Private Function Evaluation with Linear Complexity

Shuaishuai Li[1], Cong Zhang[2], Anyu Wang[1,2(✉)], and Xiaoyun Wang[1,2]

[1]Zhongguancun Laboratory, Beijing, China
[2]Tsinghua University, Beijing, China
liss@zgclab.edu.cn, zhangcong@mail.tsinghua.edu.cn, anyuwang@tsinghua.edu.cn, xiaoyunwang@tsinghua.edu.cn

Abstract

We present new frameworks for secure function evaluation (SFE) and private function evaluation (PFE) that support both Boolean and arithmetic circuits. While SFE requires multiple parties to jointly compute a *public* circuit, PFE generalizes SFE by allowing one party to keep the circuit *private*. Our work achieves the first linear-complexity PFE protocol with respect to both the number of parties n and circuit size m, significantly improving upon prior PFE constructions that require $O(mn^2)$ complexity and are limited to Boolean circuits.

1 Introduction

Secure function evaluation (SFE) enables mutually distrustful parties to jointly compute a *public* circuit C over their private inputs $x_1, \ldots, x_n$ while revealing nothing beyond the output. Private function evaluation (PFE) [AF90] strengthens SFE by additionally requiring privacy of the circuit itself. In the PFE setting, one party P_1 holds a *private* circuit C, while each party P_i contributes a private input x_i. The parties jointly compute $C(x_1, \ldots, x_n)$ without revealing any information about C beyond what is implied by the output. While SFE suffices for many scenarios, the literature has explored a wide range of PFE applications, including software diagnostics [BPSW07], privacy-preserving medical applications [BFK+09], and confidential insurance rate calculation [GKSS19]. A more comprehensive survey can be found in [AGKS20].

A straightforward approach to constructing PFE leverages universal circuits [Val76]. Specifically, the parties can execute an SFE protocol to evaluate a universal circuit U_m that satisfies $U_m(C, x) = C(x)$ for any given circuit C of size m and input x. The security properties of SFE inherently guarantee that the circuit C remains concealed during this computation. However, this method's practicality heavily depends on the complexity of the universal circuit U_m.

For Boolean circuits with m gates, despite a series of optimizations [SS08, KS08, KS16, GKS17, ZYZL19, LYZ+21, DGS+23], U_m has a size lower bound of $O(m \log m)$ [Val76]. The situation becomes significantly worse for arithmetic circuits, where the size of U_m grows to $O(m^5)$ [Raz08, SY10]. This superlinear overhead renders constructions based on universal circuits inefficient, particularly for arithmetic computations. The fundamental challenge lies in developing PFE protocols achieving linear complexity in m.

The seminal work by Abadi and Feigenbaum [AF90] proposed a two-party PFE protocol for Boolean circuits with linear complexity in the circuit size m.[1] Their protocol also has a linear round complexity

[1]Notably, their scheme assumes leakage of the number m^* of AND gates rather than the total circuit size m. However, we observe that their protocol can be straightforwardly modified to obtain a PFE scheme where the leaked information becomes m instead. In this modified version, both the (computational and communication) complexity and number of rounds would then scale linearly with m.

in m. At ASIACRYPT 2011, Katz and Malka [KM11] introduced a constant-round two-party PFE protocol for Boolean circuits that achieves $O(m)$ complexity based on linearly-homomorphic encryption (LHE). A recent implementation study [HKRS20] of this PFE protocol demonstrated that it surpasses the PFE approach based on universal circuits not only in communication efficiency but also in overall runtime performance. Following this work, Bicer et al. [BBKL22] proposed a two-party PFE with a reusability feature, which makes their protocol more efficient when the protocol is executed multiple times for the same function (by the same two parties).

At EUROCRYPT 2013, Mohassel and Sadeghian [MS13] introduced a versatile framework for PFE. Their approach decomposes PFE into two key components: (1) concealing the circuit through circuit-topology hiding (CTH), and (2) securely evaluating the gates using private gate evaluation (PGE). These components are designed independently and then seamlessly integrated to form a complete PFE protocol. The authors presented two distinct CTH realizations. The first method relies on oblivious evaluation of switching networks (OSN), which, while efficient for small circuits, becomes impractical for larger ones due to its $O(m \log m)$ complexity. In contrast, the second approach leverages LHE, achieving linear complexity.

Building upon this CTH+PGE framework, the work [MS13] developed multiple two-party and multiparty PFE protocols with varying characteristics. In the two-party setting, they constructed a constant-round protocol for Boolean circuits based on garbled circuits [Yao86], while also proposing an m-round PFE scheme for arithmetic circuits based on LHE. In the multiparty setting, they gave an m-round PFE construction for Boolean circuits. Subsequent research further enhanced these protocols. The work [BBKL19] optimized the concrete efficiency of the constant-round two-party PFE protocol for Boolean circuits using the half gates technique [ZRE15]. Another work [XJL+23] successfully reduced the round complexity of the multiparty PFE scheme to constant rounds.

Notably, in the multiparty setting involving n parties, both the PFE constructions in [MS13] and [XJL+23] exhibit quadratic complexity $O(mn^2)$ with respect to the number of parties. This quadratic scaling significantly limits their practicality for large-scale deployments. Consequently, developing a multiparty PFE scheme with linear complexity $O(mn)$ remains an open question. Furthermore, the existing multiparty PFE constructions share another limitation: they only support Boolean circuits. While solutions based on universal circuits for arithmetic circuits exist, their prohibitive $O(m^5)$ complexity makes them extremely impractical. What we truly need are protocols achieving linear complexity in the circuit size m. Therefore, the design of efficient multiparty PFE protocols for *arithmetic* circuits presents another important open problem.

In this work, we simultaneously resolve the aforementioned two open problems. Specifically, we answer the following questions affirmatively.

Does there exist a multiparty PFE scheme whose computational and communication complexity are both linear in the number of parties n and the circuit size m? Moreover, can such scheme support both Boolean and arithmetic circuits?

Other related works. Semi-PFE is a relaxed variant of PFE that permits partial circuit information leakage. The work [PSS09] considered semi-PFE where the circuit topology could be revealed. Notably, in [MS13], protecting circuit topology is achieved by the CTH functionality. The works [GKSS19, GSSY24] introduced flexible semi-PFE implementation frameworks that allow customizable circuit information leakage. Specifically, they considered that the computed circuit C contain a public component C_{pub} and a private component C_{pri} and demonstrated how to efficiently evaluate C using existing techniques for SFE and PFE.

Several works designed PFE protocols in the malicious adversary setting. The seminal work [KM11] demonstrated how to adapt their passively secure two-party PFE protocol to resist malicious behavior from the party who does not possess the circuit through efficient zero-knowledge protocols. But their protocol

cannot resist malicious circuit owner. The work [MSS14] later proposed an actively secure PFE framework with linear complexity, extending the techniques from [MS13]. While this represents significant progress, the protocol's round complexity grows linearly with circuit size. This limitation was addressed by [LWY22], who derived a constant-round variant of [MSS14]'s protocol, with [JL23] later optimizing its concrete performance.

1.1 Our Contribution

Our contributions are summarized as follows. A comparison of PFE and semi-PFE protocols is provided in Table 1.

A new SFE framework. We propose a new SFE framework based on linearly-homomorphic encryption (LHE) with two appealing features. First, only one party needs circuit knowledge and storage proportional to circuit width, while others does not need to know the circuit in advance (but circuit information may be revealed during protocol execution) and maintain only constant storage. Second, it only requires a star topology where one designated party (say P_1) communicates with all other parties. Specifically, the central party's communication cost with each other party is independent of the number n of parties, thereby achieving linear $O(n)$ total communication complexity. We note that the star topology implies that there are only $n-1$ communication links, which is optimal as any topology with $n-2$ or fewer links would be disconnected and thus insufficient for general secure computation.

PFE with linear complexity. We upgrade our LHE-based SFE to PFE that supports both Boolean and arithmetic circuits while achieving linear complexity $O(m)$ in circuit size m. In particular, the resulting multiparty PFE achieves linear complexity $O(mn)$, where n denotes the number of parties. In contrast, previous multiparty PFE protocols [MS13, XJL+23] exhibit $O(mn^2)$ complexity.

A SFE paradigm from encryption. We show that the well-known GMW protocol [GMW87] and our SFE framework can be cast into an encryption-based paradigm for SFE, with the key difference being that they employ different underlying encryption schemes. In particular, based on this paradigm, we show how to use symmetric-key encryption (SKE) scheme to design an alternative PFE protocol with linear complexity. The resulting protocol does not rely on LHE, and may therefore achieve higher computational efficiency.

Semi-PFE with reduced round complexity. Our PFE protocol requires m rounds of interactions among participants. We demonstrate how to reduce the round complexity by considering semi-PFE that selectively leaks partial circuit information. Specifically, when the circuit topology can be revealed, the round complexity can be reduced to d rounds, where d is the circuit depth. Furthermore, we show that it suffices to reveal only the circuit depth d in order to achieve $O(d)$ round complexity.

Efficient PFE from minimal homomorphism. Finally, we consider constructing PFE with linear complexity based on a minimal homomorphic assumption. Specifically, we use a rerandomizable public-key encryption (RPKE) scheme, which is a special type of PKE that allows generating fresh ciphertexts encrypting the same message from existing ciphertexts, in the sense that the resulting ciphertext is statistically indistinguishable from an honestly generated encryption of the same message. RPKE can be viewed as a homomorphic encryption scheme with minimal homomorphism: it supports only homomorphic evaluation of the identity function.

In fact, our SKE-based PFE construction already implies that RPKE can be used to construct PFE with linear complexity. Specifically, our SKE-based PFE construction relies on an SKE scheme together with secure distributed evaluation of its encryption and decryption algorithms. Since RPKE implies SKE (by treating the PKE secret key as a symmetric key) and also implies oblivious transfer (OT) [LZL23] (and therefore enables secure evaluation of arbitrary functions), RPKE can in principle be used to instantiate the construction. However, such an approach requires *non-black-box* use of RPKE, namely, securely evaluating

Party Number	Protocol Type	Work	Circuit Type	Public Circuit Information	Rounds	Complexity
2	Semi-PFE	[PSS09]	B	Topology	$O(1)$	$O(m)$
		Sect. 7.1	B,A	Topology	d	$O(m)$
		Sect. 7.2	B,A	Size,Depth	$(1+\varepsilon)d$	$O(m)$
	PFE	[AF90]	B	Size	m	$O(m)$
		[KM11, BBKL22]	B	Size	$O(1)$	$O(m)$
		[MS13]	B,A	Size	m	$O(m)$
		[MS13, BBKL19]	B	Size	$O(1)$	$O(m)$
		Sect. 4.1	B,A	Size	m	$O(m)$
n	Semi-PFE	Sect. 7.1	B,A	Topology	d	$O(mn)$
		Sect. 7.2	B,A	Size,Depth	$(1+\varepsilon)d$	$O(mn)$
	PFE	[MS13]	B	Size	m	$O(mn^2)$
		[XJL+23]	B	Size	$O(1)$	$O(mn^2)$
		Sect. 4.2	B,A	Size	m	$O(mn)$

Table 1: Comparison among different semi-PFE and PFE protocols with linear complexity (with respect to the number m of gates in the circuit). "B" represents "Boolean". "A" represents "arithmetic". Further, d is the depth of the computed circuit, and ε is any positive constant. The best entries are marked with green highlighting.

its encryption and decryption algorithms, and is therefore inefficient. We show that two-party PFE for Boolean circuits with linear complexity can instead be constructed using only *black-box* access to RPKE.

Remark 1. In standard PFE protocols, the only circuit information leaked is the total size m, which is the number of addition and multiplication gates in the circuit. Similar to [AF90], by instead revealing the number m^* of multiplication (AND) gates, the communication and round complexity of our PFE protocols can be reduced to scale linearly with m^*, with the exception of our RPKE-based PFE construction, which cannot achieve this improvement. Furthermore, in our semi-PFE protocol, the round complexity becomes linear in the multiplicative depth d^*, where only layers containing multiplication gates contribute to the depth count.

1.2 Technical Overview

We provide an overview of the techniques behind our PFE constructions. From now on, we use $[\![x]\!]$ to represent an encryption of x under some encryption scheme.

Starting point: SFE from FHE. We begin with a straightforward two-party secure function evaluation (SFE) protocol based on fully homomorphic encryption (FHE), where party P_1 collects encryptions of all inputs and carries out the desired homomorphic evaluation. This FHE-based SFE scheme is in fact also a PFE scheme, since clearly P_2 does not need to know any information about the computed circuit. However, FHE introduces significant computational overhead, particularly for homomorphic multiplications (i.e., multiplications over ciphertexts). To address this, instead of having P_1 perform homomorphic multiplications by itself, we let P_2 assist P_1 in completing the computational task.

A new SFE framework from LHE. Our idea is to remove the need for homomorphic multiplication by using interactions. Recall that the goal of homomorphic multiplication is to generate a ciphertext $[\![uv]\!]$

from the ciphertexts $[\![u]\!]$ and $[\![v]\!]$. Note that for random a, b, the following equation holds.

$$uv = (a + u - a)(b + v - b) = ab + a(v - b) + (u - a)b + (u - a)(v - b).$$

Apparently, if P_1 has $(u - a, v - b)$ and the ciphertext triple $([\![a]\!], [\![b]\!], [\![ab]\!])$, then it can compute an encryption of uv as follows.

$$[\![uv]\!] \leftarrow [\![ab]\!] + (v - b)[\![a]\!] + (u - a)[\![b]\!] + (u - a)(v - b).$$

This requires only linear homomorphism.

The remaining challenge lies in enabling P_1 to securely obtain $([\![a]\!], [\![b]\!], [\![ab]\!])$ and $(u-a, v-b)$. We first consider the two-party setting. The generation of the ciphertext triple $([\![a]\!], [\![b]\!], [\![ab]\!])$ is straightforward, i.e., P_2 simply samples two random values a, b and computes the desired ciphertexts. This requires no any homomorphic operations. Enabling P_1 to learn $(u-a, v-b)$ presents more subtle security considerations. While P_1 can easily compute the encryptions of $u - a$ and $v - b$ using only additive homomorphism, we need a secure decryption protocol to prevent P_2 from learning information about $u - a$ and $v - b$.

We utilize an *oblivious decryption* technique enabled by the additive homomorphism property of the encryption scheme. The protocol proceeds as follows: P_1 first randomizes the target ciphertext $[\![x]\!]$ by subtracting a random mask r, then sends this masked ciphertext $[\![x - r]\!]$ to P_2 for decryption. P_2 returns the decrypted result, which P_1 then unmasks to recover the desired value x.

We further extend our protocol to the multiparty setting by employing a distributed LHE scheme where a group of k parties (specifically, $k = n - 1$ in our case) can collaboratively generate a key pair $(\mathsf{pk}, \mathsf{sk})$ while ensuring that: (1) no coalition of $k-1$ parties can recover any information about the secret key, and (2) all k parties together can reconstruct the secret key. For generating the ciphertext triple $([\![a]\!], [\![b]\!], [\![ab]\!])$, each party P_i for $i \in [2, n]$ samples random values a_i and b_i, then sends their encryptions $([\![a_i]\!], [\![b_i]\!])$ under pk to P_1. Using these ciphertexts, P_1 homomorphically computes the encrypted values $[\![a]\!] \leftarrow \sum_{i\in[2,n]}[\![a_i]\!]$ and $[\![b]\!] \leftarrow \sum_{i\in[2,n]}[\![b_i]\!]$. To obtain the encrypted product $[\![c]\!]$ where $c = ab$, P_1 sends $[\![a]\!]$ to other parties who each compute $[\![c_i]\!] = b_i[\![a]\!]$ and return the results, allowing P_1 to finally compute $[\![c]\!] \leftarrow \sum_{i\in[2,n]}[\![c_i]\!]$.

A critical remaining step is enabling P_1 to securely obtain the plaintext values $u - a$ and $v - b$. This requires a distributed decryption protocol where P_1 provides an input ciphertext while other parties jointly hold the secret key. To maintain the overall protocol complexity at $O(mn)$, the distributed decryption must operate in $O(n)$ complexity. We will demonstrate how to instantiate the required LHE with existing schemes.

Protecting non-topological circuit information. To construct PFE that hides circuit information, we begin by considering a weaker variant called semi-PFE, which aims to protect partial information about the circuit C. Specifically, we focus on hiding all information except the circuit topology, which has been previously explored in [PSS09]. We call this semi-PFE topology-revealing PFE (trPFE). We give a new and efficient approach to achieve trPFE. Note that the key requirement here is to prevent all parties other than P_1 from distinguishing whether the computed gate is an addition gate or a multiplication gate.

In our SFE framework, addition gates can be computed locally by P_1 using linearly-homomorphic operations, while multiplication gates require interaction with other parties. This interaction involves two main tasks: generating encrypted triples and performing oblivious decryption. Importantly, the generation of encrypted triples does not depend on any intermediate values of the circuit evaluation, meaning it can be completed in a setup stage.

Note that preventing all parties other than P_1 from distinguishing between addition gates and multiplication gates also implies that the number m^* of multiplication gates cannot be leaked. Although P_1 only requires m^* encrypted triples, we still instruct the parties to generate m encrypted triples in advance, where m is the total number of addition and multiplication gates in the circuit. This ensures that $P_2, \ldots, P_n$ cannot infer information about m^*.

