the garbler*, prepares an encrypted version of the circuit in the form of garbled tables, which are then sent to P_2. The other party P_2, *the evaluator*, evaluates the *garbled circuit* after receiving the keys corresponding to his input wires using oblivious transfers.[1] Oblivious transfer (OT) allows the receiver P_2 to retrieve one of two messages obliviously from the sender P_1 without the receiver learning the other message or the sender learning which message was retrieved. Though OTs require expensive public-key cryptography [IR89], OT extension [Ish+03, Ash+13] allows to perform a large number of OTs more efficiently by extending a few *base OTs* and obtain many oblivious transfers using only symmetric cryptographic operations. Recent optimizations to Yao's GC protocol include *point-and-permute* [BMR90], *free-XOR* [KS08b], *fixed-key AES garbling* [Bel+13], and *half gates* [ZRE15].

3.2 Private Function Evaluation

Private function evaluation (PFE) extends SFE to the case where only one party P_1 inputs a private function f represented by circuit $\mathcal{C}_f$. The protocol must guarantee that P_2 on private input x learns the output $f(x)$ but no other information about the function f whereas P_1 learns nothing.[2] Generally, PFE protocols reveal the size of the circuit $\mathcal{C}_f$ to the participants. 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 for this is that an adversarial party P_1 could reveal the inputs of party P_2 by defining f to leak information about x, e.g., $f(x) = x$.

[1] Even though the gates are encrypted and thus the gates' types can easily be hidden from P_2, P_2 must know the *topology* of the circuit for evaluating the garbled circuit.

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

3.3 Homomorphic Encryption

Homomorphic encryption (HE) schemes allow for computations on encrypted data, i.e., operations performed on the ciphertexts are reflected in the output of decryption as if they were applied directly on the plaintexts.

The protocol of Katz and Malka [KM11] is based on additively homomorphic encryption, i.e., a HE scheme that supports only homomorphic addition. The authors of [KM11] suggest to instantiate their protocol with Paillier [Pai99] or ElGamal [Elg85] HE and mention that their protocol can be improved by using elliptic-curve cryptography (ECC). Since then, several significant improvements on additively HE were published that we consider in our implementation:

DJN. The DJN cryptosystem [DJN10], a generalization of Paillier's scheme [Pai99], has since then been optimized using CRT-based decryption [HMS12]. Our implementation is based on *libpaillier*[3] and uses this optimization.

EC ElGamal. EC ElGamal encryption offers exceptionally small ciphertexts, practical computation and an additive homomorphism over the underlying elliptic curve group. 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]. In our implementation, we use the *RELIC Toolkit* [AG09] for ECC.

BFV. Significant improvements have been made in the area of RLWE-based HE [Reg05,LPR10,Bra12,FV12]. The RLWE-based BFV scheme [FV12,Lai17] is implemented in the Microsoft SEAL library [Sea19], which is among the fastest HE libraries available today. We present a high level overview of the BFV scheme restricted to only the part of its functionality which is relevant for our application. For additional details, see [Lai17]. We note that our discussion also applies to other popular Ring-LWE-based HE schemes such as BGV [BGV12].

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 probabilistic, takes a public key pk and a message $m \in R_t$ as inputs, and outputs a ciphertext $c \in (R_q)^2$. The ciphertext output by Enc has a noise component associated with it 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 $m = \mathsf{Dec}(sk, \mathsf{Enc}(pk, m))$ works 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 functions, 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. Given ciphertexts

[3] http://hms.isi.jhu.edu/acsc/libpaillier/

$c_1 = \mathsf{Enc}(m_1)$ 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. To hide the operations applied to the 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], § 9.4). This requires larger parameters to accommodate the extra noise, and has been taken into account in our parameter selection.

4 Linear-Complexity Private Function Evaluation

In this section, we recapitulate the private function evaluation (PFE) protocol of Katz and Malka [KM11] in Sect. 4.1, introduce further improvements in Sect. 4.2, and propose efficient instantiations using EC ElGamal in Sect. 4.3 and the BFV homomorphic encryption scheme in Sect. 4.4.

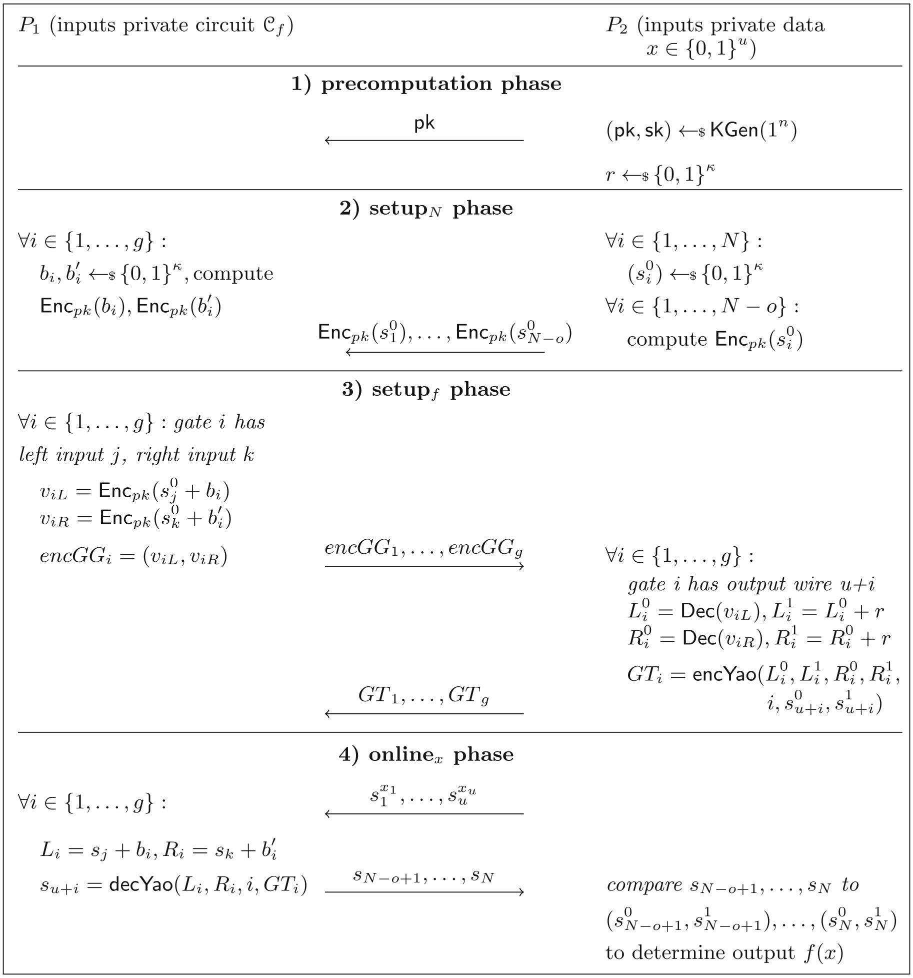
4.1 The [KM11] Protocol

The PFE protocol proposed by Katz and Malka [KM11] combines homomorphic encryption (HE) with Yao's garbled circuit (GC) protocol to hide the topology of the circuit $\mathcal{C}_f$ in addition to the parties' inputs. They give a baseline protocol and a roughly twice as efficient improved protocol. We describe the improved protocol shown in Fig. 1 and refer to the original paper for the baseline version.