To hide gate types, we have the parties also perform oblivious decryption when computing addition gates. Since $P_2, \ldots, P_n$ always use the same input (the secret key sharing) and does not receive any output from the oblivious decryption functionality, they cannot determine whether an addition or multiplication gate is being computed. This approach effectively hides all circuit information from $P_2, \ldots, P_n$ except the circuit topology, achieving the goal of trPFE.

Achieving PFE: revealing only circuit size. We further upgrade the trPFE construction to PFE. In our trPFE protocol, the parties compute the circuit in a layer-by-layer manner. This will leak two types of information about the circuit: the depth and number of gates in each layer. Since the circuit depth ranges over $[1, m]$, we enforce exactly m rounds of computation. Additionally, we restrict the parties to evaluating only one gate per round. Consequently, we can ensure no leakage of topological information.

Compared to our trPFE, the PFE protocol imposes only a limitation of evaluating a single gate per round, while introducing no additional cost. This means that our PFE can be as efficient as trPFE in certain amortized settings, even though our protocol has a higher round complexity. For example, if the computational power of the parties allows them to evaluate k gates per round, our PFE takes mT runtime per execution with T being the duration of one round since the parties can only compute a single gate per round. As a comparison, our trPFE protocol can achieve mT/k runtime. However, when evaluating k circuits of size m simultaneously, both schemes achieve mT total runtime. Thus, the round complexity disadvantage vanishes in this amortized setting.

Unifying GMW and LHE-based SFE: an alternative PFE construction. We observe that our protocol shares a similar design with the well-known GMW protocol [GMW87], where the parties evaluate the circuit gate-by-gate while maintaining each gate's output in some encoded form. In GMW, this encoding is an additive sharing, whereas in our LHE-based construction it is an LHE ciphertext. We note that in the two-party case, additive sharing can be viewed as a one-time pad (OTP) encryption, which motivates our unification of both protocols through encryption. Specifically, we propose an encryption-based SFE paradigm that can work with any encryption scheme, where each gate's output is computed as a ciphertext under the chosen encryption scheme. Instantiating this paradigm with OTP yields the GMW protocol, while using LHE produces our LHE-based scheme.

We also consider instantiating our SFE paradigm using an SKE scheme, and then show how to further achieve circuit privacy by obfuscating the evaluation of addition and multiplication gates, thereby obtaining an alternative PFE construction with linear complexity. Since the resulting protocol does not rely on LHE (and therefore avoids homomorphic operations), it can achieve higher computational efficiency. The tradeoff is that this SKE-based construction incurs higher communication complexity.

Reducing round complexity by revealing depth. Compared to trPFE, our PFE achieves stronger privacy guarantees at the cost of increased round complexity from d to m, where d represents circuit depth. We prove that revealing the circuit topology is unnecessary for achieving $O(d)$ round complexity. Specifically, we show that semi-PFE leaking circuit *depth* suffices. We call this protocol depth-revealing PFE (drPFE). Our drPFE is remarkably simple: participants compute $\lceil m/(\varepsilon d) \rceil$ gates per round ($\varepsilon > 0$ is an adjustable constant), with the constraint that all gates must belong to the same circuit layer. When the number of gates in a layer is not divided by $\lceil m/(\varepsilon d) \rceil$, the parties evaluate some *dummy* gates. Now we analyze the complexity of the resulting protocol.

Assume the j-th layer contains m_j gates, then the round complexity is

$$\sum_{j \in [d]} \lceil \frac{m_j}{\lceil m/(\varepsilon d) \rceil} \rceil < \sum_{j \in [d]} (\frac{m_j}{m/(\varepsilon d)} + 1) = (1 + \varepsilon) d.$$

This implies that the protocol can be accomplished within $(1 + \varepsilon)d = O(d)$ rounds. The parameter ε controls the tradeoff between round complexity and computational overhead. To see this, note that the

number of computed gates is

$$\sum_{j\in[d]} \lceil m/(\varepsilon d)\rceil \cdot \lceil \frac{m_j}{\lceil m/(\varepsilon d)\rceil}\rceil < \sum_{j\in[d]} \lceil m/(\varepsilon d)\rceil (\frac{m_j}{\lceil m/(\varepsilon d)\rceil} + 1) = m + d\lceil m/(\varepsilon d)\rceil.$$

This means the smaller ε can result in lower round complexity and higher computational and communication complexity.

Compared to trPFE that leaks circuit topology, drPFE protocol reveals only circuit depth. From an information perspective, this represents a significant reduction in information leakage. For circuits with m gates, the total number of possible circuits exceeds $O(m^m)$ [Vol99], meaning the amount of information that needs protection in PFE is at least $O(m \log m)$ bits. However, once the circuit topology is revealed, the number of possible circuits reduces to just 2^m, corresponding to only m bits of remaining information. This demonstrates that disclosing the circuit topology would expose the majority of the circuit information. In contrast, since the depth d can only range between 1 and m, revealing the depth leaks merely $\log m$ bits of information, representing a substantially smaller information leakage. This brings drPFE remarkably close to standard PFE in terms of the amount of information leakage.

PFE from minimal homomorphism. We prove that rerandomizable public-key encryption (RPKE) suffices to construct a two-party PFE scheme with linear (in circuit size m) complexity for Boolean circuits. In particular, the construction only requires black-box use of the RPKE scheme. To establish this result, we first show that when encrypting bits, rerandomization implies a stronger homomorphic property we call *plain homomorphism*. In particular, given a ciphertext of a bit $[\![x]\!]$ and two plaintext bits a, b, plain homomorphism enables the generation of a ciphertext $[\![ax \oplus b]\!]$. Note that plain homomorphism remains weaker than linear homomorphism since it does not support XOR operations between ciphertexts. To construct our PFE protocol, we eliminate the need for XOR between ciphertexts when evaluating both XOR and AND gates. Consequently, all required homomorphic operations can be realized solely using plain homomorphism, implied by rerandomization.

1.3 Discussion: Why CTH is Unnecessary in Our PFE Framework

In the PFE framework of Mohassel and Sadeghian [MS13], two functionalities are defined: circuit-topology hiding (CTH) and private gate evaluation (PGE). The CTH functionality ensures that a party unaware of the circuit (say, P_2) can correctly identify and use the proper input shares for each gate during evaluation, while PGE guarantees that P_2 remains oblivious to the type of gate being evaluated.

A critical observation is that their protocols rely on either additive secret sharing (for arithmetic circuits) or Yao's garbled sharing (for Boolean circuits). A common feature of these secret-sharing schemes is that any party has different shares for different secrets, which implies that P_2 who lacks knowledge of the circuit topology cannot determine which shares correspond to the inputs of the gate being computed. This necessitates the use of the CTH functionality, which guides P_2 to generate the correct shares without revealing information about circuit topology.

In contrast, our framework eliminates the need for CTH by adopting an encryption-based sharing scheme, where a sharing of a secret x is of form (e, sk), with e being the ciphertext of x under the secret key sk. The key advantage of this sharing scheme is that P_2's share, sk, remains constant for different secrets. Consequently, P_2 no longer faces the risk of using incorrect shares during gate evaluation, as its input is always the fixed share sk. This allows our framework to achieve PFE without relying on the CTH functionality.

2 Preliminaries

2.1 Notations

Let λ be the security parameter. For any integer a, b, we denote $[a, b]$ the set $\{a, a+1, \ldots, b\}$. If $a > b$, $[a, b]$ is defined to be the empty set $\varnothing$. Furthermore, we abbreviate $[1, b]$ to $[b]$. A function $\varepsilon(\lambda)$ is negligible (in λ), denoted $\varepsilon(\lambda) = \mathsf{negl}(\lambda)$, if $\varepsilon(\lambda) = O(\lambda^{-c})$ for any $c > 0$. Two distributions X and Y are computationally indistinguishable, denoted $X \approx_c Y$ if no PPT algorithm can distinguish them with a non-negligible advantage. Further, X and Y are statistically indistinguishable, denoted $X \approx_s Y$, if their statistical distance is negligible in λ.

For a value x over a ring or field, we use $[\![x]\!]$ to represent an encryption of x. Performing arithmetic operations (addition or multiplication) on ciphertexts refers to utilizing the homomorphic properties of the encryption scheme to compute ciphertexts corresponding to the transformed plaintext results. For instance, the operation $a[\![x]\!] + [\![y]\!]$ yields the ciphertext $[\![ax+y]\!]$, which requires the underlying encryption scheme to be linearly homomorphic.

2.2 Security Model

We assume the adversary $\mathcal{A}$ is static, requiring it to select which parties to compromise before the protocol begins. Moreover, the adversary is semi-honest, implying that the corrupted parties faithfully follow the protocol specification while potentially analyzing their views to extract additional information. Finally, all entities, including both adversarial and honest parties, are computationally bounded, formally modeled as probabilistic polynomial-time interactive Turing machines (PPT ITMs). Our security proofs are established within the universal composability (UC) framework introduced by Canetti [Can01]. We describe the UC framework in Appendix A.

2.3 Public-Key Encryption and Its Homomorphic Variants

We revisit the standard definition of public-key encryption (PKE) and two of its homomorphic variants called rerandomizable PKE (RPKE) and linearly-homomorphic encryption (LHE) in Appendix B.

2.4 Distributed LHE

Our multiparty PFE protocol requires a distributed variant of LHE featuring two key components: (1) a distributed key generation protocol that allows k parties to collaboratively generate a key pair while ensuring that any subset of $k-1$ parties learns no information about the secret key, and (2) a distributed decryption algorithm that enables ciphertext decryption without full secret key reconstruction. The distributed key generation must guarantee that only when all k parties cooperate can the secret key be recovered. Crucially, to achieve linear complexity (with respect to the number of parties) in our multiparty PFE constructions, both the distributed key generation and distributed decryption protocols must operate with $O(k)$ complexity.

All of the Paillier [Pai99], BCP [BCP03], and CL [CL15] schemes can be extended to the distributed setting. The choice of underlying cryptosystem determines the arithmetic circuits in our PFE constructions. When using the Paillier or BCP scheme, the message space is $\mathbb{Z}_N$ for an RSA modulus N, hence the circuit is over $\mathbb{Z}_N$. When using the CL scheme, the message space is $\mathbb{Z}_p$ for a large prime p, hence the circuit is over $\mathbb{Z}_p$. A distributed variant of Paillier can be found in [BDTZ16]. However, we did not find a distributed version of the BCP scheme. Although this extension is straightforward, we provide a detailed description of distributed BCP in Appendix C for completeness. In particular, our distributed key generation protocol requires $P_2, \ldots, P_n$ to jointly generate shares of the secret key while ensuring that all parties, including

P_1, obtain the corresponding public key. For distributed BCP, we analyze that the communication cost of this step is $2n-3$ group elements. Furthermore, our distributed decryption protocol requires $P_2, \ldots, P_n$ to jointly assist P_1 in decrypting a BCP ciphertext held by P_1, without learning any information about the underlying plaintext. For distributed BCP, we analyze that the communication cost of this step is $2n-2$ group elements. The complexity analysis of our SFE and PFE protocols with be conducted by instantiating LHE with the distributed BCP scheme. Finally, we remark that the distributed version of the CL scheme follows an analogous algorithm description with different parameter specification.

Remark 2 (Ciphertext Size Reduction). In the BCP scheme (and other ElGamal-like schemes like the CL scheme), each ciphertext is of the form $(g^r, f^x \cdot \mathsf{pk}^r)$, consisting of two group elements. We can reduce the ciphertext size by nearly half when encrypting many messages. Specifically, to encrypt k messages $x_1, \ldots, x_k$, the scheme works as follows:

1. Generate k key pairs $(\mathsf{pk}_1, \mathsf{sk}_1), \ldots, (\mathsf{pk}_k, \mathsf{sk}_k)$.
2. The ciphertext is $(g^r, f^{x_1} \cdot \mathsf{pk}_1^r, \ldots, f^{x_k} \cdot \mathsf{pk}_k^r)$.

The size of the resulting ciphertext is $k+1$ group elements instead of $2k$. When k is large, this optimization reduces the communication cost by nearly a factor of 2. Although the public key size increases (from a single group element to k group elements), it is transmitted only once.

2.5 Oblivious Decryption

Our SFE and PFE require the ability to decrypt a ciphertext in an oblivious manner, where P_1 holds the ciphertext e while the other parties possess the secret key (in the two-party case, P_2 holds the secret key, whereas in the multiparty setting, $P_2, \ldots, P_n$ hold a sharing of the secret key). The goal is to allow P_1 to obtain the decrypted result of e without learning the secret key, while simultaneously preventing the other parties from learning e. To formalize this, we define an oblivious decryption functionality as follows.

Oblivious Decryption Functionality $\mathcal{F}_{\mathsf{od}}$

The parties agree on a LHE scheme with ciphertext space $\mathcal{C}$.

1. Receive a ciphertext $e \in \mathcal{C}$ from P_1.
2. Receive an additive sharing $(\mathsf{sk}_2, \ldots, \mathsf{sk}_n)$ of the secret key from the parties $P_2, \ldots, P_n$.
3. Recover the secret key sk and use it to decrypt e to x.
4. Send x to P_1.

We implement the oblivious decryption functionality by leveraging the linear homomorphism of the underlying encryption scheme. Specifically, our protocol proceeds as follows.

1. Let $e = [\![x]\!]$ is the ciphertext to be decrypted.
2. P_1 samples a random value $r \in \mathcal{M}$ and computes the masked ciphertext $[\![x']\!] \leftarrow [\![x]\!] - r$.
3. $P_2, \ldots, P_n$ execute a distributed decryption protocol on $[\![x']\!]$, with P_1 obtaining the output x'.
4. P_1 outputs $x = x' + r$.

Instantiating the above execution with the distributed BCP cryptosystem yields an identical communication cost to BCP distributed decryption (see Appendix C), specifically $2n-2$ group elements.

3 A New Framework for SFE from LHE

In this section, we present a new two-party and multiparty SFE framework. Our two-party PFE builds upon an LHE scheme, while our multiparty PFE relies on a distributed LHE scheme. Our SFE framework adopts

a design strategy similar to the well-known GMW protocol [GMW87], where both approaches compute circuits gate-by-gate while maintaining specific output encodings of each gate. The main difference lies in that GMW employs the additive secret sharing to encode the outputs, whereas our SFE framework utilizes an encryption-based sharing. Specifically, for each gate output, one party holds the ciphertext while others share the secret key. This encryption-based sharing scheme exhibits an important property that only the ciphertext share varies across different sharings while the secret key shares remain constant. This enables our scheme to have some new features compared to GMW.

First, in our SFE framework, only one party (say, P_1) needs to know the computed circuit, eliminating the need to disclose circuit details to other parties in advance. This simplifies practical deployment by removing the requirement for all parties to agree on the target circuit, an inevitable step in GMW. Note that the protocol execution may leak circuit information in our SFE framework, which violates the privacy requirements of PFE.

Second, both GMW and our SFE framework evaluate circuits layer-by-layer, achieving linear round complexity in the circuit depth. In GMW, this necessitates all parties maintaining storage proportional to the circuit width, as they must maintain the computational results of all gates from the previous layer until the current layer's computation is complete. In contrast, our SFE framework requires only P_1 to maintain circuit-width storage, while other parties need only constant storage independent of the circuit size. This is because in our SFE framework, all parties except P_1 only perform decryption (in an oblivious manner) for P_1. Consequently, they only need to store the key and the resources required for oblivious decryption, where the latter is independent of the circuit size.

Finally, our protocol requires only a star topology, i.e., each party only needs to interact with one designated party P_1. Specifically, the communication cost between P_1 and each other party is independent of the number n of parties, thereby achieving linear $O(n)$ total communication complexity. We note that any protocol's (including GMW) interaction pattern can also be transformed into a star-shaped link by having P_1 act as a message relay (with the messages encrypted using an encryption scheme). However, unlike our SFE protocol, the resulting protocol cannot guarantee that the communication cost between P_1 and each other party is independent of n.

3.1 Two-Party SFE

Our two-party SFE protocol utilizes an LHE scheme $(\mathsf{Keygen}, \mathsf{Enc}, \mathsf{Dec})$ with message space $\mathcal{M}$. The computed circuit is also over $\mathcal{M}$. Without loss of generality, we assume each party has a single input. We provide the detailed description of our two-party SFE protocol, where we assume the parties have access to the oblivious decryption functionality $\mathcal{F}_{\mathsf{od}}$.

Protocol 1. Two-Party SFE from LHE

Parameter: A circuit C over the ring $\mathcal{M}$. Let m^* be the number of multiplication gates in C.
Input: P_1 has $x_1 \in \mathcal{M}$. P_2 has $x_2 \in \mathcal{M}$.
Output: P_1 obtains $C(x_1, x_2)$.

Setup.

1. P_2 samples a key pair $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{Keygen}(1^\lambda)$.
2. P_2 samples (a_j, b_j) and computes $c_j = a_j b_j$ for all $j \in [m^*]$.
3. P_2 encrypts (a_j, b_j, c_j) to $(\llbracket a_j \rrbracket, \llbracket b_j \rrbracket, \llbracket c_j \rrbracket)$ for all $j \in [m^*]$ under pk.
4. P_2 sends $(\mathsf{pk}, \{\llbracket a_j \rrbracket, \llbracket b_j \rrbracket, \llbracket c_j \rrbracket\}_{j \in [m^*]})$ to P_1.