The Boolean circuit to be evaluated privately has g gates, u inputs and o outputs and has size $N = u + g$. The circuit is assumed to be built of only two-input NAND gates so that their functionality does not need to be hidden. There exist established highly optimized hardware synthesis tools that optimize for a small number of NAND gates when translating the function to a circuit. Moreover, it is assumed that *"the output wires of the circuit do not connect to any other gates"* [KM11] which is achieved by adding at most o gates to the circuit. [KM11] define the wiring among the gates as follows: Incoming wires are the inputs of the g gates. Outgoing wires are the output wires of the g gates and the u input wires of the circuit. Each incoming wire must be connected to exactly one outgoing wire, but an outgoing wire may be connected to more incoming wires, enabling gates with arbitrary fan-out. In contrast, UC-based PFE requires the fan-out to be at most two which requires additional copy-gates [Val76] that increase the circuit size.

Party P_2 inputs private data x of length $|x| = u$ and acts as the *circuit garbler* from Yao's protocol. P_1 inputs the private circuit $\mathcal{C}_f$ of g gates and acts as the *circuit evaluator*. Since P_2 must remain unaware of the circuit wiring, P_2 cannot directly garble the gates. Instead, P_1 creates a so-called encrypted garbled gate encGG_i for each gate i of the circuit and P_2 decrypts these to learn the keys required to create the garbled tables as in Yao's protocol (cf. Sect. 3.1 and [LP09]). By creating the encrypted garbled gates under HE, 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). Thereby, the circuit topology remains hidden from P_2.

Four Phases of PFE Protocols. We split the protocol of [KM11] and UC-based PFE into four phases: 1) a *precomputation* phase which is run only once, 2) a $setup_N$ phase dependent on the size N of the function, 3) a $setup_f$ phase dependent on the function f, and 4) an $online_x$ phase dependent on the input x.



P_1 (inputs private circuit $\mathcal{C}_f$)		P_2 (inputs private data $x \in \{0,1\}^u$)
	1) precomputation phase	
	$\xleftarrow{\mathsf{pk}}$	$(\mathsf{pk}, \mathsf{sk}) \leftarrow_{\$} \mathsf{KGen}(1^n)$
		$r \leftarrow_{\$} \{0,1\}^{\kappa}$
	2) setup$_N$ phase	
$\forall i \in \{1, \ldots, g\}$:		$\forall i \in \{1, \ldots, N\}$:
$b_i, b'_i \leftarrow_{\$} \{0,1\}^{\kappa}$, compute		$(s_i^0) \leftarrow_{\$} \{0,1\}^{\kappa}$
$\mathsf{Enc}_{pk}(b_i), \mathsf{Enc}_{pk}(b'_i)$		$\forall i \in \{1, \ldots, N-o\}$:
	$\xleftarrow{\mathsf{Enc}_{pk}(s_1^0), \ldots, \mathsf{Enc}_{pk}(s_{N-o}^0)}$	compute $\mathsf{Enc}_{pk}(s_i^0)$
	3) setup$_f$ phase	
$\forall i \in \{1, \ldots, g\}$: *gate i has left input j, right input k*		
$v_{iL} = \mathsf{Enc}_{pk}(s_j^0 + b_i)$ $v_{iR} = \mathsf{Enc}_{pk}(s_k^0 + b'_i)$		
$encGG_i = (v_{iL}, v_{iR})$	$\xrightarrow{encGG_1, \ldots, encGG_g}$	$\forall i \in \{1, \ldots, g\}$:
		gate i has output wire u+i $L_i^0 = \mathsf{Dec}(v_{iL}), L_i^1 = L_i^0 + r$ $R_i^0 = \mathsf{Dec}(v_{iR}), R_i^1 = R_i^0 + r$
	$\xleftarrow{GT_1, \ldots, GT_g}$	$GT_i = \mathsf{encYao}(L_i^0, L_i^1, R_i^0, R_i^1, i, s_{u+i}^0, s_{u+i}^1)$
	4) online$_x$ phase	
$\forall i \in \{1, \ldots, g\}$:	$\xleftarrow{s_1^{x_1}, \ldots, s_u^{x_u}}$	
$L_i = s_j + b_i, R_i = s_k + b'_i$		
$s_{u+i} = \mathsf{decYao}(L_i, R_i, i, GT_i)$	$\xrightarrow{s_{N-o+1}, \ldots, s_N}$	*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 determine output $f(x)$

Fig. 1. The [KM11] protocol. The circuit $\mathcal{C}_f$ has u input wires, o output wires, g gates, and size $N = u + g$. The symmetric security parameter is $\kappa = 128$.

In most applications, e.g., when a server provides a service with a pre-defined function (such as privacy-preserving IDS, cf. Sect. 1), the *precomputation* and both *setup* phases can be precomputed before the client provides its input, allowing for a very fast $online_x$ phase. In other applications, the function may not be known beforehand, in which case the *precomputation* and $setup_N$ phases can be precomputed, and the $setup_f$ and $online_x$ phases are run online.

1) precomputation phase. We first determine all operations that have to be done once, independently of the protocol run: For [KM11], this includes generating and sending the public key of the HE scheme, and for UC-based PFE, the construction of the UC itself. We do not include this phase in our performance evaluation in Sect. 5.

2) setup$_N$ phase. This phase precomputes all operations that depend only on the size N of the circuit. In [KM11], P_2 creates two wire keys representing the bit values $\mathsf{0}$ and $\mathsf{1}$ for each of the $N = g+u$ outgoing wires. 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. P_2 chooses the wire key s_i^0 at random 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 $i \in \{1,\ldots,N\}$ and sends the homomorphically encrypted wire keys $\mathsf{Enc}(s_1^0)$, …, $\mathsf{Enc}(s_{N-o}^0)$ to P_1. As a preparation for the $setup_f$ phase, P_1 already creates and encrypts two random blinding values, b_i and b_i', for each gate G_i. This phase has complexity $\mathcal{O}(N)$.