Input encoding.

1. P_1 encrypts x_1 to $[\![x_1]\!]$ under pk.
2. P_2 encrypts x_2 to $[\![x_2]\!]$ under pk and sends $[\![x_2]\!]$ to P_1.

Circuit evaluation.

Evaluating an addition gate. To compute an addition gate with input enryptions $[\![u]\!], [\![v]\!]$, P_1 computes $[\![w]\!] \leftarrow [\![u]\!] + [\![v]\!]$.

Evaluating a multiplication gate. To compute the j-th multiplication gate with input enryptions $[\![u]\!], [\![v]\!]$, the parties perform the following procedures.

1. P_1 computes $[\![s]\!] \leftarrow [\![u]\!] - [\![a_j]\!], [\![t]\!] \leftarrow [\![v]\!] - [\![b_j]\!]$.
2. The parties invoke the oblivious decryption functionality $\mathcal{F}_{\mathsf{od}}$ to decrypt $[\![s]\!]$ and $[\![t]\!]$ with P_1 obtaining the output s, t.
3. P_1 computes $[\![w]\!] \leftarrow [\![c_j]\!] + s[\![b_j]\!] + t[\![a_j]\!] + st$.

Output decoding.

1. Let $[\![z]\!]$ be the ciphertext corresponding to the output gate.
2. The parties invoke the oblivious decryption functionality $\mathcal{F}_{\mathsf{od}}$ to decrypt $[\![z]\!]$ with P_1 obtaining the output z.
3. P_1 outputs z.

Theorem 1. *Assume the underlying LHE scheme is IND-CPA secure, then Protocol 1 securely computes the circuit C in the UC model against semi-honest, static adversaries.*

sketch. The correctness of the protocol is straightforward to verify. Regarding privacy, we provide a brief sketch of the argument here, while deferring the complete security proof to Appendix D.1.

The input privacy of P_1 is easily verified since P_2 receives no messages from either P_1 or the functionality $\mathcal{F}_{\mathsf{od}}$ throughout the protocol execution. For P_2's input privacy, we observe that P_1, lacking the secret key, gains no information from any ciphertexts. The only plaintext values P_1 obtains are $s = u - a_j$ and $t = v - b_j$. Crucially, since a_j and b_j are uniformly random values sampled by P_2, the values s and t are themselves uniformly random, thus revealing no information about P_2's private inputs. □

3.2 Multiparty SFE

Our multiparty SFE employs a distributed LHE scheme where parties collaboratively generate a public key while $P_2, \ldots, P_n$ obtain a secret key sharing. This ensures that any $n-2$ parties among $P_2, \ldots, P_n$ learn no information about the secret key, yet can collectively reconstruct it. The key distinction between our multiparty and two-party SFE lies in ciphertext triple generation. In the two-party case, P_2 alone samples ciphertext triples with full knowledge of their plaintext values, whereas our multiparty protocol requires $P_2, \ldots, P_n$ to jointly generate triples such that any $n-2$ of them remain oblivious to the underlying plaintexts. The detailed protocol description is as follows.

Protocol 2. Multiparty SFE from LHE

Public parameters: A circuit C over the ring $\mathcal{M}$. Let m^* be the number of multiplication gates in C.
Input: Each party P_i has an input x_i for $i \in [n]$.
Output: P_1 obtains $C(x_1, \ldots, x_n)$.

Setup.

1. The parties perform distributed key generation to agree on a public key pk. Meanwhile, $P_2, \ldots, P_n$ obtain a sharing of the secret key sk such that any $n-2$ of them have no information about sk.
2. For all $i \in [2, n]$, P_i samples $(a_{j,i}, b_{j,i})$ and encrypts them to $([\![a_{j,i}]\!], [\![b_{j,i}]\!])$ for all $j \in [m^*]$.
3. For all $i \in [2, n]$, P_i sends $\{[\![a_{j,i}]\!], [\![b_{j,i}]\!]\}_{j\in[m^*]}$ to P_1.
4. P_1 computes $[\![a_j]\!] \leftarrow \sum_{i\in[2,n]} [\![a_{j,i}]\!]$, $[\![b_j]\!] \leftarrow \sum_{i\in[2,n]} [\![b_{j,i}]\!]$ for all $j \in [m^*]$.
5. P_1 sends $\{[\![a_j]\!]\}_{j\in[m^*]}$ to all other parties.
6. For all $i \in [2, n]$, P_i computes $[\![c_{j,i}]\!] \leftarrow b_{j,i}[\![a_j]\!]$ for all $j \in [m^*]$ and sends $\{[\![c_{j,i}]\!]\}_{j\in[m^*]}$ to P_1.
7. P_1 computes $[\![c_j]\!] \leftarrow \sum_{i\in[2,n]} [\![c_{j,i}]\!]$ for all $j \in [m^*]$.
8. P_1 stores $(\mathsf{pk}, \{[\![a_j]\!], [\![b_j]\!], [\![c_j]\!]\}_{j\in[m^*]})$.

Input encoding.

1. Each party P_i encrypts x_i to $[\![x_i]\!]$ under pk.
2. For all $i \in [2, n]$, P_i sends $[\![x_i]\!]$ to P_1.

Circuit evaluation.

Evaluating an addition gate. To compute an addition gate with input enryptions $[\![u]\!], [\![v]\!]$, P_1 computes $[\![w]\!] \leftarrow [\![u]\!] + [\![v]\!]$.

Evaluating a multiplication gate. To compute the j-th multiplication gate with input enryptions $[\![u]\!], [\![v]\!]$, the parties perform the following procedures.

1. P_1 computes $[\![s]\!] \leftarrow [\![u]\!] - [\![a_j]\!]$, $[\![t]\!] \leftarrow [\![v]\!] - [\![b_j]\!]$.
2. The parties invoke the oblivious decryption functionality $\mathcal{F}_{\mathsf{od}}$ to decrypt $[\![s]\!]$ and $[\![t]\!]$ with P_1 obtaining the output s, t.
3. P_1 computes $[\![w]\!] \leftarrow [\![c_j]\!] + s[\![b_j]\!] + t[\![a_j]\!] + st$.

Output decoding.

1. Let $[\![z]\!]$ be the ciphertext corresponding to the output gate.
2. The parties invoke the oblivious decryption functionality $\mathcal{F}_{\mathsf{od}}$ to decrypt $[\![z]\!]$ with P_1 obtaining the output z.
3. P_1 outputs z.

Theorem 2. *Assume the underlying LHE scheme is IND-CPA secure, then Protocol 2 securely computes the circuit C in the UC model against semi-honest, static adversaries.*

sketch. The complete security proof appears in Appendix D.1. We provide a privacy sketch considering two cases: either P_1 is honest or corrupted. When P_1 is honest, the corrupted parties only receive messages during the setup stage. Since the setup execution is input-independent, these messages contain no information about honest parties' inputs and can be perfectly simulated. When P_1 is corrupted, the adversary cannot access the secret key as at least one honest party P_h exists among $P_2, \ldots, P_n$. Consequently, all ciphertexts observed by the adversary reveal no information about the plaintexts. The only plaintext values obtained are $s = u - a_j$ and $t = v - b_j$. Crucially, since $a_j = \sum_{i\in[2,n]} a_{j,i}$ and $b_j = \sum_{i\in[2,n]} b_{j,i}$ contain random components $a_{j,h}$ and $b_{j,h}$ generated by P_h, both a_j and b_j are uniformly distributed. This ensures s and t leak no information about honest parties' inputs. □

3.3 Complexity Analysis

We analyze both the asymptotic complexity and concrete communication complexity of our SFE protocols. When analyzing the concrete communication complexity, we assume LHE is instantiated with the BCP scheme described in Section 2.4. The analysis would be similar if substituting the underlying LHE scheme with alternative constructions. Recall that the BCP scheme represents each ciphertext with two group elements. Moreover, our analysis in Sections 2.4 and 2.5 demonstrates that distributed key generation requires $2n-3$ group elements of communication and oblivious decryption requires $2n-2$ group elements.

Two-party SFE. It is straightforward to see that Protocol 1 achieves $O(m^*)$ asymptotic communication complexity and $O(m)$ asymptotic computational complexity. Next we analyze the concrete communication complexity.

The concrete communication cost of Protocol 1 is characterized as follows: the setup stage requires distributing a public key and $3m^*$ ciphertexts; the input encoding stage involves sending a ciphertext; the circuit evaluation stage necessitates $2m^*$ calls to oblivious decryption; and the output stage requires one call to oblivious decryption. Thus the total communication cost is $1+3m^*\cdot 2+2+(2m^*+1)\cdot 2 = 10m^*+5$ group elements.

Multiparty SFE. It is easy to see that the asymptotic communication complexity of Protocol 2 is $O(m^*n)$. Regarding the asymptotic computational complexity, we note that for addition gates *only* P_1 needs to perform computation. Thus, the protocol's asymptotic computational complexity is $O(m + m^*n)$. Notably, previous multiparty SFE protocols against $n-1$ corruptions (e.g., the GMW protocol) require at least $O(mn)$ computational complexity, making the asymptotic computational complexity one of the advantages of our multiparty SFE protocol.

The concrete communication cost of Protocol 2 is characterized as follows: the setup stage requires distributed key generation and distributing $(4n-4)m^*$ ciphertexts; the input encoding stage involves sending $n-1$ ciphertexts; the circuit evaluation stage necessitates $2m^*$ calls to oblivious decryption; and the output stage requires one call to oblivious decryption. This yields a communication cost of $(2n-3)+(4n-4)m^*\cdot 2+(n-1)\cdot 2+(2m^*+1)\cdot(2n-2) = (12n-12)m^*+6n-7$ group elements.

4 PFE with Linear Complexity

In this section, we present the upgrade from our SFE (Section 3) to PFE, with two key modifications. First, since the exact number m^* of multiplication gates is sensitive information in PFE, we modify the setup stage to generate m ciphertext triples rather than m^*. Second, the protocol requires the parties to invoke the oblivious decryption functionality for both addition and multiplication gates, preventing any party except P_1 from learning the type of the computed gate.

4.1 Two-Party PFE

Protocol 3. Two-Party PFE from LHE

Parameter: m is the size of the computed circuit.
Input: P_1 has a private circuit C of size m and $x_1 \in \mathcal{M}$. P_2 has $x_2 \in \mathcal{M}$.
Output: P_1 obtains $C(x_1, x_2)$.

Setup.

1. P_2 samples a key pair $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{Keygen}(1^\lambda)$.
2. P_2 samples (a_j, b_j) and computes $c_j = a_j b_j$ for all $j \in [m]$.

3. P_2 encrypts (a_j, b_j, c_j) to $([\![a_j]\!], [\![b_j]\!], [\![c_j]\!])$ for all $j \in [m]$ under pk.
4. P_2 sends $(\mathsf{pk}, \{[\![a_j]\!], [\![b_j]\!], [\![c_j]\!]\}_{j\in[m]})$ to P_1.

Input encoding.

1. P_1 encrypts x_1 to $[\![x_1]\!]$ under pk.
2. P_2 encrypts x_2 to $[\![x_2]\!]$ under pk and sends $[\![x_2]\!]$ to P_1.

Circuit evaluation.

To compute the j-th gate with input enryptions $[\![u]\!], [\![v]\!]$, the parties perform the following procedures.

1. If the computed gate is an addition gate, P_1 sets $s = t = 0$ and encrypts them to $[\![s]\!], [\![t]\!]$. Otherwise, P_1 computes $[\![s]\!] \leftarrow [\![u]\!] - [\![a_j]\!]$ and $[\![t]\!] \leftarrow [\![v]\!] - [\![b_j]\!]$.
2. The parties invoke the oblivious decryption functionality $\mathcal{F}_{\mathsf{od}}$ to decrypt $[\![s]\!]$ and $[\![t]\!]$ with P_1 obtaining the output s, t.
3. If the computed gate is an addition gate, P_1 computes $[\![w]\!] \leftarrow [\![u]\!] + [\![v]\!]$. Otherwise, P_1 computes $[\![w]\!] \leftarrow [\![c_j]\!] + s[\![b_j]\!] + t[\![a_j]\!] + st$.

Output decoding.

1. Let $[\![z]\!]$ be the ciphertext corresponding to the output gate.
2. The parties invoke the oblivious decryption functionality $\mathcal{F}_{\mathsf{od}}$ to decrypt $[\![z]\!]$ with P_1 obtaining the output z.
3. P_1 outputs z.

Theorem 3. *Assume the underlying LHE scheme is IND-CPA secure, then Protocol 3 securely computes the private circuit C in the UC model against semi-honest, static adversaries.*

sketch. The input privacy sketch follows essentially the same reasoning as in Theorem 1. For circuit privacy, we observe that P_2's behavior remains identical when computing both addition and multiplication gates. This ensures no information about the gate types is revealed. The complete proof appears in Appendix D.3. □

4.2 Multiparty PFE

Protocol 4. Multiparty PFE from LHE

Parameter: m is the size of the computed circuit.
Input: P_1 has a private circuit C of size m. Each party P_i has $x_i \in \mathcal{M}$.
Output: P_1 obtains $C(x_1, \ldots, x_n)$.

Setup.

1. The parties perform distributed key generation to agree on a public key pk. Meanwhile, $P_2, \ldots, P_n$ obtain a sharing of the secret key sk such that any $n-2$ of them have no information about sk.
2. For all $i \in [2, n]$, P_i samples $(a_{j,i}, b_{j,i})$ and encrypts them to $([\![a_{j,i}]\!], [\![b_{j,i}]\!])$ for all $j \in [m]$.
3. For all $i \in [2, n]$, P_i sends $\{[\![a_{j,i}]\!], [\![b_{j,i}]\!]\}_{j\in[m]}$ to P_1.
4. P_1 computes $[\![a_j]\!] \leftarrow \sum_{i\in[2,n]} [\![a_{j,i}]\!]$, $[\![b_j]\!] \leftarrow \sum_{i\in[2,n]} [\![b_{j,i}]\!]$ for all $j \in [m]$.
5. P_1 sends $\{[\![a_j]\!]\}_{j\in[m]}$ to all other parties.
6. For all $i \in [2, n]$, P_i computes $[\![c_{j,i}]\!] \leftarrow b_{j,i}[\![a_j]\!]$ for all $j \in [m]$ and sends $\{[\![c_{j,i}]\!]\}_{j\in[m]}$ to P_1.

7. P_1 computes $[\![c_j]\!] \leftarrow \sum_{i\in[2,n]}[\![c_{j,i}]\!]$ for all $j \in [m]$.
8. P_1 stores $(\mathsf{pk}, \{[\![a_j]\!], [\![b_j]\!], [\![c_j]\!]\}_{j\in[m]})$.

Input encoding.

1. Each party P_i encrypts x_i to $[\![x_i]\!]$ under pk.
2. For all $i \in [2, n]$, P_i sends $[\![x_i]\!]$ to P_1.

Circuit evaluation.

To compute the j-th gate with input enryptions $[\![u]\!], [\![v]\!]$, the parties perform the following procedures.

1. If the computed gate is an addition gate, P_1 sets $s = t = 0$ and encrypts them to $[\![s]\!], [\![t]\!]$. Otherwise, P_1 computes $[\![s]\!] \leftarrow [\![u]\!] - [\![a_j]\!]$ and $[\![t]\!] \leftarrow [\![v]\!] - [\![b_j]\!]$.
2. The parties invoke the oblivious decryption functionality $\mathcal{F}_{\mathsf{od}}$ to decrypt $[\![s]\!]$ and $[\![t]\!]$ with P_1 obtaining the output s, t.
3. If the computed gate is an addition gate, P_1 computes $[\![w]\!] \leftarrow [\![u]\!] + [\![v]\!]$. Otherwise, P_1 computes $[\![w]\!] \leftarrow [\![c_j]\!] + s[\![b_j]\!] + t[\![a_j]\!] + st$.

Output decoding.

1. Let $[\![z]\!]$ be the ciphertext corresponding to the output gate.
2. The parties invoke the oblivious decryption functionality $\mathcal{F}_{\mathsf{od}}$ to decrypt $[\![z]\!]$ with P_1 obtaining the output z.
3. P_1 outputs z.

Theorem 4. *Assume the underlying LHE scheme is IND-CPA secure, then Protocol 4 securely computes the private circuit C in the UC model against semi-honest, static adversaries.*

sketch. The input privacy sketch follows essentially the same reasoning as in Theorem 2. For circuit privacy, we observe that the behavior of $P_2, \ldots, P_n$ remains identical when computing both addition and multiplication gates. This ensures no information about the gate types is revealed. The complete proof appears in Appendix D.4. □

4.3 Complexity Analysis

We analyze both the asymptotic complexity and concrete communication complexity of our PFE protocols. In particular, when analyzing the concrete communication complexity, we assume LHE is instantiated with the BCP scheme described in Section 2.4.