In the UC-based PFE protocols, the UC is garbled and sent to the evaluator, which has complexity $\Theta(N \log N)$.

3) setup$_f$ phase. This depends on the specific function f. In [KM11], party P_1 creates the encrypted garbled gates. In order to hide the wiring of the circuit from P_2, each wire key is blinded. If outgoing wires j and k are connected to the incoming wires of gate G_i, P_1 constructs the encrypted garbled gate encGG_i by making use of the additively homomorphic property of Enc as

$$\mathsf{encGG}_i = \left(\mathsf{Enc}(s_j^0 + b_i), \mathsf{Enc}(s_k^0 + b_i')\right). \tag{1}$$

P_1 then sends $\mathsf{encGG}_1, \ldots, \mathsf{encGG}_g$ to P_2. P_2 is now able to create the garbled tables and thereby acts as the *circuit garbler* from Yao's protocol. For each gate G_i, P_2 decrypts the corresponding encrypted garbled gate and retrieves the blinded wire keys for the left and the right incoming wire of the gate:

$$L_i^0 = \mathsf{Dec}(\mathsf{Enc}(s_j^0 + b_i)), \quad R_i^0 = \mathsf{Dec}(\mathsf{Enc}(s_k^0 + b_i')). \tag{2}$$

P_2 is now able to obtain the blinded wire keys $s_j^1 + b_i$ and $s_k^1 + b_i'$ by defining $L_i^1 = L_i^0 + r$ and $R_i^1 = R_i^0 + r$. Note that the blinded wire keys L_i^0, L_i^1 and R_i^0, R_i^1 are independent of the keys assigned to the outgoing wires of gates j and k. This hides the circuit topology from P_2 while still enabling P_2 to create the garbled truth table

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

→

randomly permuted truth table

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

→

garbled table GT_i

$\mathsf{sEnc}_{L_i^1\|\|R_i^0\|\|i\|\|00}(s^1_{u+i})$
$\mathsf{sEnc}_{L_i^0\|\|R_i^0\|\|i\|\|01}(s^1_{u+i})$
$\mathsf{sEnc}_{L_i^1\|\|R_i^1\|\|i\|\|10}(s^0_{u+i})$
$\mathsf{sEnc}_{L_i^0\|\|R_i^1\|\|i\|\|11}(s^1_{u+i})$

Fig. 2. encYao: creation of a garbled table [LP09]

tables. The garbled table GT_i is generated using function encYao, instantiated as shown in Fig. 2 [LP09]: The truth table of the NAND gate is randomly permuted and then for each combination of the left (L_i^0, L_i^1) and right (R_i^0, R_i^1) input key these keys are used to symmetrically encrypt the output key s^0_{u+i} or s^1_{u+i} using function sEnc which is instantiated using AES-128 (cf. Sect. 5.1 for details). We emphasize that the gates' output keys are pre-determined and the protocol of [KM11] applies additively homomorphic operations on input keys. Therefore, we cannot use GC optimizations like point-and-permute [BMR90], garbled row reduction [NPS99, Pin+09], or half-gates [ZRE15]. Instead, we have to use the classical GC from [LP09] with four entries per garbled table (GT), so each GT has size $4 \cdot (|s_{u+i}| + \sigma)$ bits, where $\sigma = 40$ is the statistical security parameter. Finally, P_2 sends $\mathsf{GT}_1, \ldots, \mathsf{GT}_g$ to P_1. This phase has complexity $\mathcal{O}(N)$.

In the UC-based PFE protocols, the wire keys specifying the values of the UC's programming bits are sent which yields $\Theta(N \log N)$ communication.

4) online$_x$ phase. In this final phase, the private data x is input by P_2. In [KM11], the wire keys $s_1^{x_1}, \ldots, s_u^{x_u}$ of the circuit input wires corresponding to P_2's input bits $x_1, \ldots, x_u$ are sent to P_1.[4] P_1 can now evaluate the garbled tables and determine the wire keys of the output wires as follows: To evaluate gate 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 for gate G_i with left input j and right input k 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 b_i, b_i' from the *setup$_N$* phase to calculate $L_i = s_j + b_i$ and $R_i = s_k + b_i'$. P_1 now decrypts the garbled table GT_i to learn the wire key $s_{u+i} = \mathsf{decYao}(L_i, R_i, i, \mathsf{GT}_i)$ as in Yao's garbled circuit protocol and continues 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. These can be mapped to plaintext outputs as in Yao's protocol. However, as mentioned in Sect. 3.2, the function holder P_1 should not learn the output of f, so the output is determined by party P_2. This phase has complexity $\mathcal{O}(N)$.

[4] The protocol can naturally be extended to the setting where also P_1 has private input data y. Either y is encoded in the private function f [PSS09], or the keys corresponding to the bits of y are obliviously sent to P_1 using oblivious transfer [Ish+03, LP09, Ash+13] as describe in [KM11].

In the UC-based PFE protocols, the wire keys corresponding to the private input x are sent, the garbled UC is evaluated, which requires $\Theta(N \log N)$ computation, and the output bits of the UC are decoded.

4.2 Optimizations of the [KM11] Protocol

In this section, we describe our optimizations to the protocol of [KM11].

Precomputation of All Homomorphic Encryptions. As described in Sect. 4.1, all homomorphic encryptions can be precomputed in the $setup_N$ phase where only the size N is known but neither $\mathcal{C}_f$ nor x. Since encryption is a relatively expensive operation, this drastically reduces the protocol runtime (see Sect. 5.2).

The wire keys are sampled randomly so depend neither on the inputs nor on the circuit $\mathcal{C}_f$, and are encrypted using the HE public key generated by P_2.

P_2 can sample and homomorphically encrypt the *encrypted wire keys* $\mathsf{Enc}(s_i^0)$, where $1 \leq i \leq N$. Similarly, P_1 can sample and encrypt the blinding values b_i, b_i', where $1 \leq i \leq g$, using P_1's public key. Here, it is necessary to exchange the public key of the HE scheme first. We argue that this is feasible in practice by P_2 publishing the public key beforehand.

Pipelining. The creation and evaluation of the garbled circuit (GC) is done in topological order which makes this process eligible for pipelining. When transmitting the garbled gates directly after creation, they can be ungarbled by the evaluator while subsequent gates are still being garbled by the garbler. This GC pipelining was proposed and implemented in [Hen+10,Hua+11].

In addition to the GC pipelining, we also implemented pipelining of the creation and evaluation of the encrypted garbled gates. The process of retrieving the wire keys from the encrypted garbled gates can then seamlessly be combined with the pipelined creation and evaluation of the GC. Since decryption of the encrypted garbled gates is the most expensive operations in the $setup_f$ phase, this significantly speeds up the protocol and reduces the time spent solely on network communication. In our experiments, we saw that pipelining improves the runtime in the $setup_f$ phase by about 25%.

Parallelization. The [KM11] protocol is very suitable for parallelization. We provide a fully parallelized implementation of 1) the creation of the encrypted wire keys by P_2 and the encrypted blinding values by P_1 in the $setup_N$ phase, 2) the creation of the encrypted garbled gates by P_1 in the $setup_f$ phase, 3) the decryption of the encrypted garbled gates and the creation of the garbled tables by P_2 in the $setup_f$ phase. Only the evaluation of the garbled tables by P_1 depends on the wire keys obtained from previous garbled tables and therefore cannot be fully parallelized.

4.3 Instantiating [KM11] with EC ElGamal

Katz and Malka suggest to use ElGamal encryption to instantiate their protocol [KM11], and briefly mention the possibility of using elliptic curve cryptography (ECC) in their protocol. 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.

In standard additively homomorphic lifted EC ElGamal, a message $m \in GF(p)$ is mapped to a curve point M as $M = m * P$. The reverse mapping used during decryption then requires solving the discrete logarithm of M which requires that m is from a small domain whereas we need to operate on $\kappa = 128$ bit keys. Instead, we observe that the only requirement for the choice of the wire keys and the blinding values in the [KM11] protocol is indistinguishability, so we can simply define curve points M as our plaintext values for wire keys and blinding values. Then, we perform plaintext additions using the ECC arithmetic on the elliptic curve when P_1 needs to apply the blinding value to a plaintext wire key in order to determine the values L_i and R_i. These points are then mapped to keys for AES using a KDF (cf. Sect. 5.1).

Analogous to standard ElGamal, we define encryption of a message M with a public key $A = a * P$ as follows:

$$\mathsf{Enc}(M) = (K, C) = (k * P, k * A + M). \tag{3}$$

Decryption of the ciphertext (K, C) can now be done as follows:

$$\mathsf{Dec}(K, C) = C - a*K = k*A + M - a*k*P = k*a*P + M - a*k*P = M. \tag{4}$$

EC ElGamal is additively homomorphic in the underlying elliptic curve group. We define the homomorphic addition of two ciphertexts as

$$\mathsf{Enc}(M_1) \oplus \mathsf{Enc}(M_2) = (K_1, C_1) \oplus (K_2, C_2) = (K_1 + K_2, C_1 + C_2). \tag{5}$$

This satisfies the additively homomorphic property over the EC group:

$$\begin{aligned}
\mathsf{Dec}(\mathsf{Enc}(M_1) \oplus \mathsf{Enc}(M_2)) &= \mathsf{Dec}((k_1 * P, k_1 * A + M_1) \oplus (k_2 * P, k_2 * A + M_2)) \\
&= \mathsf{Dec}((k_1 * P + k_2 * P, k_1 * A + M_1 + k_2 * A + M_2)) \\
&= \mathsf{Dec}((k_1 + k_2) * P, (k_1 + k_2) * A + M_1 + M_2)) \\
&= (k_1 + k_2) * A + M_1 + M_2 - a * (k_1 + k_2) * P = M_1 + M_2. \qquad (6)
\end{aligned}$$

Semantic security naturally follows from that of ElGamal based in the DDH assumption in the EC group.

4.4 Instantiating [KM11] with BFV Homomorphic Encryption

Since the linear-complexity protocol of [KM11] was proposed in 2011, significant progress has been made in the area of Ring-LWE (RLWE) based homomorphic encryption. Thus, we revise the protocol of [KM11] with an HE instantiation based on these efficient Ring-LWE HE schemes. We specifically use the BFV scheme (cf. Sect. 3.3) as implemented in Microsoft's SEAL library [Sea19]. We take the plaintext modulus as $t = 2$, which results in the smallest possible polynomial modulus degree and thus ciphertext size in our scenario. The coefficient modulus q is chosen as a product of primes $q_1 = 12289$ and $q_2 = 1099510054913$. q_1 is the smallest prime that is large enough to allow homomorphic blinding of the key values and satisfies $q_1 \equiv 1 \bmod 2n$, where n is *polynomial modulus degree* (cf. [Sea19] for details). 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, §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 $p = 2048$ as the polynomial modulus degree, which is the smallest n that maintains computational security of 128-bits for a q of 54-bits (cf. [Lai17], Table 3).

Encoding of the Wire Keys. When choosing a plaintext modulus of $t = 2$, each bit of the plaintext value is encoded as one coefficient of the polynomial. Assume we have a wire key v with a binary representation of $v = v_{127}||v_{126}||\ldots||v_0$, we define our plaintext polynomial as $v_{127}x^{127} + \ldots + v_1 x + v_0$. Since homomorphic addition is done coefficient-wise in the BFV scheme and we use $t = 2$, addition becomes equivalent to a *homomorphic XOR operation*.

Due to the requirement that each wire key has to be utilized separately when creating the encrypted garbled gates, Chinese Remainder Theorem (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 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.

Efficient Packing of the Ciphertexts. The encoding of the wire keys 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, utilizing 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.

Unfortunately, without access to the secret key, it is not possible for P_1 to homomorphically extract a subset of coefficients of the underlying plaintext, and thus a wire key. Therefore, multiple wire keys can only be packed in a response to P_2 holding the secret key.

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 $\mathsf{Enc}(s_j||s_k)$. 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. We achieve this by applying a "homomorphic (right) bit shift" of 128-bits (respectively coefficients) to one of the wire keys (by multiplying a ciphertext by the plaintext constant 2^{128}) and adding both wire keys afterwards.

These wire keys still have to be blinded to form the encrypted garbled gate encGG_i, which can now be achieved by only one homomorphic addition. Therefore, we concatenate the blinding values b_i and b'_i and homomorphically add them to $\mathsf{Enc}(s_j||s_k)$ to receive the encrypted garbled gate $\mathsf{encGG}_i = \mathsf{Enc}(s_j||s_k) + \mathsf{Enc}(b_i||b'_i) = \mathsf{Enc}((s_j + b_i)||(s_k + b'_i))$. Since P_2 is in charge of telling the wire keys apart, "unpacking" is simply done by decrypting the ciphertext and assigning 128-bits to both wire keys.