Two-party PFE. It is easy to verify that Protocol 3 achieves $O(m)$ complexity for both communication and computation asymptotically. The concrete communication cost of Protocol 3 is characterized as follows: the setup stage requires distributing a public key and $3m$ ciphertexts; the input encoding stage involves sending a ciphertext; the circuit evaluation stage necessitates $2m$ calls to oblivious decryption; and the output stage requires one call to oblivious decryption. This yields a total communication cost of $1 + 3m \cdot 2 + 2 + (2m + 1) \cdot 2 = 10m + 5$ group elements.

Multiparty PFE. Protocol 4 achieves $O(mn)$ asymptotic complexity for both communication and computation. The concrete communication cost of Protocol 4 is characterized as follows: the setup stage requires distributed key generation and distributing $(4n - 4)m$ ciphertexts; the input encoding stage involves sending $n-1$ ciphertexts; the circuit evaluation stage necessitates $2m$ calls to oblivious decryption; and the output stage requires one call to oblivious decryption. This yields a total communication cost of $2n - 3 + (4n - 4)m \cdot 2 + (n - 1) \cdot 2 + (2m + 1) \cdot (2n - 2) = (12n - 12)m + 6n - 7$ group elements.

5 A Unified Paradigm for SFE from Encryption

Our LHE-based SFE framework follows the same high-level design strategy as the classic GMW protocol [GMW87]: the parties evaluate the target circuit gate by gate, and the output of each gate is represented in an encoded form such that no coalition of up to $n-1$ parties can recover its value. In GMW, this encoding takes the form of additive secret sharing, whereas in our framework it consists of a ciphertext together with a secret key that is shared among $n-1$ parties (in the two-party case, one party holds the ciphertext and the other holds the secret key).

This structural similarity motivates a unified SFE paradigm based on encryption. The starting point is a simple but important observation: additive secret sharing can be interpreted as a form of encryption. In particular, the two-party additive secret sharing scheme corresponds to a standard one-time pad (OTP) encryption scheme, while its multiparty version corresponds to a distributed variant of OTP.

Our unified SFE paradigm abstracts both GMW and our LHE-based SFE protocol. The paradigm is parameterized by an underlying encryption scheme and evaluates circuits gate by gate on encrypted wire values. When instantiated with OTP encryption, the paradigm recovers the GMW protocol; when instantiated with LHE, it yields our SFE framework. More generally, this paradigm enables the construction of a wide range of new SFE protocols by plugging in different encryption schemes.

5.1 An Encryption-Based SFE Paradigm

Our paradigm assumes the encryption scheme's key consists of both an encryption key ek and a decryption key dk, thereby capturing both symmetric-key cases (where encryption and decryption keys are identical) and public-key scenarios (where they differ). For each message to be encrypted, we employ a distinct key pair, enabling instantiation with OTP. Naturally, using multi-message encryption would simplify the paradigm's description, where a single key pair is used.

A Paradigm for Two-Party SFE from Encryption

Parameter: A circuit C over the ring $\mathcal{M}$. Let m^* be the number of multiplication gates in C.
Input: P_1 has $x_1 \in \mathcal{M}$. P_2 has $x_2 \in \mathcal{M}$.
Output: P_1 obtains $C(x_1, x_2)$.

Setup.

1. P_2 samples $m^* + 2$ key pairs $\{(\mathsf{ek}_j, \mathsf{dk}_j)\}_{j\in[m^*+2]}$.

Input encoding.

1. P_1 and P_2 perform *oblivious encryption*, enabling P_1 to receive an encryption $[\![x_1]\!]$ of x_1 under ek_{m^*+1}.[a]
2. P_2 encrypts x_2 to $[\![x_2]\!]$ under ek_{m^*+2} and sends $[\![x_2]\!]$ to P_1.

Circuit evaluation.

The parties compute the j-th gate as follows.

1. Let $[\![u]\!]$ and $[\![v]\!]$ be the input encryptions under ek_{j_1} and ek_{j_2} respectively.
2. The parties perform some protocol to have P_1 obtain an encryption $[\![u+v]\!]$ (for an addition gate) or $[\![uv]\!]$ (for a multiplication gate) under the key ek_j. During protocol execution, P_2 takes as input the decryption keys dk_{j_1} and dk_{j_2} corresponding to encryption keys ek_{j_1} and ek_{j_2} respectively.

Output decoding.

1. Let $[\![z]\!]$ be the ciphertext under ek corresponding to the output gate.

2. The parties perform *oblivious decryption*, enabling P_1 to obtain z. During protocol execution, P_2 takes as input the decryption key dk corresponding to encryption key ek.
3. P_1 outputs z.

[a] To protect the decryption key, we cannot directly send the encryption key to P_1, as in some schemes the encryption key may leak information about the decryption key. When using public-key encryption, P_2 can safely provide the public (encryption) key to P_1.

Instantiation with OTP. We can instantiate this paradigm using OTP over the ring $\mathcal{M}$, where to encrypt an element x, we sample a random element $x_2 \in \mathcal{M}$ as the encryption key and set the ciphertext as $x_1 = x - x_2$. Observe that (x_1, x_2) forms an additive sharing of x. This implies that each input and each gate output value are computed as additive sharings, which constitutes the core idea of GMW [GMW87].

Instantiation with LHE. Naturally, instantiating this paradigm with LHE yields our LHE-based SFE framework, and we omit the technical details.

Instantiation with SKE. We also consider instantiating this paradigm with a multi-message symmetric-key encryption (SKE) scheme like AES. Compared to OTP-based implementation, SKE-based realization can reduce P_2's storage to be independent of the circuit size (P_2 only needs a single encryption/decryption key). Compared to LHE-based implementation, SKE-based realization may potentially reduce the protocol's computational cost by eliminating P_1's need to perform homomorphic operations.

When constructing SFE using SKE, we must ensure that the output of every gate is represented as an SKE ciphertext. Let ek be the key of P_2, and let $[\![u]\!]$ and $[\![v]\!]$ denote the input ciphertexts. For addition gates, the parties evaluate the circuit $\mathsf{E}_{\mathsf{ek}}(\mathsf{D}_{\mathsf{ek}}([\![u]\!]) + \mathsf{D}_{\mathsf{ek}}([\![v]\!]))$, which yields a ciphertext of $u + v$. For multiplication gates, the parties compute $\mathsf{E}_{\mathsf{ek}}(\mathsf{D}_{\mathsf{ek}}([\![u]\!])\mathsf{D}_{\mathsf{ek}}([\![v]\!]))$, which yields a ciphertext of uv.

5.2 Supporting More Parties

While our SFE paradigm initially supports only two parties, we present two approaches to extend it to more parties. Recall that our paradigm uses an encryption-based sharing where P_1 holds the ciphertext of the value x and P_2 holds the decryption key dk (together with the encryption key ek).

Distributing values. The first approach involves distributing the values among multiple parties. Specifically, for n parties, a sharing of a value x is of form $([\![x_1]\!], x_2, x_3, \ldots, x_{n-1}, \mathsf{dk})$, with $(x_1, \ldots, x_{n-1})$ being an $(n-2, n-1)$-sharing of x. Apparently, any subset of $n-1$ parties cannot reconstruct the original value.

Distributing keys. The second approach involves distributing the decryption key among multiple parties. Specifically, for n participants, a value x is shared as $([\![x]\!], \mathsf{dk}_2, \ldots, \mathsf{dk}_n)$, where $(\mathsf{dk}_2, \ldots, \mathsf{dk}_n)$ form an $(n-2, n-1)$-sharing of dk. In this case, any $n-1$ parties cannot recover the original value. Note that this is the method adopted by our multiparty SFE framework in the previous section (Section 3).

Another interesting way for key distributing is to let each dk_i be an independent decryption key with a corresponding encryption key ek_i. A sharing of a value is $([\![x]\!], \mathsf{dk}_2, \ldots, \mathsf{dk}_n)$, where $[\![x]\!]$ is obtained by iteratively encrypting x $n-1$ times using the encryption keys $\mathsf{ek}_2, \ldots, \mathsf{ek}_n$.[2] While this method lacks efficiency due to requiring evaluation on $(n-1)$-layer ciphertexts, it remains worth investigating as a feasibility study, particularly since no strong evidence exists to rule out its potential efficiency.

[2] If the message space differs from the ciphertext space, we need a method to convert ciphertexts to plaintext elements after each encryption before further encryption.

6 An Alternative PFE Construction from SKE

We extend our SFE paradigm to the PFE setting. In particular, we consider instantiating the paradigm with SKE schemes. We show how to equip the resulting SFE construction with circuit privacy, thereby obtaining an alternative PFE construction with linear complexity. A key advantage of this approach is that it avoids LHE, and thus has the potential to achieve lower computational overhead.

6.1 PFE from SKE

As the starting point, we use an SKE scheme (E, D) as the underlying encryption scheme. We let the parties $P_2, \ldots, P_n$ hold an $(n-2, n-1)$-sharing of a symmetric key ek, and let P_1 hold the ciphertext of each secret under the key ek. Since the key ek can be reused for all secret values, in order to guarantee circuit privacy it suffices to ensure that the parties $P_2, \ldots, P_n$ cannot distinguish whether the evaluated gate is an addition gate or a multiplication gate. To achieve this goal, our idea is to let the parties compute the *same* function for both addition and multiplication gates, where P_1 use an additional private input bit b to indicate the gate type ($b = 0$ for a multiplication gate, and $b = 1$ for an addition gate). Let the input ciphertexts be $[\![u]\!]$ and $[\![v]\!]$, and let the symmetric key sharing be $\langle \mathsf{ek} \rangle$. We define this function as follows.

$$f((b, [\![u]\!], [\![v]\!]), \langle \mathsf{ek} \rangle) = \mathsf{E}_{\mathsf{ek}}((1-b)\mathsf{D}_{\mathsf{ek}}([\![u]\!])\mathsf{D}_{\mathsf{ek}}([\![v]\!]) + b(\mathsf{D}_{\mathsf{ek}}([\![u]\!]) + \mathsf{D}_{\mathsf{ek}}([\![v]\!]))).$$

Here, $(b, [\![u]\!], [\![v]\!])$ is the private input of P_1, while $\langle \mathsf{ek} \rangle$ is the private input of $P_2, \ldots, P_n$. We can verify that when $b = 0$, the output is a ciphertext of uv, and when $b = 1$, the output is a ciphertext of $u + v$. Since b is kept private, the parties $P_2, \ldots, P_n$ cannot determine whether the evaluated gate is an addition gate or a multiplication gate.

We remark that it is necessary to ensure that the underlying SKE scheme is IND-CPA secure. IND-CPA security can be achieved under the CTR mode [BDJR97]. Let $(\mathsf{E}', \mathsf{D}')$ be a basic SKE scheme. In the CTR mode, for the i-th plaintext x_i, the ciphertext is $c_i = \mathsf{E}_{\mathsf{ek}}(x_i) = x_i + \mathsf{E}'_{\mathsf{ek}}(\mathsf{ctr}_i)$, and decryption computes $x_i = \mathsf{D}_{\mathsf{ek}}(c_i) = c_i - \mathsf{E}'_{\mathsf{ek}}(\mathsf{ctr}_i)$, where ctr_i denotes the i-th nonce-counter value. It is important to note that, in our PFE construction, when evaluating a gate, only party P_1 knows the indices of the input ciphertexts. Therefore, ctr_i should be treated as a private value known to P_1, rather than a public value. It is clear that the multiplicative complexity of both CTR-mode encryption and decryption matches that of the basic SKE encryption algorithm E'.

6.2 Instantiation with MiMC

We give a concrete efficiency instantiation of our SKE-based PFE construction. We choose MiMC [AGR+16] as the underlying SKE scheme and use the CTR mode to achieve IND-CPA security. Note that MiMC is defined over large fields $\mathbb{F}_p$ and is therefore well-suited for arithmetic circuits. When evaluating Boolean circuits, we can instead instantiate the SKE scheme using LowMC [ARS+15].

According to the recommended parameters in [AGR+16], the number of rounds is $r = \lceil \log p / \log 3 \rceil$. To encrypt x under key k, MiMC fixes random constants $c_1, \ldots, c_{r-1}$ and computes $y_1 = (x + k)^3$ and $y_i = (y_{i-1} + c_{i-1} + k)^3$ for $i \in [2, r]$. The final ciphertext is $y = y_r$. We consider a prime p between 2^{128} and 2^{129}. Then, $r = \lceil \log p / \log 3 \rceil \leq 82$. Thus, the encryption procedure involves $2 \cdot 82 = 164$ multiplications. In our SKE-based PFE, each gate requires secure evaluation of a circuit consisting of one SKE encryption, two SKE decryptions, and three additional multiplications. Therefore, each gate requires $3 \cdot 164 + 3 = 495$ multiplications. Using the GMW protocol, assuming that the parties have already generated Beaver triples, each multiplication gate still requires the parties to communicate $4(n-1)$ elements (in order to open two additive sharings). Therefore, ignoring the communication cost of generating Beaver triples, the MiMC-based PFE construction requires $4(n-1) \cdot 495 = 1980(n-1)$ plaintext elements per gate.

Comparison with LHE-based construction. In our LHE-based PFE construction based on LHE, each gate requires the parties to send $12(n-1)$ group elements. In the BCP scheme, each group element is 2 times the size of a plaintext element. Therefore, the communication cost per multiplication is equivalent to transmitting $24(n-1)$ plaintext elements. Apparently, our LHE-based PFE incurs higher communication cost ($1980(n-1)$ v.s. $24(n-1)$). However, it avoids expensive homomorphic operations, and may therefore achieve lower computational overhead. In particular, more efficient SKE constructions could further improve the efficiency of our SKE-based PFE.

7 Semi-PFE: Revealing Topology or Depth

Our PFE protocol's primary limitation is its linear round complexity with respect to circuit size m, stemming from the sequential gate evaluation requirement where parties must compute one gate per round. If two gates are computed simultaneously, the parties know that the inputs of either gate does not depend on the output of the other gate, thereby leaking the topological information of the circuit.

We consider semi-PFE that trade partial circuit privacy for reduced round complexity. Specifically, we give semi-PFE protocols for both topology-revealing PFE (trPFE) and depth-revealing PFE (drPFE). The former allows leaking the circuit *topology*, while the latter only allows leaking the circuit *depth* d. Both our trPFE and drPFE achieve $O(d)$ round complexity. We also demonstrate that compared to standard PFE, drPFE reveals only a small amount of information, whereas trPFE leaks the majority of the circuit information.

7.1 Topology-Revealing PFE

By permitting circuit topology leakage, our PFE framework naturally yields a trPFE protocol with d rounds of interactions, achieved simply by having parties evaluate gates according to the circuit's topological ordering.

However, we emphasize that full topology disclosure isn't actually necessary to achieve this round complexity. In fact, revealing only the circuit depth d and the number $m_1, \ldots, m_d$ of gates per layer suffices, since in our PFE protocols, the parties without the circuit compute each gate using fixed inputs (a secret key sharing among them).

7.2 Depth-Revealing PFE

We prove that revealing only the circuit depth d suffices to achieve $O(d)$ round complexity. The key idea is straightforward: the parties compute $\lceil m/(\varepsilon d)\rceil$ gates simultaneously ($\varepsilon > 0$ is an adjustable constant). If P_1 observes that a layer contains a non-integer multiple of $\lceil m/(\varepsilon d)\rceil$ gates, the parties simply evaluate some dummy gates. For a circuit with m_j gates at the j-th layer, the total round complexity becomes

$$\sum_{j\in[d]} \lceil \frac{m_j}{\lceil m/(\varepsilon d)\rceil} \rceil < \sum_{j\in[d]} (\frac{m_j}{m/(\varepsilon d)} + 1) = (1+\varepsilon)d.$$

When ε is a constant, the protocol achieves $O(d)$ round complexity. Note that ε can effectively control the tradeoff between round complexity and computational overhead, since the number of computed gates is

$$\sum_{j\in[d]} \lceil m/(\varepsilon d)\rceil \cdot \lceil \frac{m_j}{\lceil m/(\varepsilon d)\rceil} \rceil < \sum_{j\in[d]} \lceil m/(\varepsilon d)\rceil (\frac{m_j}{\lceil m/(\varepsilon d)\rceil} + 1) = m + d\lceil m/(\varepsilon d)\rceil.$$

Apparently, the smaller ε will result in lower round complexity and a higher number of computed gates.

7.3 Semi-PFE from an Information Perspective

In this section, we consider semi-PFE protocols through an information perspective. Specifically, we quantify the information leakage in various semi-PFE primitives using the notion of entropy. This can provide a quantitative metric for circuit privacy.

We first recall a basic fact regarding circuit complexity from [Vol99] (see also in [Kat05]), which states that for circuits with m gates, the total number of possible circuits exceeds $O(m^m)$. This implies the amount of information that needs protection in PFE is at least $O(m \log m)$ bits. However, once the circuit topology is revealed, the number of possible circuits reduces to just 2^m, corresponding to only m bits of remaining information. That is, disclosing the circuit topology will reveal the majority of the circuit information.

In contrast, since the depth d ranges between 1 and m, revealing the depth leaks merely $\log m$ bits of information, which is a substantially smaller information leakage compared to leaking circuit topology.