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]. Blinding of the wire keys can now be applied by concatenating 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.

Wire Key Generation Using Seed Expansion. The wire keys are encrypted by the private key owner P_2 and can be homomorphically encrypted using the *secret* key to have smaller noise and smaller ciphertext size. When encrypting with the secret key, half of the ciphertext coefficients are chosen uniformly at random from R_q. Using a pseudo-random function, one can sample these coefficients by expanding a seed sent to P_1 instead. This nearly halves the ciphertext size of the encrypted wire keys and significantly improves communication which is the major bottleneck of the scheme.[5]

5 Evaluation

In this section, we describe our implementation of the different instantiations of the [KM11] protocol and point out bottlenecks and advantages. We experimentally compare our implementations with the best existing UC-based PFE implementation of [Alh+20]. We also give estimates on the efficiency of the recent UC improvements of [Liu+20] that results in 33% smaller UCs and hence would improve UC-based PFE of [Alh+20] by around 33% in both runtime and communication (cf. dashed lines in Fig. 4 and 3). The results of our performance tests show that HE-based linear-complexity PFE supersedes UC-based PFE in runtime starting from a few thousand gates already and in communication for all

[5] Since January 2020 (version 3.4.0) the SEAL library [Sea19] supports seed expansion and encryption with the secret key. Our implementation uses this optimization.

circuit sizes. Hence, linear-complexity PFE is a viable alternative for improving the performance of private function evaluation.

5.1 Implementation

We implemented our optimized and fully parallelized version of the [KM11] protocol described in Sect. 4 using the ABY SFE framework [DSZ15]. Our implementation is available as open-source at https://encrypto.de/code/linearPFE. This is the first implementation of a linear-complexity PFE protocol. We provide a fair comparison with today's most efficient UC-based PFE implementation of [Alh+20] with complexity $\Theta(N \log N)$ which is based on the same STPC framework ABY.

We instantiate sEnc as $\mathsf{sEnc}_{k'}(m) = (AES_{k'}(0)||AES_{k'}(1)||\ldots|| AES_{k'}(\lceil(|m|+\sigma)/128\rceil - 1)) \oplus (m||0^{\sigma})$, where AES is AES-128 and $\sigma = 40$ is the statistical security parameter. The arbitrary-length key k is mapped to a 128-bit key $k' = \mathrm{KDF}(k)$ where the KDF is instantiated with *PBKDF2*.

We instantiate the DJN cryptosystem with modulus size of 3072 bits.

In our EC ElGamal-based implementation we use the eBATS B-251 binary elliptic curve. RELIC encodes each point on the elliptic curve in 33 bytes.

SEAL serializes ciphertexts as 64-bit values using a compression function. For our specific choice of parameters, this compression did not achieve ideal results. For all ciphertexts except the encrypted wire keys where a seed is used to reduce their size, we implemented our own serialization where we eliminate unnecessary zeroes and thereby reduce the ciphertext size compared to the SEAL encoding.

5.2 Experimental Evaluation

We use two identical machines with a physical connection of 10 Gbit/s bandwidth and a round-trip time of 1 ms. We refer to this as the LAN setting and also simulated a WAN setting with 100 Mbit/s bandwidth and a round-trip time of 100 ms. Each machine is equipped with an Intel Core i9-7960X CPU (32 Cores, 2.8 GHz) and 128 GB RAM. All measurements are averaged over 10 executions. Because in all PFE protocols the costs for the input x is substantially lower than for the gates, we fix the number of input bits to $u = |x| = 64$. The exact performance measures used to plot the figures are given in the full version [Hol+20].

Communication. In Fig. 3, we depict the communication of the PFE protocols. The EC ElGamal instantiation clearly outperforms all other implementations, including UC-based PFE [Alh+20] and thereby offers the best PFE scheme in terms of communication known so far. Its communication is lower than UC-based PFE of [Alh+20] by a factor of $\sim 11\times$ for circuit size $N = 10^6$.

We observe that the communication complexity of DJN-based PFE is on par with UC-based approaches. Due to its large ciphertext size, BFV-based encryption has the worst communication of our instantiations but it is only a factor of about 1.8× higher for $N = 10^6$ than that of UC-based PFE [Alh+20]. Its communication is significantly reduced by the seed expansion technique to

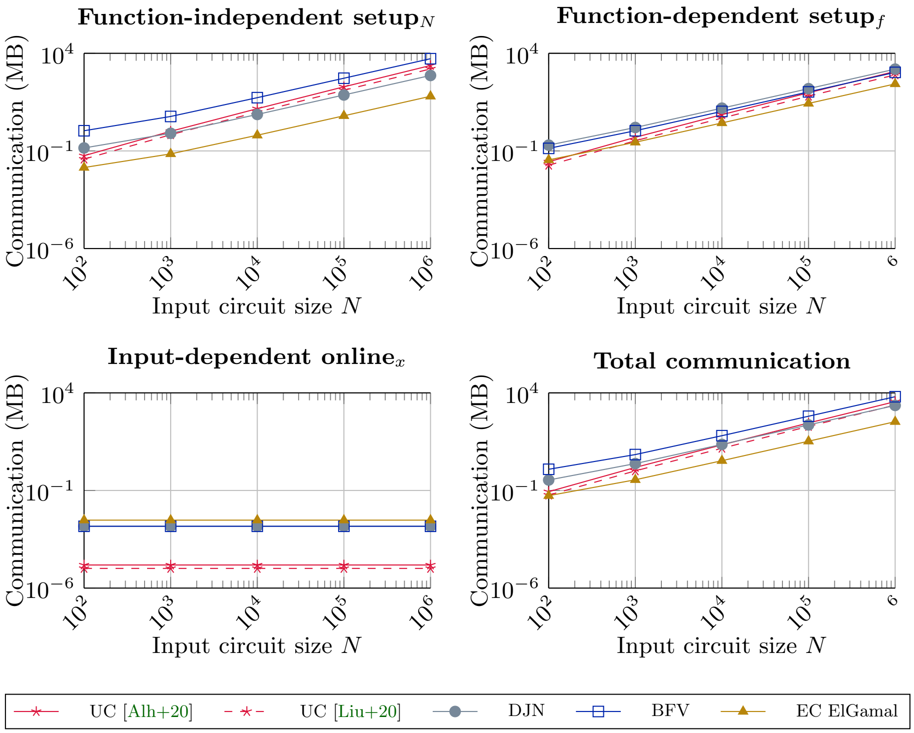


Fig. 3. Communication of PFE protocols (in MB).

reduce the size of the encrypted wire keys in the BFV scheme (cf. Sect. 3.3). In the *online*$_x$ phase, the communication of all protocols only depends on the size of the input x and is nearly negligible (only a few KB).

Runtime. In Fig. 4, we depict the runtime of our implementation compared to the most recent UC-based PFE implementation of [Alh+20].

ECC-based PFE is our fastest implementation: Compared to the state-of-the-art UC-based PFE implementation of [Alh+20], the total runtime for $N = 10^6$ gates is faster by a factor ∼3.3× in LAN and ∼7.0× in WAN.

BFV-based PFE offers promising total runtimes even though it is *less* efficient than ECC-based PFE of [Alh+20] by a factor of ∼1.4× in LAN and ∼1.8× in WAN for $N = 10^6$. The larger factor in the WAN setting results from its larger communication overhead compared to ECC-based PFE. These findings underline that though computational complexity is still relevant, communication complexity becomes the bottleneck for these PFE protocols. Therefore, the computational advances of BFV cannot compensate its larger ciphertext sizes any more. Still, our implementation instantiated with the BFV scheme beats [Alh+20] for circuits of about $N \geq 250000$ gates when function- and input-independent precomputations from the *setup*$_N$ phase are excluded cf. full version [Hol+20]).

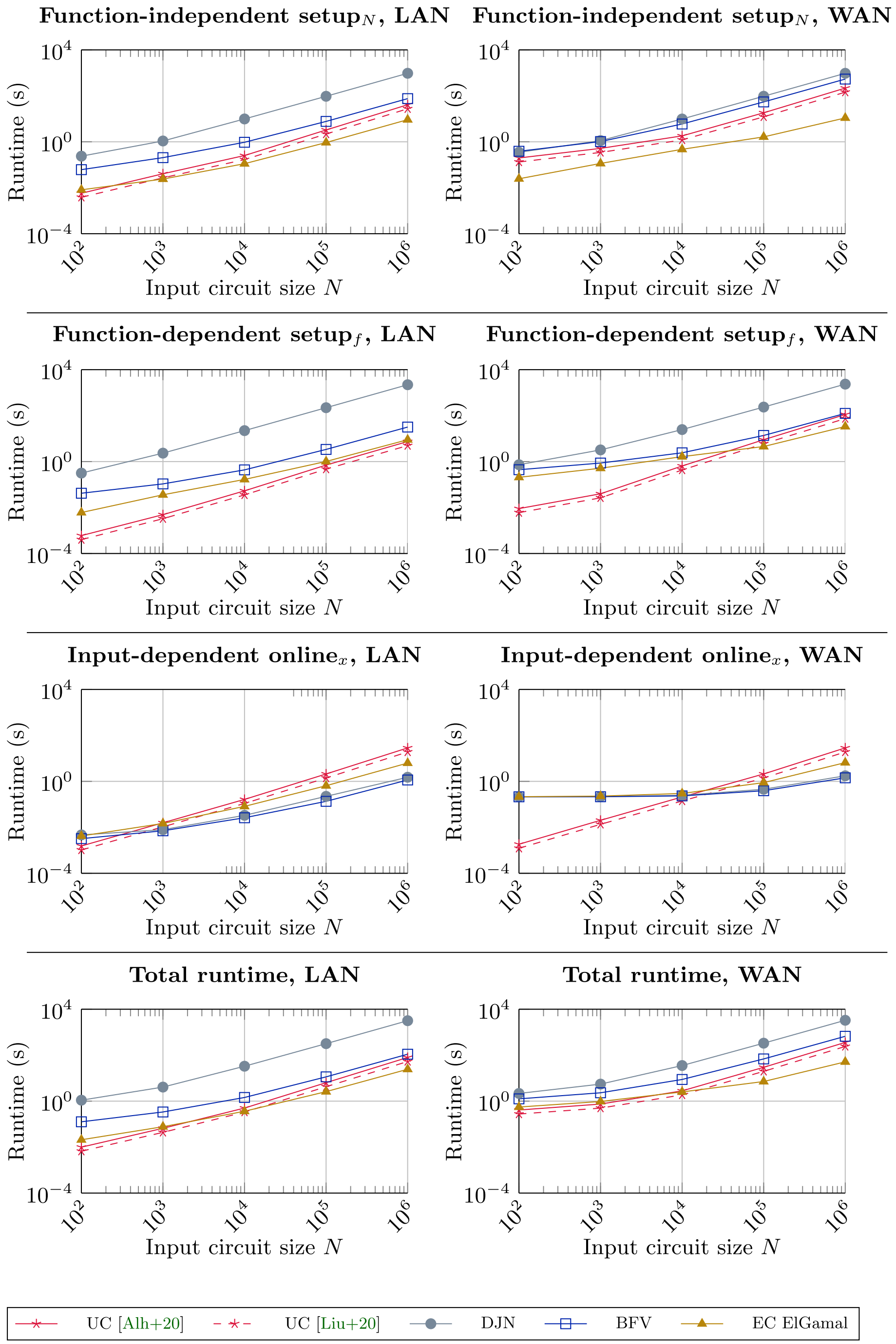


Fig. 4. Runtime of PFE protocols (in seconds).

DJN-based PFE has impractical computational overhead, i.e., about 53 minutes of runtime for $N = 10^6$ gates in LAN (compared to 24 s of the ECC-based instantiation), even with the optimizations described in Sect. 3.3. Its runtime in WAN is similar to WAN as it is dominated by computation.

Per-phase Comparison. In the $setup_N$ phase, computation and communication are independent of the function f and input x and only depend on the (maximum) size of f. This yields significant large precomputation capabilities of HE-based PFE, especially for our BFV-based instantiation.

In the $setup_f$ phase, the logarithmic overhead of UC-based PFE of [Alh+20] has a large performance impact. In contrast, HE-based protocols scale linearly and outperform UC-based PFE for $N \geq 10^6$ in LAN and $N \geq 250000$ in WAN.

In the $online_x$ phase, HE-based PFE outperforms UC-based PFE of [Alh+20] for about $N \geq 1000$ gates in LAN and $N \geq 10000$ gates in WAN. Here, the computation is dominated by GC evaluation. The logarithmic overhead of the UC size compared to the actual circuit leads to a noticeable performance drawback. Since our ECC-based implementation uses points on the elliptic curve as wire keys (encoded as 264 bit values), the GC is larger by a factor of about two compared to the BFV- and DJN-based instantiations where wire keys have size 128 bits. This impacts GC evaluation runtime and BFV-based PFE becomes the fastest instantiation in the $online_x$ phase.