Finally, recall that our trPFE protocol in fact only requires leaking the circuit depth d and the numbers $(m_1, \ldots, m_d)$ of gates in every layer. Note that there are at most $\binom{m-1}{d-1}$ possible $(m_1, \ldots, m_d)$ for a fixed d. Therefore, the total information leakage is bounded by $\log(\sum_{d\in[m]} \binom{m-1}{d-1}) = m - 1$ bits. While this leakage remains significantly larger than the $\log m$ bits revealed in drPFE, it is substantially smaller than the information disclosure from full topology exposure.

8 Optimizations

We present two optimizations for our SFE and PFE protocols: one reducing the overall communication cost in the two-party setting and the other improving the computational efficiency of the circuit evaluation stage in both the two-party and multiparty settings.

8.1 Achieving Better Overall Efficiency in the Two-Party Setting

As we have analyzed, when instantiating our protocol with the BCP cryptosystem, the communication cost for computing a multiplication gate in our two-party SFE and PFE is 10 group elements. We present an optimization that reduces the communication cost to 8 group elements.

Recall that the goal of computing a multiplication gate is to compute the ciphertext $[\![w]\!]$ from two ciphertexts $[\![u]\!]$ and $[\![v]\!]$ with $w = uv$. Our optimization starts from the following equation.

$$uv - r = (u - r_1)(v - r_2) + ur_2 + r_1v - r_1r_2 - r.$$

We denote $a = u - r_1, b = v - r_2$, and $c = ur_2 + r_1v - r_1r_2 - r$. If P_1 samples (r, r_1, r_2) as three random values, then P_1 can generate ciphertexts $([\![a]\!], [\![b]\!], [\![c]\!])$ using linear homomorphism. Specifically, (a, b, c) reveal no information about (u, v) since (r, r_1, r_2) are all random. This means P_1 can directly send the ciphertexts $([\![a]\!], [\![b]\!], [\![c]\!])$ to P_2. Even if P_2 decrypts them to obtain (a, b, c), no secret information is leaked. The advantage of this approach is that P_2 can compute the ciphertext of $[\![ab + c]\!]$ after decryption and return it to P_1. Then, P_1 can compute the ciphertext of $w = uv$ as follows.

$$[\![uv]\!] = [\![ab + c]\!] + r.$$

Now we analyze the communication cost of the above process. Note that our approach involves transmitting 4 ciphertexts: P_1 sends 3 ciphertexts to P_2, and P_2 returns 1 ciphertext. Since each ciphertext consists of 2 group elements, the total communication cost is 8 group elements.

8.2 Achieving Better Circuit Evaluation Efficiency

Our protocol requires oblivious decryption (described in Section 2.5) during the circuit evaluation stage. Compared to direct decryption where P_1 sends the ciphertexts to other parties, oblivious decryption necessitates an additional ciphertext operation ($[\![x]\!] - r$) from P_1. We present a method to eliminate oblivious decryption, thereby improving the protocol's computational efficiency of the circuit evaluation stage.

The purpose of oblivious decryption is to prevent the participants holding the secret key from learning the decrypted messages. In our protocol, each ciphertext to be decrypted is of the form $[\![u - a]\!]$, where u is the secret information while a is generated by the participants holding the secret key. The need to keep $u - a$ private stems from the fact that a is known to these participants.

By modifying the generation strategy of a so that all participants (including P_1) jointly generate them in a way that any coalition of $n - 1$ parties gains no information about a, we can eliminate the need to protect $u - a$. We present two concrete methods to achieve this.

In our protocols, a takes the form $a = a_2 + \cdots + a_n$, where each a_i is a random value sampled by P_i (note that in the two-party setting, $a = a_2$ is completely known to P_2). Our first approach modifies this by having P_1 contribute an additional share a_1, making $a = a_1 + \cdots + a_n$. This ensures that no information about a is revealed to the parties $P_2, \ldots, P_n$, though it slightly increases the setup cost.

Our second method assumes a special property of the encryption scheme called ciphertext obliviously samplability (COS), which allows sampling valid ciphertexts without knowledge of the corresponding plaintexts. Prior work [GKM+00] has shown that PKE with this property implies OT. Using the COS property, P_1 can directly sample an encrypted value $[\![a]\!]$, guaranteeing that no coalition of $n - 1$ parties learns any information about the plaintext a. Finally, we emphasize that the generation of ciphertext triples in this approach can be accomplished using the multiplication protocol in our protocols.

When combining COS with an encryption scheme supporting a single ciphertext multiplication, we can make the setup stage non-interactive (the parties still need to perform the distributed key generation process). Specifically, P_1 can directly generate the required ciphertext triple: first sampling $[\![a]\!]$ and $[\![b]\!]$ obliviously, then homomorphically computing $[\![ab]\!]$. Notably, many existing homomorphic encryption schemes (not limited to lattice-based constructions) [BGN05, MGH10, Gen09, BGV12, DPSZ12, CHKV25] already support this single-multiplication homomorphism.

9 PFE from Minimal Homomorphism

Our SFE and PFE protocols rely on an LHE scheme. LHE enables homomorphic evaluation of linear functions over ciphertexts. A natural question is whether our protocols can be based on weaker forms of homomorphic encryption.

We consider a minimal homomorphic PKE notion, namely rerandomizable PKE (RPKE) (see Section 2.3 for its formal definition), which allows the randomness in a ciphertext to be refreshed so that the resulting ciphertext is statistically indistinguishable from a freshly generated encryption of the same message. Interestingly, our SKE-based PFE construction implicitly suggests that RPKE can be used to build PFE with linear complexity, since RPKE implies both SKE and OT [LZL23]. However, such a construction requires non-black-box use of RPKE (where the parties must securely evaluate the encryption and decryption algorithms), and is therefore inefficient.

In this section, we prove that it suffices to use RPKE in a black-box manner to construct two-party SFE and PFE for Boolean circuits with linear complexity. To obtain our protocols, we first show that RPKE with a small (i.e., polynomial-size) message space implies a stronger form of homomorphism, which we call plain homomorphism. Informally, given a ciphertext $[\![x]\!]$ and any two plaintexts a, b, plain homomor-

phism allows one to generate a ciphertext $[\![ax + b]\!]$.[3] We defer the construction of a PKE scheme with plain homomorphism from RPKE to Appendix E. Note that plain homomorphism is weaker than linear homomorphism, since the latter additionally requires the ability to perform addition on ciphertexts.

9.1 Two-Party SFE from PKE with Plain Homomorphism

We first present our SFE scheme for Boolean circuits based on PKE with plain homomorphism. Our protocol consists of three stages: input encoding, circuit evaluation, and output decoding.

Input encoding. In this stage, P_2 generates a key pair $(\mathsf{pk}, \mathsf{sk})$ and uses pk to encrypt its input x_2 to $[\![x_2]\!]$, then sends $(\mathsf{pk}, [\![x_2]\!])$ to P_1, who encrypts its input x_1 to $[\![x_1]\!]$ using the received public key pk.

Circuit evaluation. We handle XOR and AND gates separately. For XOR gates, P_1 needs to compute the ciphertext $[\![u \oplus v]\!]$ from ciphertexts $[\![u]\!]$ and $[\![v]\!]$. However, plain homomorphism does not support direct XOR between ciphertexts. Our idea is to have P_1 sample two random bits r_1, r_2 and compute the ciphertexts $[\![u \oplus r_1]\!] \leftarrow [\![u]\!] \oplus r_1$ and $[\![v \oplus r_2]\!] \leftarrow [\![v]\!] \oplus r_2$. These two ciphertexts are sent to P_2, who decrypts them and returns an encryption of the XOR of decrypted plaintexts back to P_1. Consequently, P_1 obtains the ciphertext $[\![u \oplus v \oplus r_1 \oplus r_2]\!]$. Finally, P_1 computes $[\![w]\!] \leftarrow [\![u \oplus v \oplus r_1 \oplus r_2]\!] \oplus (r_1 \oplus r_2)$. It is straightforward to verify that $w = u \oplus v$. The privacy is also easy to prove. Since r_1 and r_2 are uniformly random bits, P_2 gains no additional information. Furthermore, as P_1 does not possess the secret key, it learns no information from the ciphertext returned by P_2.

For AND gates, P_1 needs to compute the ciphertext $[\![uv]\!]$ from ciphertexts $[\![u]\!]$ and $[\![v]\!]$. Our approach is inspired by the efficiency improvement approach in Section 8.1 and relies on the following equation.

$$uv \oplus r = (u \oplus r_1)(v \oplus r_2) \oplus (ur_2 \oplus r_3 \oplus r) \oplus (r_1 v \oplus r_1 r_2 \oplus r_3).$$

We have P_1 sample r, r_1, r_2, r_3 as four random bits and compute the corresponding ciphertexts

$$\begin{aligned}
[\![a]\!] &\leftarrow [\![u]\!] \oplus r_1 \\
[\![b]\!] &\leftarrow [\![v]\!] \oplus r_2 \\
[\![c_1]\!] &\leftarrow r_2[\![u]\!] \oplus r_3 \oplus r \\
[\![c_2]\!] &\leftarrow r_1[\![v]\!] \oplus r_1 r_2 \oplus r_3.
\end{aligned}$$

Apparently, this requires only plain homomorphism. P_1 then sends these ciphertexts to P_2. Upon receiving these ciphertexts, P_2 decrypts them to obtain a, b, c_1, c_2 and returns the ciphertext $[\![ab \oplus c_1 \oplus c_2]\!]$ to P_1. Finally, P_1 computes $[\![w]\!] \leftarrow [\![ab \oplus c_1 \oplus c_2]\!] \oplus r$. It is straightforward to verify that $w = uv$. The privacy guarantee follows directly since r, r_1, r_2, r_3 are all uniformly random bits, preventing P_2 from extracting any additional information from a, b, c_1, c_2. Moreover, as P_1 lacks the secret key, it cannot gain any extra information from the ciphertext returned by P_2.

Output decoding. After the circuit evaluation stage, P_1 obtains the output ciphertext $[\![z]\!]$. To decrypt this ciphertext, P_1 samples a random bit r and computes $[\![z']\!] \leftarrow [\![z]\!] \oplus r$. P_1 then sends the ciphertext $[\![z']\!]$ to P_2, who decrypts it and returns the decrypted result z' to P_1. Finally, P_1 outputs $z = z' \oplus r$. The correctness and privacy can be easily verified, and we omit them here.

9.2 Handling Circuit Privacy for PFE

To extend our SFE construction to the PFE setting, we have to make sure P_2 doesn't know whether they are computing an XOR gate or an AND gate. In the original SFE protocol, the distinction is evident since P_2 receives 2 ciphertexts for XOR gates versus 4 ciphertexts for AND gates.

[3] The term "plain" originates from the fact that it supports only evaluation between ciphertexts and *plaintexts*.

We first enforce uniform ciphertext count by having P_1 send 4 ciphertexts for both gate types. Since the 4 ciphertexts sent for an AND gate are all ciphertexts of random bits, the 2 dummy ciphertexts for an XOR gate must also be ciphertexts of random bits (as P_2 can decrypt these ciphertexts). The ciphertext generation proceeds as follows.

- **XOR gates.** Sample four random bits r, r_1, r_2, r_3, and the four ciphertexts are $(\llbracket r \rrbracket, \llbracket r_3 \rrbracket, \llbracket u \oplus r_1 \rrbracket, \llbracket v \oplus r_2 \rrbracket)$.
- **AND gates.** Sample four random bits r, r_1, r_2, r_3, and the four ciphertexts are $(\llbracket u \oplus r_1 \rrbracket, \llbracket v \oplus r_2 \rrbracket, \llbracket r_2 u \oplus r_3 \oplus r \rrbracket, \llbracket r_1 v \oplus r_1 r_2 \oplus r_3 \rrbracket)$.

Further, upon receiving these ciphertexts, P_2 should always compute the same expression. Specifically, assume these ciphertexts encrypt (a, b, c_1, c_2), we let P_2 return $\llbracket w' \rrbracket = \llbracket ab \oplus c_1 \oplus c_2 \rrbracket$ to P_1. To ensure correctness, P_1 computes the final ciphertext as follows.

- **XOR gates.** Compute $\llbracket w \rrbracket \leftarrow \llbracket w' \rrbracket \oplus (r r_3 \oplus r_1 \oplus r_2)$.
- **AND gates.** Compute $\llbracket w \rrbracket \leftarrow \llbracket w' \rrbracket \oplus r$.

We can verify that for XOR gates the output satisfies $w = u + v$, while for AND gates it holds that $w = uv$.

Remark 3. We have demonstrated that rerandomizable PKE can be used to construct two-party SFE and PFE for Boolean circuits with linear complexity. In fact, based on the results we obtained in Appendix E, our conclusion can be extended to arbitrary arithmetic circuits, provided the input space has polynomial size (in the security parameter).

10 Conclusion

In this work, we present new frameworks for SFE and PFE based on LHE. Our framework yields the first PFE protocol achieving linear complexity (in both circuit size and number of participants) that supports both Boolean and arithmetic circuits. A limitation of our protocol is its linear round complexity with respect to circuit size. To address this, we further demonstrate how to achieve round complexity linear in circuit depth by revealing either the topology or depth of the circuit. Specifically, we provide an information-theoretic argument showing that revealing circuit depth leaks only minimal information about the circuit, whereas disclosing circuit topology exposes significantly more. Our work leaves open the natural question of designing a multiparty PFE protocol achieving both *constant rounds* and *linear complexity*. Notably, this is not an open question in the context of SFE [GLM+24].

Another interesting topic arising from our work concerns the relationships between different notions of homomorphism. Our results in Appendix E show that for small message spaces, rerandomization (minimal homomorphism) implies plain homomorphism, which further implies the equivalence between additive and linear homomorphism. A natural direction is to explore similar (positive or negative) results when the message space has super-polynomial size.

References

[AF90] Martín Abadi and Joan Feigenbaum. Secure circuit evaluation. J. Cryptol., pages 1–12, 1990.

[AGKS20] Masaud Y. Alhassan, Daniel Gunther, Agnes Kiss, and Thomas Schneider. Efficient and scalable universal circuits. Journal of Cryptology, pages 1216–1271, 2020.

[AGR+16] Martin R. Albrecht, Lorenzo Grassi, Christian Rechberger, Arnab Roy, and Tyge Tiessen. Mimc: Efficient encryption and cryptographic hashing with minimal multiplicative complexity. In Jung Hee Cheon and Tsuyoshi Takagi, editors, ASIACRYPT, pages 191–219, 2016.

[ARS+15] Martin R. Albrecht, Christian Rechberger, Thomas Schneider, Tyge Tiessen, and Michael Zohner. Ciphers for MPC and FHE. In Elisabeth Oswald and Marc Fischlin, editors, EUROCRYPT, pages 430–454, 2015.

[BBKL19] Muhammed Ali Bingöl, Osman Biçer, Mehmet Sabir Kiraz, and Albert Levi. An efficient 2-party private function evaluation protocol based on half gates. Comput. J., pages 598–613, 2019.

[BBKL22] Osman Biçer, Muhammed Ali Bingöl, Mehmet Sabir Kiraz, and Albert Levi. Highly efficient and re-executable private function evaluation with linear complexity. IEEE Trans. Dependable Secur. Comput., pages 835–847, 2022.

[BCP03] Emmanuel Bresson, Dario Catalano, and David Pointcheval. A simple public-key cryptosystem with a double trapdoor decryption mechanism and its applications. In Chi-Sung Laih, editor, ASIACRYPT, pages 37–54, 2003.

[BDJR97] Mihir Bellare, Anand Desai, Eron Jokipii, and Phillip Rogaway. A concrete security treatment of symmetric encryption. In Proceedings 38th annual symposium on foundations of computer science, pages 394–403. IEEE, 1997.

[BDTZ16] Carsten Baum, Ivan Damgård, Tomas Toft, and Rasmus Winther Zakarias. Better preprocessing for secure multiparty computation. In Mark Manulis, Ahmad-Reza Sadeghi, and Steve A. Schneider, editors, ACNS, pages 327–345, 2016.

[BFK+09] Mauro Barni, Pierluigi Failla, Vladimir Kolesnikov, Riccardo Lazzeretti, Ahmad-Reza Sadeghi, and Thomas Schneider. Secure evaluation of private linear branching programs with medical applications. In Michael Backes and Peng Ning, editors, ESORICS, pages 424–439, 2009.

[BGN05] Dan Boneh, Eu-Jin Goh, and Kobbi Nissim. Evaluating 2-dnf formulas on ciphertexts. In Joe Kilian, editor, TCC, pages 325–341, 2005.

[BGV12] Zvika Brakerski, Craig Gentry, and Vinod Vaikuntanathan. (leveled) fully homomorphic encryption without bootstrapping. In Shafi Goldwasser, editor, Innovations in Theoretical Computer Science, pages 309–325, 2012.

[BPSW07] Justin Brickell, Donald E. Porter, Vitaly Shmatikov, and Emmett Witchel. Privacy-preserving remote diagnostics. In Peng Ning, Sabrina De Capitani di Vimercati, and Paul F. Syverson, editors, CCS, pages 498–507, 2007.

[Can01] Ran Canetti. Universally composable security: A new paradigm for cryptographic protocols. In FOCS, pages 136–145, 2001.

[CHKV25] Henry Corrigan-Gibbs, Alexandra Henzinger, Yael Tauman Kalai, and Vinod Vaikuntanathan. Somewhat homomorphic encryption from linear homomorphism and sparse LPN. In Serge Fehr and Pierre-Alain Fouque, editors, EUROCRYPT, pages 3–33, 2025.

[CL15] Guilhem Castagnos and Fabien Laguillaumie. Linearly homomorphic encryption from $$\mathsf {DDH}$$. In Kaisa Nyberg, editor, CT-RSA, pages 487–505, 2015.

[Coc01] Clifford C. Cocks. An identity based encryption scheme based on quadratic residues. In Bahram Honary, editor, Cryptography and Coding, 8th IMA International Conference, pages 360–363, 2001.

[DGS+23] Yann Disser, Daniel Günther, Thomas Schneider, Maximilian Stillger, Arthur Wigandt, and Hossein Yalame. Breaking the size barrier: Universal circuits meet lookup tables. In International Conference on the Theory and Application of Cryptology and Information Security, pages 3–37. Springer, 2023.

[DPSZ12] Ivan Damgård, Valerio Pastro, Nigel P. Smart, and Sarah Zakarias. Multiparty computation from somewhat homomorphic encryption. In Reihaneh Safavi-Naini and Ran Canetti, editors, CRYPTO, pages 643–662, 2012.

[Gam84] Taher El Gamal. A public key cryptosystem and a signature scheme based on discrete logarithms. In G. R. Blakley and David Chaum, editors, CRYPTO, pages 10–18, 1984.

[Gen09] Craig Gentry. Fully homomorphic encryption using ideal lattices. In Michael Mitzenmacher, editor, STOC, pages 169–178, 2009.

[GKM+00] Yael Gertner, Sampath Kannan, Tal Malkin, Omer Reingold, and Mahesh Viswanathan. The relationship between public key encryption and oblivious transfer. In FOCS, pages 325–335, 2000.

[GKS17] Daniel Günther, Ágnes Kiss, and Thomas Schneider. More efficient universal circuit constructions. In International Conference on the Theory and Application of Cryptology and Information Security, pages 443–470. Springer, 2017.

[GKSS19] Daniel Günther, Ágnes Kiss, Lukas Scheidel, and Thomas Schneider. Poster: Framework for semi-private function evaluation with application to secure insurance rate calculation. In Lorenzo Cavallaro, Johannes Kinder, XiaoFeng Wang, and Jonathan Katz, editors, CCS, pages 2541–2543, 2019.

[GLM+24] Vipul Goyal, Junru Li, Ankit Kumar Misra, Rafail Ostrovsky, Yifan Song, and Chenkai Weng. Dishonest majority constant-round MPC with linear communication from DDH. In Kai-Min Chung and Yu Sasaki, editors, ASIACRYPT, pages 167–199, 2024.

[GMW87] Oded Goldreich, Silvio Micali, and Avi Wigderson. How to play any mental game or A completeness theorem for protocols with honest majority. In Alfred V. Aho, editor, STOC, pages 218–229, 1987.

[GSSY24] Daniel Günther, Joachim Schmidt, Thomas Schneider, and Hossein Yalame. Fluent: A tool for efficient mixed-protocol semi-private function evaluation. In 2024 Annual Computer Security Applications Conference (ACSAC), pages 733–746. IEEE, 2024.

[HKRS20] Marco Holz, Ágnes Kiss, Deevashwer Rathee, and Thomas Schneider. Linear-complexity private function evaluation is practical. In Liqun Chen, Ninghui Li, Kaitai Liang, and Steve A. Schneider, editors, ESORICS, pages 401–420, 2020.

[JL23] Hanyu Jia and Xiangxue Li. PFE: linear active security, double-shuffle proofs, and low-complexity communication. In Chunpeng Ge and Moti Yung, editors, Inscrypt, pages 227–252, 2023.

[Kat05] Jonathan Katz. Lecture on circuit complexity. 2005.

[KM11] Jonathan Katz and Lior Malka. Constant-round private function evaluation with linear complexity. In Dong Hoon Lee and Xiaoyun Wang, editors, ASIACRYPT, pages 556–571, 2011.

[KS08] Vladimir Kolesnikov and Thomas Schneider. A practical universal circuit construction and secure evaluation of private functions. In International Conference on Financial Cryptography and Data Security, pages 83–97. Springer, 2008.

[KS16] Ágnes Kiss and Thomas Schneider. Valiant's universal circuit is practical. In Annual International Conference on the Theory and Applications of Cryptographic Techniques, pages 699–728. Springer, 2016.

[LWY22] Yi Liu, Qi Wang, and Siu-Ming Yiu. Making private function evaluation safer, faster, and simpler. In Goichiro Hanaoka, Junji Shikata, and Yohei Watanabe, editors, PKC, pages 349–378, 2022.

[LYZ+21] Hanlin Liu, Yu Yu, Shuoyao Zhao, Jiang Zhang, Wenling Liu, and Zhenkai Hu. Pushing the limits of valiant's universal circuits: Simpler, tighter and more compact. In Annual International Cryptology Conference, pages 365–394. Springer, 2021.

[LZL23] Shuaishuai Li, Cong Zhang, and Dongdai Lin. Oblivious transfer from rerandomizable pke. In International Conference on Information and Communications Security, pages 128–145. Springer, 2023.

[MGH10] Carlos Aguilar Melchor, Philippe Gaborit, and Javier Herranz. Additively homomorphic encryption with d-operand multiplications. In Tal Rabin, editor, CRYPTO, pages 138–154, 2010.

[MS13] Payman Mohassel and Seyed Saeed Sadeghian. How to hide circuits in MPC an efficient framework for private function evaluation. In Thomas Johansson and Phong Q. Nguyen, editors, EUROCRYPT, pages 557–574, 2013.

[MSS14] Payman Mohassel, Seyed Saeed Sadeghian, and Nigel P. Smart. Actively secure private function evaluation. In Palash Sarkar and Tetsu Iwata, editors, ASIACRYPT, pages 486–505, 2014.

[Pai99] Pascal Paillier. Public-key cryptosystems based on composite degree residuosity classes. In Jacques Stern, editor, EUROCRYPT, pages 223–238, 1999.

[PSS09] Annika Paus, Ahmad-Reza Sadeghi, and Thomas Schneider. Practical secure evaluation of semi-private functions. In Michel Abdalla, David Pointcheval, Pierre-Alain Fouque, and Damien Vergnaud, editors, ACNS, pages 89–106, 2009.

[Raz08] Ran Raz. Elusive functions and lower bounds for arithmetic circuits. In Cynthia Dwork, editor, STOC, pages 711–720, 2008.

[Reg05] Oded Regev. On lattices, learning with errors, random linear codes, and cryptography. In Harold N. Gabow and Ronald Fagin, editors, STOC, pages 84–93, 2005.

[SS08] Ahmad-Reza Sadeghi and Thomas Schneider. Generalized universal circuits for secure evaluation of private functions with application to data classification. In International Conference on Information Security and Cryptology, pages 336–353. Springer, 2008.

[SY10] Amir Shpilka and Amir Yehudayoff. Arithmetic circuits: A survey of recent results and open questions. Found. Trends Theor. Comput. Sci., pages 207–388, 2010.

[Val76] Leslie G. Valiant. Universal circuits (preliminary report). In Ashok K. Chandra, Detlef Wotschke, Emily P. Friedman, and Michael A. Harrison, editors, STOC, pages 196–203, 1976.

[Vol99] Heribert Vollmer. Introduction to Circuit Complexity - A Uniform Approach. 1999.

[XJL+23] Yongfeng Xu, Hanyu Jia, Xiangxue Li, Qiang Li, Yue Bao, and Xintian Hou. Constant-round multiparty private function evaluation with (quasi-)linear complexities. In Mehdi Tibouchi and Xiaofeng Wang, editors, ACNS, pages 115–142, 2023.

[Yao86] Andrew Chi-Chih Yao. How to generate and exchange secrets (extended abstract). In FOCS, pages 162–167, 1986.

[ZRE15] Samee Zahur, Mike Rosulek, and David Evans. Two halves make a whole - reducing data transfer in garbled circuits using half gates. In Elisabeth Oswald and Marc Fischlin, editors, EUROCRYPT, pages 220–250, 2015.

[ZYZL19] Shuoyao Zhao, Yu Yu, Jiang Zhang, and Hanlin Liu. Valiant's universal circuits revisited: an overall improvement and a lower bound. In International Conference on the Theory and Application of Cryptology and Information Security, pages 401–425. Springer, 2019.

A The Universal Composability Framework

Security in the universal composability (UC) framework [Can01] is formally defined through the computational indistinguishability between the real and ideal world executions. In both of these two worlds, an entity called environment $\mathcal{Z}$ is introduced.

Real world. The environment $\mathcal{Z}$ in the real world provides inputs to and collecting outputs from all parties $P_1, \ldots, P_n$. Moreover, $\mathcal{Z}$ instruct the adversary $\mathcal{A}$ to attack the protocol π. Finally, the environment $\mathcal{Z}$ outputs a bit indicating which world it lies in. The parties may additionally access to an ideal functionality $\mathcal{H}$.

Ideal world. In the ideal world, the environment $\mathcal{Z}$ still provide inputs to all parties and collecting their outputs. However, rather than executing actual protocols, parties now simply forward inputs from $\mathcal{Z}$ to an ideal functionality $\mathcal{F}$ and receive outputs from $\mathcal{F}$. These is also an ideal adversary $\mathcal{S}$, which emulates real-world adversary $\mathcal{A}$'s attack through by interacting with both $\mathcal{F}$ and $\mathcal{Z}$. Ultimately, $\mathcal{Z}$ terminates by outputting a single bit representing its judgment about whether it interacted with the real protocol or this idealized simulation.

Security definition. We denote the output of $\mathcal{Z}$ in the real world as $\mathsf{Real}^{\mathcal{H}}_{\pi,\mathcal{A},\mathcal{Z}}(z)$ and in the ideal world as $\mathsf{Ideal}^{\mathcal{H}}_{\mathcal{F},\mathcal{S},\mathcal{Z}}(z)$. Here, z is an auxiliary input of the environment.

Definition 1 (UC Security)**.** A protocol π is said to securely realize functionality $\mathcal{F}$ in the $\mathcal{H}$-hybrid model if for every PPT adversary $\mathcal{A}$, there exists a PPT simulator $\mathcal{S}$ such that the two distributions $\mathsf{Real}^{\mathcal{H}}_{\pi,\mathcal{A},\mathcal{Z}}(z)$ and $\mathsf{Ideal}^{\mathcal{H}}_{\mathcal{F},\mathcal{S},\mathcal{Z}}(z)$ are computationally indistinguishable.

B Public-Key Encryption and Its Homomorphic Variants

We recall the standard definitions of public-key encryption (PKE) and two of its homomorphic variants.

Definition 2 (Public-Key Encryption)**.** A public-key encryption scheme consists of three PPT algorithms $(\mathsf{Keygen}, \mathsf{Enc}, \mathsf{Dec})$.

- $\mathsf{Keygen}(1^\lambda)$. On input 1^λ, output a pair of keys $(\mathsf{pk}, \mathsf{sk})$.
- $\mathsf{Enc}(\mathsf{pk}, x)$. On input a public key pk and a message $x \in \mathcal{M}$, output a ciphertext e.
- $\mathsf{Dec}(\mathsf{sk}, e)$. On input a secret key sk and a ciphertext e, output a plaintext x.

These algorithms satisfy the following properties.

- **Correctness.** For all $\lambda \in \mathbb{N}, x \in \mathcal{M}$ and every $(\mathsf{pk}, \mathsf{sk})$ output by $\mathsf{Keygen}(1^\lambda)$, it holds that

  $$\Pr[\mathsf{Dec}(\mathsf{sk}, \mathsf{Enc}(\mathsf{pk}, x)) = x] = 1 - \mathsf{negl}(\lambda).$$

  The probability is taken over the internal coins of Enc.
- **IND-CPA security.** For any two messages x_0, x_1 and every $(\mathsf{pk}, \mathsf{sk})$ output by $\mathsf{Keygen}(1^\lambda)$, it holds that

  $$\mathsf{Enc}(\mathsf{pk}, x_0) \approx_c \mathsf{Enc}(\mathsf{pk}, x_1).$$

  The probability is taken over the internal coins of Enc.

Definition 3 (Rerandomizable PKE)**.** Rerandomizable PKE (RPKE) is a PKE scheme $(\mathsf{Keygen}, \mathsf{Enc}, \mathsf{Dec})$ equipped with an additional algorithm Rand. This algorithm satisfies that for any ciphertext e of a message x under a public key pk, it holds that

$$\mathsf{Rand}(\mathsf{pk}, e) \approx_s \mathsf{Enc}(\mathsf{pk}, x).$$

Definition 4 (Linearly-Homomorphic Encryption)**.** PKE with linear homomorphism, or simply linearly-homomorphic encryption (LHE), equips a PKE scheme $(\mathsf{Keygen}, \mathsf{Enc}, \mathsf{Dec})$ with an additional algorithm Lin. This algorithm satisfies that for any two ciphertexts e_1 and e_2 encrypting x and y respectively under a public key pk and a message $a \in \mathcal{M}$, it holds that

$$\mathsf{Lin}(\mathsf{pk}, a, e_1, e_2) \approx_s \mathsf{Enc}(\mathsf{pk}, ax + y).$$

RPKE can be viewed as homomorphic encryption with *minimal* homomorphism, meaning that it supports only evaluation of the identity function. RPKE can be instantiated with many schemes, such as those by ElGamal [Gam84], Cock [Coc01], or Regev [Reg05]. Further, LHE (and also RPKE) can be instantiated with the Paillier [Pai99], BCP [BCP03], or CL [CL15] scheme.

C The Distributed BCP Scheme

The distributed version of the BCP cryptosystem [BCP03] is described as follows.

Definition 5 (Distributed BCP Cryptosystem)**.** Select four distinct odd primes p, q, p', q' where $p = 2p'+1$ and $q = 2q' + 1$, with both p' and q' being λ-bit primes. Compute $N = pq$ and $N' = p'q'$. Let $\mathbb{G} = \mathrm{QR}_{N^2}$ denote the cyclic group of quadratic residues modulo N^2 with a generator g. Note that the order of g is NN'. Define $\mathbb{F}$ as the subgroup of $\mathbb{G}$ generated by $f = 1 + N$, noting that f has order N. For any element $f' \in \mathbb{F}$, its discrete logarithm base $1 + N$ can be efficiently computed as $\mathsf{DL}(f') = (f' \bmod N^2 - 1)/N$. The message space, ciphertext space, public key space, secret key space, and randomness space of the scheme are $\mathcal{M} = \mathbb{Z}_N, \mathcal{C} = \mathbb{G}^2, \mathcal{PK} = \mathbb{G}, \mathcal{SK} = \mathbb{Z}_{NN'}$, and $\mathcal{R} = \mathbb{Z}_{N^2}$.

- **Key generation.** Sample $\mathsf{sk} \leftarrow \mathcal{SK}$ and compute $\mathsf{pk} = g^{\mathsf{sk}}$. Return $(\mathsf{pk}, \mathsf{sk})$.
- **Encryption.** To encrypt a message $x \in \mathcal{M}$, sample $r \leftarrow \mathcal{R}$ and compute $e' = g^r, e'' = f^x \cdot \mathsf{pk}^r$. Return $e = (e', e'')$.
- **Decryption.** To decrypt a ciphertext $e = (e', e'') \in \mathcal{C}$, compute and return $x = \mathsf{DL}(e'' \cdot (e')^{-\mathsf{sk}})$.
- **Linear homomorphism.** For a message $a \in \mathcal{M}$ and two ciphertext $e_1 = (e'_1, e''_1), e_2 = (e'_2, e''_2)$, compute
$$(e', e'') = ((e'_1)^a \cdot e'_2, (e''_1)^a \cdot e''_2).$$
Sample $r \leftarrow \mathcal{R}$ and return $e = (e' \cdot g^r, e'' \cdot \mathsf{pk}^r)$.

Distributed key generation. In the multiparty setting, no parties can know N' since this would allow them to factorize N. Thus, we cannot let each party sample a random value from $\mathbb{Z}_{NN'}$ as their secret key share. Instead, each P_i samples sk_i from $\mathbb{Z}_{N^2}$ and sends $\mathsf{pk}_i = g^{\mathsf{sk}_i}$ to P_1. The resulting public key $\mathsf{pk} = \prod_{i\in[k]} \mathsf{pk}_i$ is statistically indistinguishable from a freshly generated one. This follows from the fact that the distribution

$$\{\mathsf{sk} : \mathsf{sk}' \leftarrow \mathbb{Z}_{N^2}, \mathsf{sk} = \mathsf{sk}' \bmod NN'\}.$$

is statistically close to the uniform distribution over $\mathbb{Z}_{NN'}$, since N is nearly a multiple of N'. Specifically, $(N \bmod N')/N' = 2/p' + 2/q' + 1/(p'q')$ is negligible in λ.

Note that the distributed key generation involves each party P_i sending a group element (i.e., pk_i) to P_1 and receiving the final public key, which takes $2(k-1)$ group elements of communication if there are k parties. Our protocols require $P_2, \ldots, P_n$ to perform distributed key generation while ultimately delivering the public key to P_1, and the communication cost is $2(n-2)+1 = 2n-3$ group elements.