When excluding precomputation of the $setup_N$ phase from the total runtime, BFV-based PFE outperforms UC-based PFE of [Alh+20] for about $N \geq 250000$ in LAN and WAN, and ECC-based PFE outperforms [Alh+20] for about $N \geq 10000$ in LAN and about $N \geq 25000$ in WAN (cf. full version [Hol+20]).

Summary. In this paper, we optimize and implement the linear-complexity PFE protocol of [KM11]. Our elliptic curve ElGamal-based implementation outperforms the state-of-the-art UC-based PFE implementation of [Alh+20] not only in communication, but also in total runtime: For private circuits of size $N = 10^6$, our implementation is ∼3.3× faster in a LAN and ∼7.0× faster in a WAN setting and scales with $\mathcal{O}(N)$ instead of $\Theta(N \log N)$.

Acknowledgement. This project received funding from the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation program (grant agreement No. 850990 PSOTI). It was co-funded by the Deutsche Forschungsgemeinschaft (DFG) – SFB 1119 CROSSING/236615297 and GRK 2050 Privacy & Trust/251805230, and by BMBF and HMWK within ATHENE.

References

[AG09] Aranha, D.F., Gouvêa, C.: RELIC cryptographic toolkit (2009). https://github.com/relic-toolkit

[Alh+20] Alhassan, M.Y., Günther, D., Kiss, Á., Schneider, T.: Efficient and scalable universal circuits. J. Cryptol. **33**(3), 1216–1271 (2020). https://doi.org/10.1007/s00145-020-09346-z

[Ash+13] Asharov, G., Lindell, Y., Schneider, T., Zohner M.: More efficient oblivious transfer and extensions for faster secure computation. In: CCS 2013, pp. 535–548. ACM (2013)

[Bel+13] Bellare, M., Hoang, V.T., Keelveedhi, S., Rogaway, P.: Efficient garbling from a fixed-key blockcipher. In: S&P 2013, pp. 478–492. IEEE (2013)

[BGV12] Brakerski, Z., Gentry, C., Vaikuntanathan, V.: (Leveled) fully homomorphic encryption without bootstrapping. In: Innovations in Theoretical Computer Science (ITCS 2012), pp. 309–325. ACM (2012)

[Biç+18] Biçer, O., Bingöl, M.A., Kiraz, M.S., Levi, A.: Highly efficient and reusable private function evaluation with linear complexity. Cryptology ePrint Archive, Report 2018/515 (2018). https://ia.cr/2018/515

[Bin+18] Bingöl, M.A., Biçer, O., Kiraz, M.S., Levi, A.: An efficient 2-party private function evaluation protocol based on half gates. Comput. J. **62**(4), 598–613 (2018)

[BMR90] Beaver, D., Micali, S., Rogaway, P.: The round complexity of secure protocols. In: STOC 1990, pp. 503–513. ACM (1990)

[Bra12] Brakerski, Z.: Fully homomorphic encryption without modulus switching from classical GapSVP. In: Safavi-Naini, R., Canetti, R. (eds.) CRYPTO 2012. LNCS, vol. 7417, pp. 868–886. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-32009-5_50

[DJN10] Damgård, I., Jurik, M., Nielsen, J.B.: A generalization of Paillier's public-key system with applications to electronic voting. Int. J. Inf. Secur. **9**(6), 371–385 (2010). https://doi.org/10.1007/s10207-010-0119-9

[DSZ15] Demmler, D., Schneider, T., Zohner, M.: ABY - a framework for efficient mixed-protocol secure two-party computation. In: NDSS 2015. The Internet Society (2015)

[Elg85] ElGamal, T.: A public key cryptosystem and a signature scheme based on discrete logarithms. Trans. Inf. Theory **31**(4), 469–472 (1985)

[FAL06] Frikken, K.B., Atallah, M.J., Li, J.: Attribute-based access control with hidden policies and hidden credentials. IEEE Trans. Comput. **55**(10), 1259–1270 (2006)

[FAZ05] Frikken, K.B., Atallah, M.J., Zhang, C.: Privacy-preserving credit checking. In: ACM Conference on Electronic Commerce (EC 2005), pp. 147–154. ACM (2005)

[Fel+19] Felsen, S., Kiss, Á., Schneider, T., Weinert, C.: Secure and private function evaluation with Intel SGX. In: CCSW 2019, pp. 165–181. ACM (2019)

[FV12] Fan, J., Vercauteren, F.: Somewhat practical fully homomorphic encryption. Cryptology ePrint Archive, Report 2012/144 (2012). https://ia.cr.org/2012/144

[GKS17] Günther, D., Kiss, Á., Schneider, T.: More efficient universal circuit constructions. In: Takagi, T., Peyrin, T. (eds.) ASIACRYPT 2017. LNCS, vol. 10625, pp. 443–470. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-70697-9_16

[GMW87] Goldreich, O., Micali, S., Wigderson, A.: How to play ANY mental game. In: STOC 1987, pp. 218–229. ACM (1987)

[Gün+19] Günther, D., Kiss, Á., Scheidel, L., Schneider, T.: Framework for semi-private function evaluation with application to secure insurance rate calculation. CCS 2019 Posters/Demos (2019)

[Hen+10] Henecka, W., Kögl, S., Sadeghi, A.-R., Schneider, T., Wehrenberg, I.: TASTY: tool for automating secure two-party computations. In: CCS 2010, pp. 451–462. ACM (2010)

[HMS12] Hu, Y., Martin, W.J., Sunar, B.: Enhanced flexibility for homomorphic encryption schemes via CRT. In: ACNS 2012 (Industrial Track) (2012)

[Hol+20] Holz, M., Kiss, Á., Rathee, D., Schneider, T.: Linear-complexity private function evaluation is practical (full version). Cryptology ePrint Archive, Report 2020/853 (2020). https://ia.cr/2020/853

[Hua+11] Huang, Y., Evans, D., Katz, J., Malka, L.: Faster secure two-party computation using garbled circuits. In: USENIX Security 2011. USENIX (2011)

[IR89] Impagliazzo, R., Rudich, S.: Limits on the provable consequences of one-way permutations. In: STOC 1989, pp. 44–61. ACM (1989)

[Ish+03] Ishai, Y., Kilian, J., Nissim, K., Petrank, E.: Extending oblivious transfers efficiently. In: Boneh, D. (ed.) CRYPTO 2003. LNCS, vol. 2729, pp. 145–161. Springer, Heidelberg (2003). https://doi.org/10.1007/978-3-540-45146-4_9