Distributed decryption. Assume P_1 wants to decrypt a ciphertext $e = (e', e'')$ without learning the secret key. P_1 sends e' to all other parties. Each party P_i returns $d_i = (e')^{\mathsf{sk}_i}$. The decryption is

$e''((e')^{\mathsf{sk}_1} \prod_{i\in[2,k]} d_i)^{-1}$.[4] Note that $d_i = g^{r\cdot \mathsf{sk}_i}$ for a randomness r. The decisional Diffie–Hellman (DDH) assumption over $\mathbb{G}$ ensures that $(g^r, g^{\mathsf{sk}_i}, g^{r\cdot \mathsf{sk}_i})$ is computationally indistinguishable from the uniform distribution over $\mathbb{G}^3$. Thus, revealing d_i will not compromise the privacy of the secret key. This conclusion still holds even if all $\{d_i\}_i$ are revealed. Note that the communication cost of distributed decryption is $2(n-1)$ group elements.

D Missing Security Proofs

D.1 Proof of Theorem 1

Proof. **Correctness.** We prove that P_1 obtains the correct output. Note that the correctness of computing addition gates and output can be easily verified. We only need to prove that when computing a multiplication gate, the value w satisfies that $w = uv$. Note that

$$\begin{aligned} w =& c_j + sb_j + ta_j + st \\ =& c_j + (u - a_j)b_j + (v - b_j)a_j + (u - a_j)(v - b_j) \\ =& c_j - a_jb_j + ub_j + u(v - b_j) \\ =& uv. \end{aligned}$$

This completes the correctness proof.

Privacy. We consider two cases, depending on which party is corrupted.

Case 1: P_1 is corrupted. The simulator $\mathcal{S}$ receives the input x_1 from the environment. By transmitting it to the functionality computing C, $\mathcal{S}$ obtains the output $C(x_1, x_2)$. $\mathcal{S}$ emulates the protocol as follows.

1. *Simulation for setup.* Since P_2 provides no input during the setup stage, $\mathcal{S}$ can perfectly simulate the execution of this stage, generating the corresponding key pair $(\mathsf{pk}, \mathsf{sk})$ and ciphertext triples $\{[\![a_j]\!], [\![b_j]\!], [\![c_j]\!]\}_{j\in[m^*]}$).
2. *Simulation for input encoding.* Since P_1 does not know the secret key, $\mathcal{S}$ can simply encrypt 0 and simulate P_2 sending the resulting ciphertext to P_1.
3. *Simulation for circuit evaluation.* No simulation is required for addition gates since their evaluation is non-interactive. To simulate the multiplication gate evaluation, the simulator $\mathcal{S}$ simply samples two random values s and t and simulates the functionality $\mathcal{F}_{\mathsf{od}}$ by returning s and t to P_1.
4. *Simulation for output decoding.* The simulator $\mathcal{S}$ simply simulates the functionality $\mathcal{F}_{\mathsf{od}}$ by returning $C(x_1, x_2)$ to P_1.

We now prove the indistinguishability between the ideal world and the real world by examining the following hybrid executions.

- Hybrid_0. This hybrid is identical to the ideal world.
- Hybrid_1. This hybrid is identical to the ideal world, except that during the input encoding stage, the simulator $\mathcal{S}$ returns a ciphertext corresponding to the real input x_2 to P_1.
- Hybrid_2. This hybrid is identical to Hybrid_1, except that during the circuit evaluation stage, the simulator $\mathcal{S}$ (simulating the functionality $\mathcal{F}_{\mathsf{od}}$) returns the real computed result s, t to P_1.
- Hybrid_3. This hybrid is identical to Hybrid_2, except that during the output decoding stage, the simulator $\mathcal{S}$ (simulating the functionality $\mathcal{F}_{\mathsf{od}}$) returns the real output z to P_1. Note that Hybrid_3 is exactly the real world.

[4] Since the order of g is unknown to any party, we let P_1 first compute the group element $(e')^{\mathsf{sk}_1} \prod_{i\in[2,k]} d_i$ and then compute its inverse in the ring $\mathbb{Z}_{N^2}$ using the extended Euclidean algorithm. This approach requires only a single invocation of the extended Euclidean algorithm, as opposed to computing inverses for each d_i individually, thereby improving computational efficiency.

We need to prove that every two consecutive hybrids are computationally indistinguishable.

- $\mathsf{Hybrid}_0 \approx_c \mathsf{Hybrid}_1$. The indistinguishability between these two hybrids relies on the IND-CPA security of the underlying LHE scheme, which ensures that P_1 cannot distinguish between the ciphertext of the real input x_2 and the ciphertext of zero.
- $\mathsf{Hybrid}_1 \approx_c \mathsf{Hybrid}_2$. The difference between these two hybrids lies in the messages s and t. In Hybrid_1, s and t are random values, whereas in Hybrid_2, $s = u - a_j$ and $t = v - b_j$. Note that a_j and b_j are uniformly random and P_1 only knows their ciphertexts (without the secret key), hence the IND-CPA security of the underlying LHE scheme ensures that $s = u - a_j$ and $t = v - b_j$ are computationally indistinguishable from random. Thus, Hybrid_1 and Hybrid_2 are indistinguishable.
- $\mathsf{Hybrid}_2 \approx_c \mathsf{Hybrid}_3$. These two hybrids differ in the message that P_1 receives from the functionality $\mathcal{F}_{\mathsf{od}}$. In Hybrid_2, P_1 obtains the ideal output $C(x_1, x_2)$, whereas in Hybrid_3 (the real world), P_1 receives the real-world output z. However, the correctness of our protocol ensures that $z = C(x_1, x_2)$, making Hybrid_2 and Hybrid_3 indistinguishable.

Based on the above analysis, we conclude that the ideal world (i.e., Hybrid_0) and the real world (i.e., Hybrid_3) are computationally indistinguishable.
Case 2: P_2 is corrupted. No simulation is required for this case since in the real protocol, P_2 receives no messages from either P_1 or the functionality $\mathcal{F}_{\mathsf{od}}$. □

D.2 Proof of Theorem 2

Proof. **Correctness.** We first prove that each generated triple (a_j, b_j, c_j) satisfies $c_j = a_j b_j$, which follows directly from the equation below.

$$c_j = \sum_{i \in [2,n]} c_{j,i} = \sum_{i \in [2,n]} b_{j,i} a_j = b_j a_j.$$

The correctness of computing addition gates and output can be easily verified. The correctness proof for multiplication gates follows the same idea as in our two-party SFE protocol.
Privacy. Let $A \subseteq [n]$ be the set of corrupted parties and $H = [n] \backslash A$ be the set of honest parties. We consider two cases, depending on whether P_1 is corrupted.
Case 1: P_1 is corrupted. The simulator $\mathcal{S}$ receives the inputs $\{x_i\}_{i \in A}$ from the environment. By transmitting them to the functionality computing C, $\mathcal{S}$ obtains the output $C(x_1, \ldots, x_n)$. $\mathcal{S}$ emulates the protocol as follows.

1. *Simulation for setup.* Since the parties $P_2, \ldots, P_n$ provide no input during the setup stage, $\mathcal{S}$ can perfectly simulate the execution of this stage, generating the required key pair $(\mathsf{pk}, \mathsf{sk})$ and ciphertext triples $\{[\![a_j]\!], [\![b_j]\!], [\![c_j]\!]\}_{j \in [m^*]}$). Here, sk is shared among $P_2, \ldots, P_n$.
2. *Simulation for input encoding.* Since at least one honest party exists among $P_2, \ldots, P_n$, the corrupted parties learn no information about the secret key sk. Thus, for each $i \in A$, the simulator $\mathcal{S}$ can simply encrypt 0 and simulate P_i sending the resulting ciphertext to P_1.
3. *Simulation for circuit evaluation.* No simulation is required for addition gates since their evaluation is non-interactive. To simulate the multiplication gate evaluation, the simulator $\mathcal{S}$ simply samples two random values s and t and simulates the functionality $\mathcal{F}_{\mathsf{od}}$ by returning s and t to P_1.
4. *Simulation for output decoding.* The simulator $\mathcal{S}$ simply simulates the functionality $\mathcal{F}_{\mathsf{od}}$ by returning $C(x_1, \ldots, x_n)$ to P_1.

We now prove the indistinguishability between the ideal world and the real world by examining the following hybrid executions.

- Hybrid_0. This hybrid is identical to the ideal world.

- Hybrid_1. This hybrid is identical to the ideal world, except that during the input encoding stage, the simulator $\mathcal{S}$ returns ciphertexts corresponding to the real inputs $\{x_i\}_{i\in H}$ to P_1.
- Hybrid_2. This hybrid is identical to Hybrid_1, except that during the circuit evaluation stage, the simulator $\mathcal{S}$ (simulating the functionality $\mathcal{F}_{\mathsf{od}}$) returns the real computed result s, t to P_1.
- Hybrid_3. This hybrid is identical to Hybrid_2, except that during the output decoding stage, the simulator $\mathcal{S}$ (simulating the functionality $\mathcal{F}_{\mathsf{od}}$) returns the real output z to P_1. Note that Hybrid_3 is exactly the real world.

We need to prove that every two consecutive hybrids are computationally indistinguishable.

- $\mathsf{Hybrid}_0 \approx_c \mathsf{Hybrid}_1$. Since the corrupted parties learn no information about the secret key sk, the IND-CPA security of the underlying LHE scheme ensures that P_1 cannot distinguish between the ciphertext of the real inputs $\{x_i\}_{i\in H}$ and the ciphertexts of zero. Therefore, these two hybrids are computationally indistinguishable.
- $\mathsf{Hybrid}_1 \approx_c \mathsf{Hybrid}_2$. The difference between these two hybrids lies in the messages s and t. In Hybrid_1, s and t are random values, whereas in Hybrid_2, $s = u - a_j$ and $t = v - b_j$. Note that $a_j = \sum_{i\in[2,n]} a_{j,i}$ and $b_j = \sum_{i\in[2,n]} b_{j,i}$. Since at least one honest party P_k exists among $P_2, \ldots, P_n$, both a_j and b_j are uniformly random and the corrupted parties only know their ciphertexts. The IND-CPA security of the underlying LHE scheme ensures that $s = u - a_j$ and $t = v - b_j$ are computationally indistinguishable from random. Thus, Hybrid_1 and Hybrid_2 are indistinguishable.
- $\mathsf{Hybrid}_2 \approx_c \mathsf{Hybrid}_3$. These two hybrids differ in the message that P_1 receives from the functionality $\mathcal{F}_{\mathsf{od}}$. In Hybrid_2, P_1 obtains $C(x_1, \ldots, x_n)$, whereas in Hybrid_3 (the real world), P_1 receives the real-world output z. However, the correctness of our protocol ensures that $z = C(x_1, \ldots, x_n)$, making Hybrid_2 and Hybrid_3 indistinguishable.

Based on the above analysis, we conclude that the ideal world (i.e., Hybrid_0) and the real world (i.e., Hybrid_3) are computationally indistinguishable.

Case 2: P_1 is honest. In this case, the corrupted parties only receive message from the honest P_1 during the setup stage. However, in the setup stage, all parties provide no input, hence the simulator can perfectly simulate all messages received by corrupted parties. □

D.3 Proof of Theorem 3

Proof. **Correctness.** The correctness proof follows the same idea as that for Protocol 1 and is therefore omitted here.

Privacy. We consider two cases, depending on which party is corrupted.

Case 1: P_1 is corrupted. The simulator $\mathcal{S}$ receives the input x_1 and circuit C from the environment. By transmitting them to the ideal functionality computing C, $\mathcal{S}$ obtains the output $C(x_1, x_2)$. $\mathcal{S}$ emulates the protocol as follows.

1. *Simulation for setup.* Since P_2 provides no input during the setup stage, $\mathcal{S}$ can perfectly simulate the execution of this stage, generating the corresponding key pair $(\mathsf{pk}, \mathsf{sk})$ and ciphertext triples $\{[\![a_j]\!], [\![b_j]\!], [\![c_j]\!]\}_{j\in[m]}$).
2. *Simulation for input encoding.* Since P_1 does not know the secret key, $\mathcal{S}$ can simply encrypt 0 and simulate P_2 sending the resulting ciphertext to P_1.
3. *Simulation for circuit evaluation.* To simulate the gate evaluation, the simulator $\mathcal{S}$ assigns $s = t = 0$ if the computed gate is an addition gate and sets s, t to random values otherwise. Then, it simulates the functionality $\mathcal{F}_{\mathsf{od}}$ returning s and t to P_1.
4. *Simulation for output decoding.* The simulator $\mathcal{S}$ simply simulates the functionality $\mathcal{F}_{\mathsf{od}}$ by returning $C(x_1, x_2)$ to P_1.

We now prove the indistinguishability between the ideal world and the real world by examining the following hybrid executions.

- Hybrid_0. This hybrid is identical to the ideal world.
- Hybrid_1. This hybrid is identical to the ideal world, except that during the input encoding stage, the simulator $\mathcal{S}$ returns a ciphertext corresponding to the real input x_2 to P_1.
- Hybrid_2. This hybrid is identical to Hybrid_1, except that during the circuit evaluation stage, the simulator $\mathcal{S}$ (simulating the functionality $\mathcal{F}_{\mathsf{od}}$) returns the real computed result s, t to P_1.
- Hybrid_3. This hybrid is identical to Hybrid_2, except that during the output decoding stage, the simulator $\mathcal{S}$ (simulating the functionality $\mathcal{F}_{\mathsf{od}}$) returns the real output z to P_1. Note that Hybrid_3 is exactly the real world.

Notice that the indistinguishability proofs between Hybrid_0 and Hybrid_1 as well as Hybrid_2 and Hybrid_3 follow the same approach as the privacy proof for Protocol 1, hence we only need to demonstrate the indistinguishability between Hybrid_1 and Hybrid_2.

- $\mathsf{Hybrid}_1 \approx_c \mathsf{Hybrid}_2$. These two hybrids differ only in the messages s and t. In Hybrid_1, s and t are random values when the computed gate is a multiplication gate and $s = t = 0$ otherwise. In Hybrid_2, $s = u - a_j$ and $t = v - b_j$ when the computed gate is a multiplication gate and $s = t = 0$ otherwise. Therefore, we only need to prove the indistinguishability of (s, t) between Hybrid_1 and Hybrid_2 when the computed gate is a multiplication gate. Note that a_j and b_j are uniformly random and P_1 only knows their ciphertexts (without the secret key), hence the IND-CPA security of the underlying LHE scheme ensures that $s = u - a_j$ and $t = v - b_j$ are computationally indistinguishable from random. Thus, Hybrid_1 and Hybrid_2 are indistinguishable.

Therefore, we can conclude that the ideal world (i.e., Hybrid_0) and the real world (i.e., Hybrid_3) are computationally indistinguishable.
Case 2: P_2 is corrupted. No simulation is required for this case since in the real protocol, P_2 receives no messages from either P_1 or the functionality $\mathcal{F}_{\mathsf{od}}$. □

D.4 Proof of Theorem 4

Proof. **Correctness.** The correctness proof follows the same idea as that for Protocol 2 and is therefore omitted here.
Privacy. Let $A \subseteq [n]$ be the set of corrupted parties and $H = [n] \backslash A$ be the set of honest parties. We consider two cases, depending on whether P_1 is corrupted.
Case 1: P_1 is corrupted. The simulator $\mathcal{S}$ receives the inputs $\{x_i\}_{i \in A}$ and circuit C from the environment. By transmitting them to the functionality computing C, $\mathcal{S}$ obtains the output $C(x_1, \ldots, x_n)$. $\mathcal{S}$ emulates the protocol as follows.

1. *Simulation for setup.* Since the parties $P_2, \ldots, P_n$ provide no input during the setup stage, $\mathcal{S}$ can perfectly simulate the execution of this stage, generating the required key pair $(\mathsf{pk}, \mathsf{sk})$ and ciphertext triples $\{[\![a_j]\!], [\![b_j]\!], [\![c_j]\!]\}_{j \in [m]})$. Here, sk is shared among $P_2, \ldots, P_n$.
2. *Simulation for input encoding.* Since at least one honest party exists among $P_2, \ldots, P_n$, the corrupted parties learn no information about the secret key sk. Thus, for each $i \in A$, the simulator $\mathcal{S}$ can simply encrypt 0 and simulate P_i sending the resulting ciphertext to P_1.
3. *Simulation for circuit evaluation.* To simulate the gate evaluation, the simulator $\mathcal{S}$ assigns $s = t = 0$ if the computed gate is an addition gate and sets s, t to random values otherwise. Then, it simulates the functionality $\mathcal{F}_{\mathsf{od}}$ returning s and t to P_1.
4. *Simulation for output decoding.* The simulator $\mathcal{S}$ simply simulates the functionality $\mathcal{F}_{\mathsf{od}}$ by returning $C(x_1, \ldots, x_n)$ to P_1.

We now prove the indistinguishability between the ideal world and the real world by examining the following hybrid executions.

- Hybrid_0. This hybrid is identical to the ideal world.
- Hybrid_1. This hybrid is identical to the ideal world, except that during the input encoding stage, the simulator $\mathcal{S}$ returns ciphertexts corresponding to the real inputs $\{x_i\}_{i\in H}$ to P_1.
- Hybrid_2. This hybrid is identical to Hybrid_1, except that during the circuit evaluation stage, the simulator $\mathcal{S}$ (simulating the functionality $\mathcal{F}_{\mathsf{od}}$) returns the real computed result s, t to P_1.
- Hybrid_3. This hybrid is identical to Hybrid_2, except that during the output decoding stage, the simulator $\mathcal{S}$ (simulating the functionality $\mathcal{F}_{\mathsf{od}}$) returns the real output z to P_1. Note that Hybrid_3 is exactly the real world.

Notice that the indistinguishability proofs between Hybrid_0 and Hybrid_1 as well as Hybrid_2 and Hybrid_3 follow the same approach as the privacy proof for Protocol 2, hence we only need to demonstrate the indistinguishability between Hybrid_1 and Hybrid_2.

- $\mathsf{Hybrid}_1 \approx_c \mathsf{Hybrid}_2$. These two hybrids differ only in the messages s and t. In Hybrid_1, s and t are random values when the computed gate is a multiplication gate and $s = t = 0$ otherwise. In Hybrid_2, $s = u - a_j$ and $t = v - b_j$ when the computed gate is a multiplication gate and $s = t = 0$ otherwise. Therefore, we only need to prove the indistinguishability of (s, t) between Hybrid_1 and Hybrid_2 when the computed gate is a multiplication gate. Note that $a_j = \sum_{i\in[2,n]} a_{j,i}$ and $b_j = \sum_{i\in[2,n]} b_{j,i}$. Since at least one honest party P_k exists among $P_2, \ldots, P_n$, both a_j and b_j are uniformly random and the corrupted parties only know their ciphertexts. The IND-CPA security of the underlying LHE scheme ensures that $s = u - a_j$ and $t = v - b_j$ are computationally indistinguishable from random. Thus, Hybrid_1 and Hybrid_2 are indistinguishable.

Consequently, we can conclude that the ideal world (i.e., Hybrid_0) and the real world (i.e., Hybrid_3) are computationally indistinguishable.

Case 2: P_1 is honest. In this case, the corrupted parties only receive message from the honest P_1 during the setup stage. However, in the setup stage, all parties provide no input, hence the simulator can perfectly simulate all messages received by corrupted parties. □

E Rerandomization Implies Plain Homomorphism

We first consider the case where the plaintexts are bits, and prove the following result.

Theorem 5. *Assume the existence of an RPKE scheme, then there is a bit-PKE scheme that supports plain homomorphism.*

Proof. Let $(\mathsf{Keygen}, \mathsf{Enc}, \mathsf{Dec}, \mathsf{Rand})$ be an RPKE scheme. We assume its message space contains 0 and 1.[5] We construct a bit-PKE scheme $(\mathsf{Keygen}', \mathsf{Enc}', \mathsf{Dec}')$ as follows.

- $\mathsf{Keygen}'(1^\lambda)$. Run $\mathsf{Keygen}(1^\lambda)$ to generate a key pair $(\mathsf{pk}, \mathsf{sk})$.
- $\mathsf{Enc}'(\mathsf{pk}, x)$. Run $e_1 \leftarrow \mathsf{Enc}(\mathsf{pk}, x)$ and $e_2 \leftarrow \mathsf{Enc}(\mathsf{pk}, x \oplus 1)$ and output the ciphertext pair (e_1, e_2).
- $\mathsf{Dec}'(\mathsf{sk}, (e_1, e_2))$. Run $x \leftarrow \mathsf{Dec}(\mathsf{sk}, e_1)$ and output x.

Given a ciphertext $[\![x]\!] = (e_1, e_2)$ and two plaintext bits a, b, we can generate a ciphertext of $ax \oplus b$ as follows.

1. **Generating $[\![ax]\!]$.** If $a = 0$, compute $(e_1', e_2') \leftarrow (\mathsf{Enc}(\mathsf{pk}, 0), \mathsf{Enc}(\mathsf{pk}, 1))$. Otherwise, set $(e_1', e_2') \leftarrow (e_1, e_2)$.

[5] As long as the message space contains at least two messages, we can always encode any two distinct messages as 0 and 1 respectively.

2. **Generating $[\![ax \oplus b]\!]$.** Output $(\mathsf{Rand}(\mathsf{pk}, e'_1), \mathsf{Rand}(\mathsf{pk}, e'_2))$ if $b = 0$ and $(\mathsf{Rand}(\mathsf{pk}, e'_2), \mathsf{Rand}(\mathsf{pk}, e'_1))$ otherwise.

It is straightforward to verify that the final output ciphertext is indeed an encryption of $ax \oplus b$. Moreover, the IND-CPA security of the scheme naturally follows from the IND-CPA security of the underlying RPKE scheme. □

Achieving larger message space. The above scheme supports only bit operations. We generalize it to support operations over any ring $\mathcal{M}$. However, the resulting scheme is only secure and efficient when the size of $\mathcal{M}$ is polynomial in λ (that is, the messages are short). We have the following theorem.

Theorem 6. *Assume the existence of an RPKE scheme with message space $\mathcal{M}$ of size polynomial in λ, then there is a PKE scheme with the same message space that supports plain homomorphism. In particular, the ciphertext size is $O(|\mathcal{M}|^2)$.*

Proof. Let $(\mathsf{Keygen}, \mathsf{Enc}, \mathsf{Dec}, \mathsf{Rand})$ be an RPKE scheme with message space $\mathcal{M}$ being a ring. Let $\mathcal{M}' = \mathcal{M} \backslash \{0\}$. We construct a PKE scheme $(\mathsf{Keygen}', \mathsf{Enc}', \mathsf{Dec}')$ with message space $\mathcal{M}$ as follows.

- $\mathsf{Keygen}'(1^\lambda)$. Run $\mathsf{Keygen}(1^\lambda)$ to generate a key pair $(\mathsf{pk}, \mathsf{sk})$.
- $\mathsf{Enc}'(\mathsf{pk}, x)$. Run $e_{i,j} \leftarrow \mathsf{Enc}(\mathsf{pk}, ix + j)$ for all $(i, j) \in \mathcal{M}' \times \mathcal{M}$ and output the ciphertext matrix $(e_{i,j})_{(i,j)\in\mathcal{M}'\times\mathcal{M}}$.
- $\mathsf{Dec}'(\mathsf{sk}, (e_{i,j})_{(i,j)\in\mathcal{M}'\times\mathcal{M}})$. Run $x \leftarrow \mathsf{Dec}(\mathsf{sk}, e_{1,0})$ and output x.

Given a ciphertext $[\![x]\!] = (e_{i,j})_{(i,j)\in\mathcal{M}'\times\mathcal{M}}$ and two plaintexts $a, b \in \mathcal{M}$, we can generate a ciphertext of $ax + b$ as follows.

1. Compute $e_{0,j} \leftarrow \mathsf{Enc}(\mathsf{pk}, j)$ for all $j \in \mathcal{M}$.
2. **Generating $[\![ax]\!]$.** Set $e'_{i,j} \leftarrow e_{ia,j}$ for all $(i, j) \in \mathcal{M}' \times \mathcal{M}$.
3. **Generating $[\![ax + b]\!]$.** Compute $e''_{i,j} \leftarrow \mathsf{Rand}(\mathsf{pk}, e'_{i,j+ib})$ for all $(i, j) \in \mathcal{M}' \times \mathcal{M}$ and output $(e''_{i,j})_{(i,j)\in\mathcal{M}'\times\mathcal{M}}$.

We prove that $(e''_{i,j})_{(i,j)\in\mathcal{M}'\times\mathcal{M}}$ is indeed a ciphertext of $ax + b$ under the new PKE scheme. Note that under the RPKE scheme, $e''_{i,j}$ and $e'_{i,j+ib}$ encrypt the same message, and $e'_{i,j+ib}$ and $e_{ia,j+ib}$ also encrypt the same message. Since $e_{ia,j+ib}$ is an RPKE ciphertext of $iax + j + ib = i(ax + b) + j$, it follows that $e''_{i,j}$ is also an RPKE ciphertext of $i(ax + b) + j$. Therefore, $(e''_{i,j})_{(i,j)\in\mathcal{M}'\times\mathcal{M}}$ is a ciphertext of $ax + b$ under the new PKE scheme. Finally, the IND-CPA security of the scheme naturally follows from the IND-CPA security of the underlying RPKE scheme when $\mathcal{M}$ is of size polynomial in λ. □

For clarity of exposition, we describe the resulting PKE scheme supporting plain homomorphism with message space $\mathcal{M} = \mathbb{Z}_3$.

Example 1 (PKE with Plain Homomorphism over $\mathbb{Z}_3$)**.** Let $(\mathsf{Keygen}, \mathsf{Enc}, \mathsf{Dec}, \mathsf{Rand})$ be an RPKE scheme with message space $\mathbb{Z}_3$. Our PKE scheme supporting plain homomorphism with message space $\mathbb{Z}_3$ is given as follows.

- $\mathsf{Keygen}'(1^\lambda)$. Run $\mathsf{Keygen}(1^\lambda)$ to generate a key pair $(\mathsf{pk}, \mathsf{sk})$.
- $\mathsf{Enc}'(\mathsf{pk}, x)$. Run $e_{i,j} \leftarrow \mathsf{Enc}(\mathsf{pk}, ix + j)$ for all $(i, j) \in \mathbb{Z}_3^* \times \mathbb{Z}_3$ and output the ciphertext matrix

$$\begin{pmatrix} e_{1,0} & e_{1,1} & e_{1,2} \\ e_{2,0} & e_{2,1} & e_{2,2} \end{pmatrix}.$$

- $\mathsf{Dec}'(\mathsf{sk}, \left(\begin{smallmatrix} e_{1,0} & e_{1,1} & e_{1,2} \\ e_{2,0} & e_{2,1} & e_{2,2} \end{smallmatrix}\right))$. Run $x \leftarrow \mathsf{Dec}(\mathsf{sk}, e_{1,0})$ and output x.

Given a ciphertext $[\![x]\!] = \left(\begin{smallmatrix} e_{1,0} & e_{1,1} & e_{1,2} \\ e_{2,0} & e_{2,1} & e_{2,2} \end{smallmatrix}\right)$ and two plaintexts $a, b \in \mathcal{M}$, an encryption of $ax + b$ is generated as follows.

1. Compute $e_{0,j} \leftarrow \mathsf{Enc}(\mathsf{pk}, j)$ for all $j \in \mathbb{Z}_3$.

2. Output

$$\begin{pmatrix} \mathsf{Rand}(\mathsf{pk}, e_{a,b}) & \mathsf{Rand}(\mathsf{pk}, e_{a,b+1}) & \mathsf{Rand}(\mathsf{pk}, e_{a,b-1}) \\ \mathsf{Rand}(\mathsf{pk}, e_{-a,-b}) & \mathsf{Rand}(\mathsf{pk}, e_{-a,-b+1}) & \mathsf{Rand}(\mathsf{pk}, e_{-a,-b-1}) \end{pmatrix}.$$

Trading homomorphism for reduced ciphertext size. Our RPKE-based PKE scheme supporting plain homomorphism has ciphertext size $O(|\mathcal{M}|^2)$, where $\mathcal{M}$ is the message space. Plain homomorphism allows computing ciphertext $[\![ax + b]\!]$ from ciphertext $[\![x]\!]$ and plaintexts a, b. That is, plain homomorphism supports both *ciphertext-plaintext multiplication* and *ciphertext-plaintext addition*. We define *semi-plain homomorphism* as supporting either ciphertext-plaintext multiplication or ciphertext-plaintext addition, but not requiring both operations simultaneously. We prove that requiring only semi-plain homomorphism can reduce the ciphertext size to $O(|\mathcal{M}|)$. Specifically, we obtain the following result.

Theorem 7. *Assume the existence of an RPKE scheme with message space $\mathcal{M}$ of size polynomial in λ, then there is a PKE scheme with the same message space that supports semi-plain homomorphism. In particular, the ciphertext size is $O(|\mathcal{M}|)$.*

Proof. Let $(\mathsf{Keygen}, \mathsf{Enc}, \mathsf{Dec}, \mathsf{Rand})$ be an RPKE scheme with message space $\mathcal{M}$. Let $\mathcal{M}' = \mathcal{M} \backslash \{0\}$. We first present a PKE scheme $(\mathsf{Keygen}', \mathsf{Enc}', \mathsf{Dec}')$ and prove that it supports ciphertext-plaintext multiplication.

- $\mathsf{Keygen}'(1^\lambda)$. Run $\mathsf{Keygen}(1^\lambda)$ to generate a key pair $(\mathsf{pk}, \mathsf{sk})$.
- $\mathsf{Enc}'(\mathsf{pk}, x)$. Run $e_i \leftarrow \mathsf{Enc}(\mathsf{pk}, ix)$ for all $i \in \mathcal{M}'$ and output the ciphertext vector $(e_i)_{i \in \mathcal{M}'}$.
- $\mathsf{Dec}'(\mathsf{sk}, (e_i)_{i \in \mathcal{M}'})$. Run $x \leftarrow \mathsf{Dec}(\mathsf{sk}, e_1)$ and output x.

Given a ciphertext $[\![x]\!] = (e_i)_{i \in \mathcal{M}'}$ and a plaintext $a \in \mathcal{M}$, we can generate a ciphertext of ax as follows.

1. Compute $e_0 \leftarrow \mathsf{Enc}(\mathsf{pk}, 0)$.
2. Compute $e'_i \leftarrow \mathsf{Rand}(\mathsf{pk}, e_{ia})$ for all $i \in \mathcal{M}'$ and output $(e'_i)_{i \in \mathcal{M}'}$.

We prove that $(e'_i)_{i \in \mathcal{M}'}$ is indeed a ciphertext of ax under the new PKE scheme. Note that under the RPKE scheme, e'_i and e_{ia} encrypt the same message, i.e., $i(ax)$. Therefore, $(e'_i)_{i \in \mathcal{M}'}$ is a ciphertext of ax under the new PKE scheme.

We further give a PKE scheme $(\mathsf{Keygen}'', \mathsf{Enc}'', \mathsf{Dec}'')$ and prove that it supports ciphertext-plaintext addition.

- $\mathsf{Keygen}''(1^\lambda)$. Run $\mathsf{Keygen}(1^\lambda)$ to generate a key pair $(\mathsf{pk}, \mathsf{sk})$.
- $\mathsf{Enc}''(\mathsf{pk}, x)$. Run $e_j \leftarrow \mathsf{Enc}(\mathsf{pk}, x + j)$ for all $j \in \mathcal{M}$ and output the ciphertext vector $(e_j)_{j \in \mathcal{M}}$.
- $\mathsf{Dec}''(\mathsf{sk}, (e_j)_{j \in \mathcal{M}})$. Run $x \leftarrow \mathsf{Dec}(\mathsf{sk}, e_0)$ and output x.

Given a ciphertext $[\![x]\!] = (e_j)_{j \in \mathcal{M}}$ and a plaintext $b \in \mathcal{M}$, we can generate a ciphertext of $x + b$ as follows.

1. Compute $e''_j \leftarrow \mathsf{Rand}(\mathsf{pk}, e_{j+b})$ for all $j \in \mathcal{M}$ and output $(e''_j)_{j \in \mathcal{M}}$.

We prove that $(e''_j)_{j \in \mathcal{M}}$ is a ciphertext of $x + b$ under the new PKE scheme. Note that under the RPKE scheme, e''_j and e_{j+b} encrypt the same message, i.e., $(x + b) + j$. Therefore, $(e''_j)_{j \in \mathcal{M}}$ is a ciphertext of $x + b$ under the new PKE scheme. □

Remark 4 (On Additive and Linear Homomorphism). Recall that our SFE and PFE frameworks rely on a linearly homomorphic PKE scheme. Informally, linear homomorphism enables generating a ciphertext $[\![ax+y]\!]$ from a ciphertext pair $([\![x]\!], [\![y]\!])$ and a plaintext a. In some literature, this property is also referred to as "additive homomorphism". However, such terminology is not strictly accurate. Syntactically, additive homomorphism only requires ciphertext-ciphertext addition, i.e., the capability to generate a ciphertext $[\![x + y]\!]$ from a ciphertext pair $([\![x]\!], [\![y]\!])$. That is, additive homomorphism does not inherently support ciphertext-plaintext multiplication. Of course, additive homomorphism does support *ciphertext-integer* multiplication. However, plaintexts cannot always be treated as integers. For instance, assume plaintexts are degree-k polynomials over $\mathbb{Z}_p$ ($k > 0$). Ciphertext-plaintext multiplication allows ciphertexts

to be multiplied by *arbitrary* plaintext polynomials. Nevertheless, ciphertext-integer multiplication only supports multiplication between ciphertexts and elements in $\mathbb{Z}_p$ (i.e., degree-zero plaintext polynomials). Therefore, we can say that the equivalence between additive and linear homomorphism remains unclear. Specifically, while linear homomorphism necessarily implies additive homomorphism, the converse does not hold in general.

Surprisingly, our results obtained in this section actually demonstrate that additive and linear homomorphism are equivalent when the message space is small (of size polynomial in the security parameter λ). This follows because additive homomorphism naturally implies rerandomization, and our results further demonstrate that rerandomization enables ciphertext-plaintext multiplication. Consequently, additive homomorphism enables ciphertext-plaintext multiplication and thus implies linear homomorphism. That is, additive and linear homomorphism are equivalent.