[KM11] Katz, J., Malka, L.: Constant-round private function evaluation with linear complexity. In: Lee, D.H., Wang, X. (eds.) ASIACRYPT 2011. LNCS, vol. 7073, pp. 556–571. Springer, Heidelberg (2011). https://doi.org/10.1007/978-3-642-25385-0_30

[Kob87] Koblitz, N.: Elliptic curve cryptosystems. Math. Comput. **48**(177), 203–209 (1987)

[KS08a] Kolesnikov, V., Schneider, T.: A practical universal circuit construction and secure evaluation of private functions. In: Tsudik, G. (ed.) FC 2008. LNCS, vol. 5143, pp. 83–97. Springer, Heidelberg (2008). https://doi.org/10.1007/978-3-540-85230-8_7

[KS08b] Kolesnikov, V., Schneider, T.: Improved garbled circuit: free XOR gates and applications. In: Aceto, L., Damgård, I., Goldberg, L.A., Halldórsson, M.M., Ingólfsdóttir, A., Walukiewicz, I. (eds.) ICALP 2008. LNCS, vol. 5126, pp. 486–498. Springer, Heidelberg (2008). https://doi.org/10.1007/978-3-540-70583-3_40

[KS16] Kiss, Á., Schneider, T.: Valiant's universal circuit is practical. In: Fischlin, M., Coron, J.-S. (eds.) EUROCRYPT 2016. LNCS, vol. 9665, pp. 699–728. Springer, Heidelberg (2016). https://doi.org/10.1007/978-3-662-49890-3_27

[KSS13] Kolesnikov, V., Sadeghi, A.-R., Schneider, T.: A systematic approach to practically efficient general two-party secure function evaluation protocols and their modular design. J. Comput. Secur. **21**(2), 283–315 (2013)

[Lai17] Laine, K.: Simple encrypted arithmetic library 2.3.1. Microsoft Research (2017). https://www.microsoft.com/en-us/research/uploads/prod/2017/11/sealmanual-2-3-1.pdf

[Liu+20] Liu, H., Yu, Y., Zhao, S., Zhang, J., Liu, W.: Pushing the limits of Valiant's universal circuits: simpler, tighter and more compact. Cryptology ePrint Archive, Report 2020/161 (2020). https://ia.cr/2020/161

[LMS16] Lipmaa, H., Mohassel, P., Sadeghian, S.S.: Valiant's universal circuit: improvements, implementation, and applications. Cryptology ePrint Archive, Report 2016/17 (2016). https://ia.cr/2016/017

[LP09] Lindell, Y., Pinkas, B.: A proof of security of Yao's protocol for two-party computation. J. Cryptol. **22**(2), 161–188 (2009). https://doi.org/10.1007/s00145-008-9036-8

[LPR10] Lyubashevsky, V., Peikert, C., Regev, O.: On ideal lattices and learning with errors over rings. In: Gilbert, H. (ed.) EUROCRYPT 2010. LNCS, vol. 6110, pp. 1–23. Springer, Heidelberg (2010). https://doi.org/10.1007/978-3-642-13190-5_1

[Mil86] Miller, V.S.: Use of elliptic curves in cryptography. In: Williams, H.C. (ed.) CRYPTO 1985. LNCS, vol. 218, pp. 417–426. Springer, Heidelberg (1986). https://doi.org/10.1007/3-540-39799-X_31

[MS13] Mohassel, P., Sadeghian, S.: How to hide circuits in MPC an efficient framework for private function evaluation. In: Johansson, T., Nguyen, P.Q. (eds.) EUROCRYPT 2013. LNCS, vol. 7881, pp. 557–574. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-38348-9_33

[MSS14] Mohassel, P., Sadeghian, S., Smart, N.P.: Actively secure private function evaluation. In: Sarkar, P., Iwata, T. (eds.) ASIACRYPT 2014. LNCS, vol. 8874, pp. 486–505. Springer, Heidelberg (2014). https://doi.org/10.1007/978-3-662-45608-8_26

[Nik+14] Niksefat, S., Sadeghiyan, B., Mohassel, P., Sadeghian, S.: ZIDS: a privacy-preserving intrusion detection system using secure two-party computation protocols. Comput. J. **57**(4), 494–509 (2014)

[NPS99] Naor, M., Pinkas, B., Sumner, R.: Privacy preserving auctions and mechanism design. In: ACM Conference on Electronic Commerce (EC 1999), pp. 129–139. ACM (1999)

[Pai99] Paillier, P.: Public-key cryptosystems based on composite degree residuosity classes. In: Stern, J. (ed.) EUROCRYPT 1999. LNCS, vol. 1592, pp. 223–238. Springer, Heidelberg (1999). https://doi.org/10.1007/3-540-48910-X_16

[Pin+09] Pinkas, B., Schneider, T., Smart, N.P., Williams, S.C.: Secure two-party computation is practical. In: Matsui, M. (ed.) ASIACRYPT 2009. LNCS, vol. 5912, pp. 250–267. Springer, Heidelberg (2009). https://doi.org/10.1007/978-3-642-10366-7_15

[PSS09] Paus, A., Sadeghi, A.-R., Schneider, T.: Practical secure evaluation of semi-private functions. In: Abdalla, M., Pointcheval, D., Fouque, P.-A., Vergnaud, D. (eds.) ACNS 2009. LNCS, vol. 5536, pp. 89–106. Springer, Heidelberg (2009). https://doi.org/10.1007/978-3-642-01957-9_6

[Reg05] Regev, O.: On lattices, learning with errors, random linear codes, and cryptography. In: STOC 2005, pp. 84–93. ACM (2005)

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

[Val76] Valiant, L.G.: Universal circuits (preliminary report). In: STOC 1976, pp. 196–203. ACM (1976)

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

[Yao86] Yao, A.C.-C.: How to generate and exchange secrets. In: FOCS 1986, pp. 162–167. IEEE (1986)

[Zha+19] Zhao, S., Yu, Yu., Zhang, J., Liu, H.: Valiant's universal circuits revisited: an overall improvement and a lower bound. In: Galbraith, S.D., Moriai, S. (eds.) ASIACRYPT 2019. LNCS, vol. 11921, pp. 401–425. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-34578-5_15

[ZRE15] Zahur, S., Rosulek, M., Evans, D.: Two halves make a whole. In: Oswald, E., Fischlin, M. (eds.) EUROCRYPT 2015. LNCS, vol. 9057, pp. 220–250. Springer, Heidelberg (2015). https://doi.org/10.1007/978-3-662-46803-6_8